# OGC WPS 2.0 REST / JSON Binding Extension

# Table of Contents

| |
|---|
| **Open Geospatial Consortium** |
| Submission Date: <yyyy-mm-dd> |
| Approval Date: <yyyy-mm-dd> |
| Publication Date: <yyyy-mm-dd> |
| External identifier of this OGC® document: http://www.opengis.net/doc/IS/wps-rest/1.0 |
| Internal reference number of this OGC® document: 18-062 |
| Version: 1.0-draft.2 |
| Category: OGC® Implementation Specification |
| Editor: Benjamin Pross |

# OGC WPS 2.0 REST/JSON Binding Extension



## Warning

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

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

| |
|---|
| Document type: OGC® Standard |
| Document subtype: Interface |
| Document stage: Draft |
| Document language: English |

License Agreement

Permission is hereby granted by the Open Geospatial Consortium, ("Licensor"), free of charge and subject to the terms set forth below, to any person obtaining a copy of this Intellectual Property and any associated documentation, to deal in the Intellectual Property without restriction (except as set forth below), including without limitation the rights to implement, use, copy, modify, merge, publish, distribute, and/or sublicense copies of the Intellectual Property, and to permit persons to whom the Intellectual Property is furnished to do so, provided that all copyright notices on the intellectual property are retained intact and that each person to whom the Intellectual Property is furnished agrees to the terms of this Agreement.

If you modify the Intellectual Property, all copies of the modified Intellectual Property must include, in addition to the above copyright notice, a notice that the Intellectual Property includes modifications that have not been approved or adopted by LICENSOR.

THIS LICENSE IS A COPYRIGHT LICENSE ONLY, AND DOES NOT CONVEY ANY RIGHTS UNDER ANY PATENTS THAT MAY BE IN FORCE ANYWHERE IN THE WORLD.

THE INTELLECTUAL PROPERTY IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE, AND NONINFRINGEMENT OF THIRD PARTY RIGHTS. THE COPYRIGHT HOLDER OR HOLDERS INCLUDED IN THIS NOTICE DO NOT WARRANT THAT THE FUNCTIONS CONTAINED IN THE INTELLECTUAL PROPERTY WILL MEET YOUR REQUIREMENTS OR THAT THE OPERATION OF THE INTELLECTUAL PROPERTY WILL BE UNINTERRUPTED OR ERROR FREE. ANY USE OF THE INTELLECTUAL PROPERTY SHALL BE MADE ENTIRELY AT THE USER'S OWN RISK. IN NO EVENT SHALL THE COPYRIGHT HOLDER OR ANY CONTRIBUTOR OF INTELLECTUAL PROPERTY RIGHTS TO THE INTELLECTUAL PROPERTY BE LIABLE FOR ANY CLAIM, OR ANY DIRECT, SPECIAL, INDIRECT OR CONSEQUENTIAL DAMAGES, OR ANY DAMAGES WHATSOEVER RESULTING FROM ANY ALLEGED INFRINGEMENT OR ANY LOSS OF USE, DATA OR PROFITS, WHETHER IN AN ACTION OF CONTRACT, NEGLIGENCE OR UNDER ANY OTHER LEGAL THEORY, ARISING OUT OF OR IN CONNECTION WITH THE IMPLEMENTATION, USE, COMMERCIALIZATION OR PERFORMANCE OF THIS INTELLECTUAL PROPERTY.

This license is effective until terminated. You may terminate it at any time by destroying the Intellectual Property together with all copies in any form. The license will also terminate if you fail to comply with any term or condition of this Agreement. Except as provided in the following sentence, no such termination of this license shall require the termination of any third party end-user sublicense to the Intellectual Property which is in force as of the date of notice of such termination. In addition, should the Intellectual

Property, or the operation of the Intellectual Property, infringe, or in LICENSOR's sole opinion be likely to infringe, any patent, copyright, trademark or other right of a third party, you agree that LICENSOR, in its sole discretion, may terminate this license without any compensation or liability to you, your licensees or any other party. You agree upon termination of any kind to destroy or cause to be destroyed the Intellectual Property together with all copies in any form, whether held by you or by any third party.

Except as contained in this notice, the name of LICENSOR or of any other holder of a copyright in all or part of the Intellectual Property shall not be used in advertising or otherwise to promote the sale, use or other dealings in this Intellectual Property without prior written authorization of LICENSOR or such copyright holder. LICENSOR is and shall at all times be the sole entity that may authorize you or any third party to use certification marks, trademarks or other special designations to indicate compliance with any LICENSOR standards or specifications. This Agreement is governed by the laws of the Commonwealth of Massachusetts. The application to this Agreement of the United Nations Convention on Contracts for the International Sale of Goods is hereby expressly excluded. In the event any provision of this Agreement shall be deemed unenforceable, void or invalid, such provision shall be modified so as to make it valid and enforceable, and as so modified the entire Agreement shall remain in full force and effect. No decision, action or inaction by LICENSOR shall be construed to be a waiver of any rights or remedies available to it.

## i. Abstract

In many cases geospatial or location data, including data from sensors, must be processed before the information can be used effectively. The OGC Web Processing Service (WPS) Interface Standard provides a standard interface that simplifies the task of making simple or complex computational processing services accessible via web services. Such services include well-known processes found in GIS software as well as specialized processes for spatio-temporal modeling and simulation. While the OGC WPS standard was designed with spatial processing in mind, it can also be used to readily insert non-spatial processing tasks into a web services environment. The WPS standard provides a robust, interoperable, and versatile protocol for process execution on web services. It supports both immediate processing for computational tasks that take little time and asynchronous processing for more complex and time consuming tasks. Moreover, the WPS standard defines a general process model that is designed to provide an interoperable description of processing functions. It is intended to support process cataloguing and discovery in a distributed environment. The OGC WPS REST/JSON Binding extension builds on the WPS 2.0 standard and defines the processing standards to communicate over a RESTful protocol using JSON encodings. This binding definition will be a newer and more modern way of programming and interacting with resources over the web while allowing better integration into existing software packages.

## ii. Keywords

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

geoprocessing, ogcdoc, OGC document, processes, WPS, REST, JSON

## iii. Preface

This extension is a continuation of WPS 2.0, a standard for web-based processing of geospatial data. It defines how the interfaces for WPS 2.0 operations should be constructed and interpreted using a REST based protocol with JSON encoding. Within the current version of WPS 2.0, bindings are defined for HTTP/POST using XML encodings and HTTP/GET using KVP encodings. Also in the current WPS 2.0 standard, a core conceptual model is provided that may be used to specify a WPS in different architectures such as REST or SOAP. Therefore, this extension is a natural fit to what is already defined in the standard.

## iv. Submitting organizations

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

- 52°North GmbH

- Intergraph Corporation (Hexagon Geospatial)

**v. Submitters**

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

| Name | Representing | OGC Member |
|---|---|---|
| Benjamin Pross | 52°North GmbH | Yes |
| Stan Tillman | Intergraph Corporation (Hexagon Geospatial) | Yes |

# Chapter 1. Scope

This document specifies the interface to a general-purpose Web Processing Service (WPS) using a RESTful protocol transporting JSON encoded requests and responses. A WPS is a web service that enables the execution of computing processes and the retrieval of metadata describing their purpose and functionality. Typically, these processes combine raster, vector, and/or coverage data with well-defined algorithms to produce new raster, vector, and/or coverage information. The document is an extension to the OGC WPS 2.0 Interface Standard [14-065]. It should be considered as another binding extension to HTTP/POST + XML and HTTP/GET + KVP as defined in Section 10 (Binding Extensions for WPS Operations) of the WPS 2.0 standard.

# Chapter 2. 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.

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

- Any one of the conformance levels specified in Annex B (normative).
- Any one of the Distributed Computing Platform profiles specified in Annexes TBD through TBD (normative).

