OGC® DOCUMENT: 18-062R2

External identifier of this OGC® document: http://www.opengis.net/doc/IS/ogcapi-processes-1/1.0

Open Geospatial Consortium

OGC API - PROCESSES - PART 1: CORE

STANDARD

APPROVED

Version: 1.0.0
Submission Date: 2021-06-18
Approval Date: 2021-08-23
Publication Date: 2021-12-20
Editor: Benjamin Pross, Panagiotis (Peter) A. Vretanos

License Agreement

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

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

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

This license is effective until terminated. You may terminate it at any time by destroying the Intellectual Property together with all copies in any form. The license will also terminate if you fail to comply with any term or condition of this Agreement. Except as provided in the following sentence, no such termination of this license shall require the termination of any third party end-user sublicense to the Intellectual Property which is in force as of the date of notice of such termination. In addition, should the Intellectual Property, or the operation of the Intellectual Property, infringe, or in LICENSOR's sole opinion be likely to infringe, any patent, copyright, trademark or other right of a third party, you agree that LICENSOR, in its sole discretion, may terminate this license without any compensation or liability to you, your licensees or any other party. You agree upon termination of any kind to destroy or cause to be destroyed the Intellectual Property together with all copies in any form, whether held by you or by any third party.

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

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

Suggested additions, changes and comments on this document are welcome and encouraged. Such suggestions may be submitted using the online change request form on OGC web site: http://portal.opengeospatial.org/public_ogc/change_request.php

Copyright notice

Note

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

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

CONTENTS

LIST OF TABLES

LIST OF FIGURES

LIST OF RECOMMENDATIONS

I ABSTRACT

The OGC API — Processes — Part 1: Core Standard supports the wrapping of computational tasks into executable processes that can be offered by a server through a Web API and be invoked by a client application. The standard specifies a processing interface to communicate over a RESTful protocol using JavaScript Object Notation (JSON) encodings. The standard leverages concepts from the OGC Web Processing Service (WPS) 2.0 Interface Standard but does not require implementation of a WPS.

By way of background and context, in many cases geospatial or location data, including data from sensors, must be processed before the information can be effectively used. The WPS 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 Geographic Information Systems (GIS) as well as specialized processes for spatiotemporal modeling and simulation. While the 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 OGC API — Processes Standard 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 OGC API — Processes Standard addresses all of the use cases that were addressed by the WPS Standard, while also leveraging the OpenAPI specification and a resource-oriented approach.

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

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

RESOURCE	PATH	HTTP METHOI	PARAME	DOCUMENT REFERENCE
Landing page	`/`	GET	N/A	Clause 7.2
Conformance classes	`/conformance`	GET	N/A	Clause 7.4
Process list	`/processes`	GET	N/A	Clause 7.9
Process description	`/processes/{process ID}`	GET	process ID (in path)	Clause 7.10
Process execution	`/processes/{process ID}/execution`	POST	process ID (in path),	Clause 7.11

RESOURCE	PATH	HTTP METHOI	PARAME	DOCUMENT REFERENCE
			Execute request (contained in body)	
Job status info	`/jobs/{jobID}`	GET	jobID (in path)	Clause 7.12
Job results	`/jobs/{jobID}/results`	GET	jobID (in path)	Clause 7.13

In general, the HTTP GET operation is used to provide access to the resources described above. However, in order to execute a process, the HTTP POST method is used to send an execution request to the server.

Additionally, the /jobs endpoint can be used to grant access to a list of jobs.

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

RESOURCE	PATH	HTTP METHOI	PARAME	DOCUMENT REFERENCE
Job list	`/jobs`	GET	N/A	Clause 11

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

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

RESOURCE	PATH	HTTP METHOI	PARAME	DOCUMENT REFERENCE
Job status info	`/jobs/{jobID}`	DELETE	jobID (in path)	Clause 13

The OGC API — Processes — Part 1: Core Standard supports the wrapping of computational tasks into executable processes that can be offered by a server through a Web API and be invoked by a client application. Examples of computational processes that can be supported by implementations of this specification include raster algebra, geometry buffering, constructive area geometry, routing, imagery analysis and several others.

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 SECURITY CONSIDERATIONS

The OGC API — Processes Standard 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. The API makes use of different HTTP methods, namely GET, POST and DELETE. (Note that future extensions could introduce additional HTTP methods.)

HTTP methods can be classified as

- Safe, meaning that they do not alter the state of (a resource on) the server, and
- Idempotent, meaning that can be executed an indefinite number of times and deliver the same result.

Table 4 gives an overview of the classification of HTTP the methods used in this standard:

Table 4 — Classification of HTTP methods

HTTP METHOD	SAFE	IDEMPOTENT
GET	yes	yes
POST	no	no
DELETE	no	yes

Source RFC 7231, Hypertext Transfer Protocol (HTTP/1.1): Semantics and Content

The following resources can be retrieved using the safe HTTP GET operation and can contain sensitive information:

Requirements class “Core”:

- Process list
- Process description
- Job status info
- Job results

Requirements class “Job list”

- list

The following API operations use unsafe HTTP methods, modify resources and therefore require special attention:

Requirements class “Core”:

- Execute, HTTP POST

Requirements class “Dismiss”

- Dismiss, HTTP DELETE

III.A. Operations using HTTP GET

Most of the operations defined in this standard are use the safe HTTP GET operation. However, the resources that are returned by these operations contain information that could be used to exploit the API. Table 5 gives an overview of the resources specified in this standard and what kind of information they contain.

Table 5 — Requirements class ‘Core’ — Overview of core operations and returned sensitive information

RESOURCE	PATH	HTTP METHOI	INFORMATION DELIVERED
Landing page	/	GET	General information about the service, links to API endpoints
Conformance classes	/conformance	GET	List of conformance classes
Process list	/processes	GET	Process identifiers, links to process descriptions
Process description	/processes/{processID}	GET	Information about a process, e.g. inputs/outputs
Job status info	/jobs/{jobID}	GET	Status info, links to results or exceptions
Job results	/jobs/{jobID}/results	GET	Job results

The resources and contained information in more detail:

- The landing page contains links to the API endpoints and so leads to all other resources the API offers.
- The list of conformance classes could contain information about extensions like “dismiss” that pose additional security issues.

- The process list contains process identifiers and links to the respective process descriptions.
- The process description contains all necessary information needed to execute a process. This information can be used to send an JSON execute request to the API that will pass initial sanity checks, for example checks for the correct input/output identifiers. If this barrier is taken by an attacker, issues as discussed in section Clause III.B can occur.
- The job status info contains not only status information, but for finished processes also links to results / exceptions. The results of a process execution are a valuable resource as well as the exceptions that could contain hints about why the execution has failed.

Table 6 — Requirements class 'Job List' — Overview of operations and returned sensitive information

RESOURCE	PATH	HTTP METHOI	INFORMATION DELIVERED
Job list	`/jobs`	GET	List of job ids and status info, links to results or exceptions

The retrieval of the job list of a process returns the job ids and links to the respective job status.

III.B. Execute operation

The execute operation uses HTTP POST to create new processing jobs (process executions). As discussed above, the HTTP POST method is not safe and it poses the following threats if misused:

- The processing can use up considerable server resources, for example computing time, network traffic (when accessing referenced inputs) or storage space for inputs and outputs.
- Malicious inputs can be provided. Either inline in the execute request JSON or referenced.

Table 7 — Requirements class 'Core' — Overview of the execute operation and returned sensitive information

RESOURCE	PATH	HTTP METHOI	INFORMATION DELIVERED
Job status info	`/jobs`	POST	Job id, status info, (links to) results or exceptions

The ids that are used for new jobs and that are returned in the status info document should be created in a non-guessable way, for example using UUIDs. This will prevent random attempts to get job status information, results / exceptions or even cancel jobs / delete job artifacts.

RECOMMENDATION 1
/rec/job-list/access-control-job-list
Servers implementing the conformance class 'Job List' SHOULD have an access control in place for the /jobs endpoint to prevent misuse of job-ids.

III.C. Dismiss operation

The optional dismiss extension uses the HTTP DELETE method and can be used to

- Cancel a running job, and
- Remove artifacts of a finished job.

Both usages pose security related issues. The cancellation of a running job (if not done on purpose) is wasting the resources that the job has used until it was cancelled. The same goes for the unwanted removal of artifacts of a finished job. If the dismiss extension is implemented, access control for the operation should be considered. The dismiss operation is idempotent, as it is specified by this standard to be called using a specific job identifier. The first dismiss request to that identifier will result in a HTTP 200 (OK) status code. Continued dismiss requests using the same identifier result in a HTTP 410 (Gone) error code, but nothing else is changed on the server. A successful dismiss request returns a status info document containing the job identifier and the status "dismissed". This status info document has no further security implications.

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	AFFILIATION
Benjamin Pross *(editor)*	52°North GmbH
Stan Tillman	Hexagon
Panagiotis (Peter) A. Vretanos *(editor)*	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

1 SCOPE

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.

2

CONFORMANCE

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. OpenAPI 3.0 specification is one option for defining the Processing API. As such a requirements class has been specified for OpenAPI 3.0, which depends on the requirements class *Core*:

- OpenAPI Specification 3.0.

An implementation of the *Core* requirements class may also decide to use other API definition representations in addition to, 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.

NOTE: OpenAPI 3.0 offers an open, powerful and vendor neutral description format. While the use of OpenAPI 3.0 for the formal definition of the API is not mandatory, the requests/responses of the API specified in this standard are defined using OpenAPI 3.0 schemas. See also the note regarding /req/core/landingpage-success

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

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

This class defines an information model, encoded in JSON, which 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 8 — 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

3

NORMATIVE REFERENCES

3 NORMATIVE REFERENCES

The following documents are referred to in the text in such a way that some or all of their content constitutes requirements of this document. For dated references, only the edition cited applies. For undated references, the latest edition of the referenced document (including any amendments) applies.

Policy SWG: OGC 08-131r3, *The Specification Model — Standard for Modular specifications*. Open Geospatial Consortium (2009). https://portal.ogc.org/files/?artifact_id=34762&version=2

Arliss Whiteside Jim Greenwood : OGC 06-121r9, *OGC Web Service Common Implementation Specification*. Open Geospatial Consortium (2010). https://portal.ogc.org/files/?artifact_id=38867

Matthias Mueller: OGC 14-065, *OGC® WPS 2.0 Interface Standard*. Open Geospatial Consortium (2015). http://docs.opengeospatial.org/is/14-065/14-065r0.html

T. Dierks, C. Allen: IETF RFC 2246, *The TLS Protocol Version 1.0*. Internet Engineering Task Force, Fremont, CA (1999). https://raw.githubusercontent.com/relaton/relaton-data-ietf/master/data/reference.RFC.2246.xml

R. Fielding, J. Gettys, J. Mogul, H. Frystyk, L. Masinter, P. Leach, T. Berners-Lee: IETF RFC 2616, *Hypertext Transfer Protocol — HTTP/1.1*. Internet Engineering Task Force, Fremont, CA (1999). https://raw.githubusercontent.com/relaton/relaton-data-ietf/master/data/reference.RFC.2616.xml

J. Franks, P. Hallam-Baker, J. Hostetler, S. Lawrence, P. Leach, A. Luotonen, L. Stewart: IETF RFC 2617, *HTTP Authentication: Basic and Digest Access Authentication*. Internet Engineering Task Force, Fremont, CA (1999). https://raw.githubusercontent.com/relaton/relaton-data-ietf/master/data/reference.RFC.2617.xml

E. Levinson: IETF RFC 2387, *The MIME Multipart/Related Content-type*. Internet Engineering Task Force, Fremont, CA (1998). https://raw.githubusercontent.com/relaton/relaton-data-ietf/master/data/reference.RFC.2387.xml

E. Rescorla: IETF RFC 2818, *HTTP Over TLS*. Internet Engineering Task Force, Fremont, CA (2000). https://raw.githubusercontent.com/relaton/relaton-data-ietf/master/data/reference.RFC.2818.xml

T. Berners-Lee, R. Fielding, L. Masinter: IETF RFC 3986, *Uniform Resource Identifier (URI): Generic Syntax*. Internet Engineering Task Force, Fremont, CA (2005). https://raw.githubusercontent.com/relaton/relaton-data-ietf/master/data/reference.RFC.3986.xml

A. Phillips, M. Davis: IETF RFC 4646, *Tags for Identifying Languages*. Internet Engineering Task Force, Fremont, CA (2006). https://raw.githubusercontent.com/relaton/relaton-data-ietf/master/data/reference.RFC.4646.xml

R. Fielding, J. Reschke: IETF RFC 7231, *Hypertext Transfer Protocol (HTTP/1.1): Semantics and Content*. Internet Engineering Task Force, Fremont, CA (2014). https://

raw.githubusercontent.com/relaton/relaton-data-ietf/master/data/reference.RFC.7231.xml

J. Snell: IETF RFC 7240, *Prefer Header for HTTP*. Internet Engineering Task Force, Fremont, CA (2014). https://raw.githubusercontent.com/relaton/relaton-data-ietf/master/data/reference.RFC.7240.xml

M. Nottingham: IETF RFC 8288, *Web Linking*. Internet Engineering Task Force, Fremont, CA (2017). https://raw.githubusercontent.com/relaton/relaton-data-ietf/master/data/reference.RFC.8288.xml

T. Bray: IETF RFC 8259, *The JavaScript Object Notation (JSON) Data Interchange Format*. Internet Engineering Task Force, Fremont, CA (2017). https://raw.githubusercontent.com/relaton/relaton-data-ietf/master/data/reference.RFC.8259.xml

JSON Schema Validation: A Vocabulary for Structural Validation of JSON. https://json-schema.org/draft/2020-12/json-schema-validation.html

4

TERMS, DEFINITIONS AND ABBREVIATED TERMS

4 TERMS, DEFINITIONS AND ABBREVIATED TERMS

This document uses the terms defined in OGC Policy Directive 49, 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 document and OGC documents do not use the equivalent phrases in the ISO/IEC Directives, Part 2.

