OGC API - Processes - Part 1

Core

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/ogcapi-processes-1/1.0

Internal reference number of this OGC® document: 18-062

Version: 1.0-draft.4

Category: OGC® Implementation Specification

Editor: Benjamin Pross

OGC API - Processes - Part 1: Core

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.

Table of Contents

i. Abstract

In many cases geospatial or location data, including data from sensors, must be processed before the information can be effectively used. The OGC Web Processing Service (WPS) Interface Standard provides a standard interface that simplifies the task of making simple or complex computational geospatial 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, the standard could 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. WPS 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 API - Processes API builds on the WPS 2.0 standard and defines the processing standards to communicate over a RESTful protocol using JSON encodings. This API is a newer and more modern way of programming and interacting with resources over the web while allowing better integration into existing software packages.

The resources that are provided by a server implementing the OGC API - Processes are listed in Table 1 below and include the capabilities document of the server, the list of processes available (Process list and Process description), jobs (running processes) and results of process executions.

Table 1. Requirements class 'Core' - Overview of core resources, applicable HTTP methods and links to the document sections

Resource	Path	HTTP method	Parameter	Document reference
Landing page	`/`	GET	N/A	7.2 API landing page
Conformance classes	`/conformance`	GET	N/A	7.4 Declaration of conformance classes
Process list	`/processes`	GET	N/A	7.7 Retrieve a process list
Process description	`/processes/{processID}`	GET	processID (in path)	7.8 Retrieve a process description
Job status info	`/processes/{processID}` `/jobs/{jobID}`	GET	processID, jobID (in path)	7.10 Retrieve status information about a job
Job results	`/processes/{processID}` `/jobs/{jobID}/results`	GET	processID, jobID (in path)	7.11 Retrieve job results

Resource	Path	HTTP method	Parameter	Document reference
Job status info or results	`/processes/{processID}/jobs`	POST	processID (in path), Execute request (contained in body)	7.9 Create a new job

In general, the HTTP GET operation is used to provide access to the resources described above. However, in order to create a new job, the HTTP POST method is used to create a new job by sending an execute request to the server.

Additionally, a list of jobs for a specific process can be requested.

Table 2. Requirements class 'Job list' - Overview of resources, applicable HTTP methods and links to the document sections

Resource	Path	HTTP method	Parameter	Document reference
Job list	`/processes/{processID}/jobs`	GET	processID (in path)	11 Requirements Class "Job list"

As a further addition to the operations accessible through HTTP GET and POST methods, in order to cancel a job execution and/or remove traces of the job execution the DELETE method can be used.

Table 3. Requirements class 'Dismiss' - Overview of resources, applicable HTTP methods and links to the document sections

Resource	Path	HTTP method	Parameter	Document reference
Job status info	`/processes/{processID}/jobs/{jobID}`	DELETE	processID, jobID (in path)	13 Requirements Class "Dismiss"

ii. Keywords

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

ogcdoc, OGC document, OGC API, Geospatial API, processes, Web Processing Service, WPS, JSON, HTML, geoprocessing, API, OpenAPI, HTML

iii. Preface

The Processing API is a continuation of WPS 2.0, a standard for web-based processing of geospatial data. The Processing API 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, the Processing API 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
- Hexagon
- CubeWerx Inc.
- Ecere Corporation
- Terradue Srl
- European Space Agency (ESA)
- Spacebel

v. Submitters

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

Name	**Representing**
Benjamin Pross *(editor)*	52°North GmbH
Stan Tillman	Hexagon
Panagiotis (Peter) A. Vretanos	CubeWerx Inc.
Jérôme Jacovella-St-Louis	Ecere Corporation
Pedro Gonçalves	Terradue Srl
Gérald Fenoy	Gérald Fenoy (Individual Member)
Cristiano Lopes	European Space Agency (ESA)
Christophe Noel	Spacebel

Chapter 1. Scope

This OGC Standard specifies a Web API that enables the execution of computing processes and the retrieval of metadata describing their purpose and functionality. Typically, these processes combine raster, vector, coverage and/or point cloud data with well-defined algorithms to produce new raster, vector, coverage and/or point cloud information.

Chapter 2. Conformance

This standard defines seven requirements / conformance classes.

The standardization targets of all conformance classes are "Web APIs."

The main requirements class is:

- Core.

The *Core* specifies requirements that all Web APIs have to implement.

Two requirements classes depend on the *Core* and specify representations for the resources specified in the *Core*:

- JSON, and
- HTML.

The JSON encoding is mandatory.

The *Core* does not mandate any encoding or format for the formal definition of the API. One option is the OpenAPI 3.0 specification and a requirements class has been specified for OpenAPI 3.0, which depends on the *Core*:

- OpenAPI Specification 3.0.

An implementation of the *Core* requirements class may also decide to use other API definition representations in addition or instead of an OpenAPI 3.0 definition. Examples for alternative API definitions: OpenAPI 2.0 (Swagger), future versions of the OpenAPI specification, an OWS Common 2.0 capabilities document or WSDL.

The *Core* is intended to be a minimal useful API for the execution of processes from the geospatial domain. It is designed to map the operations of a Web Processing Service 2.0.

The *Core* does not mandate the use of any specific process description to specify the interface of a process. Instead this standard defines and recommends the use of the following conformance class:

- OGC Process Description

which defines an information model, encoded in JSON, that may be used to specify the interface of a process.

Three additional conformance classes are specified that extend the basic functionality of an API:

- Job list, and
- Callback, and
- Dismiss.

Additional capabilities such as support for transactions, extended job monitoring, etc., may be specified in future parts of the OGC API - Processes series or as vendor-specific extensions.

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.