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

# Chapter 3. References

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

OGC 14-065, OGC WPS 2.0 Interface Standard, version 2.0.2

OGC 06-121r9, OGC Web Service Common Specification, version 2.0

OGC 08-131r3 – The Specification Model – A Standard for Modular Specifications

IETF RFC 4646: Tags for Identifying Languages

IETF RFC 3986: Uniform Resource Identifier (URI): Generic Syntax

ISO 8601:2004, Data elements and interchange formats – Information interchange – Representation of dates and times

XML Schema Part 2: Datatypes Second Edition, W3C Recommendation 28 October 2004.

# Chapter 4. Terms and Definitions

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

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

## 4.1. Process

A process p is a function that for each input returns a corresponding output

```
p: X → Y
```

where X denotes the domain of arguments x and Y denotes the co-domain of values y. Within this specification, process arguments are referred to as process inputs and result values are referred to as process outputs. Processes that have no process inputs represent value generators that deliver constant or random process outputs.

## 4.2. Process description

A process description is an information model that specifies the interface of a process. A process description is used for a machine-readable description of the process itself but also provides some basic information about the process inputs and outputs.

## 4.3. Process input

Process inputs are the arguments of a process and refer to data provided to a process. Each process input is an identifiable item.

## 4.4. Process output

Process outputs are the results of a process and refer to data returned by a process. Each process output is an identifiable item.

## 4.5. Process profile

A process profile is a description of a process on an interface level. Process profiles may have different levels of abstraction and cover several aspects. On a generic level, a process

profile may only refer to the provided functionality of a process, i.e. by giving a verbal or formal definition how the outputs are derived from the inputs. On a concrete level a process profile may completely define inputs and outputs including data type definitions and formats.

## 4.6. WPS Server

A WPS Server is a web server that provides access to simple or complex computational processing services

## 4.7. Process offering

A process offering is an identifiable process that may be executed on a particular service instance. A process offering contains a process description as well as service-specific information about the supported execution protocols (e.g. synchronous and asynchronous execution).

## 4.8. Process execution

The execution of a process is an action that calculates the outputs of a given process for a given set of data inputs.

## 4.9. Job

The (processing) job is a server-side object created by a processing service for a particular process execution. A job may be latent in the case of synchronous execution or explicit in the case of asynchronous execution. Since the client has only oblique access to a processing job, a Job ID is used to monitor and control a job.

## 4.10. Service profiles for WPS

A service profile for WPS is a conformance class that defines the general capabilities of a WPS server, by (1) specifying the supported service operations, (2) the process model, (3) the supported process execution modes, (4) the supported operation binding(s).

## 4.11. REST or RESTful

Representational state transfer. REST-compliant Web services allow requesting systems to access and manipulate textual representations of Web resources using a uniform and predefined set of stateless operations.

## 4.12. JSON

JavaScript Object Notation is a lightweight data-interchange format. It is easy for humans to read and write and it is easy for machines to parse and generate.

# Chapter 5. Conventions

This sections provides details and examples for any conventions used in the document. Examples of conventions are symbols, abbreviations, use of XML schema, or special notes regarding how to read the document.

## 5.1. Identifiers

The normative provisions in this specification are denoted by the URI

http://www.opengis.net/spec/WPS/2.0/req/service/binding/rest-json

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

Permissions that appear in this document are denoted by partial URIs which are relative to the following base:

http://www.opengis.net/spec/WPS/2.0/per/service/binding/rest-json

Recommendations that appear in this document are denoted by partial URIs which are relative to the following base:

http://www.opengis.net/spec/WPS/2.0/rec/service/binding/rest-json

## 5.2. Abbreviated Terms

| Abbreviated Term | Meaning |
|---|---|
| CRS | Coordinate Reference System |
| GML | Geography Markup Language |
| HTTP | Hypertext Transfer Protocol |
| ISO | International Organization for Standardization |
| KVP | Keyword Value Pair |
| MIME | Multipurpose Internet Mail Extensions |
| OGC | Open Geospatial Consortium |
| URI | Universal Resource Identifier |
| URL | Uniform Resource Locator |
| WPS | Web Processing Service |
| XML | Extensible Markup Language |
| REST | Representational State Transfer |

| | |
|---|---|
| JSON | JavaScript Object Notation |

## 5.3. Use of the Term "Process"

The term process is one of the most used terms both in the information and geosciences domain. If not stated otherwise, this specification uses the term process as an umbrella term for any algorithm, calculation or model that either generates new data or transforms some input data into output data as defined in section 4.1 of the WPS 2.0 standard.

## 5.4. Namespace Conventions

| Prefix | Namespace URI | Description |
|---|---|---|
| ows | http://www.opengis.net/ows/2.0 | OWS Common 2.0 XML Schema |
| xlink | http://www.w3.org/1999/xlink | Definitions for XLINK |
| xml | http://www.w3.org/XML/1998/namespace | XML (required for xml:lang) |
| xs | http://www.w3.org/2001/XMLSchema | XML Schema |

# Chapter 6. Overview

## 6.1. Resources to be provided by WPS

The resources that are provided by a WPS REST server are listed in Table 1 below and include the capabilities document of the server, the list of processes available (ProcessCollection and Process), jobs (running processes) and results of process executions.

TODO: check chapter numbers

| Resource | Path | HTTP method | Document reference |
|---|---|---|---|
| Landing page | `/` | GET | 7.2 API landing page |
| API definition | `/api` | GET | 7.3 API definition |
| Conformance classes | `/conformance` | GET | 7.4 Declaration of conformance classes |
| Available processes | `/processes` | GET | 7.8 Retrieve a process collection |
| Process description | `/processes/{processID}` | GET | 7.9 Retrieve a process description |
| Job collection | `/processes/{processID}/jobs` | GET | 7.10 Retrieve a job collection |
| Job status info | `/processes/{processID}/jobs/{jobID}` | GET | 7.12 Retrieve status information about a job |
| Job result | `/processes/{processID}/jobs/{jobID}/results` | GET | 7.13 Retrieve a job result |

*Table 1. Overview of resources, applicable HTTP methods and links to the document sections*

## 6.2. Operations on WPS resources

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 crate a new job by sending an execute request to the server. The operation is listed in Table 2 below.

| Description | Path | HTTP method | Parameter | Document reference |
|---|---|---|---|---|
| Create a new job | `/processes/{processID}/jobs` | POST | Execute request (contained in body) | 7.11 Create a new job |

*Table 2. Overview of resources, applicable HTTP methods and links to the document sections*

This standard uses JSON as encoding for requests and responses. The inputs and outputs of a process can have any format. The formats of are defined at the time of job creation and are fixed for the specific job.

Support for HTML is recommended as HTML is the core language of the World Wide Web. A server that supports HTML will support browsing the data with a web browser and it will enable search engines to crawl and index the processes.

# Chapter 7. Requirement Class "Core"

The following section describes the core requirements class.

## 7.1. Overview

| Requirements Class | |
|---|---|
| http://www.opengis.net/spec/WPS/2.0/req/service/binding/rest-json/core | |
| Target type | Web service |
| Dependency | RFC 2616 (HTTP/1.1) |
| Dependency | RFC 2818 (HTTP over TLS) |
| Dependency | RFC 5988 (Web Linking) |

A server that implements the WPS REST/JSON Binding provides access to processes.

Each WPS has a single `LandingPage` (path `/`) that provides links to

- the `APIDefinition` (path `/api`),
- the `Conformance` statements (path `/conformance`),
- the `processes` metadata (path `/processes`).

The `APIDefinition` describes the capabilities of the server that can be used by clients to connect to the server or by development tools to support the implementation of servers and clients. Accessing the `APIDefinition` using HTTP GET returns a description of the API.

Accessing the `Conformance` using HTTP GET returns a list of URIs of requirements classes implemented by the WPS server.