This document also uses terms defined in the OGC Standard for Modular specifications (OGC 08-131r3), also known as the ‘ModSpec’. The definitions of terms such as standard, specification, requirement, and conformance test are provided in the ModSpec.

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

4.1. Terms and definitions

4.1.1. **process**

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

$p: X \rightarrow Y$

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.

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.

4.1.2. **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.

4.1.3. 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.

4.1.4. Link

The term “link” is commonly used as substitute for URL or URI. In this standard, “link” refers to an element described by the schema for a link as shown at link.yaml. This is a JSON element containing properties like “rel” (relation) and “href”. The value of the “href” property is an URI.

4.1.5. Link header

HTTP Link header, as defined in RFC 8288 (Web Linking).

4.1.6. 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.

4.1.7. 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.

4.1.8. 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.

4.1.9. 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).

4.1.10. 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.

4.1.11. REST

The Representational State Transfer (REST) style is an abstraction of the architectural elements within a distributed hypermedia system. REST focuses on the roles of components, the constraints upon their interaction with other components, and their interpretation of significant data elements. It encompasses the fundamental constraints upon components, connectors, and data that define the basis of the Web architecture, and thus the essence of its behavior as a network-based application. An API that conforms to the REST architectural principles/ constraints is called a **RESTful API**. (Source: OGC 18-088)

4.2. Abbreviated terms

API	Application Programming Interface
CITE	Compliance Interoperability & Testing Evaluation

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

5

CONVENTIONS

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.

- **http://www.opengis.net/def/rel/ogc/1.0/conformance**: Refers to a resource that identifies the specifications that the link's context conforms to.
- **http://www.opengis.net/def/rel/ogc/1.0/exceptions**: The target URI points to exceptions of a failed process.
- **http://www.opengis.net/def/rel/ogc/1.0/execute**: The target URI points to the execution endpoint of the server.
- **http://www.opengis.net/def/rel/ogc/1.0/job-list**: The target URI points to the list of jobs.
- **http://www.opengis.net/def/rel/ogc/1.0/processes**: The target URI points to the list of processes the API offers.
- **http://www.opengis.net/def/rel/ogc/1.0/results**: The target URI points to the results of a job.

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

5.3. Use of HTTPS

For simplicity, this document only refers to the HTTP protocol. This is not meant to exclude the use of HTTPS. This 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.

5.4. 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.

6

OVERVIEW

6 OVERVIEW

The OGC API — Processes Standard builds on the WPS 2.0 Standard and is modularized. This means 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 is not specified in WPS 2.0.

6.1. Encodings

JSON is the encoding for requests and responses. The inputs and outputs of a process can be any format. The formats 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.

7

REQUIREMENTS CLASS "CORE"

7 REQUIREMENTS CLASS "CORE"

The following section describes the core requirements class.

7.1. Overview

REQUIREMENTS CLASS 1	
http://www.opengis.net/spec/ogcapi-processes-1/1.0/req/core	
Obligation	requirement
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 Standard provides access to processes.

Each implementation of the OGC API — Processes Standard has a single `LandingPage` (path `/`) that provides links to

- The `APIDefinition` (no fixed path),
- The `Conformance` statements (path `/conformance`),
- The `processes` metadata (path `/processes`).

Note that additional requirements classes may introduce additional links for the landing page.

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 offered by the OGC API — Processes implementation, 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.

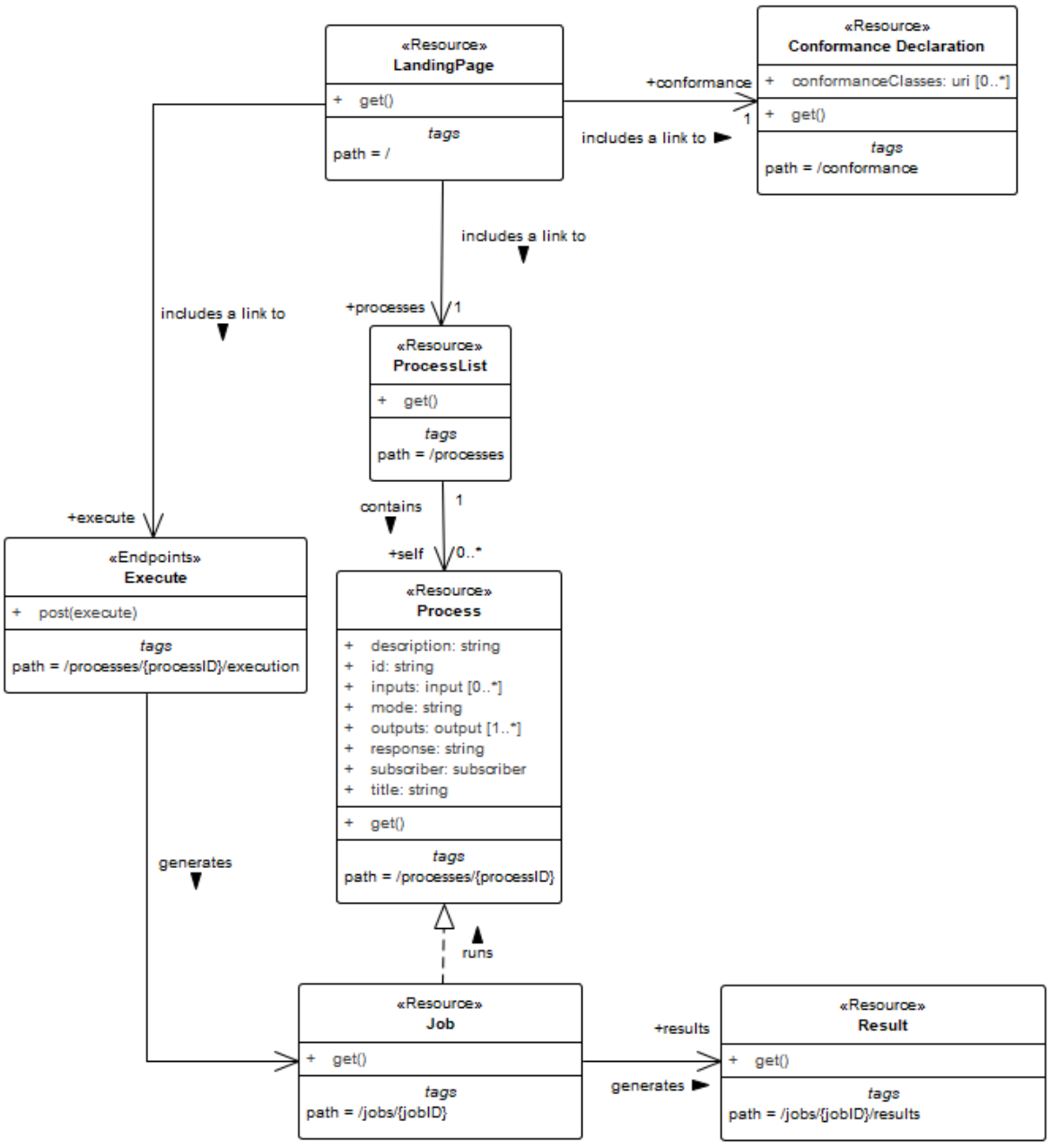

Figure 1 — Resources in the Core requirements class

The OGC API — Processes Standard utilizes elements of <draft> the OGC API-Common standard. Table 9 Identifies the API-Common Requirements Classes which are applicable to each section of this standard.

Table 9 — 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 defines the requirements 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:

REQUIREMENT 2

- the API definition (relation type 'service-desc' or 'service-doc')
- `/conformance` (relation type 'http://www.opengis.net/def/rel/ogc/1.0/conformance')
- `/processes` (relation type 'http://www.opengis.net/def/rel/ogc/1.0/processes')

NOTE 1: The term "*...based upon the OpenAPI 3.0 schema...*" used in the requirements of this specification means that OpenAPI 3.0 is used to define:

- all the required properties of the respective request/response schema,
- and any optional properties of the respective request/response schema.

It also means that unless explicitly excluded these schemas are extensible with additional properties not defined in the schema using the `additionalProperties` mechanism defined in the OpenAPI 3.0 specification.

```
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 Standard
  links:
    type: array
    items:
      $ref: "link.yaml"
```

Figure 2 — Schema for the landing page

NOTE 2: This schema can also be obtained from `landingPage.yaml`.

```
type: object
required:
  - href
properties:
  href:
    type: string
  rel:
    type: string
    example: service
  type:
    type: string
    example: application/json
  hreflang:
    type: string
    example: en
  title:
    type: string
```

Figure 3 — Schema for a link

NOTE 3: This schema can also be obtained from `link.yaml`.

Example — Landing page response document:

```
{
 "links": [{
  "href": "http://processing.example.org/oapi-p?f=application/json",
  "rel": "self",
  "type": "application/json",
  "title": "This document"
 },{
  "href": "http://processing.example.org/oapi-p?f=text/html",
  "rel": "alternate",
  "type": "text/html",
  "title": "This document as HTML"
 },
 {
  "href": "http://processing.example.org/oapi-p/api?f=application/json",
  "rel": "service-desc",
  "type": "application/json",
  "title": "API definition for this endpoint as JSON"
 },
 {
  "href": "http://processing.example.org/oapi-p/api?f=text/html",
  "rel": "service-desc",
  "type": "text/html",
  "title": "API definition for this endpoint as HTML"
 },
 {
  "href": "http://processing.example.org/oapi-p/conformance",
  "rel": "http://www.opengis.net/def/rel/ogc/1.0/conformance",
  "type": "application/json",
  "title": "OGC API - Processes conformance classes implemented by this server"
 },
 {
  "href": "http://processing.example.org/oapi-p/processes",
  "rel": "http://www.opengis.net/def/rel/ogc/1.0/processes",
  "type": "application/json",
  "title": "Metadata about the processes"
 },
 {
  "href": "http://processing.example.org/oapi-p/jobs",
  "rel": "http://www.opengis.net/def/rel/ogc/1.0/job-list",
  "title": "The endpoint for job monitoring"
 }]
}
```

7.2.3. Error situations

See Clause 7.5.1 for general guidance.

7.3. Retrieve an API definition

The following section defines the requirements to retrieve an API definition.

7.3.1. Operation

Every implementation of OGC API — Processes provides an API definition that describes the capabilities of the server. This definition is 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 URIs of all API definitions referenced from the landing page SHALL support the HTTP GET method.

PERMISSION 1
/per/core/api-definition-uri
The API definition is metadata about the API and strictly not part of the API itself, but it MAY be hosted as a sub-resource to the base path of the API, for example, at path `/api`. There is no need to include the path of the API definition in the API definition itself.

Note that multiple API definition formats can be supported.

7.3.2. Response

REQUIREMENT 4
/req/core/api-definition-success A successful execution of the operation to get the API definition document SHALL be reported as a response with a HTTP status code `200`.
The server SHALL return an API definition document.

RECOMMENDATION 2
/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.

NOTE: Two common approaches are:

- an additional path for each encoding of each resource (this can be expressed, for example, using format specific suffixes like ".html");
- an additional query parameter (for example, "accept" or "f") that overrides the Accept header of the HTTP request.

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 implementation of 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 Clause 7.5.1 for general guidance.

7.4. Declaration of conformance classes

7.4.1. Operation

To support "generic" clients for accessing servers implementing OGC API — Processes 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

REQUIREMENT 6

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 confClasses.yaml and list all OGC API — Processes conformance classes that the server conforms to.

```
type: object
required:
  - conformsTo
properties:
  conformsTo:
    type: array
    items:
      type: string
      example: "http://www.opengis.net/spec/ogcapi-processes-1/1.0/conf/core"
```

Figure 4 — Schema for the list of conformance classes

NOTE: This schema can also be obtained from `confClasses.yaml`.

Example — 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 Clause 7.5.1 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 10 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 10 — 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.
`204`	The request was successful but did not generate any content.
`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.
`410`	The target resource is no longer available at the origin server.
`429`	The user has sent too many requests in a given amount of time (“rate limiting”).
`500`	An internal error occurred in the server.
`501`	The server does not support the functionality required to fulfill the request.

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

PERMISSION 2
/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 10, too.

When a server encounters an error in the processing of a request, it may wish to include information in addition to the status code in the response. Since Web API interactions are often machine-to-machine, a machine-readable report would be preferred. RFC 7807 addresses this need by providing "Problem Details" response schemas for both JSON and XML.

RECOMMENDATION 3
/rec/core/problem-details
A server SHOULD include a "Problem Details" report in an error response in accordance with RFC 7807.

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"). Cross-origin resource sharing is a mechanism that allows restricted resources on a web page to be requested from another domain outside the domain from which the first resource was served. A typical example is a web-application accessing processes and data from multiple servers.

RECOMMENDATION 4
/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 5
/rec/core/access-control-expose-headers
If the server is intended to be accessed from the browser and if Cross-origin resource sharing is supported, the `Access-Control-Expose-Headers` header SHOULD be used and the header

RECOMMENDATION 5

SHOULD contain the value `location` to enable the browser to access the `location` header of the response.

RECOMMENDATION 6

/rec/core/html

To support browsing an implementation of OGC API — Processes with a web browser and to enable search engines to crawl and index a process, implementations SHOULD consider to support an HTML encoding.

7.7. Limit parameter

Several resources defined in this specification (see Retrieve a process list, Retrieve a job list) use the `limit` parameter to control the number of results that are presented in a response.

- The client can request a limit it is interested in.
- The server likely has a default value for the limit, and a maximum limit.
- If the server has any more results available than it returns (the number it returns is less than or equal to the requested/default/maximum limit) then the server will include a link to the next set of results.

So (using the default/maximum values of 10/1000 from the OpenAPI fragment in requirement `/req/core/pl-limit-definition` or `/req/job-list/limit-definition`):

- If you ask for 10 results, you will get 0 to 10 (as requested) and if there are more, a `next` link;
- If you don't specify a limit, you will get 0 to 10 (default) and if there are more, a `next` link;
- If you ask for 5000 results, you might get up to 1000 (server-limited) and if there are more, a `next` link;
- If you follow the next link from the previous response, you might get up to 1000 additional results and if there are more, a `next` link.