Table 4. Conformance class URIs

Conformance class	**URI**
Core	http://www.opengis.net/spec/ogcapi-processes-1/1.0/conf/core
OGC Process Description	http://www.opengis.net/spec/ogcapi-processes-1/1.0/conf/ogc-process-description
JSON	http://www.opengis.net/spec/ogcapi-processes-1/1.0/conf/json
HTML	http://www.opengis.net/spec/ogcapi-processes-1/1.0/conf/html
OpenAPI Specification 3.0	http://www.opengis.net/spec/ogcapi-processes-1/1.0/conf/oas30
Job list	http://www.opengis.net/spec/ogcapi-processes-1/1.0/conf/job-list
Callback	http://www.opengis.net/spec/ogcapi-processes-1/1.0/conf/callback
Dismiss	http://www.opengis.net/spec/ogcapi-processes-1/1.0/conf/dismiss

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 2616. Hypertext Transfer Protocol - HTTP/1.1. http://tools.ietf.org/html/rfc2616

IETF RFC 2617. HTTP Authentication: Basic and Digest Access Authentication. https://tools.ietf.org/html/rfc2617

IETF RFC 2246. Transport Layer Security. http://tools.ietf.org/html/rfc2246

IETF RFC 2818. HTTP Over TLS. http://tools.ietf.org/html/rfc2818

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

IETF RFC 4646: Tags for Identifying Languages. https://tools.ietf.org/html/rfc4646

IETF RFC 6749: The OAuth 2.0 Authorization Framework. https://tools.ietf.org/html/rfc6749

IETF RFC 8288: Web Linking https://tools.ietf.org/html/rfc8288

Chapter 4. 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.

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. 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.

4.3. JSON

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

4.4. 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.

4.5. 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.

4.6. 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.

4.7. 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 output

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

4.9. 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.

4.10. 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.

4.11. 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).

4.12. WPS Server

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

Chapter 5. Conventions

This section 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/ogcapi-processes-1/1.0

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

5.2. Link relations

To express relationships between resources, RFC 8288 (Web Linking) is used.

The following registered link relation types are used in this document.

- **alternate**: Refers to a substitute for the link's context.
- **license**: Refers to a license associated with the link's context.
- **service-desc**: Identifies service description for the context that is primarily intended for consumption by machines.
    - API definitions are considered service descriptions.
- **service-doc**: Identifies service documentation for the context that is primarily intended for human consumption.
- **self**: Conveys an identifier for the link's context.
- **status**: Identifies a resource that represents the context's status.
- **up**: Refers to a parent document in a hierarchy of documents.

In addition the following link relation types are used for which no applicable registered link relation type could be identified.

- **conformance**: Refers to a resource that identifies the specifications that the link's context conforms to.
- **exceptions**: The target URI points to exceptions of a failed process.
- **execute**: The target URI points to the execution endpoint of a process.
- **process-desc**: The target URI points to a specific process description.
- **processes**: The target URI points to the list of processes the API offers.
- **results**: The target URI points to the results of a process.

Each resource representation includes an array of links. Implementations are free to add additional links for all resources provided by the API.

5.3. Abbreviated Terms

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

5.4. 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.

5.5. Use of HTTPS

For simplicity, this document only refers to the HTTP protocol. This is not meant to exclude the use of HTTPS. It is simply a shorthand notation for "HTTP or HTTPS". In fact, most servers are expected to use HTTPS, not HTTP.

OGC Web API standards do not prohibit the use of any valid HTTP option. However, implementers should be aware that optional capabilities which are not in common use could be an impediment to interoperability.

5.6. HTTP URIs

This document does not restrict the lexical space of URIs used in the API beyond the requirements

of the HTTP and URI Syntax IETF RFCs. If URIs include reserved characters that are delimiters in the URI subcomponent, these have to be percent-encoded. See Clause 2 of RFC 3986 (URI Syntax) for details.

Chapter 6. Overview

The OGC API - Processes builds on the WPS 2.0 standard and is modularized, meaning that there is a separation between

- Core requirements, that specify basic capabilities and can easily be mapped to existing OGC Web Processing Services;
- More advanced functionality, that was not specified in WPS 2.0.

6.1. Encodings

This standard uses JSON as the encoding for requests and responses. The inputs and outputs of a process can be 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 with a web browser and will enable search engines to crawl and index the processes.

Chapter 7. Requirements Class "Core"

The following section describes the core requirements class.

7.1. Overview

Requirements Class	
http://www.opengis.net/spec/ogcapi-processes-1/1.0/req/core	
Target type	Web API
Dependency	API - Common Core
Dependency	RFC 2616 (HTTP/1.1)
Dependency	RFC 2818 (HTTP over TLS)
Dependency	RFC 8288 (Web Linking)

A server that implements the OGC API - Processes provides access to processes.

Each OGC API - Processes has a single `LandingPage` (path `/`) that provides links to

- the `APIDefinition` (no fixed path),
- 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 `Conformance` using HTTP GET returns a list of URIs of requirements classes implemented by the server.

The list of processes contains a summary of each process the OGC API - Processes 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 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.