The list of processes contains a summary of each process the WPS offers, including the link to a more detailed description of the process.

The process description contains information about inputs and outputs and a link to the execution-endpoint for the process.

A HTTP GET request to the execution-endpoint delivers a list of completed executions (jobs).

A HTTP POST request to the execution-endpoint creates a new job. The inputs and outputs need to be passed in a JSON execute-request.

The URL for accessing status information is delivered in the HTTP header `location`.

After a process is finished (status = success/failed), the results/exceptions can be retrieved.

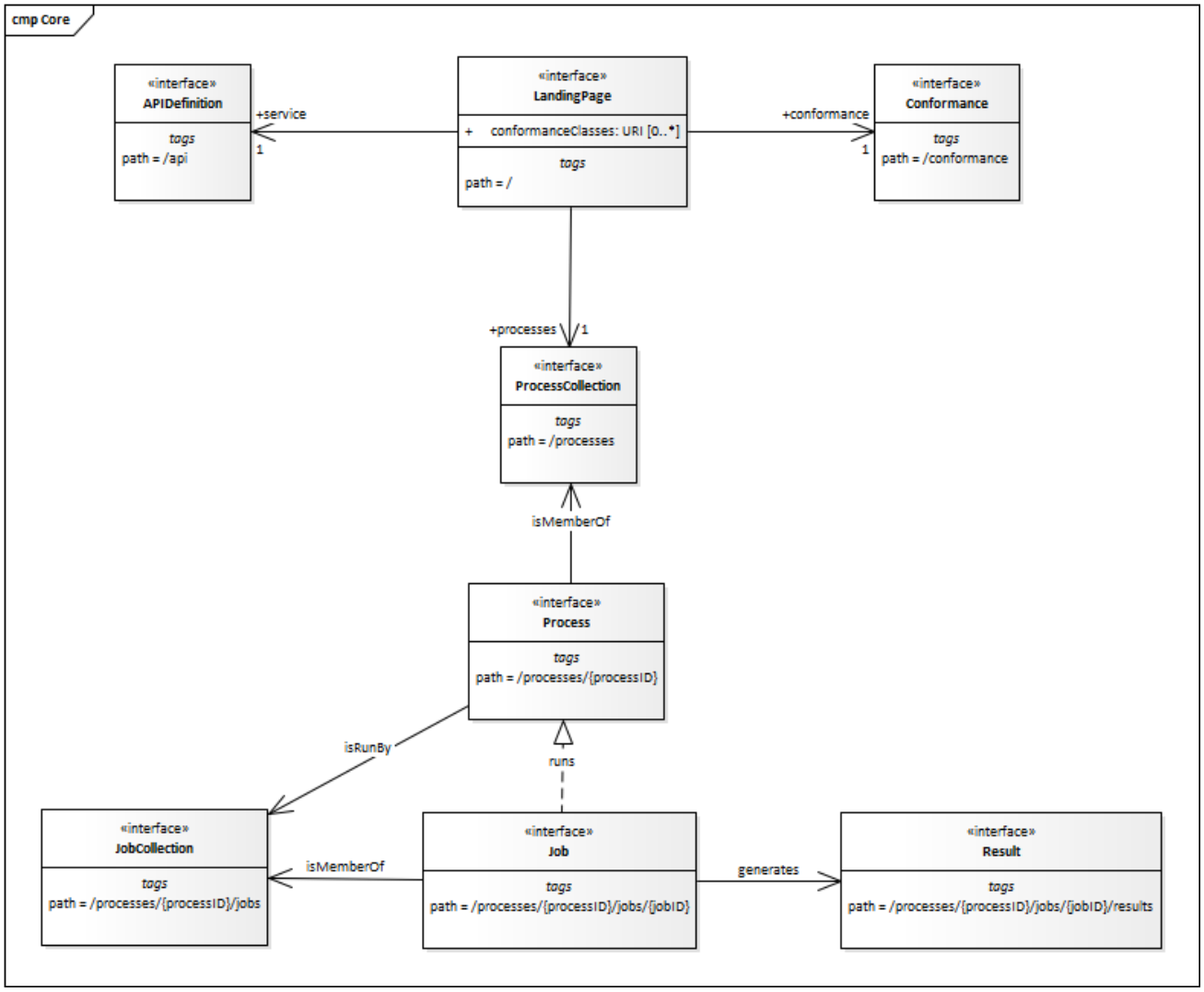


*Figure 1. Resources in the Core requirements class*

# 7.2. Retrieve the API landing page

The following section describes a method to retrieve an API landing page.

## 7.2.1. Operation

| | |
|---|---|
| **Requirement 1** | **/core/root-op**<br>**The server SHALL support the HTTP GET operation at the path /.** |

## 7.2.2. Response

| Requirement 2 | /core/root-success<br>A successful execution of the operation SHALL be reported as a response with a HTTP status code 200. The content of that response SHALL be based upon the OpenAPI 3.0 schema root.yaml and include at least links to the following resources: * /api (relation type 'service') * /conformance (relation type 'conformance') * /processes (relation type 'processes') |
|---|---|

*Schema for the landing page*

```
type: object
required:
  - links
properties:
  links:
    type: array
    items:
      $ref: link.yaml
```

*Example 1. Landing page response document*

```
{
    "links": [{
        "href": "http://processing.example.org/",
        "rel": "self",
        "type": "application/json",
        "title": "this document"
    },
    {
        "href": "http://processing.example.org/api",
        "rel": "service",
        "type": "application/openapi+json;version=3.0",
        "title": "the API definition"
    },
    {
        "href": "http://processing.example.org/conformance",
        "rel": "conformance",
        "type": "application/json",
        "title": "WPS 2.0 REST/JSON Binding conformance classes
implemented by this server"
    },
    {
        "href": "http://processing.example.org/processes",
        "rel": "processes",
        "type": "application/json",
        "title": "Metadata about the processes"
    }]
}
```

### 7.2.3. Error situations

See HTTP status codes for general guidance.

## 7.3. Retrieve an API definition

The following section describes a method to retrieve an API definition.

### 7.3.1. Operation

Every WPS provides an API definition that describes the capabilities of the server and which can be used by developers to understand the API, by software clients to connect to

the server, or by development tools to support the implementation of servers and clients.

| Requirement 3 | /core/api-definition-op<br>The server SHALL support the HTTP GET operation at the path `/api`. |
|---|---|

### 7.3.2. Response

| Requirement 4 | /core/api-definition-success<br>A successful execution of the operation SHALL be reported as a response with a HTTP status code `200`. The server SHALL return an API definition document. |
|---|---|

| Recommendation 1 | /core/api-definition-oas<br>If the API definition document uses the OpenAPI Specification 3.0, the document SHOULD conform to the OpenAPI Specification 3.0 requirements class. |
|---|---|

If multiple API definition formats are supported by a server, use content negotiation to select the desired representation.

The API definition document describes the API. In other words, there is no need to include the `/api` operation in the API definition itself.

The idea is that any WPS can be used by developers that are familiar with the API definition language(s) supported by the server. For example, if an OpenAPI definition is used, it should be possible to create a working client using the OpenAPI definition. The developer may need to learn a little bit about geometry data types, etc., but it should not be required to read this standard to access the data via the API.

### 7.3.3. Error situations

See HTTP status codes for general guidance.