This document make requirements and recommendations about links in general and the `next` link in particular at the appropriate resource end points.

This document does not mandate any specific implementation approach for the `next` links.

An implementation could use opaque links that are managed by the server. It is up to the server to determine how long these links can be dereferenced. Clients should be prepared to receive a 404 response.

Another implementation approach is to use an implementation-specific parameter that specifies the index within the result set from which the server begins presenting results in the response, like `offset` or the `startIndex` parameter that was used in WFS 2.0.

The API will return no `next` link, if it has returned all selected results, and the server knows that. However, the server may not be aware that it has already returned all selected results. For example, if the request states `limit=10` and the query to the backend returns 10 results, the server may not know, if there are more results or not (in most cases there will be more results), unless the total number of results is also computed, which may be too costly. The server will then add the next link, and if there are no more results, dereferencing the next link will return an empty results list and no next link. This behavior is consistent with the statements above.

Clients should not assume that paging is safe against changes to dataset while a client iterates through `next` links. If a server provides opaque links these could be safe and maintain the state during the original request. Using a parameter for the start index, however, will not be safe.

This document also makes recommendations about include a `prev` link at the appropriate resource end points. Providing `prev` links supports navigating back and forth between pages, but depending on the implementation approach it may be too complex to implement.

7.8. Link headers

RECOMMENDATION 7	
/rec/core/link-header	
A	Links included in payload of responses SHOULD also be included as `Link` headers in the HTTP response according to RFC 8288, Clause 3. This recommendation does not apply, if there are a large number of links included in a response or a link is not known when the HTTP headers of the response are created.

7.9. Retrieve a process list

The following section defines the requirements to retrieve the available processes offered by the server.

7.9.1. Operation

7.9.1.1. Process list

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

7.9.1.2. Parameter `limit`

REQUIREMENT 9	
/req/core/pl-limit-definition	

A	The operation SHALL support a parameter `limit` with the following characteristics (using an OpenAPI Specification 3.0 fragment):
	`name: limit`
	`in: query`
	`required: false`
	`schema:`
	`  type: integer`
	`  minimum: 1`
	`  maximum: 10000`
	`  default: 10`
	`style: form`
	`explode: false`

PERMISSION 3	
/per/core/limit-default-minimum-maximum	
A	The values for `minimum`, `maximum` and `default` in requirement `/req/core/limit-definition` are only examples and MAY be changed.

REQUIREMENT 10	
/req/core/pl-limit-response	
A	The response SHALL not contain more process summaries than specified by the optional `limit` parameter.

REQUIREMENT 10	
B	If the API definition specifies a maximum value for `limit` parameter, the response SHALL not contain more process summaries than this maximum value.

PERMISSION 4	
/per/core/limit-response	
A	The server MAY return fewer process summaries than requested (but not more).

7.9.2. Response

REQUIREMENT 11
/req/core/process-list-success
A successful execution of the *process* 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.

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

Figure 5 — Schema for the process list

NOTE 1: This schema can also be obtained from `processList.yaml`.

```
allOf:
  - $ref: "descriptionType.yaml"
  - type: object
    required:
      - id
      - version
    properties:
      id:
        type: string
      version:
        type: string
      jobControlOptions:
        type: array
```

```
      items:
        $ref: "jobControlOptions.yaml"
    outputTransmission:
      type: array
      items:
        $ref: "transmissionMode.yaml"
    links:
      type: array
      items:
        $ref: "link.yaml"
```

Figure 6 — Schema for a process summary

NOTE 2: This schema can also be obtained from `processSummary.yaml`.

(see also: `descriptionType.yaml`).

```
type: string
enum:
  - sync-execute
  - async-execute
  - dismiss
```

Figure 7 — Schema for the job control options

NOTE 3: This schema can also be obtained from `jobControlOptions.yaml`.

```
type: string
enum:
  - value
  - reference
default:
  - value
```

Figure 8 — Schema for the transmission mode

NOTE 4: This schema can also be obtained from `transmissionMode.yaml`.

The number of process summaries returned depends on the server and the parameter `limit`.

See the discussion about the `limit` parameter in the Limit parameter section.

REQUIREMENT 12	
/req/core/pl-links	
A	A `200`-response SHALL include the following links: • a link to this response document (relation: `self`), • a link to the response document in every other media type supported by the service (relation: `alternate`).

See the discussion about the `next` links in the Limit parameter section.

RECOMMENDATION 8
/rec/core/next-1

RECOMMENDATION 8	
A	A `200`-response SHOULD include a link to the next page (relation: `next`) of process summaries, if more process summaries have been selected than returned in the response.

RECOMMENDATION 9	
/rec/core/next-2	
A	Dereferencing a next page link (relation: `next`) SHOULD return additional process summaries from the set of selected process summaries that have not yet been returned.

RECOMMENDATION 10	
/rec/core/next-3	
A	The number of process summaries in a response to dereferencing a next page link (relation: `next`) SHOULD follow the same rules as for the response to the original query and again include a next page link (relation: `next`), if there are more process summaries in the selection that have not yet been returned.

See the discussion about the `prev` link in the Limit parameter section.

PERMISSION 5	
/per/core/prev	
A	A response to dereferencing a next page link (relation: `next`) MAY include a previous page link (relation: `prev`) to the resource that included the next page link (relation: `next`).

Example 1 — A HTTP GET request for retrieving the list of offered processes encoded as JSON.:

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

Example 2 — A Process list encoded as JSON.:

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

```
        "value",
        "reference"
      ],
      "links": [
        {
          "href": "https://processing.example.org/oapi-p/processes/
EchoProcess",
          "type": "application/json",
          "rel": "self",
          "title": "process description"
        }
      ]
    }
  ],
  "links": [
    {
      "href": "https://processing.example.org/oapi-p/processes?f=json",
      "rel": "self",
      "type": "application/json"
    },
    {
      "href": "https://processing.example.org/oapi-p/processes?f=html",
      "rel": "alternate",
      "type": "text/html"
    }
  ]
}
```


7.9.3. Error situations

See Clause 7.5.1 for general guidance.

7.10. Retrieve a process description

The following section defines the requirements to retrieve metadata about a process.

7.10.1. Operation

REQUIREMENT 13
/req/core/process
The server SHALL support the HTTP GET operation at the path `/processes/{processID}`.

7.10.2. Response

REQUIREMENT 14	
/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 11
/rec/core/ogc-process-description
Implementations SHOULD consider supporting the OGC process description.

7.10.3. Error situations

See Clause 7.5.1 for general guidance.

REQUIREMENT 15
/req/core/process-exception/no-such-process
If the operation is executed using an invalid process identifier, the response SHALL be HTTP status code 404. The content of that response SHALL be based upon the OpenAPI 3.0 schema exception.yaml. The `type` of the exception SHALL be "http://www.opengis.net/def/exceptions/ogcapi-processes-1/1.0/no-such-process".

7.11. Execute a process

This section describes the requirements for executing a process.

Depending on the description of the process and the negotiated process execution mode, process execution may result in the creation of a job resource.

7.11.1. Operation

REQUIREMENT 16
/req/core/process-execute-op
The server SHALL support the HTTP POST operation at the path `/processes/{processID}/execution`.

7.11.2. Request body

7.11.2.1. Content schema

REQUIREMENT 17
/req/core/process-execute-request
The content of a request the request body SHALL be based upon the OpenAPI 3.0 schema execute.yaml.

```
type: object
properties:
  inputs:
    additionalProperties:
      oneOf:
        - $ref: "inlineOrRefData.yaml"
        - type: array
          items:
            $ref: "inlineOrRefData.yaml"
  outputs:
    additionalProperties:
      $ref: "output.yaml"
  response:
    type: string
    enum:
      - raw
      - document
    default:
      - raw
  subscriber:
    $ref: "subscriber.yaml"
```

Figure 9 — Schema for execute

NOTE 1: This schema can also be obtained from `execute.yaml`.

```
oneOf:
  - $ref: "inputValueNoObject.yaml"
  - $ref: "qualifiedInputValue.yaml"
```

```
  - $ref: "link.yaml"
```

Figure 10 — Schema for an in-line or referenced process input value

NOTE 2: This schema can also be obtained from `inlineOrRefData.yaml`.

(see also: inputValueNoObject.yaml, qualifiedInputValue.yaml, link.yaml)

```
type: object
properties:
  format:
    $ref: "format.yaml"
  transmissionMode:
    $ref: "transmissionMode.yaml"
```

Figure 11 — Schema for a process output

NOTE 3: This schema can also be obtained from `output.yaml`.

(see also: format.yaml, transmissionMode.yaml)

7.11.2.2. Process inputs

Overview:

Each process input is a name/value pair that appears in the `inputs` section of an execute request.

The name of each input is its identifier as specified by the input's definition in the process description.

Process input values in an execute request can be specified in-line or by reference.

REQUIREMENT 18	
/req/core/process-execute-inputs	
A	The server SHALL support process input values specified in-line in an execute request (i.e. by value).
B	The server SHALL support process input values specified by reference (i.e. using a link).

As shown in inlineOrRefData.yaml, a process input value that is specified in-line in an execute request can be:

- a simple literal value,
- an array,
- a qualified value,
- a binary value,

- or a bounding box.

Simple literal values:

A simple literal value can be a string, number, integer or Boolean.

```
oneOf:
  - type: string
  - type: number
  - type: integer
  - type: boolean
  - type: array
  - $ref: "binaryInputValue.yaml"
  - $ref: "bbox.yaml"
```

Figure 12 — Schema for a simple literal value

NOTE 1: This schema can also be obtained from `inputValueNoObject.yaml`.

(see also: binaryInputValue.yaml, bbox.yaml)

Example 1 — Simple literal value examples.: A string literal:

```
"stringInput": "String value"
```

A date string:

```
"dateInput": "2021-05-24T20:40:13-05:00"
```

A number:

```
"numberInput": 3.14159
```

An integer:

```
"integerInput": 10
```

A Boolean:

```
"booleanInput": true
```

Array of values:

Array elements, as per inlineOrRefData.yaml, can be:

- simple literals,
- embedded arrays,
- qualified values,
- binary values,
- bounding box values,
- or references to values using links.

REQUIREMENT 19	
/req/core/process-execute-input-array	
Conditions	The process input is defined in the process description as having a maximum cardinality of greater than one (maxOccurs>1).
A	The server SHALL support process input values encoded as an array.
B	This SHALL be true even if the input consists of a single value.

Example 2 — Array value examples.: An array of simple values:

```
"arrayOfSimpleValues": [1, 2, 4, 10, 7]
```

An array with a single simple value:

```
"arrayOfSimpleValues": ["a"]
```

An array of arrays of simple values:

```
"arrayOfArrays": [[1,2,3,4], ["a","b","c","d"]]
```

An array of objects values:

```
"arrayOfQualifiedValues": [
  {
    "value": {
      "measurement": 10.3,
      "uom": "m",
      "reference": "https://ucum.org/ucum-essence.xml"
    }
  },
  {
    "value": {
      "measurement": 10.5,
      "uom": "m",
      "reference": "https://ucum.org/ucum-essence.xml"
    }
  },
  {
    "value": {
      "measurement": 10.9,
      "uom": "m",
      "reference": "https://ucum.org/ucum-essence.xml"
    }
  },...
],
```

NOTE 2: In an execute request, as per requirement /req/core/process-execute-input-inline-

object, object values must be encoded as qualified values to prevent unintended ambiguity with the built-in value types (i.e. bounding boxes, links or qualified values).
Qualified values:

A qualified value is a value that can be optionally qualified with a `format` parameter.

Qualified values can be used to encode process input values that, according to their definition in the process description, can be of multiple media types. The `format` parameter is used to identify the specific media type being provided as the process input.

Qualified values can also be used to encode object-valued process inputs in order to avoid ambiguity with the built-in value schemas defined in this standard (i.e. bbox.yaml, link.yaml or qualifiedInputValue.yaml itself).

The actual *value* in a qualified value object is specified using the `value` key. The value of the `value` key is an instance of inputValue.yaml.

```
allOf:
  - $ref: "format.yaml"
  - type: object
    required:
      - value
    properties:
      value:
        $ref: "inputValue.yaml"
```

Figure 13 — Schema for a qualified value

NOTE 3: This schema can also be obtained from `qualifiedInputValue.yaml`.

```
type: object
properties:
  mediaType:
    type: string
  encoding:
    type: string
  schema:
    oneOf:
      - type: string
        format: url
      - type: object
```

Figure 14 — Schema for a format qualifier

NOTE 4: This schema can also be obtained from `format.yaml`.

```
oneOf:
  - $ref: "inputValueNoObject.yaml"
  - type: object
```

Figure 15 — Schema of a process input value

NOTE 5: This schema can also be obtained from `inputValue.yaml`.

(see also: inputValueNoObject.yaml)

REQUIREMENT 20	
/req/core/process-execute-input-inline-object	
Conditions	a) The process input value is specified in-line in an execute request.

REQUIREMENT 20	
	b) The process input is defined as an object according to its schema from the process description.
A	The server SHALL support process input values encoded as qualified values (qualifiedValue.yaml).
B	The value of the `value` key SHALL be an *object* instance of input Value.yaml.

REQUIREMENT 21	
/req/core/process-execute-input-mixed-type	
Conditions	a) The process input value is specified in-line in an execute request. b) The schema of the process input from the process description indicates that a value instance can be one of multiple input media types. In JSON Schema this is denoted by the use of the `oneOf` construct.
A	The server SHALL support process input values encoded as qualified values (qualifiedValue.yaml).
B	The value of the `value` key SHALL be an instance of inputValue. yaml.
C	The `format` parameter of the qualified value (qualifiedValue.yaml) SHALL be used to indicate, for this value instance, the specific input type selected from the list of type choices defined by the input value's schema from the process description.