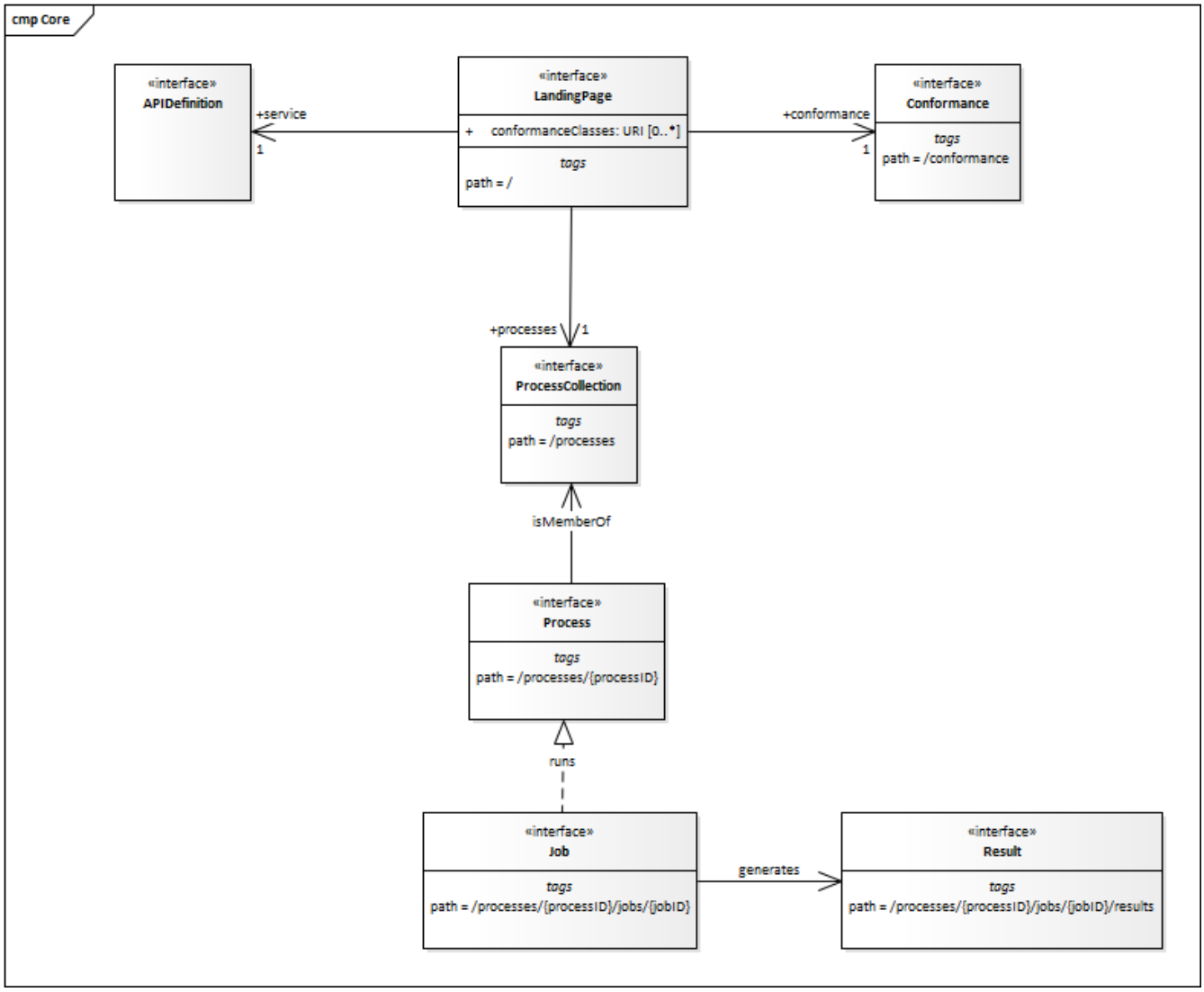

Figure 1. Resources in the Core requirements class

The OGC API - Processes standard is build upon the OGC API-Common standard. Table 5 Identifies the API-Common Requirements Classes which are applicable to each section of this standard.

Table 5. Mapping API - Processes Sections to API-Common Requirements Classes

API - Processes Section	API-Common Requirements Class
API Landing Page	http://www.opengis.net/spec/ogcapi_common-1/1.0/req/core
API Definition	http://www.opengis.net/spec/ogcapi_common-1/1.0/req/core
Declaration of Conformance Classes	http://www.opengis.net/spec/ogcapi_common-1/1.0/req/core
OpenAPI 3.0	http://www.opengis.net/spec/ogcapi_common-1/1.0/req/oas30
HTML	http://www.opengis.net/spec/ogcapi_common-1/1.0/req/html

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	**/req/core/landingpage-op** **The server SHALL support the HTTP GET operation at the path `/`.**

7.2.2. Response

Requirement 2	**/req/core/landingpage-success** **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 landingpage.yaml and include at least links to the following resources: * the API definition (relation type 'service-desc' or 'service-doc') * `/conformance` (relation type 'conformance') * `/processes` (relation type 'processes')**

Schema for the landing page

```
type: object
required:
  - links
properties:
  title:
    type: string
    example: Example processing server
  description:
    type: string
    example: Example server implementing the OGC API - Processes 1.0
  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/",
        "rel": "alternate",
        "type": "text/html",
        "title": "this document"
    },
    {
        "href": "http://processing.example.org/api",
        "rel": "service-desc",
        "type": "application/openapi+json;version=3.0",
        "title": "the API definition"
    },
    {
        "href": "http://processing.example.org/conformance",
        "rel": "conformance",
        "type": "application/json",
        "title": "OGC API - Processes 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 OGC API - Processes 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	/req/core/api-definition-op **The server SHALL support the HTTP GET operation at the path `/api`.**

7.3.2. Response

Requirement 4	/req/core/api-definition-success **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	/rec/core/api-definition-oas **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 OGC API - Processes 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 geospatial data types, etc., but it should not be required to read this standard to access the processes and results 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	/req/core/conformance-op **The server SHALL support the HTTP GET operation at the path `/conformance`.**

7.4.2. Response

Requirement 6	**/req/core/conformance-success** **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 OGC API - Processes 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/ogcapi_processes/1.0/req/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 JSON as encodings.

```
{
    "conformsTo": [
        "http://www.opengis.net/spec/ogcapi-processes-1/1.0/conf/core",
        "http://www.opengis.net/spec/ogcapi-processes-1/1.0/conf/json",
        "http://www.opengis.net/spec/ogcapi-processes-1/1.0/conf/html",
        "http://www.opengis.net/spec/ogcapi-processes-1/1.0/conf/oas30"
    ]
}
```

7.4.3. Error situations

See HTTP status codes for general guidance.

7.5. Use of HTTP 1.1

Requirement 7	**/req/core/http** **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 6 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.

Table 6. Typical HTTP status codes

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 authorized 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.
`429`	The user has sent too many requests in a given amount of time ("rate limiting").
`500`	An internal error occurred in the server.

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

Permission 1	**/per/core/additional-status-codes** **Servers MAY support other capabilities of the HTTP protocol and, therefore, MAY return other status codes than those listed in Table 6, too.**

7.6. Support for cross-origin requests

Access to content from a HTML page is by default prohibited for security reasons, if the content is located on another host than the webpage ("same-origin policy"). A typical example is a web-application accessing processes and data from multiple servers.

Recommendation 2	**/rec/core/cross-origin** **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	/rec/core/html **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 list

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

7.7.1. Operation

Requirement 8	/req/core/process-list **The server SHALL support the HTTP GET operation at the path `/processes`.**

7.7.2. Response

Requirement 9	/req/core/process-list-success **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 processList.yaml.**

Schema for the process list

```
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.

```
GET /processes HTTP/1.1
Host: processing.example.org
```

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": "process-desc",
            "title": "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	**/req/core/process** **The server SHALL support the HTTP GET operation at the path** `/processes/{processID}`**.**

7.8.2. Response

Requirement 11	/req/core/process-success
A	A successful execution of the operation SHALL be reported as a response with a HTTP status code 200.
B	The content of the response SHALL be a process description.