## 7.4. Declaration of conformance classes

### 7.4.1. Operation

To support "generic" clients for accessing Web Processing Services in general - and not "just" a specific API / server, the server has to declare the requirements classes it implements and conforms to.

| Requirement 5 | /core/conformance-op<br>The server SHALL support the HTTP GET operation at the path `/conformance`. |
|---|---|

### 7.4.2. Response

| Requirement 6 | /core/conformance-success<br>A successful execution of the operation SHALL be reported as a response with a HTTP status code 200. The content of that response SHALL be based upon the OpenAPI 3.0 schema req-classes.yaml and list all WPS REST/JSON Binding requirements classes that the server conforms to. |
|---|---|

*Schema for the list of requirements classes*

```
    type: object
    required:
      - conformsTo
    properties:
      conformsTo:
        type: array
        items:
          type: string
          example:
"http://www.opengis.net/spec/WPS/2.0/req/service/binding/rest-json/1.0/core"
```

*Example 2. Requirements class response document*

This example response in JSON is for a server that supports OpenAPI 3.0 for the API definition and HTML and GeoJSON as encodings for features.

```
{
  "conformsTo": [
    "http://www.opengis.net/spec/WPS/2.0/req/service/binding/rest-
json/1.0/core",
    "http://www.opengis.net/spec/WPS/2.0/req/service/binding/rest-
json/1.0/oas30",
    "http://www.opengis.net/spec/WPS/2.0/req/service/binding/rest-
json/1.0/html"
  ]
}
```

### 7.4.3. Error situations

See HTTP status codes for general guidance.

## 7.5. Use of HTTP 1.1

| Requirement 7 | /core/http<br>**The server SHALL conform to HTTP 1.1. If the server supports HTTPS, the server SHALL also conform to HTTP over TLS.** |
|---|---|

### 7.5.1. HTTP status codes

Table 3 lists the main HTTP status codes that clients should be prepared to receive.

This includes, for example, support for specific security schemes or URI redirection.

In addition, other error situations may occur in the transport layer outside of the server.

| Status code | Description |
|---|---|
| `200` | A successful request. |
| `201` | The request was successful and one or more new resources have being created. |
| `400` | The server cannot or will not process the request due to an apparent client error. For example, a query parameter had an incorrect value. |
| `401` | The request requires user authentication. The response includes a `WWW-Authenticate` header field containing a challenge applicable to the requested resource. |
| `403` | The server understood the request, but is refusing to fulfill it. While status code `401` indicates missing or bad authentication, status code `403` indicates that authentication is not the issue, but the client is not authorised to perform the requested operation on the resource. |
| `404` | The requested resource does not exist on the server. For example, a path parameter had an incorrect value. |
| `405` | The request method is not supported. For example, a POST request was submitted, but the resource only supports GET requests. |
| `406` | The `Accept` header submitted in the request did not support any of the media types supported by the server for the requested resource. |
| `500` | An internal error occurred in the server. |

*Table 3. Typical HTTP status codes*

More specific guidance is provided for each resource, where applicable.

| Permission 1 | /core/additional-status-codes<br>**Servers MAY support other capabilities of the HTTP protocol and, therefore, MAY return other status codes than those listed in Table 3, too.** |
|---|---|

## 7.6. Support for cross-origin requests

To access data from a HTML page where the data is on another host than the webpage is by default prohibited for security reasons ("same-origin policy"). A typical example is a web-application accessing feature data from multiple distributed datasets.

| Recommendation 2 | /core/cross-origin<br>If the server is intended to be accessed from the browser, cross-origin requests SHOULD be supported. Note that support can also be added in a proxy layer on top of the server. |
|---|---|

Two common mechanisms to support cross-origin requests are:

- Cross-origin resource sharing (CORS)
- JSONP (JSON with padding)

| Recommendation 3 | /core/html<br>To support browsing a WPS with a web browser and to enable search engines to crawl and index a dataset, implementations SHOULD consider to support an HTML encoding. |
|---|---|

## 7.7. Retrieve a process collection

The following section describes a method to retrieve the available processes offered by the server.

### 7.7.1. Operation

| Requirement 8 | /core/process-collection<br>The server SHALL support the HTTP GET operation at the path `/processes`. |
|---|---|

### 7.7.2. Response

| Requirement 9 | /core/process-collection-success<br>A successful execution of the operation SHALL be reported as a response with a HTTP status code 200. The content of that response SHALL be based upon the OpenAPI 3.0 schema processCollection.yaml. |
|---|---|

*Schema for the process collection*

```
    type: array
    items:
      $ref: 'processSummary.yaml'
```

**NOTE** | References to additional schemas can found in Annex TODO

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

```
https://processing.example.org/processes
```

*Example of Process list encoded as JSON.*

```
[
    {
        "id": "EchoProcess",
        "title": "EchoProcess",
        "version": "1.0.0",
        "jobControlOptions": ["async-execute", "sync-execute"],
        "outputTransmission": ["value", "reference"],
        "links": [
          {
            "href": "https://processing.example.org/processes/EchoProcess",
            "type": "application/json",
            "rel": "self",
            "title": "default process description"
          }
        ]
    },...
]
```

### 7.7.3. Error situations

See HTTP status codes for general guidance.

## 7.8. Retrieve a process description

The following section describes a method to retrieve metadata about a process.

### 7.8.1. Operation

| Requirement 10 | /core/process<br>The server SHALL support the HTTP GET operation at the path /processes/{processID}. |
|---|---|

### 7.8.2. Response

| Requirement 11 | /core/process-success<br>A successful execution of the operation SHALL be reported as a response with a HTTP status code 200. The content of that response SHALL be based upon the OpenAPI 3.0 schema process.yaml. |
|---|---|

*Schema for a process*

```
    allOf:
       - $ref: 'processSummary.yaml'
       - type: object
         properties:
            inputs:
              type: array
              items:
                 $ref: 'inputDescription.yaml'
            outputs:
              type: array
              items:
                 $ref: 'outputDescription.yaml'
            links:
              type: array
              items:
                 $ref: 'link.yaml'
```

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

```
https://processing.example.org/processes/EchoProcess
```

*Example of a process encoded as JSON.*

```
{
    "id": "EchoProcess",
    "title": "EchoProcess",
    "version": "1.0.0",
    "jobControlOptions": ["async-execute", "sync-execute"],
    "outputTransmission": ["value", "reference"],
```

```
    "inputs": [{
        "id": "boundingboxInput",
        "title": "boundingboxInput",
        "input": {
            "supportedCRS": [{
                "default": true,
                "crs": "EPSG:4326"
            }]
        },
        "minOccurs": 1,
        "maxOccurs": 1
    },
    {
        "id": "literalInput",
        "title": "literalInput",
        "input": {
            "literalDataDomain": {
                "dataType": {
                  "name": "double"
                },
                "valueDefinition": {
                    "anyValue": true
                }
            }
        },
        "minOccurs": 1,
        "maxOccurs": 1
    },
    {
        "id": "complexInput",
        "title": "complexInput",
        "input": {
            "formats": [{
                "default": true,
                "mimeType": "application/xml"
            },
            {
                "mimeType": "application/xml"
            },
            {
                "mimeType": "text/xml"
            }]
        },
        "minOccurs": 1,
        "maxOccurs": 1
    }],
    "outputs": [{
```

```
        "id": "boundingboxOutput",
        "title": "boundingboxOutput",
        "output": {
            "supportedCRS": [{
                "default": true,
                "crs": "EPSG:4326"
            }]
        }
    },
    {
        "id": "literalOutput",
        "title": "literalOutput",
        "output": {
            "literalDataDomain": {
                "dataType": {
                  "name": "double"
                },
                "valueDefinition": {
                    "anyValue": true
                }
            }
        }
    },
    {
        "id": "complexOutput",
        "title": "complexOutput",
        "output": {
            "formats": [{
                "default": true,
                "mimeType": "application/xml"
            },
            {
                "mimeType": "application/xml"
            },
            {
                "mimeType": "text/xml"
            }]
        }
    }],
    "links": [
      {
        "href": "https://processing.example.org/processes/EchoProcess/jobs",
        "rel": "execute",
        "title": "Execute endpoint"
      }
    ]
}
```

### 7.8.3. Error situations

See HTTP status codes for general guidance.

| Requirement 12 | /core/process-exception/no-such-process<br>If the operation is executed using an invalid process identifier, the response shall have HTTP status code 404. The content of that response SHALL be based upon the OpenAPI 3.0 schema exception.yaml. The exception code of the exception shall be "NoSuchProcess". |
|---|---|

## 7.9. Retrieve a job collection

The following section describes a method to retrieve a collection of existing jobs of a process.