Example 3 — Qualified value examples.: An example of a complex process input value.

```
"complexObjectInput": {
  "value": {
    "property1": "value1",
    "property2": "value2",
    "property3": "value3"
  }
}
```

In this second example, the property, `geometryInput` has a cardinality of greater than 1 and value instances can be one of a number of enumerated media types. The schema of `geometryInput` from the OGC process description for the process might be:

```
"geometryInput": {
  "title": "Geometry input",
  "description": "This is an example of a geometry input.  In this case the
 geometry can be expressed as a GML of GeoJSON geometry.",
  "minOccurs": 2,
  "maxOccurs": 5,
  "schema": {
    "oneOf": [
      {
```

```
        "type": "string",
        "contentMediaType": "application/gml+xml; version=3.2",
        "contentSchema": "http://schemas.opengis.net/gml/3.2.1/geometryBasic2d.
xsd"
      },
      {
        "$ref": "http://schemas.opengis.net/ogcapi/features/part1/1.0/openapi/
schemas/geometryGeoJSON.json"
      }
    ]
  }
},
```

and an instance of this process input in an execute request might be:

```
"geometryInputs": [
  {
    "value": "<gml:Polygon gml:id=\"GID1\" srsName=\"urn:ogc:def:crs:OGC::
CRS84\"><gml:exterior><gml:LinearRing><gml:posList>-77.024519 38.810529 -77.
024635 38.810973 -77.024704 38.810962 -77.024776 38.811239 -77.024957 38.81121
 -77.024905 38.811012 -77.024905 38.811012 -77.024865 38.810857 -77.025024 38.
810832 -77.025071 38.811012 -77.025203 38.810992 -77.02506 38.810444 -77.024519
 38.810529</gml:posList></gml:LinearRing></gml:exterior></gml:Polygon>",
    "mediaType": "application/gml+xml; version=3.2"
  },
  {
    "value": {
      "type": "Polygon",
      "coordinates": [[[ -176.5814819, -44.10896301 ],
                       [ -176.5818024, -44.10964584 ],
                       [ -176.5844116, -44.11236572 ],
                       [ -176.5935974, -44.11021805 ],
                       [ -176.5973511, -44.10743332 ],
                       [ -176.5950928, -44.10562134 ],
                       [ -176.5858459, -44.1043396  ],
                       [ -176.5811157, -44.10667801 ],
                       [ -176.5814819, -44.10896301 ]]]
    },
    "mediaType": "application/geo+json"
  }
]
```

In this case, the `mediaType` parameter is used to indicate the specific type of geometry being passed as input in each case; GML Polygon for the first element of the array and GeoJSON Polygon for the second element.

Binary values:

In some cases, for example in order to pass through firewalls, binary input values need to be encoded in-line in an execute request as a string.

```
type: string
format: byte
```

Figure 16 — Schema for an in-line binary value

NOTE 6: This schema can also be obtained from `binaryInputValue.yaml`.

REQUIREMENT 22
/req/core/process-execute-input-inline-binary

REQUIREMENT 22	
Conditions	a) The process input value is specified in-line in an execute request. b) The process input value is binary.
A	The service SHALL support binary values encoded as base64-encoded strings.

Example 4 — Binary value examples.: This is an example of an image process input whose media type is defined in the process description. The schema definition for this process input might be:

```
"schema": {
  "type": "string",
  "contentEncoding": "binary",
  "contentMediaType": "application/tiff; application=geotiff"
}
```

and an example instance value in an execute request might be:

```
"imageInput": "R0lGODdhNAHCAfcAAAcHDD
+Gs4sLDQpDaqGFdaHE54dJPEoECUlGRteKgcdITgokG4hoVkpY
\ngNzHwKKkqOLm7RRjlEgpHU9iZ44lHQYqVdmki6doVmhHOMOIeJG20HiDjCcKBglIeadISrso
\nJGooFNbN2d2qr8aljyklHwQJQkdvkWaKxIdrb442LidLeGhMTp6LkeP1+Kh3aiUuVAoUHmlu
\ngkcwNYdZRmkJDYGcsDFokElVYyk1NsWWhLEPDtmQldrUyoyFhrjo+Nna5d+4tMGstspoXgc4\n.
..qgu7sSu7qbtCs2u7t6u6rLsrp4u7veu76eO6vyu8w0u8xWu8x4u8yau8shu8y
+u8zwu90Su9\n00u91Wu914u92au928u9whsQADs="
```

NOTE 7: Even though the schema indicates that the input type is `binary`, when the input value

is encoded in-line in an execute request, as per requirement /req/core/process-execute-input-inline-binary, the binary value is encoded as base64-encoded string.
In this second example, the image input can be one of a number of value types denoted in JSON Schema by the use of the `oneOf[]` construct. An example schema for this a process input might be:

```
"schema": {
  "oneOf": [
    {
      "type": "string",
      "contentEncoding": "binary",
      "contentMediaType": "application/tiff; application=geotiff"
    },
    {
      "type": "string",
      "contentEncoding": "binary",
      "contentMediaType": "application/jp2"
    }
  ]
}
```

and a JPEG2000 instance example in an execute request might be:

```
{
  "value":
 "VBORw0KGgoAAAANSUhEUgAABvwAAAa4CAYAAABMB35kAAABhGlDQ1BJQ0MgcHJvZmlsZQAA
\nKJF9kT1Iw0AcxV9TpSL1A+xQxCFDdbIgKuKoVShChVArtOpgcumH0KQhSXFxFFwLDn4sVh1c
\nnHV1cBUEwQ8QNzcnRRcp8X9JoUWMB8f9eHfvcfcOEOplplkdY4Cm22Y6mRCzuRUx9IogouhH\n .
.. \nj3Z5mX7/PCPVRJV92rpHK24xcJrzk20+tkeYlCPqcZNO3Lpni1OJWatPCcmgGDEqx7Om6lfa
```

```
\nppM4k4BTe9+bsn3L9/9/yWhA0PwQGW8ipCZsnZt9lsdrYEM8z/M8z/M8z/M8z/M8z/
MzLWY1\nAAAACUlEQVQ871H6P6JI+TxS5Wn2AAAAAElFTkSuQmCC",
  "mediaType": "application/jp2"
}
```

Bounding box values:

A process input value instance can be a bounding box.

REQUIREMENT 23
/req/core/process-execute-input-inline-bbox
Servers SHALL support process input values that conform to the bbox.yaml schema.

```
type: object
required:
  - bbox
properties:
  bbox:
    type: array
    oneOf:
      - minItems: 4
        maxItems: 4
      - minItems: 6
        maxItems: 6
    items:
      type: number
  crs:
    type: string
    format: uri
    default: "http://www.opengis.net/def/crs/OGC/1.3/CRS84"
    enum:
      - "http://www.opengis.net/def/crs/OGC/1.3/CRS84"
      - "http://www.opengis.net/def/crs/OGC/0/CRS84h"
```

Figure 17 — Schema for a bounding box value

NOTE 8: This schema can also be obtained from `bbox.yaml`.

This schema is meant to be a template for defining bounding box process inputs. If the specified `default` and `enum` are suitable for your purposes then you can reference this file directly in your process description.

PERMISSION 6
/per/core/process-execute-input-inline-bbox
Servers MAY copy the contents of bbox.yaml into another file and adjust the `default` and `enum` values are required.

Input validation

Process inputs in an execute request and the corresponding process input definition in the process description have a validation relationship. That is to say, that the schema of a process

input definition from the process description can be used to validate the component of the corresponding process input value in an execute request that is an instance of inputValue.yaml.

Consider a process input named `complexObjectInput` with the following definition from an OGC process description:

```
"complexObjectInput": {
  "title": "Complex Object Input Example",
  "description": "This is an example of a complex object input.",
  "schema": {
    "type": "object",
    "required": [
      "property1",
      "property5"
    ],
    "properties": {
      "property1": {
        "type": "string"
      },
      "property2": {
        "type": "string",
        "format": "uri"
      },
      "property3": {
        "type": "number"
      },
      "property4": {
        "type": "string",
        "format": "dateTime"
      },
      "property5": {
        "type": "boolean"
      }
    }
  }
}
```

and the following instance in an execute request:

```
"inputs": [
  .
  .
  .
  "complexObjectInput": {
    "value": {
      "property1": "value1",
      "property2": "value2",
      "property3": "value5"
    }
  },
  .
  .
  .
]
```

The process input value in this execute request is an instance of a qualified value. For the purposes of validation, the server need only validate the component of the qualified value that is an instance of inputValue.yaml against the `schema` fragment from the OGC process description. Specifically, the validation target is:

```
{
  "property1": "value1",
  "property2": "value2",
```

```
  "property3": "value5"
}
```

NOTE 9: This example makes use of an OGC process description. However, any other process description vocabulary may be used and applied, for the purpose of validation, in a similar manner.

REQUIREMENT 24	
/req/core/process-execute-input-validation	
A	For process input values specified in-line in an execute request, the server SHALL validate each component of a process input value that is an instance of inputValue.yaml using the definition of the corresponding input from the process description.
B	For process input values specified by reference in an execute request, the server SHALL resolve the value and then validate it as if the value had been specified in-line in the execute request (i. e. as per requirement ***A***).

7.11.2.3. Execution mode

A process may be executed synchronously or asynchronously.

In which of these two modes a server responds is a function of the job control options specified in the process description and the presence or absence of the HTTP `Prefer` header (IETF RFC 7240).

REQUIREMENT 25	
/req/core/process-execute-default-execution-mode	
Conditions	The execute request **is not** accompanied with the HTTP `Prefer` header.
A	The server SHALL respond asynchronously if, according to the job control options in the process description, the process can only be executed asynchronously.
B	The server SHALL respond synchronously if, according to the job control options in the process description, the process can only be executed synchronously.
C	The server SHALL respond synchronously if, according to the job control options in the process description, the process can be executed in either mode.

REQUIREMENT 26

/req/core/process-execute-auto-execution-mode

Conditions	The execute request **is** accompanied with the HTTP `Prefer` header asserting a `respond-async` preference.
A	The server SHALL respond asynchronously if, according to the job control options in the process description, the process can only be executed asynchronously.
B	The server SHALL respond synchronously if, according to the job control options in the process description, the process can only be executed synchronously.
C	The server SHALL respond, at its discretion, either synchronously or asynchronously if, according to the job control options in the process description, the process can be executed in either mode.

RECOMMENDATION 12

/rec/core/process-execute-honor-prefer

A	If an execute request is accompanied with the HTTP `Prefer` header asserting a `respond-async` preference, then the server SHOULD honor that preference and response asynchronously if, according to the job control options in the process description, the process can be executed asynchronously.
B	If an execute request is accompanied with the HTTP `Prefer` header asserting a `wait` preference, then the server SHOULD honor that preference in the decision to execute the process asynchronously if, according to the job control options in the process description, the process can be executed asynchronously.

RECOMMENDATION 13

/rec/core/process-execute-handle-prefer

A client that accompanies an execute request with the HTTP `Prefer` header asserting a `respond-async` preference and/or a `wait` preference SHOULD be prepared to receive either an asynchronous or a synchronous response.

RECOMMENDATION 14

/rec/core/process-execute-preference-applied

If an execute request is accompanied with the HTTP `Prefer` header then, in the response, servers SHOULD include the HTTP `Preference-Applied` response header as an indication as to which ‘Prefer` tokens were honoured by the server.

7.11.2.4. Response type

The `response` parameter selects the form used to present processing results.

- A `response` value of `document` indicates that the server should generate a response document (see results.yaml) that contains each of the requested process outputs.
- A `response` value of `raw` indicates that the server should present the process responses in their native form without the use of a containing document.

The default value, if the parameter is not specified is `raw`.

7.11.2.5. Process outputs

Overview:

Each process output is a named object that appears in the `outputs` section of an execute request. The name of each output is its identifier as specified by the output's definition in the process description.

Default outputs

REQUIREMENT 27
/req/core/process-execute-default-outputs
If a process is defined as having one or more outputs and the `outputs` parameter is omitted in an execute request, this SHALL be equivalent to having requested all the defined outputs in the execute request.

Output value transmission mode

Like inputs, output values can be transmitted in-line in a result document or referenced using a link. The `transmissionMode` parameter in the execute request controls how the server responds.

```
type: string
enum:
  - value
  - reference
default:
  - value
```

Figure 18 — Schema for the transmission mode

NOTE: This schema can also be obtained from `transmissionMode.yaml`.

Output value format

A process output can be defined in the process description as being of one or more media types. In cases where a specific output can be presented in one of a number of media types, the `format` parameter in the execute request can be used to indicate the format that should be used to present the process output value in the server's response.