The Core does not mandate the use of a specific process description to specify the interface of a process. That said, the Core requirements class makes the following recommendation:

Recommendation 4	/rec/core/ogc-process-description Implementations SHOULD consider supporting the OGC process description.

7.8.3. Error situations

See HTTP status codes for general guidance.

Requirement 12	/req/core/process-exception/no-such-process 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. Create a new job

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

7.9.1. Operation

Requirement 13	/req/core/job-creation The server SHALL support the HTTP POST operation at the path `/processes/{processID}/jobs`.

7.9.2. Request body

Requirement 14	/req/core/job-creation-request The content of a request to create a new job SHALL be based upon the OpenAPI 3.0 schema execute.yaml.

Schema for execute

```
type: object
required:
  - outputs
  - mode
  - response
properties:
  inputs:
    type: array
    items:
      $ref: "input.yaml"
  outputs:
    type: array
    items:
      $ref: "output.yaml"
  mode:
    type: string
    enum:
      - sync
      - async
      - auto
  response:
    type: string
    enum:
      - raw
      - document
  subscriber:
    $ref: "subscriber.yaml"
```

The creation of a job can be done synchronously or asynchronously.

Requirement 15	/req/core/job-creation-mode
A	To create a job asynchronously, the "mode" attribute of the execute request body SHALL be set to "async".
B	To create a job synchronously, the "mode" attribute of the execute request body SHALL be set to "sync".
C	To let the server decide the execution mode, the "mode" attribute of the execute request body SHALL be set to "auto".

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"
    }],
    "response": "document",
    "mode": "async"
}
```

7.9.3. Response

In case of asynchronous execution, the requirements below apply:

Requirement 16	**/req/core/job-creation-success-async** **A successful execution of the operation SHALL be reported as a response with a HTTP status code 201.**

Requirement 17	**/req/core/job-creation-success-header-async** **The 201 response of the operation SHALL return a HTTP header named 'Location' which contains a link to the newly created job.**

For synchronous execution, the following requirement applies:

Requirement 18	/req/core/job-creation-success-sync
A	A successful execution of the operation SHALL be reported as a response with a HTTP status code 200.
B	If the "response" attribute of the execute request was set to "document", the content of the response SHALL be based upon the OpenAPI 3.0 schema results.yaml
C	If the "response" attribute of the execute request was set to "raw", the content of the response SHALL only include the one output selected by the execute request body.

7.9.4. 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]).

7.10. Retrieve status information about a job

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

7.10.1. Operation

Requirement 19	**/req/core/job** **The server SHALL support the HTTP GET operation at the path `/processes/{processID}/jobs/{jobID}`.**

7.10.2. Response

Requirement 20	/req/core/job-success 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
         - dismissed
   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.

```
GET /processes/EchoProcess/jobs/81574318-1eb1-4d7c-af61-4b3fbcf33c4f HTTP/1.1
Host: processing.example.org
```

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"
    }
  ]
}
```


7.10.3. 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	**/req/core/job-exception/no-such-job** **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".**

7.11. Retrieve job results

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

7.11.1. Operation

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

7.11.2. Response

Requirement 23	/req/core/job-result-success **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.

```
GET /processes/EchoProcess/jobs/81574318-1eb1-4d7c-af61-4b3fbcf33c4f/result HTTP/1.1
Host: processing.example.org
```

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/>"
        }
    }
]
```


7.11.3. 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	**/req/core/job-result-exception/no-such-job** **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	**/req/core/job-result-exception/result-not-ready** **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	**/req/core/job-result-failed** **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 Class "OGC Process Description"

The following section describes the `OGC Process Description` requirements class.

8.1. Overview

Requirements Class	
http://www.opengis.net/spec/ogcapi-processes-1/1.0/req/ogc-process-description	
Target type	Web API
Dependency	OGC API - Processes Core
Dependency	JSON

The OGC process description is an information model that may be used to specify the interface of a process. This model is an evolution of the process description model originally defined in the OGC WPS 2.0.2 Interface Standard and as such provides a bridge from legacy implementations into the OGC API Framework.

The process description allows the following information to be specified:

- An identifier for the process
- Descriptive metadata about the process;
  - a title
  - a narrative description of the process
  - keywords that can be associated with the process
  - references to additional metadata
- A description of each process input
- A description of each process output
- A job control specification that indicates whether the process can be invoked synchronously, asynchronously, or either.
- An output transmission specification that indicates how the results of a process are retrieved; either by value or by reference
- A section for additional parameters that are intended for communities of use to extend the process description as required

This clause defines a JSON-encoding of the OGC process description.

8.2. OGC process description

Requirement 27	/req/ogc-process-description/json-encoding A JSON-encoded OGC process description shall validate against the OpenAPI 3.0 schema: process.yaml.

Schema for a process (process.yaml)

```
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"
      }
    ]
}
```

Chapter 9. Security Considerations

The OGC API - Processes specifies a Web API that enables the execution of computing processes, the retrieval of metadata describing their purpose and functionality and the retrieval of the results of the process execution.

Access control should be considered for the following resources the API specifies:

Requirements class "Core":

- Process description
- Process list
- Job status info
- Job result

Requirements class "Job list"

- Job list

The following API operations modify resources and therefore require special attention:

Requirements class "Core":

- Execute, HTTP POST

Requirements class "Dismiss"

- Dismiss, HTTP DELETE

9.1. Security related HTTP Status Codes

9.1.1. Authentication

The HTTP 1.1 protocol (RFC 2616) supports the use of HTTP authentication via the HTTP status code 401: Unauthorized.

9.1.2. Access control

Access control is supported by the HTTP protocol through the status code 403: Forbidden.

9.2. Security related HTTP Headers

As per IETF RFC 2616, the **Authorization** request-header should be used to send credentials to the API. If the client application tries to perform a CRUD (create, read, update, delete) operation on a protected resource with no or wrong credentials, the server will answer with HTTP status code 401 along with a **WWW-Authenticate** response-header containing at least one challenge applicable to the target resource.

9.3. OAuth 2.0 and OpenID Connect 1.0

OAuth 2.0 (RFC 6749) is a widely used authorization protocol that enables resource protection for Web-APIs. On top of OAuth 2.0 sits OpenID Connect 1.0, which allows different client applications to verify the identity of a user.

9.3.1. OAuth 2.0 Bearer Token

Access tokens contain credentials and are used to identify a user and to grant access to protected resources. Bearer Tokens are the predominant type of access token used with OAuth 2.0[1].

The Bearer Token needs to be sent to the server using the **Authorization** header, for example:

```
GET /processes/EchoProcess HTTP/1.1
Host: processing.example.org
Authorization: Bearer eyJhbGciOiJSUz...
```

9.3.2. Claims and scopes

In OpenID Connect, claims are key/value pairs that contain information about a user. A typical example of a claim is "name":"John Doe". Scopes are tied to claims and are used to specify what access privileges are being requested.[2]. Custom claims can be created to serve the needs of a Web-API.

9.4. Validation

The OGC API - Processes specifies the execute operation using the HTTP POST method, which will be used as example for requests that contain content. The same guidance applies when using different HTTP methods like PUT, which can also contain content.

9.4.1. Validation of incoming Content-types

When sending POST requests, the client will specify the **Content-Type** header of the request content. The server should always check that the **Content-Type** header and the actual request content are of the same type. If there is no **Content-Type** header or the Content-Type is unexpected, the request should be rejected.

9.4.2. Input validation

After the Content-Type was checked, the received content of POST requests itself should be validated. Incorrect input validation can lead to injection attacks, memory leakage, and compromised systems[3].

9.5. Request rate limitation

To prevent distributed denial of service (DDoS) attacks or other misuse of the API, the number of requests in a given time can be limited. When the limit is exceeded, the HTTP error code 429 (too

many requests) should be returned and access tokens could be blocked temporarily.

[1] https://oauth.net/2/bearer-tokens/

[2] https://developer.okta.com/blog/2017/07/25/oidc-primer-part-1

[3] https://www.whitehatsec.com/glossary/content/input-validation

Chapter 10. Requirements classes for encodings

10.1. Overview

This clause specifies two pre-defined requirements classes for encodings to be used with the OGC API Processes.

- JSON
- HTML

The JSON encoding is mandatory.

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

10.2. Requirement Class "JSON"

This section defines the requirements class JSON.

Requirements Class	
http://www.opengis.net/spec/ogcapi-processes-1/1.0/req/json	
Target type	Web API
Dependency	OGC API - Processes Core
Dependency	JSON

Requirement 28	**/req/json/definition** `200`**-responses of the server SHALL support the following media type: *** `application/json`

10.3. Requirement Class "HTML"

This section defines the requirements class HTML.

Requirements Class	
http://www.opengis.net/spec/ogcapi-processes-1/1.0/req/html	
Target type	Web API
Dependency	OGC API - Processes Core
Dependency	API - Common HTML
Dependency	HTML5

Requirement 29	/req/html/definition Every `200`-response of an operation of the server SHALL support the media type `text/html`.

Requirement 30	/req//html/content 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 11. Requirements Class "OpenAPI 3.0"

11.1. Basic requirements

APIs conforming to this requirements class are documented as an OpenAPI Document.

Requirements Class	
http://www.opengis.net/spec/ogcapi-processes-1/1.0/req/oas30	
Target type	Web service
Dependency	OGC API - Processes 1.0 Core
Dependency	API - Common OpenAPI 3.0
Dependency	OpenAPI Specification 3.0.1

Requirement 31	**/req/oas30/oas-definition-1**
A	An OpenAPI definition in JSON using the media type `application/vnd.oai.openapi+json;version=3.0` and a HTML version of the API definition using the media type `text/html` SHALL be available.

Requirement 32	**/req/oas30/oas-definition-2** **The JSON representation SHALL conform to the OpenAPI Specification, version 3.0.**

TODO check if additional text is needed, e.g. for the additional process descriptions

Requirement 33	**/req/oas30/oas-impl** **The server SHALL implement all capabilities specified in the OpenAPI definition.**

11.2. Complete definition

Requirement 34	**/req/oas30/completeness** **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.

11.3. Exceptions

Requirement 35	**/req/oas30/exceptions-codes** **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/exception.ya
ml
  text/html:
    schema:
      type: string
```

11.4. Security

Requirement 36	**/req/oas30/security** **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.

Chapter 12. Requirements Class "Job list"

This requirement class specifies how to retrieve a job list from the API.

Requirements Class	
http://www.opengis.net/spec/ogcapi-processes-1/1.0/req/job-list	
Target type	Web API
Dependency	OGC API - Processes Core

12.1. Operation

Requirement 37	**/req/job-list/job-list-op** **The server SHALL support the HTTP GET operation at the path** `/processes/{processID}/jobs`**.**

12.2. Response

Requirement 38	**/req/job-list/job-list-success** **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 jobList.yaml.**