### 7.9.1. Operation

| Requirement 13 | /core/job-collection<br>The server SHALL support the HTTP GET operation at the path `/processes/{processID}/jobs`. |
|---|---|

### 7.9.2. Response

| Requirement 14 | /core/job-collection-success<br>A successful execution of the operation SHALL be reported as a response with a HTTP status code 200. The content of that response SHALL be based upon the OpenAPI 3.0 schema jobCollection.yaml. |
|---|---|

*Schema for the job collection*

```
type: array
items:
  type: string
```

*Example of HTTP GET request for retrieving the list of jobs of a process encoded as JSON.*

```
http://processing.example.org/processes/EchoProcess/jobs
```

*Example of a job list encoded as JSON.*

```
[
    "8f5c13b9-6da2-4447-a683-69a5f364323b",
    "904358bd-a151-49f3-af74-79edf7ccb288"
]
```

### 7.9.3. Error situations

See HTTP status codes for general guidance.

If the process with the specified identifier doesn't exist on the server, the status code of the response will be `404` (see [rc_no-such-process]).

## 7.10. Create a new job

The following section describes a method to create a new job, i.e. execute a process.

### 7.10.1. Operation

| | |
|---|---|
| **Requirement 15** | **/core/job-creation**<br>**The server SHALL support the HTTP POST operation at the path `/processes/{processID}/jobs`.** |

### 7.10.2. Request

| | |
|---|---|
| **Requirement 16** | **/core/job-creation-request**<br>**The content of a request to create a new job SHALL be based upon the OpenAPI 3.0 schema execute.yaml.** |

*Schema for execute*

```yaml
type: object
required:
  - outputs
properties:
  inputs:
    type: array
    items:
      $ref: 'input.yaml'
  outputs:
    type: array
    items:
      $ref: 'output.yaml'
```

*Example of an execute request*

```
{
    "inputs": [{
        "id": "complexInput",
        "input": {
            "format": {
                "mimeType": "application/xml"
            },
            "value": {
                "inlineValue": "<test/>"
            }
        }
    },
    {
        "id": "literalInput",
        "input": {
            "dataType": {
              "name": "double"
            },
            "value": "0.05"
        }
    },
    {
        "id": "boundingboxInput",
        "input": {
            "bbox": [51.9, 7, 52, 7.1],
            "crs": "EPSG:4326"
        }
    }],
    "outputs": [{
        "id": "literalOutput",
        "transmissionMode": "value"
    },
    {
        "id": "boundingboxOutput",
        "transmissionMode": "value"
    },
    {
        "id": "complexOutput",
        "format": {
            "mimeType": "application/xml"
        },
        "transmissionMode": "value"
    }]
}
```

### 7.10.3. Response

| Requirement 17 | /core/job-creation-success<br>A successful execution of the operation SHALL be reported as a response with a HTTP status code 201. |
|---|---|

| Requirement 18 | /core/job-creation-success-header<br>The 201 response of the operation SHALL return a HTTP header named 'Location' which contains a link to the newly created job. |
|---|---|

### 7.10.4. Error situations

See HTTP status codes for general guidance.

If the process with the specified identifier doesn't exist on the server, the status code of the response will be `404` (see [rc_no-such-process]).

## 7.11. Retrieve status information about a job

The following section describes a method to retrieve information about the status of a job.

### 7.11.1. Operation

| Requirement 19 | /core/job<br>The server SHALL support the HTTP GET operation at the path `/processes/{processID}/jobs/{jobID}`. |
|---|---|

### 7.11.2. Response

| Requirement 20 | /core/job-success<br>A successful execution of the operation SHALL be reported as a response with a HTTP status code 200. The content of that response SHALL be based upon the OpenAPI 3.0 schema statusInfo.yaml. |
|---|---|

*Schema for status info*

```
type: object
required:
    - jobID
    - status
properties:
   jobID:
      type: string
   status:
      type: string
      enum:
        - accepted
        - running
        - successful
        - failed
   message:
      type: string
   progress:
      type: integer
      minimum: 0
      maximum: 100
   links:
      type: array
      items:
         $ref: 'link.yaml'
```

*Example of HTTP GET request for retrieving status information about a job encoded as JSON.*

```
http://processing.example.org/processes/EchoProcess/jobs/81574318-1eb1-4d7c-
af61-4b3fbcf33c4f
```

*Example of a job encoded as JSON.*


```
{
  "jobID" : "81574318-1eb1-4d7c-af61-4b3fbcf33c4f",
  "status": "accepted",
  "message": "Process started",
  "progress": 0,
  "links": [
    {
      "href":
"http://processing.example.org/processes/EchoProcess/jobs/81574318-1eb1-4d7c-
af61-4b3fbcf33c4f",
      "rel": "self",
      "type": "application/json",
      "title": "this document"
    }
  ]
}
```


### 7.11.3. Error situations

See HTTP status codes for general guidance.

If the process with the specified identifier doesn't exist on the server, the status code of the response will be `404` (see [rc_no-such-process]).

| Requirement 21 | /core/job-exception/no-such-job<br>If the operation is executed using an invalid job identifier, the response shall have HTTP status code 404. The content of that response SHALL be based upon the OpenAPI 3.0 schema **exception.yaml**. The exception code of the exception shall be "NoSuchJob". |
|---|---|

## 7.12. Retrieve a job result

The following section describes a method to retrieve the result of a job. In case the job execution failed, an exception is returned.

### 7.12.1. Operation

| Requirement 22 | /core/job-result<br>The server SHALL support the HTTP GET operation at the path `/processes/{processID}/jobs/{jobID}/results`. |
|---|---|

### 7.12.2. Response

| Requirement 23 | /core/job-result-success<br>A successful execution of the operation SHALL be reported as a response with a HTTP status code 200. The content of that response SHALL be based upon the OpenAPI 3.0 schema result.yaml. |
|---|---|

*Schema for the result of a job*

```
type: array
items:
   $ref: 'outputInfo.yaml'
```

*Schema for output info*

```
type: object
required:
    - id
    - value
properties:
    id:
      type: string
    value:
       $ref: 'valueType.yaml'
```

*Example of HTTP GET request for retrieving the result a job encoded as JSON.*

```
http://processing.example.org/processes/EchoProcess/jobs/81574318-1eb1-4d7c-
af61-4b3fbcf33c4f/result
```

*Example of a result encoded as JSON.*

```
[
   {
    "id": "literalOutput",
        "value": {
            "inlineValue": 0.05
        }
    },
    {
        "id": "boundingboxOutput",
        "value": {
            "inlineValue": {
            "bbox": [51.9, 7, 52, 7.1],
                "crs": "EPSG:4326"
            }
        }
    },
    {
        "id": "complexOutput",
        "value": {
            "inlineValue": "<test/>"
        }
    }
]
```

### 7.12.3. Error situations

See HTTP status codes for general guidance.