7.11.3. Example

Example — An execute request.:

```
{
  "inputs": {
    "stringInput": "Value2",
    "measureInput": {
      "value": {
        "measurement": 10.3,
        "uom": "m",
        "reference": "https://ucum.org/ucum-essence.xml"
      }
    },
    "dateInput": "2021-03-06T07:21:00",
    "doubleInput": 3.14159,
    "arrayInput": [1,2,3,4,5,6],
    "complexObjectInput": {
      "value": {
        "property1": "value1",
        "property2": "value2",
        "property5": true
      }
    },
    "geometryInput": [
      {
        "value": "<gml:Polygon gml:id=\"GID1\" srsName=\"urn:ogc:def:crs:OGC:
:CRS84\"><gml:exterior><gml:LinearRing><gml:posList>-77.024519 38.810529 -77.
024635 38.810973 -77.024704 38.810962 -77.024776 38.811239 -77.024957 38.81121
 -77.024905 38.811012 -77.024905 38.811012 -77.024865 38.810857 -77.025024 38.
810832 -77.025071 38.811012 -77.025203 38.810992 -77.02506 38.810444 -77.024519
 38.810529</gml:posList></gml:LinearRing></gml:exterior></gml:Polygon>",
        "mediaType": "application/gml+xml; version=3.2"
      },
      {
        "value": {
          "type": "Polygon",
          "coordinates": [[[ -176.5814819, -44.10896301 ],
                           [ -176.5818024, -44.10964584 ],
                           [ -176.5844116, -44.11236572 ],
                           [ -176.5935974, -44.11021805 ],
                           [ -176.5973511, -44.10743332 ],
                           [ -176.5950928, -44.10562134 ],
                           [ -176.5858459, -44.1043396  ],
                           [ -176.5811157, -44.10667801 ],
                           [ -176.5814819, -44.10896301 ]]]
        }
      }
    ],
    "boundingBoxInput": {
      "bbox": [ 51.9, 7, 52, 7.1 ],
      "crs": "http://www.opengis.net/def/crs/OGC/1.3/CRS84"
    },
    "imagesInput": [
      {
```

```
        "href": "https://www.someserver.com/ogcapi/Daraa/collections/Daraa_
DTED/styles/Topographic/coverage?...",
        "type": "application/tiff; application=geotiff"
      },
      {
        "value":
 "VBORw0KGgoAAAANSUhEUgAABvwAAAa4CAYAAABMB35kAAABhGlDQ1BJQ0MgcHJvZmlsZQAA
\nKJF9kT1Iw0AcxV9TpSL1A+xQxCFDdbIgKuKoVShChVArtOpgcumH0KQhSXFxFFwLDn4sVh1c
\nnHV1cBUEwQ8QNzcnRRcp8X9JoUWMB8f9eHfvcfcOEOplplkdY4Cm22Y6mRCzuRUx9IogouhH\n .
.. \nj3Z5mX7/PCPVRJV92rpHK24xcJrzk20+tkeYlCPqcZNO3Lpni1OJWatPCcmgGDEqx7Om6lfa
\nppM4k4BTe9+bsn3L9/9/yWhA0PwQGW8ipCZsnZt9lsdrYEM8z/M8z/M8z/M8z/
MzLWY1\nAAAACUlEQVQ871H6P6JI+TxS5Wn2AAAAAElFTkSuQmCC",
        "encoding": "base64",
        "mediaType": "application/jp2"
      }
    ],
    "featureCollectionInput": {
      "value": "<?xml version=\"1.0\" encoding=\"UTF-8\"?><FeatureCollection
 xmlns=\"http://schemas.myserver.com/namespaces/null\" xmlns:gml=\"http://www.
opengis.net/gml/3.2\" xmlns:xsi=\"http://www.w3.org/2001/XMLSchema-instance
\" xsi:schemaLocation=\"http://schemas.myserver.com/namespaces/null https:
//www.pvretano.com/myserver/ogcapi/daraa/schema?f=GML32&amp;collectionids=
TransportationGroundCrv http://www.opengis.net/gml/3.2 http://schemas.opengis.
net/schemas/gml/3.2.1/gml.xsd\">...",
      "mediaType": "application/gml+xml; version=3.2"
    }
  },
  "outputs": {
    "stringOutput": {
      "transmissionMode": "value"
    },
    "measureOutput": {
      "transmissionMode": "value"
    },
    "dateOutput": {
      "transmissionMode": "value"
    },
    "doubleOutput": {
      "transmissionMode": "value"
    },
    "arrayOutput": {
      "transmissionMode": "value"
    },
    "complexObjectOutput": {
      "transmissionMode": "value"
    },
    "geometryOutput": {
      "transmissionMode": "value"
    },
    "boundingBoxOutput": {
      "transmissionMode": "value"
    },
    "imageOutput": {
      "format": { "mediaType": "application/tiff; application=geotiff" },
      "transmissionMode": "value"
    },
    "featureCollectionOutput": {
      "transmissionMode": "value"
    }
  },
  "response": "document"
}
```

7.11.4. Response

The manner in which a server responds to a process execution request is determined by the following parameters:

- the negotiated execution mode (synchronous or asynchronous),
- the `response` parameter (`document` or `raw`),
- the `transmissionMode` parameter (`value` or `reference`)
- and the number of outputs requested.

The following table maps the possible responses based on the combinations of these execute parameters.

Table 11 — Table mapping execute responses based on the input parameter values.

NEGOTIAT EXECUTIO MODE	RESPON	TRANSMISSI MODE	# OU	HTT COI	MEDIA TYPE	BODY CONTENT	REQUIREMENT
sync	raw	value	1	200	[as per output definition from process description]	[output in requested format]	/req/core/process-execute-sync-raw-value-one
			*	200	multipart/related	[one output per part]	/req/core/process-execute-sync-raw-value-multi
		reference	1 *	204	none	[empty with `Link` headers]	/req/core/process-execute-sync-raw-ref
		mixed	*	200	multipart/related	[one output per part]	/req/core/process-execute-sync-raw-mixed-multi
	document	value	1 *	200	application/json	results.yaml	/req/core/process-execute-sync-document
		reference	1 *				

NEGOTIAT EXECUTIO MODE	RESPON	TRANSMISSI MODE	# OU	HTT COI	MEDIA TYPE	BODY CONTENT	REQUIREMENT
async	*any*	*any*	*any*	201	application/json	statusInfo.yaml	/req/core/process-execute-success-async

NOTE: The value *any* in a cell means *"for any valid value of the execute parameter represented by that cell"*. This table shows all possible combinations of execute parameters that are specified by this standard. Not all of these combinations need to be implemented by a server conforming to this standard. For example, if a server only offers processes that support multiple outputs **by value**, then the server **must** support `multipart/related` responses as indicated in Table 11. If, on the other hand, the server only offers processes that support multiple outputs **by reference**, then the server does not need to support `multipart/related` responses.

```
additionalProperties:
  $ref: "inlineOrRefData.yaml"
```

Figure 19 — Schema for a processing results presented as a document

NOTE 1: This schema can also be obtained from `results.yaml`.

This schema defines a map using the respective output identifier as the key. The value of an output can be returned as an in-line value or by reference.

For a synchronous execution, the following requirements apply:

REQUIREMENT 28	
/req/core/process-execute-sync-raw-value-one	
Conditions	a) The negotiated execution mode is synchronous, `response=raw`, `transmissionMode=value` b) The number of requested outputs is 1.
A	The server SHALL respond with an HTTP status code of 200.
B	The media type of the response SHALL be as specified by the output definition from the process description.
C	The content of response SHALL be the requested process output in the requested output format.

REQUIREMENT 29	
/req/core/process-execute-sync-raw-value-multi	
Conditions	a) The negotiated execution mode is synchronous, `response=raw`, `transmissionMode=value` b) The number of requested outputs is greater than 1.

REQUIREMENT 29	
A	The server SHALL respond with an HTTP status code of 200.
B	The media type of the response SHALL be `multipart/related`.
C	The content of response shall conform to RFC 2387, The MIME Multipart/Related Content-type.
D	Each requested output shall be encoded into one part of the response.
E	The `Content-ID` header for each part SHALL be the corresponding output identifier.

REQUIREMENT 30	
/req/core/process-execute-sync-raw-ref	
Conditions	a) The negotiated execution mode is synchronous, `response=raw`, `transmissionMode=reference` b) The number of requested outputs is 1 or more.
A	The server SHALL respond with an HTTP status code of 204.
B	The response SHALL include one or more `Link` headers that reference each requested output.

REQUIREMENT 31	
/req/core/process-execute-sync-raw-mixed-multi	
Conditions	a) The negotiated execution mode is synchronous, `response=raw`, b) Multiple transmission modes (`value`,`reference`) are requested. c) The number of requested outputs is more than 1.
A	The server SHALL respond with an HTTP status code of 200.
B	The media type of the response SHALL be `multipart/related`.
C	The content of response shall conform to RFC 2387, The MIME Multipart/Related Content-type.
D	Each requested output shall be encoded into one part of the response.
E	The `Content-ID` header for each part SHALL be the corresponding output identifier.

REQUIREMENT 31	
F	For outputs requested by reference, the `Content-Location` header SHALL be included pointing to the output and the body of the part SHALL be empty.
G	For outputs requested by value, the body of the response SHALL contain the output value.

REQUIREMENT 32	
/req/core/process-execute-sync-document	
Conditions	a) The negotiated execution mode is synchronous, `response=document`, `transmissionMode=value` or `transmissionMode=reference` b) The number of requested outputs is 1 or more.
A	The server SHALL respond with an HTTP status code of 200.
B	The media type of the response SHALL be `application/json`
C	The content of response SHALL conform to the results.yaml schema.

RECOMMENDATION 15	
/rec/core/process-execute-sync-document-ref	
Conditions	a) The negotiated execution mode is synchronous, `response=document`, `transmissionMode=reference` b) The number of outputs requested is 1 or more. c) The requested output value is a simple scalar value.
A	For a simple scalar values servers SHOULD consider supporting links to `text/plain` files that contain the output values.

This specification does not mandate that servers create a job as a result of executing a process synchronously. However the following permission is given:

PERMISSION 7
/per/core/process-execute-sync-job
Servers MAY support the creation of a job for synchronously executed processes.

For servers that implement this permission and do create a job as a result of synchronous execution of a process, the following requirement applies:

REQUIREMENT 33	
/req/core/job-results-success-sync	
Conditions	The server creates a job when a processes is executed synchronously.
A	A successful execution of the operation SHALL include an HTTP `Link` header with `rel=monitor` pointing to the created job.

The job reference in the header can then be used to re-fetch the results of the original synchronous execution.

In the case of asynchronous execution, the following requirement applies:

REQUIREMENT 34	
/req/core/process-execute-success-async	
Conditions	a) The negotiated execution mode is asynchronous.
A	A successful execution of the operation SHALL be reported as a response with a HTTP status code 201.
B	The header of the response SHALL return the HTTP `Location` header that contains a link to the newly created job.
C	The content of the response SHALL be based upon the JSON Schema fragment statusInfo.yaml.

7.11.5. Error situations

See Clause 7.5.1 for general guidance.

If the process with the specified identifier does not exist on the server, see requirement /req/core/process-exception/no-such-process.

7.12. Retrieve status information about a job

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

7.12.1. Operation

REQUIREMENT 35
/req/core/job
The server SHALL support the HTTP GET operation at the path `/jobs/{jobID}`.

7.12.2. Response

REQUIREMENT 36
/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.

```
type: object
required:
   - jobID
   - status
   - type
properties:
   processID:
      type: string
   type:
      type: string
      enum:
        - process
   jobID:
      type: string
   status:
      $ref: "statusCode.yaml"
   message:
      type: string
   created:
      type: string
      format: date-time
   started:
      type: string
      format: date-time
   finished:
      type: string
      format: date-time
   updated:
      type: string
      format: date-time
   progress:
      type: integer
      minimum: 0
      maximum: 100
   links:
```

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

Figure 20 — Schema for status info

NOTE 1: This schema can also be obtained from `statusInfo.yaml`.

```
type: string
nullable: false
enum:
  - accepted
  - running
  - successful
  - failed
  - dismissed
```

Figure 21 — Schema for status codes

NOTE 2: This schema can also be obtained from `statusCode.yaml`.

The job status information includes several optional date-time fields that represent milestones in the life cycle of a job. The following are recommended for servers that decide to populate some or all of these date-time fields:

RECOMMENDATION 16	
/rec/core/job-status	
A	Servers SHOULD set the value of the `processID` field if it is known.
B	Servers SHOULD set the value of the `created` field when a job has been accepted and queued for execution.
C	Servers SHOULD set the value of the `started` field when a job begins execution and is consuming compute resources.
D	Servers SHOULD set the value of the `finished` field when the execution of a job has completed and the process is no longer consuming compute resources.
E	Whenever the `status` field of the job changes, servers SHOULD revise the value of the `updated` field.

NOTE 3: Once a job has finished execution and is no longer consuming compute resources, the duration of processing can be computed as `finished-started`. The `updated` field, however, may still be revised as the system continues processing outputs, storing results, releasing compute resources, etc.

Example 1 — A HTTP GET request for retrieving status information about a job encoded as JSON.:

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

Example 2 — A job encoded as JSON.:

```
{
  "jobID" : "81574318-1eb1-4d7c-af61-4b3fbcf33c4f",
  "status": "accepted",
  "message": "Process started",
  "progress": 0,
  "created": "2021-05-04T10:13:00+05:00",
  "links": [
    {
      "href": "http://processing.example.org/oapi-p/jobs/81574318-1eb1-4d7c-
af61-4b3fbcf33c4f",
      "rel": "self",
      "type": "application/json",
      "title": "this document"
    }
  ]
}
```


7.12.3. Error situations

See Clause 7.5.1 for general guidance.

If the process with the specified identifier does not exist on the server, see requirement /req/core/process-exception/no-such-process.

REQUIREMENT 37
/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 `type` of the exception SHALL be "http://www.opengis.net/def/exceptions/ogcapi-processes-1/1.0/no-such-job".

7.13. Retrieve job results

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

7.13.1. Operation

REQUIREMENT 38
/req/core/job-results
The server SHALL support the HTTP GET operation at the path `/jobs/{jobID}/results`.

7.13.2. Response

The manner in which a server responds when retrieving job results depends on the values of the following parameters used in the execute request that created the job:

- the negotiated execution mode (synchronous or asynchronous),
- the `response` parameter (`document` or `raw`),
- the `transmissionMode` parameter (`value` or `reference`)
- and the number of output requested.

The following table maps the possible responses based on the combinations of these execute parameters.

Table 12 — Table mapping get results responses based on the input parameter values used on the original execute request.

NEGOTIA EXECUTE MODE	RESPOI	TRANSMISS MODE	# OU	REQUIF	HT CO	MEDIA TYPE	BODY CONTENT	REQUIREMENT
sync	*any*	*any*	*any*	no	[if the server creates a job on synchronous execute, job results may be re-fetched at any time as per async]			
async	raw	value	1		200	[as per output definition from process description]	[output in requested output format]	/req/core/process-execute-sync-raw-value-one
			*		200	multipart/related	[one output per part]	/req/core/process-execute-sync-raw-value-multi
		reference	1	yes	204	none	[empty with `Link` headers]	/req/core/process-execute-sync-raw-ref
			*					
		mixed	*		200	multipart/related	[one output per part]	/req/core/process-execute-sync-raw-mixed-multi
	document	value	1		200	application/json	results.yaml	/req/core/process-execute-sync-document
			*					

NEGOTIA EXECUTE MODE	RESPOI	TRANSMISS MODE	# OU	REQUIF	HT CO	MEDIA TYPE	BODY CONTENT	REQUIREMENT
		reference	1					

NOTE: The value *any* in a cell means *"for any valid value of the execute parameter represented by that cell"*.