Schema for the job list

```
type: array
items:
  $ref: "jobInfo.yaml"
```

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.

```
[
    {
        "jobID": "8ca109b4-3b86-4a9c-a284-a6d50f91019e",
        "status": "running",
        "message": "Perform step 1/2",
        "progress": 50,
        "links": [
            {
                "href":
"http://processing.example.org/processes/EchoProcess/jobs/8ca109b4-3b86-4a9c-a284-
```

```
a6d50f91019e",
                "rel": "status",
                "type": "application/json",
                "hreflang": "en",
                "title": "Job status"
            }
        ]
    },
    {
        "id": "0cf773a5-282a-4e23-96cc-f5dab18123e5",
        "infos": {
            "jobID": "0cf773a5-282a-4e23-96cc-f5dab18123e5",
            "status": "successful",
            "message": "EchoProcess job finished successful",
            "progress": 100,
            "links": [
                {
                    "href":
"http://processing.example.org/processes/EchoProcess/jobs/0cf773a5-282a-4e23-96cc-
f5dab18123e5",
                    "rel": "status",
                    "type": "application/json",
                    "hreflang": "en",
                    "title": "Job status"
                },
                {
                    "href":
"http://processing.example.org/processes/EchoProcess/jobs/0cf773a5-282a-4e23-96cc-
f5dab18123e5/results",
                    "rel": "results",
                    "type": "application/json",
                    "hreflang": "en",
                    "title": "Job result"
                }
            ]
        }
    },
    {
        "id": "63aadd9c-c0e5-4a7f-80f0-228dbb158f09",
        "infos": {
            "jobID": "63aadd9c-c0e5-4a7f-80f0-228dbb158f09",
            "status": "failed",
            "message": "EchoProcess job failed",
            "progress": 100,
            "links": [
                {
                    "href":
"http://processing.example.org/processes/EchoProcess/jobs/63aadd9c-c0e5-4a7f-80f0-
228dbb158f09",
                    "rel": "status",
                    "type": "application/json",
```

```
                    "hreflang": "en",
                    "title": "Job status"
                },
                {
                    "href":
"http://processing.example.org/processes/EchoProcess/jobs/63aadd9c-c0e5-4a7f-80f0-
228dbb158f09/results",
                    "rel": "exceptions",
                    "type": "application/json",
                    "hreflang": "en",
                    "title": "Job exception"
                }
            ]
        }
    }
]
```


12.3. 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]).

Chapter 13. Requirements Class "Callback"

This conformance class specifies a callback mechanism for completed jobs. In contrast to the pull-based mechanism specified in Create a new job and Retrieve status information about a job, this conformance class specifies a push-based mechanism, where a subscriber-URL is passed to the API in the execute request. After the job is completed, the result response is sent to the specified URL.

Requirements Class	
http://www.opengis.net/spec/ogcapi-processes-1/1.0/req/callback	
Target type	Web API
Dependency	OGC API - Processes Core

Requirement 39	**/req/callback/job-callback** **The server SHALL support callback functions for jobs.**

Example for a callback in the execute operation

```
callbacks:
    jobCompleted:
      '{$request.body#/subscriber/successUri}':
        post:
          requestBody:
            content:
              application/json:
                schema:
                  $ref: '#/components/schemas/result'
          responses:
            '202':
              description: Results received successfully
```

If the server implements this conformance class, the optional subscriber element of the execute request JSON must be used.

It is possible to add multiple callbacks for getting progress updates and notifications of a successful job completion or of a failure.

Further guidance about how to use callbacks can be found in the OpenAPI documentation.

Chapter 14. Requirements Class "Dismiss"

This requirement class specifies how to dismiss a job. Dismiss can be seen as canceling a running job or removing artifacts of a finished job.

Requirements Class	
http://www.opengis.net/spec/ogcapi-processes-1/1.0/req/dismiss	
Target type	Web API
Dependency	OGC API - Processes Core

14.1. Operation

Requirement 40	**/req/dismiss/job-dismiss-op** **The server SHALL support the HTTP DELETE operation at the path** `/processes/{processID}/jobs/{jobID}`**.**

14.2. Response

Requirement 41	**/req/dismiss/job-dismiss-success** **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. The status SHALL be set to "dismissed".**

Example of a dismissed job encoded as JSON.

```
{
  "jobID" : "81574318-1eb1-4d7c-af61-4b3fbcf33c4f",
  "status": "dismissed",
  "message": "Job dismissed",
  "progress": 56,
  "links": [
    {
      "href": "http://processing.example.org/processes/EchoProcess/jobs",
      "rel": "up",
      "type": "application/json",
      "title": "The job list for the current process"
    }
  ]
}
```

14.3. 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 42	**/req/core/job-exception/no-such-job** **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".**

Chapter 15. Media Types

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

- `application/json` for all resources.

The typical HTML media type for all "web pages" in a server would be:

- `text/html`.

The media type for an OpenAPI 3.0 definition is `application/vnd.oai.openapi+json;version=3.0` (JSON) or . `application/vnd.oai.openapi;version=3.0` (YAML).

NOTE	The OpenAPI media types have not been registered yet with IANA and can change in the future.

Chapter 16. Additional API Building Blocks

The core requirements classes of this standard are designed for the following workflow:

1. Access the list of available processes
2. Access the description of a specific process
3. Create an execute JSON request (based on the description) and send it to the server via POST
4. Process the status info and/or results

This workflow is useful for generic client that are implemented against the JSON schemas and paths specified in this standard. Generic clients can communicate with any server implementing the OGC API - Processes. However, here may be limitations regarding the handling of input and output formats.

The approach describe above requires implementers of clients to have knowledge about the standard.

An alternative that could make it easier for implementers that are not familiar with OGC (API) standards is permitting deviations from strictly following the schemas and paths specified in this standard.

Permission 2	**/per/core/alternative-process-description** **Servers MAY support alternative means of describing the inputs and outputs of a process.**

This permission allows server implementations to describe a process, such as by defining the request and response body of a POST request to a process endpoint.

Permission 3	**/per/core/alternative-process-paths** **Servers MAY support alternative API paths.**

This permission allows server implementations to specify alternative paths to processes and jobs.

An example of an OpenAPI document making use of the building blocks is shown in the following:

```
openapi: 3.0.2
info:
  title: Alternative OGC API - Processes
  description: This is an alternative OGC API - Processes
  contact:
    email: you@your-company.com
  license:
    name: Apache 2.0
    url: http://www.apache.org/licenses/LICENSE-2.0.html
  version: 1.0.0
paths:
  /buffer:
    post:
```

```
      summary: execute buffer process
      operationId: executeBuffer
      requestBody:
        description: buffer inputs
        content:
          application/json:
            schema:
              $ref: '#/components/schemas/bufferExecute'
      responses:
        "200":
          description: buffer created
          content:
            application/json:
              schema:
                $ref: '#/components/schemas/bufferResult'
        "400":
          description: invalid input
components:
  schemas:
    bufferExecute:
      required:
      - data
      - width
      type: object
      properties:
        data:
          maxItems: 10
          minItems: 1
          type: array
          description: this is possible to provide the abstract in here
          items:
            oneOf:
            - type: string
              format: application/geo+json
            - type: string
              format: application/gml+xml
        width:
          maximum: 100
          minimum: 1
          type: integer
          default: 20
    bufferResult:
      type: object
      properties:
        outputs:
          type: array
          items:
            oneOf:
            - type: string
              format: application/geo+json
            - type: string
```

```
            format: application/gml+xml
```

The goals of these additional API building blocks are:

- enabling an more seamless integration of this API with other OGC API standards and
- enabling the use of tools to auto-generate clients from the API description.