If the process with the specified id doesn't exist on the server, the status code of the response will be 404 (see [rc_no-such-process]).

| | |
|---|---|
| **Requirement 24** | **/core/job-result-exception/no-such-job**<br>**If the operation is executed using an invalid job identifier, the response shall have HTTP status code 404. The content of that response SHALL be based upon the OpenAPI 3.0 schema exception.yaml. The exception code of the exception shall be "NoSuchJob".** |

| | |
|---|---|
| **Requirement 25** | **/core/job-result-exception/result-not-ready**<br>**If the operation is executed on a running job with a valid job identifier, the response shall have HTTP status code 404. The content of that response SHALL be based upon the OpenAPI 3.0 schema exception.yaml. The exception code of the exception shall be "ResultNotReady".** |

| | |
|---|---|
| **Requirement 26** | **/core/job-result-failed**<br>**If the operation is executed on a failed job using a valid job identifier, the response shall have a HTTP error code that corresponds to the reason of the failure. The content of that response SHALL be based upon the OpenAPI 3.0 schema exception.yaml. The exception code shall correspond to the reason of the failure, e.g. InvalidParameterValue for invalid input data.** |

# Chapter 8. Requirements classes for encodings

## 8.1. Overview

This clause specifies four pre-defined requirements classes for encodings to be used in a WPS. These encodings are commonly used encodings for spatial data on the web:

- JSON
- HTML

The JSON encoding is mandatory.

The Core requirements class includes recommendations to support HTML and JSON as encodings, where practical.

## 8.2. Requirement Class "JSON"

This section defines the requirements class JSON.

| **Requirements Class** | |
|---|---|
| http://www.opengis.net/spec/WPS/2.0/req/service/binding/rest-json/json | |
| Target type | Web service |
| Dependency | WPS REST/JSON Binding Core |
| Dependency | JSON |

| **Requirement 27** | **/json/definition**<br>**200-responses of the server SHALL support the following media type:** * `application/json` |
|---|---|

## 8.3. Requirement Class "HTML"

This section defines the requirements class HTML.

| **Requirements Class** | |
|---|---|
| http://www.opengis.net/spec/WPS/2.0/req/service/binding/rest-json/html | |
| Target type | Web service |
| Dependency | WPS REST Binding 1.0 Core |
| Dependency | HTML5 |

| Requirement 28 | /html/definition<br>Every 200-response of an operation of the server SHALL support the media type text/html. |
|---|---|

| Requirement 29 | /html/content<br>Every 200-response of the server with the media type "text/html" SHALL be a HTML 5 document that includes the following information in the HTML body: * all information identified in the schemas of the Response Object in the HTML <body/>, and * all links in HTML <a/> elements in the HTML <body/>. |
|---|---|

# Chapter 9. Requirements class "OpenAPI 3.0"

## 9.1. Basic requirements

APIs conforming to this requirements class document themselves by an OpenAPI Document.

| Requirements Class | |
|---|---|
| http://www.opengis.net/spec/WPS/2.0/req/service/binding/rest-json/oas30 | |
| Target type | Web service |
| Dependency | WPS REST JSON Binding 1.0 Core |
| Dependency | OpenAPI Specification 3.0.1 |

| Requirement 30 | /req/oas30/oas-definition-1<br>The service SHALL provide an OpenAPI definition in JSON and HTML at the path `/api` using the media type `application/openapi+json;version=3.0`. |
|---|---|

**CAUTION**

ISSUE 117

The OpenAPI media type has not been registered yet with IANA and will likely change. We need to update the media type after registration.

| Requirement 31 | /req/oas30/oas-definition-2<br>The JSON representation SHALL conform to the OpenAPI Specification, version 3.0. |
|---|---|

**CAUTION**

Related to ISSUE 90

If we have a rigid path pattern there seems to be no need to add requirements for fixed operationId values. However, if the path pattern would be flexible, maybe we should require specific operationIds for selected resources?

Two example OpenAPI documents are included in Annex B.

| Requirement 32 | /req/oas30/oas-impl<br>The server SHALL implement all capabilities specified in the OpenAPI definition. |
|---|---|

## 9.2. Complete definition

| Requirement 33 | /req/oas30/completeness<br>The OpenAPI definition SHALL specify for each operation all HTTP Status Codes and Response Objects that the server uses in responses. This includes the successful execution of an operation as well as all error situations that originate from the server. |
|---|---|

Note that APIs that, for example, are access-controlled (see Security), support web cache validation, CORS or that use HTTP redirection will make use of additional HTTP status codes beyond regular codes such as `200` for successful GET requests and `400`, `404` or `500` for error situations. See HTTP status codes.

Clients have to be prepared to receive responses not documented in the OpenAPI definition. For example, additional errors may occur in the transport layer outside of the server.

## 9.3. Exceptions

| Requirement 34 | /req/oas30/exceptions-codes<br>For error situations that originate from the server, the API definition SHALL cover all applicable HTTP Status Codes. |
|---|---|

*Example 3. An exception response object definition*

```
description: An error occurred.
content:
  application/json:
    schema:
      $ref:
https://raw.githubusercontent.com/opengeospatial/OAPI/openapi/schemas/e
xception.yaml
  text/html:
    schema:
      type: string
```

## 9.4. Security

| Requirement 35 | /req/oas30/security<br>For cases, where the operations of the server are access-controlled, the security scheme(s) SHALL be documented in the OpenAPI definition. |
|---|---|

The OpenAPI specification currently supports the following security schemes:

- HTTP authentication,
- an API key (either as a header or as a query parameter),
- OAuth2's common flows (implicit, password, application and access code) as defined in RFC6749, and
- OpenID Connect Discovery.

**CAUTION**

ISSUE 41
How does a client determine which security protocols/standards/etc. a server supports?

# Chapter 10. Media Types

JSON media types that would typically be used in a WPS that supports JSON are

- `application/geo+json` for feature collections and features, and
- `application/json` for all other resources.

XML media types that would typically occur in on OGC API that supports XML are

- `application/gml+xml;version=3.2` for any GML 3.2 feature collections and features,
- `application/gml+xml;version=3.2;profile=http://www.opengis.net/def/profile/ogc/2.0/gml-sf0` for GML 3.2 feature collections and features conforming to the GML Simple Feature Level 0 profile,
- `application/gml+xml;version=3.2;profile=http://www.opengis.net/def/profile/ogc/2.0/gml-sf2` for GML 3.2 feature collections and features conforming to the GML Simple Feature Level 2 profile, and
- `application/xml` for all other resources.

The typical HTML media type for all "web pages" in a WPS would be `text/html`.

The media type for an OpenAPI definition in JSON is `application/openapi+json;version=3.0`.

**CAUTION**

ISSUE 117

The OpenAPI media type has not been registered yet with IANA and will likely change. We need to update the media type after registration.

# Annex A: Abstract Test Suite (Normative)

## A.1. Overview

Compliance testing for $standard_name$ and similar standards must answer three questions:

1. Are the capabilities advertised through the API Description compliant with the standard?
2. Does the API implement those capabilities as advertised?
3. Do the resources returned by the micro-services meet the structure and content requirements of the standard?

Further complicating the issue, an API may expose resources in addition to those defined by the standard. A test engine must be able to traverse the API description document, identify test points, and ignore resource paths which are not to be tested. The process for identifying test points is provided in Section A.4.3.

## A.2. Conventions

The following conventions apply to this Abstract Test Suite:

### A.2.1. Path Templates

Path templates are used throughout these test suites. Path templating refers to the usage of curly braces “{}” to mark a section of a URL path that can be replaced using path parameters. The terms used to describe portions of these templates are based on the URL syntax described in RFC 3986.

- scheme: http | https
- authority: DNS name of the server with optional port number
- path: The slash delimited identifier for a resource on the server
- query: query parameters following the "?" character
- fragment: identifies an element within the resource. Preceded by the "#" character

### A.2.2. API Description Document

The $standard_name$ standard does not mandate a standard format for the API Description Document. However, some form of standard is needed if tests are to be accurately described and implemented. Therefore, this Abstract Test Suite asserts that the

API Description document is compliant with OpenAPI 3.0. This Test Suite will be updated if and when an alternative is commonly adopted.