The following requirements apply when retrieving the results of a job that was created by executing a process synchronously OR was created by execution a process synchronously on a server that creates a job even on synchronous execution:

REQUIREMENT 39
/req/core/job-results-async-raw-value-one
The server SHALL response as per requirement /req/core/process-execute-sync-raw-value-one.

REQUIREMENT 40
/req/core/job-results-async-raw-value-multi
The server SHALL respond as per requirement /req/core/process-execute-sync-raw-value-multi.

REQUIREMENT 41
/req/core/job-results-async-raw-mixed-multi
The server SHALL respond as per requirement /req/core/process-execute-sync-raw-mixed-multi.

REQUIREMENT 42
/req/core/job-results-async-raw-ref
The server SHALL response as per requirement /req/core/process-execute-sync-raw-ref.

REQUIREMENT 43
/req/core/job-results-async-document
The server SHALL response as per requirement /req/core/process-execute-sync-document.

Example 1 — A HTTP GET request for retrieving the result a job encoded as JSON.:

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

Example 2 — A result encoded as JSON.:


```
{
  "stringOutput": "Value2",
  "measureOutput": {
    "value": {
      "measurement": "10.3",
      "uom": "m",
      "reference": "https://ucum.org/ucum-essence.xml"
    }
  },
  "dateOutput": "2021-03-06T07:21:00",
  "doubleOutput": "3.14159",
  "arrayOutput": [1,2,3,4,5,6],
  "complexObjectOutput": {
    "value": {
      "property1": "value1",
      "property2": "value2",
      "property5": true
    }
  },
  "geometryOutput": [
    {
      "value": "<gml:Polygon gml:id=\"GID1\" srsName=\"urn:ogc:def:crs:OGC::
CRS84\"><gml:exterior><gml:LinearRing><gml:posList>-77.024519 38.810529 -77.
024635 38.810973 -77.024704 38.810962 -77.024776 38.811239 -77.024957 38.81121
 -77.024905 38.811012 -77.024905 38.811012 -77.024865 38.810857 -77.025024 38.
810832 -77.025071 38.811012 -77.025203 38.810992 -77.02506 38.810444 -77.024519
 38.810529</gml:posList></gml:LinearRing></gml:exterior></gml:Polygon>",
      "mediaType": "application/gml+xml; version=3.2"
    },
    {
      "value": {
        "type": "Polygon",
        "coordinates": [[[ -176.5814819,-44.10896301 ],
                         [ -176.5818024,-44.10964584 ],
                         [ -176.5844116,-44.11236572 ],
                         [ -176.5935974,-44.11021805 ],
                         [ -176.5973511,-44.10743332 ],
                         [ -176.5950928,-44.10562134 ],
                         [ -176.5858459,-44.1043396  ],
                         [ -176.5811157,-44.10667801 ],
                         [ -176.5814819,-44.10896301 ]]]
      }
    }
  ],
  "boundingBoxOutput": {
    "bbox": [51.9,7,52,7.1],
    "crs": "http://www.opengis.net/def/crs/OGC/1.3/CRS84"
  },
  "imagesOutput": [
    {
      "href": "https://www.someserver.com/ogcapi/Daraa/collections/Daraa_DTED/
styles/Topographic/coverage?...",
      "type": "application/tiff; application=geotiff"
    },
    {
      "value":
 "VBORw0KGgoAAAANSUhEUgAABvwAAAa4CAYAAABMB35kAAABhGlDQ1BJQ0MgcHJvZmlsZQAA
\nKJF9kT1Iw0AcxV9TpSL1A+xQxCFDdbIgKuKoVShChVArtOpgcumH0KQhSXFxFFwLDn4sVh1c
\nnHV1cBUEwQ8QNzcnRRcp8X9JoUWMB8f9eHfvcfcOEOplplkdY4Cm22Y6mRCzuRUx9IogouhH\n .
.. \nj3Z5mX7/PCPVRJV92rpHK24xcJrzk20+tkeYlCPqcZNO3Lpni1OJWatPCcmgGDEqx7Om6lfa
```

```
\nppM4k4BTe9+bsn3L9/9/yWhA0PwQGW8ipCZsnZt9lsdrYEM8z/M8z/M8z/M8z/M8z/
MzLWY1\nAAAACUlEQVQ871H6P6JI+TxS5Wn2AAAAAElFTkSuQmCC",
      "encoding": "base64",
      "mediaType": "application/tiff; application=geotiff"
    }
  ],
  "featureCollectionOutput": {
    "value": "<?xml version=\"1.0\" encoding=\"UTF-8\"?><FeatureCollection
 xmlns=\"http://schemas.myserver.com/namespaces/null\" xmlns:gml=\"http://www.
opengis.net/gml/3.2\" xmlns:xsi=\"http://www.w3.org/2001/XMLSchema-instance
\" xsi:schemaLocation=\"http://schemas.myserver.com/namespaces/null https:
//www.pvretano.com/myserver/ogcapi/daraa/schema?f=GML32&amp;collectionids=
TransportationGroundCrv http://www.opengis.net/gml/3.2 http://schemas.opengis.
net/schemas/gml/3.2.1/gml.xsd\">...",
    "mediaType": "application/gml+xml; version=3.2"
  }
}
```


7.13.3. Error situations

See Clause 7.5.1 for general guidance.

REQUIREMENT 44

/req/core/job-results-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 `type` of the exception SHALL be "http://www.opengis.net/def/exceptions/ogcapi-processes-1/1.0/no-such-job".

REQUIREMENT 45

/req/core/job-results-exception/results-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 `type` of the exception SHALL be "http://www.opengis.net/def/exceptions/ogcapi-processes-1/1.0/result-not-ready".

REQUIREMENT 46

/req/core/job-results-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 `type` of the exception SHALL correspond to the reason of the failure, e.g. InvalidParameterValue for invalid input data.

8

REQUIREMENTS CLASS "OGC PROCESS DESCRIPTION"

8 REQUIREMENTS CLASS "OGC PROCESS DESCRIPTION"

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

8.1. Overview

REQUIREMENTS CLASS 2	
http://www.opengis.net/spec/ogcapi-processes-1/1.0/req/ogc-process-description	
Obligation	requirement
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 but also includes elements of the OpenAPI Specification. Specifically, this process description languages uses JSON Schema fragments to define the input and output parameters of a process. As such, this process description provides a bridge from legacy implementations to 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 specified using a JSON Schema fragment.
- A description of each process output specified using a JSON Schema fragment.

- 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

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

8.2. OGC process description

REQUIREMENT 47
/req/ogc-process-description/json-encoding
A JSON-encoded OGC process description SHALL validate against the JSON Schema: process.yaml.

```
allOf:
  - $ref: "processSummary.yaml"
  - type: object
    properties:
      inputs:
        additionalProperties:
          $ref: "inputDescription.yaml"
      outputs:
        additionalProperties:
          $ref: "outputDescription.yaml"
```

Figure 22 — Schema for a process

NOTE 1: This schema can also be obtained from `process.yaml`

(see also: `processSummary.yaml`)

The schema imports the elements from the process summary and specifies an object for the definition of process inputs and another object for the definition of process outputs.

REQUIREMENT 48	
/req/ogc-process-description/inputs-def	
A	Each process input definition SHALL be listed in the `inputs` section according to the JSON Schema: inputDescription.yaml.
B	The key of each process input in the `inputs` section of the process definition SHALL be the identifier for that input.

```
allOf:
  - $ref: "descriptionType.yaml"
  - type: object
    required:
      - schema
    properties:
      minOccurs:
        type: integer
        default: 1
      maxOccurs:
        oneOf:
          - type: integer
            default: 1
          - type: string
            enum:
              - "unbounded"
      schema:
        $ref: "schema.yaml"
```

Figure 23 — Schema for a process input

NOTE 2: This schema can also be obtained from `inputDescription.yaml`

(see also: `descriptionType.yaml`).

REQUIREMENT 49	
/req/ogc-process-description/input-def	
A	The schema of each process **input** value SHALL be specified using the `schema` parameter.
B	The value of the `schema` parameter SHALL be a JSON fragment that validates according to the JSON Schema: schema.yaml.
C	Servers SHALL use this schema fragment to validate the components of a process input in an execute request that is an instance of inputValue.yaml.

NOTE 3: The schema fragment specified as the value of the `schema` parameter can be used to validate the corresponding process input value in an execute request.

REQUIREMENT 50
/req/ogc-process-description/input-binary

A server SHALL support the following schema for binary include values:
`type: string`
`format: byte`

REQUIREMENT 51
/req/ogc-process-description/input-mixed-type

REQUIREMENT 51	
A	An input that can be of mixed type SHALL be defined using the `oneOf` JSON Schema keyword.
B	Each sub-schema SHALL be a JSON fragment that validates according to the JSON Schema: schema.yaml.
C	The first sub-schema in the `oneOf` array SHALL be considered the default format.

The following JSON Schema fragment illustrates how to define an input of mixed type. In this case, the `imageInput` input can be one of a couple of image media types.

```
"imageInput": {
  "schema": {
    "oneOf": [
      {
        "type": "string",
        "contentEncoding": "binary",
        "contentMediaType": "application/tiff; application=geotiff"
      },
      {
        "type": "string",
        "contentEncoding": "binary",
        "contentMediaType": "application/jp2"
      }
    ]
  }
}
```

Figure 24 — Mixed type input example

RECOMMENDATION 17	
/rec/ogc-process-description/format-key	
A	Servers SHOULD use the `format` key in the schema description of a process input or output (key: `schema`) to provide additional semantic context that can aid in the interpretation and validation of process input or output values in an execute request.

The JSON Schema specification defines a set of values for the `format` key. This specification extends this list by defining the following additional key values for use specifically in OGC process descriptions.

Table 13 — Additional values for the JSON schema format key for OGC Process Description

KEY VALUE	SHORT CODE	DESCRIPTION
http://www.opengis.net/def/format/ogcapi-processes/0/geojson-feature-collection	geojson-feature-collection	Indicates that the object is an instance of a GeoJSON feature collection (featureCollectionGeoJSON.yaml).

KEY VALUE	SHORT CODE	DESCRIPTION
http://www.opengis.net/def/format/ogcapi-processes/0/geojson-feature	geojson-feature	Indicates that the object is an instance of a GeoJSON feature (featureGeoJSON.yaml).
http://www.opengis.net/def/format/ogcapi-processes/0/geojson-geometry	geojson-geometry	Indicates that the object is an instance of a GeoJSON geometry (geometryGeoJSON.yaml).
http://www.opengis.net/def/format/ogcapi-processes/0/ogc-bbox	ogc-bbox	Indicates that the object is an instance of an OGC bounding box (bbox.yaml).

NOTE: This list of values has been submitted to the OGC Naming Authority for registration in their definition server.

RECOMMENDATION 18

/rec/ogc-process-description/format-short-code

In addition to the key values listed in Table 13, servers SHOULD also accept the short codes.

Situations might arise where communities of interest wish to extend this list of values for their own purposes.

RECOMMENDATION 19

/rec/ogc-process-description/format-value-registration

Servers wishing to extend this list of `format` key values, SHOULD officially register such values with the OGC Naming Authority.

The following JSON Schema fragment illustrates the use of the `format` key to include a semantic hint to a process input that is a geometry.

```
"geometryInput": {
  "title": "Geometry input",
  "description": "This is an example of a geometry input.  In this case the
 geometry can be expressed as a GML of GeoJSON geometry.",
  "minOccurs": 2,
  "maxOccurs": 5,
  "schema": {
    "oneOf": [
      {
        "type": "string",
        "contentMediaType": "application/gml+xml; version=3.2",
        "contentSchema": "http://schemas.opengis.net/gml/3.2.1/geometryBasic2d.
xsd"
      },
      {
        "allOf": [
          {
            "format": "geojson-geometry"
```

```
          },
          {
            "$ref": "http://schemas.opengis.net/ogcapi/features/part1/1.0/
openapi/schemas/geometryGeoJSON.yaml"
          }
        ]
      }
    ]
  }
}
```

Figure 25 — Example of semantic hints using the format key

```
allOf:
  - $ref: "descriptionType.yaml"
  - type: object
    required:
      - schema
    properties:
      schema:
        $ref: "schema.yaml"
```

Figure 26 — Schema for a process output

NOTE 4: This schema can also be obtained from `outputDescription.yaml`

(see also: `descriptionType.yaml`).

REQUIREMENT 52	
/req/ogc-process-description/outputs-def	
A	Each process output definition SHALL be listed in the `outputs` section according to the JSON Schema: outputDescription.yaml.
B	The key of each process input in the `output` section of the process definition SHALL be the identifier for that output.

REQUIREMENT 53	
/req/ogc-process-description/output-def	
A	The schema of each process **output** SHALL be specified using the `schema` parameter.
B	The value of the `schema` parameter SHALL be a JSON fragment that validates according to the JSON Schema: schema.yaml.

REQUIREMENT 54	
/req/ogc-process-description/output-mixed-type	
A	An output that can be of mixed type SHALL be defined using the `oneOf` JSON Schema keyword.

REQUIREMENT 54	
B	Each sub-schema SHALL be a JSON fragment that validates according to the JSON Schema: schema.yaml.
C	The first sub-schema in the `oneOf` array SHALL be considered the default format.

Example — Example OGC Process Description.: The following URL is an example of retrieving a process description from the `/processes/{processID}` endpoint.