Annex A: Abstract Test Suite (Normative)

A.1. Conformance Class Core

Conformance Class	
http://www.opengis.net/spec/ogcapi-processes-1/1.0/conf/core	
Target type	Web API
Requirements class	Requirements Class "Core"

A.1.1. General Tests

A.1.1.1. HTTP 1.1

a) Test Purpose:

Validate that the OGC API - Processes 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.1.2. Retrieve the API Description

A.1.2.1. 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.1.2.2. Landing Page Validation

a) Test Purpose:

Validate that the landing page complies with the required 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 OGC API - Processes contents.

d) References:

Requirement 2

A.1.2.3. 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 OGC API - Processes 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.1.2.4. 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.1.3. Identify the Test Points

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

A.1.3.1. 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 OGC API - Processes specification

c) Method:

FOR EACH paths property in the OpenAPI document If the path name is one of those specified in the OGC API - Processes 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.1.3.2. 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.1.3.3. 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.1.4. Processing the OpenAPI Document

A.1.4.1. Validate /api path

a) Test Purpose:

Validate the API definition provided through the /api path as 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.1.4.2. 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.1.4.3. 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.1.4.4. 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.1.4.5. Validate Get Processes Operation Response

a) Test Purpose:

Validate the response to the Get Processes Operation.

b) Pre-conditions:

- A Process List document has been retrieved

c) Test Method:

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

d) References:

Requirements 10, 11, and 12

A.1.4.6. 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 JSON-encoded OGC Process Description

d) References:

Requirement 10,11

A.1.4.7. 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.1.4.8. Validate the Get Jobs Operation Response

a) Test Purpose:

Validate the Get Jobs Operation Response.

b) Pre-conditions:

- A list 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 jobList.yaml

d) References:

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

A.1.4.9. 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.1.4.10. 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

A.2. Conformance Class OGC Process Description

Conformance Class	
http://www.opengis.net/spec/ogcapi-processes-1/1.0/conf/ogc-process-description	
Target type	Web API
Requirements class	Requirements Class "OGC Process Description"

A.2.1. Validate the OGC Process Description

A.2.1.1. Validate the JSON-encoded OGC Process Description

a) Test Purpose:

Validate the JSON-encoded OGC Process Description document.

b) Pre-conditions:

- The OGC Process Description document has been retrieved encoded in JSON.

c) Test Method:

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

d) References:

Requirement JSON OGC Process Description

A.3. Conformance Class JSON

Conformance Class	
http://www.opengis.net/spec/ogcapi-processes-1/1.0/conf/json	
Target type	Web API
Requirements class	Requirements Class "Core"

TODO

A.4. Conformance Class Job list

Conformance Class	
http://www.opengis.net/spec/ogcapi-processes-1/1.0/conf/job-list	
Target type	Web API
Requirements class	Requirements Class "Core"

TODO

A.5. Conformance Class Callback

Conformance Class	
http://www.opengis.net/spec/ogcapi-processes-1/1.0/conf/callback	
Target type	Web API

Requirements class	Requirements Class "Core"

TODO

A.6. Conformance Class Dismiss

Conformance Class	
http://www.opengis.net/spec/ogcapi-processes-1/1.0/conf/dismiss	
Target type	Web API
Requirements class	Requirements Class "Core"

TODO

A.7. Conformance Class HTML

Conformance Class	
http://www.opengis.net/spec/ogcapi-processes-1/1.0/conf/html	
Target type	Web API
Requirements class	Requirements Class "HTML"
Dependency	Conformance Class 'Core'

A.8. Conformance Class OpenAPI 3.0

Conformance Class	
http://www.opengis.net/spec/ogcapi-processes-1/1.0/conf/oas30	
Target type	Web API
Requirements class	Requirements Class "OpenAPI Specification 3.0"
Dependency	Conformance Class 'Core'

Abstract Test 1	**/conf/oas30/completeness**
Test Purpose	Verify the completeness of an OpenAPI document.
Requirement	/req/oas30/completeness

Test Method	Verify that for each operation, the OpenAPI document describes all HTTP Status Codes and Response Objects that the API uses in responses.

Abstract Test 2	**/conf/oas30/exceptions-codes**
Test Purpose	Verify that the OpenAPI document fully describes potential exception codes.
Requirement	/req/oas30/exceptions-codes
Test Method	Verify that for each operation, the OpenAPI document describes all HTTP Status Codes that may be generated.

Abstract Test 3	**/conf/oas30/oas-definition-1**
Test Purpose	Verify that JSON and HTML versions of the OpenAPI document are available.
Requirement	/req/oas30/oas-definition-1
Test Method	1. Verify that an OpenAPI definition in JSON is available using the media type `application/vnd.oai.openapi+json;version=3.0` and link relation `service-desc` 2. Verify that an HTML version of the API definition is available using the media type `text/html` and link relation `service-doc`.

Abstract Test 4	**/conf/oas30/oas-definition-2**
Test Purpose	Verify that the OpenAPI document is valid JSON.
Requirement	/req/oas30/oas-definition-2
Test Method	Verify that the JSON representation conforms to the OpenAPI Specification, version 3.0.

Abstract Test 5	**/conf/oas30/oas-impl**
Test Purpose	Verify that all capabilities specified in the OpenAPI definition are implemented by the API.

Requirement	/req/oas30/oas-impl
Test Method	1. Construct a path from each URL template including all server URL options and all enumerated path parameters. 2. For each path defined in the OpenAPI document, validate that the path performs in accordance with the API definition and the API-Features standard.

Abstract Test 6	**/conf/oas30/security**
Test Purpose	Verify that any authentication protocols implemented by the API are documented in the OpenAPI document.
Requirement	/req/oas30/security
Test Method	1. Identify all authentication protocols supported by the API. 2. Validate that each authentication protocol is described in the OpenAPI document by a Security Schema Object and its use is specified by a Security Requirement Object.