### A.2.3. Resource Encodings

The $standard_name$ standard does not mandate a standard encoding for resources returned by the API. Yet a compliance test requires some minimal level of expected behavior. Therefore, this Abstract Test Suite asserts that the API returns resources encoded in HTML and GeoJSON. Since no compliance test suite exists for these encodings at this time, the resources shall be presented to the test operator for human inspection.

### A.2.4. Processing Security Objects

OpenAPI does not provide a standard way to associate a security requirement with a single server URI. Therefore, $standard_name$ compliance tests will have to make that association through the runtime challenge-response transaction. At this time the role of the Security Objects should be considered advisory.

Security Requirements can be defined at both the OpanAPI root level and at the Operation Object level. The following rules should be followed to understand the scope of a Security Requirement:

- The Security Requirements defined at the root level are the default requirements for all operations and servers.
- If Security Requirements are defined at the Operation level, then those Requirements, and not the ones defined at the OpenAPI level, shall be used with that operation.
- An empty set of Security Requirements at the Operation level indicates that there are no security requirements for that operation.

Note: this allows operations to opt-out of security requirements defined at the OpenAPI level.

### A.2.5. Parameters

The following observations apply for $standard_name$ parameters:

1. $standard_name$ does not use cookies.
2. Query parameters follow common Web practice
3. Header parameters are restricted to custom headers
4. For path parameters, the name of the parameter must match the name of the variable in the path template in the path object

Parameters are defined at the Path Item and Operation level. Parameters defined at the Path Item level must apply to all operations under that Path item. These parameters may be modified at the Operation level but they may not be removed.

### A.2.6. Testable Paths

A testable path is a path which corresponds to one of the paths defined in the $standard_name$ specification. There are three alternatives for making this determination:

1. The path URI matches – this is the simplest approach but may be subject to error
2. Use mandatory tags in the tags field of the Operation Object
3. Use standardized operation ids for the operationId field of the Operation Object

A testable path is validated against the rules for that path. At a minimum that includes:

1. Building a list of all parameters which are defined in the standard
2. Validate that the mandatory parameters are present and required
3. Validate type, format, etc. for each parameter in the list.
4. Validate that there are no mandatory parameters which are not on the list.

## A.3. Requirements Trace Matrix

| |
|---|
| **Requirement 1:** API Landing Page Operation<br>The server SHALL support the HTTP GET operation at the path /.<br>Tests: A.4.2.1 |
| **Requirement 2:** API Landing Page Response<br>A successful execution of the operation SHALL be reported as a response with a HTTP status code 200. The content of that response SHALL be based upon the OpenAPI 3.0 schema root.yaml and include at least links to the following resources:<br>- /api (relation type 'service')<br>- /conformance (relation type 'conformance')<br>- /collections (relation type 'data')<br>Tests: A.4.2.2 |
| **Requirement 3:** API Definition Operation<br>The server SHALL support the HTTP GET operation at the path /api.<br>Tests: A.4.2.3 |
| **Requirement 4:** API Definition Response<br>A successful execution of the operation SHALL be reported as a response with a HTTP status code 200. The server SHALL return an API definition document.<br>Tests: A.4.2.3, A.4.2.4, A.4.4.1 |
| **Requirement 5:** Conformance Class Operation<br>The server SHALL support the HTTP GET operation at the path /conformance.<br>Tests: A.4.4.2 |

| |
|---|
| **Requirement 6:** Conformance Class Response<br>A successful execution of the operation SHALL be reported as a response with a HTTP status code 200. The content of that response SHALL be based upon the OpenAPI 3.0 schema req-classes.yaml and list all $standard_name$ requirements classes that the server conforms to.<br>Tests: A.4.4.3 |
| **Requirement 7:** HTTP 1.1<br>The server SHALL conform to HTTP 1.1.<br>If the server supports HTTPS, the server SHALL also conform to HTTP over TLS.<br>Tests: A.4.1.1 |

# A.4. Abstract Test

The Test Approach used in the $standard_name$ Abstract Test Suite includes four steps:

1. Identify the test points
2. Verify that API descriptions of the test points comply with the $standard_name$ standard
3. Verify that the micro-services at each test point behave in accordance with the $standard_name$ standard.
4. Verify that the resources returned at each test point are in accordance with the $standard_name$ standard and any referenced content standard.

Identification of test points is a new requirement with $standard_name$. Since an API is not a Web Service, there may be RESTful endpoints advertised which are not intended to be targets of the compliance testing. Section A.4.2 describes the process for crawling the API Description document and extracting those URLs which should be tested as well as the path(s) they should be tested with. The concatenation of a Server URL with a path forms a test point.

Section A.4.3 describes how the test points are exercised to determine compliance with the $standard_name$ standard.

## A.4.1. General Tests

### A.4.1.1. HTTP 1.1

#### a) Test Purpose:

Validate that the $standard_name$ services advertised through the API conform with HTTP 1.1.

**b) Pre-conditions:**

none

**c) Test Method:**

1. All compliance tests shall be configured to use the HTTP 1.1 protocol exclusively.

**d) References:**

Requirement 7

## A.4.2. Retrieve the API Description

### A.4.2.1. Landing Page Retrieval

**a) Test Purpose:**

Validate that a landing page can be retrieved from the expected location.

**b) Pre-conditions:**

- A URL to the server hosting the landing page is known.
- The test client can authenticate to the server.
- The test client has sufficient privileges to access the landing page.

**c) Test Method:**

1. Issue an HTTP GET request to the URL {root}/
2. Validate that a document was returned with a status code 200
3. Validate the contents of the returned document using test A.4.2.2

**d) References:**

Requirement 1

### A.4.2.2. Landing Page Validation

**a) Test Purpose:**

Validate that the landing page complies with the require structure and contents.

**b) Pre-conditions:**

- The landing page has been retrieved from the server

**c) Test Method:**

1. Validate the landing page against the root.yaml schema
2. Validate that the landing page includes a “service” link to API Definition
3. Validate that the landing page includes a “conformance” link to the conformance class document
4. Validate that the landing page includes a “processes” link to the $standard_name$ contents.

**d) References:**

Requirement 2

### A.4.2.3. OpenAPI Document Retrieval

Note: The URI for the API definition is provided through the landing page. However, that does not mean that the API definition resides on the same server as the landing page. Test clients should be prepared for a $standard_name$ implementation which is distributed across multiple servers.

**a) Test Purpose:**

Validate that the API Definition document can be retrieved from the expected location.

**b) Pre-conditions:**

- A URL to the server hosting the API Definition document is known.
- The test client can authenticate to the server.
- The test client has sufficient privileges to assess the API Definition document.

**c) Test Method:**

1. Issue an HTTP GET request to the URL {root}/api
2. Validate that a document was returned with a status code 200
3. Validate the contents of the returned document using test A.4.2.4

**d) References:**

Requirements 3 and 4

#### A.4.2.4. API Definition Validation

**a) Test Purpose:**

Validate that the API Definition page complies with the require structure and contents.

**b) Pre-conditions:**

- The API Definition document has been retrieved from the server

**c) Test Method:**

1. Validate the API Definition document against the OpenAPI 3.0 schema
2. Identify the Test Points as described in test A.4.3
3. Process the API Definition document as described in test A.4.4

**d) References:**

Requirement 4

### A.4.3. Identify the Test Points

Identification of the test points is a pre-condition to performing a compliance test. Thisprocess starts with A.4.3.1.

#### A.4.3.1. Identify Test Points:

**a) Purpose:**

To identify the test points associated with each Path in the OpenAPI document

**b) Pre-conditions:**

- An OpenAPI document has been obtained
- A list of URLs for the servers to be included in the compliance test has been provided
- A list of the paths specified in the $standard_name$ specification

**c) Method:**

FOR EACH paths property in the OpenAPI document If the path name is one of those specified in the $standard_name$ specification Retrieve the Server URIs using A.4.3.2. FOR EACH Server URI Concatenate the Server URI with the path name to form a test point. Add that test point to the list.