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

The description of the example `EchoProcess` process might be:


```
{
  "id": "EchoProcess",
  "title": "Echo Process",
  "description": "This process accepts and number of input and simple echoes
 each input as an output.",
  "version": "1.0.0",
  "jobControlOptions": [
    "async-execute",
    "sync-execute"
  ],
  "outputTransmission": [
    "value",
    "reference"
  ],
  "inputs": {
    "stringInput": {
      "title": "String Literal Input Example",
      "description": "This is an example of a STRING literal input.",
      "schema": {
        "type": "string",
        "enum": [
          "Value1",
          "Value2",
          "Value3"
        ]
      }
    },
    "measureInput": {
      "title": "Numerical Value with UOM Example",
      "description": "This is an example of a NUMERIC literal with an
 associated unit of measure.",
      "schema": {
        "type": "object",
        "required": [
          "value",
          "uom"
        ],
        "properties": {
          "measurement": {
            "type": "number"
          },
          "uom": {
            "type": "string"
          },
          "reference": {
            "type": "string",
            "format": "uri"
```

```
          }
        }
      }
    },
    "dateInput": {
      "title": "Date Literal Input Example",
      "description": "This is an example of a DATE literal input.",
      "schema": {
        "type": "string",
        "format": "dateTime"
      }
    },
    "doubleInput": {
      "title": "Bounded Double Literal Input Example",
      "description": "This is an example of a DOUBLE literal input that is
bounded between a value greater than 0 and 10.  The default value is 5.",
      "schema": {
        "type": "number",
        "format": "double",
        "minimum": 0,
        "maximum": 10,
        "default": 5,
        "exclusiveMinimum": true
      }
    },
    "arrayInput": {
      "title": "Array Input Example",
      "description": "This is an example of a single process input that is
an array of values.  In this case, the input array would be interpreted as a
single value and not as individual inputs.",
      "schema": {
        "type": "array",
        "minItems": 2,
        "maxItems": 10,
        "items": {
          "type": "integer"
        }
      }
    },
    "complexObjectInput": {
      "title": "Complex Object Input Example",
      "description": "This is an example of a complex object input.",
      "schema": {
        "type": "object",
        "required": [
          "property1",
          "property5"
        ],
        "properties": {
          "property1": {
            "type": "string"
          },
          "property2": {
            "type": "string",
            "format": "uri"
          },
          "property3": {
            "type": "number"
          },
          "property4": {
            "type": "string",
            "format": "dateTime"
          },
```

```
          "property5": {
            "type": "boolean"
          }
        }
      }
    },
    "geometryInput": {
      "title": "Geometry input",
      "description": "This is an example of a geometry input.  In this case the
 geometry can be expressed as a GML of GeoJSON geometry.",
      "minOccurs": 2,
      "maxOccurs": 5,
      "schema": {
        "oneOf": [
          {
            "type": "string",
            "contentMediaType": "application/gml+xml; version=3.2",
            "contentSchema": "http://schemas.opengis.net/gml/3.2.1/
geometryBasic2d.xsd"
          },
          {
            "allOf": [
              {
                "format": "geojson-geometry"
              },
              {
                "$ref": "http://schemas.opengis.net/ogcapi/features/part1/1.0/
openapi/schemas/geometryGeoJSON.yaml"
              }
            ]
          }
        ]
      }
    },
    "boundingBoxInput": {
      "title": "Bounding Box Input Example",
      "description": "This is an example of a BBOX literal input.",
      "schema": {
        "allOf": [
          {
            "format": "ogc-bbox"
          },
          {
            "$ref": "../../openapi/schemas/bbox.yaml"
          }
        ]
      }
    },
    "imagesInput": {
      "title": "Inline Images Value Input",
      "description": "This is an example of an image input.  In this case,
 the input is an array of up to 150 images that might, for example, be a set
 of tiles.  The oneOf[] conditional is used to indicate the acceptable image
 content types; GeoTIFF and JPEG 2000 in this case.  Each input image in the
 input array can be included inline in the execute request as a base64-encoded
 string or referenced using the link.yaml schema.  The use of a base64-encoded
 string is implied by the specification and does not need to be specified in
 the definition of the input.",
      "minOccurs": 1,
      "maxOccurs": 150,
      "schema": {
        "oneOf": [
          {
```

```
            "type": "string",
            "contentEncoding": "binary",
            "contentMediaType": "application/tiff; application=geotiff"
          },
          {
            "type": "string",
            "contentEncoding": "binary",
            "contentMediaType": "application/jp2"
          }
        ]
      }
    },
    "featureCollectionInput": {
      "title": "Feature Collection Input Example.",
      "description": "This is an example of an input that is a feature
 collection that can be encoded in one of three ways: as a GeoJSON feature
 collection, as a GML feature collection retrieved from a WFS or as a KML
 document.",
      "schema": {
        "oneOf": [
          {
            "type": "string",
            "contentMediaType": "application/gml+xml; version=3.2"
          },
          {
            "type": "string",
            "contentSchema": "https://schemas.opengis.net/kml/2.3/ogckml23.
xsd",
            "contentMediaType": "application/vnd.google-earth.kml+xml"
          },
          {
            "allOf": [
              {
                "format": "geojson-feature-collection"
              },
              {
                "$ref": "https://geojson.org/schema/FeatureCollection.json"
              }
            ]
          }
        ]
      }
    }
  },
  "outputs": {
    "stringOutput": {
      "schema": {
        "type": "string",
        "enum": [
          "Value1",
          "Value2",
          "Value3"
        ]
      }
    },
    "measureOutput": {
      "schema": {
        "type": "object",
        "required": [
          "value",
          "uom"
        ],
        "properties": {
```

```
      "measurement": {
        "type": "number"
      },
      "uom": {
        "type": "string"
      },
      "reference": {
        "type": "string",
        "format": "uri"
      }
    }
  }
},
"dateOutput": {
  "schema": {
    "type": "string",
    "format": "dateTime"
  }
},
"doubleOutput": {
  "schema": {
    "type": "number",
    "format": "double",
    "minimum": 0,
    "maximum": 10,
    "default": 5,
    "exclusiveMinimum": true
  }
},
"arrayOutput": {
  "schema": {
    "type": "array",
    "minItems": 2,
    "maxItems": 10,
    "items": {
      "type": "integer"
    }
  }
},
"complexObjectOutput": {
  "schema": {
    "type": "object",
    "required": [
      "property1",
      "property5"
    ],
    "properties": {
      "property1": {
        "type": "string"
      },
      "property2": {
        "type": "string",
        "format": "uri"
      },
      "property3": {
        "type": "number"
      },
      "property4": {
        "type": "string",
        "format": "dateTime"
      },
      "property5": {
        "type": "boolean"
```

```
          }
        }
      }
    },
    "geometryOutput": {
      "schema": {
        "oneOf": [
          {
            "type": "string",
            "contentMediaType": "application/gml+xml",
            "contentSchema": "http://schemas.opengis.net/gml/3.2.1/
geometryBasic2d.xsd"
          },
          {
            "allOf": [
              {
                "format": "geojson-geometry"
              },
              {
                "$ref": "http://schemas.opengis.net/ogcapi/features/part1/1.0/
openapi/schemas/geometryGeoJSON.yaml"
              }
            ]
          }
        ]
      }
    },
    "boundingBoxOutput": {
      "schema": {
        "allOf": [
           {
             "format": "ogc-bbox"
           },
           {
             "$ref": "../../openapi/schemas/bbox.yaml"
           }
        ]
      }
    },
    "imagesOutput": {
      "schema": {
        "oneOf": [
          {
            "type": "string",
            "contentEncoding": "binary",
            "contentMediaType": "application/tiff; application=geotiff"
          },
          {
            "type": "string",
            "contentEncoding": "binary",
            "contentMediaType": "application/jp2"
          }
        ]
      }
    },
    "featureCollectionOutput": {
      "schema": {
        "oneOf": [
          {
            "type": "string",
            "contentMediaType": "application/gml+xml; version=3.2"
          },
          {
```

```
            "type": "string",
            "contentMediaType": "application/vnd.google-earth.kml+xml",
            "contentSchema": "https://schemas.opengis.net/kml/2.3/ogckml23.xsd"
          },
          {
            "allOf": [
              {
                "format": "geojson-feature-collection"
              },
              {
                "$ref": "https://geojson.org/schema/FeatureCollection.json"
              }
            ]
          }
        ]
      }
    }
  },
  "links": [
    {
      "href": "https://processing.example.org/oapi-p/processes/EchoProcess/
execution",
      "rel": "http://www.opengis.net/def/rel/ogc/1.0/execute",
      "title": "Execute endpoint"
    }
  ]
}
```


The `EchoProcess` process simply echoes each process input value it is given.

9

REQUIREMENTS CLASSES FOR ENCODINGS

9 REQUIREMENTS CLASSES FOR ENCODINGS

9.1. 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.

Note that any server that supports multiple encodings will have to support a mechanism to mint encoding-specific URIs for resources in order to express links, for example, to alternate representations of the same resource. This document does not mandate any particular approach how this is supported by the server.

As clients simply need to dereference the URI of the link, the implementation details and the mechanism how the encoding is included in the URI of the link are not important. Developers interested in the approach of a particular implementation, for example, to manipulate ("hack") URIs in the browser address bar, can study the API definition.

NOTE: Two common approaches are:

- an additional path for each encoding of each resource (this can be expressed, for example, using format specific suffixes like `.html`);
- an additional query parameter (for example, "accept" or "f") that overrides the Accept header of the HTTP request.

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

9.2. Requirement Class "JSON"

This section defines the requirements class JSON.

REQUIREMENTS CLASS 3
http://www.opengis.net/spec/ogcapi-processes-1/1.0/req/json

REQUIREMENTS CLASS 3	
Obligation	requirement
Target type	Web API
Dependency	OGC API — Processes Core
Dependency	JSON

REQUIREMENT 55
/req/json/definition
`200`-responses of the server SHALL support the following media type: • `application/json` for the following API endpoints: • `/` • `/conformance` • `/processes` • `/processes/{processID}` • `/jobs/{jobID}` and for the following API endpoint: • `/processes/{processID}/execution` when the `response` parameter is set to the value `document` and/or the negotiated execution mode is asynchronous.

9.3. Requirement Class “HTML”

This section defines the requirements class HTML.

REQUIREMENTS CLASS 4	
http://www.opengis.net/spec/ogcapi-processes-1/1.0/req/html	
Obligation	requirement
Target type	Web API

REQUIREMENTS CLASS 4	
Dependency	OGC API — Processes Core
Dependency	API — Common HTML
Dependency	W3C HTML 5

REQUIREMENT 56
/req/html/definition
Every `200`-response of an operation of the server SHALL support the media type `text/html`.

REQUIREMENT 57
/req/html/content
Every `200`-response of the server with the media type “text/html” SHALL be a W3C 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/>`.

10

REQUIREMENTS CLASS "OPENAPI 3.0"

10 REQUIREMENTS CLASS "OPENAPI 3.0"

10.1. Basic requirements

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

REQUIREMENTS CLASS 5	
http://www.opengis.net/spec/ogcapi-processes-1/1.0/req/oas30	
Obligation	requirement
Target type	Web API
Dependency	OGC API — Processes 1.0 Core
Dependency	API — Common OpenAPI 3.0
Dependency	OpenAPI Specification 3.0.1

REQUIREMENT 58	
/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 59
/req/oas30/oas-definition-2
The JSON representation SHALL conform to the OpenAPI Specification, version 3.0.

REQUIREMENT 60
/req/oas30/oas-impl
The server SHALL implement all capabilities specified in the OpenAPI definition.

10.2. Complete definition

REQUIREMENT 61

/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 Clause 7.5.1.

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.

10.3. Exceptions

REQUIREMENT 62

/req/oas30/exceptions-codes

For error situations that originate from the server, the API definition SHALL cover all applicable HTTP Status Codes.

Example — An exception response object definition:

```
description: An error occurred.
content:
  application/json:
    schema:
      $ref: https://beta.schemas.opengis.net/ogcapi/common/part1/0.1/core/
openapi/schemas/exception.yaml
  text/html:
    schema:
      type: string
```

10.4. Security

REQUIREMENT 63
/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.

11

REQUIREMENTS CLASS "JOB LIST"

11 REQUIREMENTS CLASS "JOB LIST"

11.1. Overview

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

REQUIREMENTS CLASS 6	
http://www.opengis.net/spec/ogcapi-processes-1/1.0/req/job-list	
Obligation	requirement
Target type	Web API
Dependency	OGC API — Processes Core

11.2. Operation

11.2.1. Job list

REQUIREMENT 64
/req/job-list/job-list-op
The server SHALL support the HTTP GET operation at the path `/jobs`.

RECOMMENDATION 20
/rec/job-list/job-list-landing-page
A link to the following resource SHOULD be added to the API landing page: /jobs (relation type 'http://www.opengis.net/def/rel/ogc/1.0/job-list')

11.2.2. Parameter `type`

REQUIREMENT 65	
/req/job-list/type-definition	

A	The operation SHALL support a parameter `type` with the following characteristics (using an OpenAPI Specification 3.0 fragment):
	`name: type`
	`in: query`
	`required: false`
	`schema:`
	`  type: array`
	`  items:`
	`  type: string`

REQUIREMENT 66	
/req/job-list/type-response	
A	If the parameter is provided and its value is `process` then only jobs created by an OGC processes API SHALL be included in the response.
B	If the parameter is omitted, then all jobs SHALL be included in the response.

11.2.3. Parameter `processID`

REQUIREMENT 67	
/req/job-list/processID-mandatory	
A	If the server supports this conformance class, the optional `processID` property in the statusInfo.yaml schema SHALL be mandatory.

REQUIREMENT 68	
/req/job-list/processID-definition	

A	The operation SHALL support a parameter `processID` with the following characteristics (using an OpenAPI Specification 3.0 fragment):
	`name: processID`
	`in: query`
	`required: false`

REQUIREMENT 68	

	schema:
	  type: array
	  items:
	  type: string

REQUIREMENT 69
/req/job-list/processid-response
If the parameter is specified with the operation, only jobs that have a value for the `process ID` property (see: statusInfo.yaml) that matches one of the values specified for the `processID` parameter SHALL be included in the response.

11.2.4. Parameter `status`

REQUIREMENT 70	
/req/job-list/status-definition	

A	The operation SHALL support a parameter `status` with the following characteristics (using an OpenAPI Specification 3.0 fragment):
	`name: status`
	`in: query`
	`required: false`
	`schema:`
	`  type: array`
	`  items:`
	`  type: string`

REQUIREMENT 71
/req/job-list/status-response
If the parameter is specified with the operation, only jobs that have a value for the `status` property (see: statusInfo.yaml) that matches one of the specified values of the `status` parameter SHALL be included in the response.