A.9. Overview

Compliance testing for OGC API - Processes 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.10. Conventions

The following conventions apply to this Abstract Test Suite:

A.10.1. 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.10.2. API Description Document

The OGC API - Processes 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.10.3. Resource Encodings

The OGC API - Processes standard mandates JSON as standard encoding for resources returned by the API. Therefore, this Abstract Test Suite asserts that the API returns resources encoded in JSON.

A.10.4. Processing Security Objects

OpenAPI does not provide a standard way to associate a security requirement with a single server URI. Therefore, OGC API - Processes 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 OpenAPI 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.10.5. Parameters

The following observations apply for OGC API - Processes parameters:

1. OGC API - Processes 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.10.6. Testable Paths

A testable path is a path which corresponds to one of the paths defined in the OGC API - Processes 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.11. Requirements Trace Matrix

TODO update

Requirement 1: API Landing Page Operation The server SHALL support the HTTP GET operation at the path /. Tests: A.4.2.1
Requirement 2: API Landing Page Response 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') - /lists (relation type 'data') Tests: A.4.2.2
Requirement 3: API Definition Operation The server SHALL support the HTTP GET operation at the path /api. Tests: A.4.2.3
Requirement 4: API Definition Response 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. Tests: A.4.2.3, A.4.2.4, A.4.4.1

Requirement 5: Conformance Class Operation The server SHALL support the HTTP GET operation at the path /conformance. Tests: A.4.4.2
Requirement 6: Conformance Class Response 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 OGC API - Processes requirements classes that the server conforms to. Tests: A.4.4.3
Requirement 7: HTTP 1.1 The server SHALL conform to HTTP 1.1. If the server supports HTTPS, the server SHALL also conform to HTTP over TLS. Tests: A.4.1.1

A.12. Abstract Test

The Test Approach used in the OGC API - Processes Abstract Test Suite includes four steps:

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

Identification of test points is a new requirement with OGC API - Processes. 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 OGC API - Processes standard.

Annex B: Revision History

Date	Release	Editor	Primary clauses modified	Description
2017-03-07	0.1	Benjamin Pross	all	initial version
2018-05-16	0.1	Stan Tillman	1-5	Update section 1-5
2018-07-25	1.0-draft	Benjamin Pross	all	1.0-draft
2018-08-15	1.0-draft	Benjamin Pross	all	Restructuring, added requirements classes
2018-11-29	1.0-draft	Benjamin Pross	7	Update schemas and examples
2019-02-20	1.0-draft	Benjamin Pross	7	Fix for #3
2019-03-21	1.0-draft	Benjamin Pross	6,7,8,9,10	Alignment with OAPI Common, adjust schemas
2019-03-27	1.0-draft	Tom Kralidis, Benjamin Pross	6,7,8,9,10	Fix for #7, align bbox schema to WFS
2019-03-28	1.0-draft	Benjamin Pross	7	Formatting
2019-03-29	1.0-draft	Benjamin Pross	7	Adjust schemas and examples
2019-04-16	1.0-draft	Benjamin Pross	7	Adjust schemas, fix validation errors, add more data types
2019-06-05	1.0-draft	Gérald Fenoy	7	Allow unbounded for maxOccurs, Fix issue with ValueDefinition references
2019-06-12	1.0-draft	Benjamin Pross	7	Possible solution for #26

Date	Release	Editor	Primary clauses modified	Description
2019-06-19	1.0-draft	Gérald Fenoy	7	Add additionalParameter.yaml, update metadata.yaml and, descriptionType.yaml, fix intendation
2019-06-20	1.0-draft	Brad Hards	6,7	Fix typo noted during OGC API presentation, fix for #34
2019-08-09	1.0-draft.2	Benjamin Pross	7	1.0-draft.2, use plural for results path, remove wrapper
2019-08-21	1.0-draft.2	Benjamin Pross	7	adjust schemas, examples and figures, remove section about web caching
2019-10-01	1.0-draft.3	Benjamin Pross	7	1.0-draft.3, minor edits
2019-10-10	1.0-draft.3	Gérald Fenoy, Tom Kralidis	7	Add implementations, Use status in place of infos in jobInfo definition
2019-10-22	1.0-draft.3	Benjamin Pross	7	Remove mandatory path /api, fix for #50
2020-01-06	1.0-draft.3	Francis Charette	7	Add implementation
2020-01-28	1.0-draft.3	Gérald Fenoy	7	Adjust schemas and examples
2020-02-03	1.0-draft.3	Benjamin Pross	7	Fix for #63
2020-02-18	1.0-draft.3	Chris Durbin	7	Fix for #61

Date	Release	Editor	Primary clauses modified	Description
2020-04-01	1.0-draft.3	Benjamin Pross	7	Add optional subscriber property to execute request, avoid duplication, create own type for entities with properties name and reference
2020-04-06	1.0-draft.3	Benjamin Pross	5,7	Abbreviate process-description link relation to process-desc, update example, alphabetical ordering of link relations
2020-04-09	1.0-draft.3	Benjamin Pross	7	Rename root.yaml to landingPage.yaml, add title and description to root.yaml
2020-04-28	1.0-draft.3	Benjamin Pross	7	Move examples, responses and parameters from core asciidoc to external files
2020-04-29	1.0-draft.3	Benjamin Pross	11	Add Requirements Class 'Callback'
2020-04-30	1.0-draft.3	Benjamin Pross	6,11	Move overview table to abstract, allow multiple URIs for callbacks

Date	Release	Editor	Primary clauses modified	Description
2020-05-05	1.0-draft.3	Gérald Fenoy	12	Add Requirements Class 'Dismiss', fix includes and section headers
2020-05-8	1.0-draft.3	Benjamin Pross	14	Add section with info about additional/alternative building blocks
2020-05-11	1.0-draft.3	Benjamin Pross	12	Move 'Job List' from core to separate Requirements Class
2020-05-12	1.0-draft.3	Panagiotis (Peter) A. Vretanos	N/A	Create a home for extensions to the core, initial check in of draft transactions extension, add placeholders for the quotation and billing APIs
2020-05-12	1.0-draft.3	Stan Tillman	6,7,8,9,10	Review
2020-05-20	1.0-draft.3	Panagiotis (Peter) A. Vretanos	2,7	Separate the OGC process description into its own conformance class.