**d) References:**

None

### A.4.3.2. Identify Server URIs:

**a) Purpose:**

To identify all server URIs applicable to an OpenAPI Operation Object

**b) Pre-conditions:**

- Server Objects from the root level of the OpenAPI document have been obtained
- A Path Item Object has been retrieved
- An Operation Object has been retrieved
- The Operation Object is associated with the Path Item Object
- A list of URLs for the servers to be included in the compliance test has been provided

**c) Method:**

1) Identify the Server Objects which are in-scope for this operation

- IF Server Objects are defined at the Operation level, then those and only those Server Objects apply to that Operation.
- IF Server Objects are defined at the Path Item level, then those and only those Server Objects apply to that Path Item.
- IF Server Objects are not defined at the Operation level, then the Server Objects defined for the parent Path Item apply to that Operation.
- IF Server Objects are not defined at the Path Item level, then the Server Objects defined for the root level apply to that Path.
- IF no Server Objects are defined at the root level, then the default server object is assumed as described in the OpenAPI specification.

2) Process each Server Object using A.4.3.3.

3) Delete any Server URI which does not reference a server on the list of servers to test.

**d) References:**

None

### A.4.3.3. Process Server Object:

**a) Purpose:**

To expand the contents of a Server Object into a set of absolute URIs.

**b) Pre-conditions:**

- A Server Object has been retrieved

**c) Method:**

Processing the Server Object results in a set of absolute URIs. This set contains all of the URIs that can be created given the URI template and variables defined in that Server Object.

1. If there are no variables in the URI template, then add the URI to the return set.
2. For each variable in the URI template which does not have an enumerated set of valid values:
    - generate a URI using the default value,
    - add this URI to the return set,
    - flag this URI as non-exhaustive
3. For each variable in the URI template which has an enumerated set of valid values:
    - generate a URI for each value in the enumerated set,
    - add each generated URI to the return set.
4. Perform this processing in an iterative manner so that there is a unique URI for all possible combinations of enumerated and default values.
5. Convert all relative URIs to absolute URIs by rooting them on the URI to the server hosting the OpenAPI document.

**d) References:**

None

## A.4.4. Processing the OpenAPI Document

### A.4.4.1. Validate /api path

**a) Test Purpose:**

Validate API definition provided through the /api path it the athoritative definition of this API. Validate that this resource exists at the expected location and that it complies with the appropirate schema.

**b) Pre-conditions:**

- A URL to the server hosting the API definition document is known

**c) Test Method:**

1. Issue an HTTP GET request to the URL {root}/api
2. Validate that a document was returned with a status code of 200
3. Validate the returned document against the OpenAPI 3.0 schema

**d) References:**

Requirement 4

### A.4.4.2. Validate Conformance Operation

**a) Test Purpose:**

Validate that Conformance Operation behaves as required.

**b) Pre-conditions:**

- Path = /conformance

**c) Test Method:**

DO FOR each /conformance test point

1. Issue an HTTP GET request using the test point URI
2. Go to test A.4.4.3.

**d) References:**

Requirement 5

### A.4.4.3. Validate Conformance Operation Response

**a) Test Purpose:**

Validate the response to the Conformance Operation.

**b) Pre-conditions:**

- Path = /conformance
- A Conformance document has been retrieved

**c) Test Method:**

1. Validate the retrieved document against the classes.yaml schema.
2. Record all reported compliance classes and associate that list with the test point. This information will be used in latter tests.

**d) References:**

Requirement 6

### A.4.4.4. Validate the Get Processes Operation

**a) Test Purpose:**

Validate that the Get Processes Operation behaves as required.

**b) Pre-conditions:**

- Path = /processes/

**c) Test Method:**

- Issue an HTTP GET request using the test point URI
- Go to test Validate Get Processes Operation Response

**d) References:**

Requirement 9

### A.4.4.5. Validate Get Processes Operation Response

**a) Test Purpose:**

Validate the response to the Get Processes Operation.

**b) Pre-conditions:**

- A Process Collection document has been retrieved

**c) Test Method:**

1. Validate the retrieved document against the processCollection.yaml schema.
2. Validate each Process Description using test Validate the Get Process Description Operation

**d) References:**

Requirements 10, 11, and 12

### A.4.4.6. Validate the Get Process Description Operation

**a) Test Purpose:**

Validate that the Get Process Description Operation behaves as required.

**b) Pre-conditions:**

- Path = /processes/

**c) Test Method:**

DO FOR each /processes/{processId} test point

- Issue an HTTP GET request using the test point URI
- Go to test Validate the Process Description Operation Response

**d) References:**

Requirement 15

### A.4.4.7. Validate the Process Description Operation Response

**a) Test Purpose:**

Validate the response to the Process Description Operation.

**b) Pre-conditions:**

- A Process Description document has been retrieved

**c) Test Method:**

1. Validate the retrieved document against the processOffering.yaml schema.

**d) References:**

Requirement 16

### A.4.4.8. Validate the Get Jobs Operation

**a) Test Purpose:**

Validate that the Get Jobs Operation behaves as required.

**b) Pre-conditions:**

- A process id is provided by test Validate Get Processes Operation Response
- Path = /processes/{processId}/jobs

**c) Test Method:**

- Issue an HTTP GET request using the test point URI
- Go to test Validate the Get Jobs Operation Response

**d) References:**

Requirement 17

### A.4.4.9. Validate the Get Jobs Operation Response

**a) Test Purpose:**

Validate the Get Jobs Operation Response.

**b) Pre-conditions:**

- A collection of Jobs has been retrieved

**c) Test Method:**

1. Validate the structure of the response as follows:

- For HTML use Human inspection
- For JSON use jobCollection.yaml

**d) References:**

Requirements 24, 25, 26, 27, 28 and 29

### A.4.4.10. Execute Operation

**a) Test Purpose:**

Validate that the Execute Operation behaves as required.

**b) Pre-conditions:**

- A process id is provided by test Validate Get Processes Operation Response
- Inputs are provided.
- Path = /processes/{processId}/jobs

**c) Test Method:**

- Issue an HTTP POST request using the test point URI
- TODO: Inputs/Execute request in body
- TODO sync/async
- Go to test Validate the Execute Operation Response

**d) References:**

Requirement 30

### A.4.4.11. Validate the Execute Operation Response

**a) Test Purpose:**

Validate the Execute Operation Response.

**b) Pre-conditions:**

- An Execute request has been issued to the server.

**c) Test Method:**

1. Validate the structure of the response as follows:
    - For HTML use Human Inspection
    - For JSON use statusInfo.yaml
2. Validate that the following links are included in the response document:
    - To itself
    - TODO when successful, link to result must be there
3. Validate that all links include the rel and type link parameters.

**d) References:**

Requirements 31 and 32

# Annex B: Revision History

| Date | Release | Editor | Primary clauses modified | Description |
|---|---|---|---|---|
| 2017-03-09 | 0.1 | Benjamin Pross | all | initial version |
| 2017-xx-xx | 0.2 | Benjamin Pross | 6 | Update REST/JSON section |
| 2017-10-16 | 0.3 | Stan Tillman | 1-5 | Update section 1-5 |
| 2018-08-15 | 0.3 | Benjamin Pross | all | Restructuring, added requirements classes |

# Annex C: Bibliography

**NOTE**

*Example Bibliography (Delete this note).*

The TC has approved Springer LNCS as the official document citation type.

Springer LNCS is widely used in technical and computer science journals and other publications

- For citations in the text please use square brackets and consecutive numbers: [1], [2], [3]

– Actual References:

[n] Journal: Author Surname, A.: Title. Publication Title. Volume number, Issue number, Pages Used (Year Published)

[n] Web: Author Surname, A.: Title, http://Website-Url

[1] OGC: OGC Testbed 12 Annex B: Architecture. (2015).