11.2.5. Parameter `datetime`

REQUIREMENT 72
/req/job-list/datetime-definition

REQUIREMENT 72	

A	The operation SHALL support a parameter `datetime` with the following characteristics (using an OpenAPI Specification 3.0 fragment):
	`name: datetime`
	`in: query`
	`required: false`
	`schema:`
	`  type: string`

B	The value of the `datetime` parameter is either a date-time value or a time interval. The parameter value SHALL conform to the following syntax (using ABNF):
	`interval-closed = date-time "/" date-time`
	`interval-open-start = [".."] "/" date-time`
	`interval-open-end = date-time "/" [".."]`
	`interval = interval-closed / interval-open-start / interval-open-end`
	`datetime = date-time / interval`

C	The syntax of `date-time` is specified by RFC 3339, 5.6.
D	Open ranges in time intervals at the start or end are supported using a double-dot (`..`) or an empty string for the start/end.

REQUIREMENT 73
/req/job-list/datetime-response
If the parameter is specified with the operation, only jobs that have a value for the `created` property (see: statusInfo.yaml that intersects the temporal information in the `datetime` parameter SHALL be included in the response.

11.2.6. Parameter `minDuration`, `maxDuration`

REQUIREMENT 74	
/req/job-list/duration-definition	

A	The operation SHALL support a parameter `minDuration` with the following characteristics (using an OpenAPI Specification 3.0 fragment):
	`name: minDuration`
	`in: query`
	`required: false`
	`schema:`
	`  type: array`
	`  items:`
	`  type: integer`

REQUIREMENT 74	

B	The operation SHALL support a parameter `maxDuration` with the following characteristics (using an OpenAPI Specification 3.0 fragment):
	`name: maxDuration`
	`in: query`
	`required: false`
	`schema:`
	`  type: array`
	`  items:`
	`  type: integer`
	`style: form`
	`explode: false`

REQUIREMENT 75	
/req/job-list/status-response	
Conditions	a) If the `status` parameter is not specified then only jobs that are running (status: `running`) or have completed execution (`successful`, `failed` or `dismissed`) SHALL be considered for inclusion in the response. b) If the `status` parameter is specified, then only jobs with the specified status SHALL be considered for inclusion in the response.
A	If only the `minDuration` parameter is specified with the operation, only jobs with the appropriate status and a duration of at least the specified `minDuration` value SHALL be included in the response.
B	If only the `maxDuration` parameter is specified with the operation, only jobs with the appropriate status and a duration of no longer than the specified `maxDuration` value SHALL be included in the response.
C	If both the `minDuration` and `maxDuration` parameters are specified with the operation, only jobs with the appropriate status and a duration of at least the specified `minDuration` value and no longer than the specified `maxDuration` value SHALL be included in the response.
D	The value of the `minDuration` and `maxDuration` parameters SHALL be number of seconds.
E	For running jobs, the duration SHALL be computed at runtime as the time the operation was invoked minus the value of the `started` parameter (see: statusInfo.yaml).
F	For completed jobs, the duration SHALL be computed as the value of the `finished` parameter minus the value of the `started` parameter (see: statusInfo.yaml).

REQUIREMENT 75	
G	Jobs for which runtime statistics are not included in the status information or are incomplete for computing the duration of the job SHALL be omitted from the response.

11.2.7. Parameter `limit`

REQUIREMENT 76	
/req/job-list/limit-definition	

A	The operation SHALL support a parameter `limit` with the following characteristics (using an OpenAPI Specification 3.0 fragment):
	`name: limit`
	`in: query`
	`required: false`
	`schema:`
	`  type: integer`
	`  minimum: 1`
	`  maximum: 10000`
	`  default: 10`

PERMISSION 8	
/per/job-list/limit-default-minimum-maximum	
A	The values for `minimum`, `maximum` and `default` in requirement `/req/job-list/limit-definition` are only examples and MAY be changed.

REQUIREMENT 77	
/req/job-list/limit-response	
A	The response SHALL not contain more jobs than specified by the optional `limit` parameter.
B	If the API definition specifies a maximum value for `limit` parameter, the response SHALL not contain more jobs than this maximum value.

PERMISSION 9	
/per/job-list/limit-response	
A	The server MAY return fewer jobs than requested (but not more).

11.3. Response

REQUIREMENT 78
/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.

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

Figure 27 — Schema for the job list

NOTE: This schema can also be obtained from `jobList.yaml`.

(see also: statusInfo.yaml, link.yaml)

The schema defines an array of status info elements and includes a links section for navigation links within the API.

The number of jobs returned depends on the server and the parameter `limit`.

See the discussion about the `limit` parameter in the Limit parameter section.

REQUIREMENT 79	
/req/job-list/links	
A	A `200`-response SHALL include the following links: • a link to this response document (relation: `self`), • a link to the response document in every other media type supported by the service (relation: `alternate`).

See the discussion about the `next` links in the Limit parameter section.

RECOMMENDATION 21	
/rec/job-list/next-1	
A	A `200`-response SHOULD include a link to the next page (relation: `next`) of jobs, if more jobs have been selected than returned in the response.

RECOMMENDATION 22	
/rec/job-list/next-2	
A	Dereferencing a next page link (relation: `next`) SHOULD return additional jobs from the set of selected jobs that have not yet been returned.

RECOMMENDATION 23	
/rec/job-list/next-3	
A	The number of jobs in a response to dereferencing a next page link (relation: `next`) SHOULD follow the same rules as for the response to the original query and again include a next page link (relation: `next`), if there are more jobs in the selection that have not yet been returned.

See the discussion about the `prev` link in the Limit parameter section.

PERMISSION 10	
/per/job-list/prev	
A	A response to dereferencing a next page link (relation: `next`) MAY include a previous page link (relation: `prev`) to the resource that included the next page link (relation: `next`).

Example 1 — A HTTP GET request for retrieving a list of jobs encoded as JSON.:

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

Example 2 — A job list encoded as JSON.:

```
{
  "jobs": [
    {
      "processID": "Voronoi",
      "jobID": "8ca109b4-3b86-4a9c-a284-a6d50f91019e",
      "status": "running",
      "message": "Perform step 1/2",
      "progress": 50,
      "links": [
        {
```

```
          "href": "http://processing.example.org/oapi-p/jobs/8ca109b4-3b86-
4a9c-a284-a6d50f91019e",
          "rel": "status",
          "type": "application/json",
          "hreflang": "en",
          "title": "Job status"
        }
      ]
    },
    {
      "processID": "EchoProcess",
      "jobID": "0cf773a5-282a-4e23-96cc-f5dab18123e5",
      "status": "successful",
      "message": "EchoProcess job finished successful",
      "progress": 100,
      "links": [
        {
          "href": "http://processing.example.org/oapi-p/jobs/0cf773a5-282a-
4e23-96cc-f5dab18123e5",
          "rel": "status",
          "type": "application/json",
          "hreflang": "en",
          "title": "Job status"
        },
        {
          "href": "http://processing.example.org/oapi-p/jobs/0cf773a5-282a-
4e23-96cc-f5dab18123e5/results",
          "rel": "http://www.opengis.net/def/rel/ogc/1.0/results",
          "type": "application/json",
          "hreflang": "en",
          "title": "Job result"
        }
      ]
    },
    {
      "processID": "EchoProcess",
      "jobID": "63aadd9c-c0e5-4a7f-80f0-228dbb158f09",
      "status": "failed",
      "message": "EchoProcess job failed",
      "progress": 100,
      "links": [
        {
          "href": "http://processing.example.org/oapi-p/jobs/63aadd9c-c0e5-
4a7f-80f0-228dbb158f09",
          "rel": "status",
          "type": "application/json",
          "hreflang": "en",
          "title": "Job status"
        },
        {
          "href": "http://processing.example.org/oapi-p/jobs/63aadd9c-c0e5-
4a7f-80f0-228dbb158f09/results",
          "rel": "http://www.opengis.net/def/rel/ogc/1.0/exceptions",
          "type": "application/json",
          "hreflang": "en",
          "title": "Job exception"
        }
      ]
    }
  ],
  "links": [
    {
      "href": "http://processing.example.org/jobs?limit3&f=json",
```

```
      "rel": "self",
      "type": "application/json"
    },
    {
      "href": "http://processing.example.org/jobs?f=html",
      "rel": "alternate",
      "type": "text/html"
    },
    {
       "href": "http//processing.example.org/jobs?offset=4&limit=3&f=json",
       "rel": "next"
    }

  ]
}
```

11.4. Error situations

See Clause 7.5.1 for general guidance.

If the process with the specified identifier does not exist on the server, the status code of the response SHALL be `404` (see Clause 7.10.3, Requirement 15).

12

REQUIREMENTS CLASS "CALLBACK"

12 REQUIREMENTS CLASS "CALLBACK"

The Callback conformance class specifies a callback mechanism for completed jobs. In contrast to the pull-based mechanism specified in Clause 7.11 and Clause 7.12, 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 7	
http://www.opengis.net/spec/ogcapi-processes-1/1.0/req/callback	
Obligation	requirement
Target type	Web API
Dependency	OGC API — Processes Core

REQUIREMENT 80
/req/callback/job-callback
The server SHALL support callback functions for jobs.

Example — A callback in the execute operation:

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

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

Adding multiple callbacks is possible for getting progress updates and notifications of the success or failure of a job completion.

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

13

REQUIREMENTS CLASS "DISMISS"

13 REQUIREMENTS CLASS "DISMISS"

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

REQUIREMENTS CLASS 8	
http://www.opengis.net/spec/ogcapi-processes-1/1.0/req/dismiss	
Obligation	requirement
Target type	Web API
Dependency	OGC API — Processes Core

13.1. Operation

REQUIREMENT 81
/req/dismiss/job-dismiss-op
The server SHALL support the HTTP DELETE operation at the path `/jobs/{jobID}`.

13.2. Response

REQUIREMENT 82
/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 — 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/oapi-p/jobs",
      "rel": "up",
      "type": "application/json",
      "title": "The job list of this server"
    }
  ]
}
```

13.3. Error situations

See Clause 7.5.1 for general guidance.

If the process with the specified identifier does not exist on the server, the status code of the response SHALL be `404` (see /req/core/process-exception/no-such-process).

If the job with the specified identifier does not exist, the status code of the response SHALL be `404` (see /req/core/job-results-exception/no-such-job).

14

MEDIA TYPES

14 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.

15

ADDITIONAL API BUILDING BLOCKS

15 ADDITIONAL API BUILDING BLOCKS

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

a) Access the list of available processes

b) Access the description of a specific process

c) Create an execute JSON request (based on the description) and send it to the server via POST

d) Process the status info and/or results

This workflow is useful for generic clients 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 Standard. However, there may be limitations regarding the handling of input and output formats.

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

This standard uses the OpenAPI specification to define the JSON schemas and OpenAPI MAY also be used to describe the concrete API (see Clause 7.3). A variety of tools for automatic code generation exist for the OpenAPI specification. This makes it very easy for client and server implementers to work with APIs defined using OpenAPI. However, as the OGC API — Processes Standard defines several JSON schemas and leaves the concrete data types for input and outputs open, the automatic code generation cannot be used to its full extent. To cope with this and thus make the implementation of clients / servers easier for those that are not familiar with OGC (API) standards, additional alternatives to the process description and the paths to processes and jobs are permitted.

The following permissions do not affect the mandatory core requirements.

PERMISSION 11
/per/core/alternative-process-description
Servers MAY support alternative means of describing the inputs and outputs of a process.

The alternative-process-description 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 using the OpenAPI specification directly (see this example).

PERMISSION 12
/per/core/alternative-process-paths

PERMISSION 12

Servers MAY support alternative API paths.

The alternative-process-paths permission allows server implementations to specify alternative paths to processes and jobs.

An example of an OpenAPI document making use of these 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
```

Figure 28

The goals of these additional API building blocks are:

- Enabling a more seamless integration of this API with other OGC API standards and
- Enabling the use of tools to auto-generate clients / servers from the API description.

A

ANNEX A (NORMATIVE) ABSTRACT TEST SUITE

A

ANNEX A (NORMATIVE) ABSTRACT TEST SUITE

A.1. Introduction

OGC Web Application Programming Interfaces (APIs) are not Web Services in the traditional sense. Rather, they define the behavior and content of a set of Resources exposed through a Web API. Therefore, an API may expose resources in addition to those defined by the standard. A test engine must be able to traverse the API, identify and validate test points, and ignore resource paths which are not to be tested.

The following requirement applies for a server implementing the OGC API — Processes — Part 1: Core under test:

REQUIREMENT A.1

/req/core/test-process

If a server implementing the OGC API — Processes — Part 1: Core is tested using CITE tests, the server SHALL offer at least one testable process. Please refer to Annex A.1, Recommendation A.1 for further guidance.

RECOMMENDATION A.1

/rec/core/test-process

If a server implementing the OGC API — Processes — Part 1: Core is tested using CITE tests, the server SHOULD offer one of the following options:

a) An Echo process that returns any input that is provided, without any actual processing.

b) Provide example input data for a specific process.

The process logic SHOULD include a delay, whether through actual processing or a simple sleep mechanism, in order to test asynchronous execution.

A.2. Conformance Class Core

CONFORMANCE CLASS A.1	
http://www.opengis.net/spec/ogcapi-processes-1/1.0/conf/core	
Requirements class	Requirements Class “Core”
Target type	Web API

A.2.1. Landing Page /

ABSTRACT TEST A.1	
/conf/core/landingpage-op	
Requirement	/req/core/landingpage-op
Test purpose	Validate that a landing page can be retrieved from the expected location.
Test method	1. Issue an HTTP GET request to the root URL / 2. Validate the contents of the returned document using test /conf/core/landingpage-success.

ABSTRACT TEST A.2	
/conf/core/landingpage-success	
Requirement	/req/core/landingpage-success
Test purpose	Validate that the landing page complies with the require structure and contents.
Test method	1. Validate that a document was returned with an HTTP status code or 200. 2. Validate the landing page for all supported media types using the resources and tests identified in Table A.1

ABSTRACT TEST A.2	
	3. For formats that require manual inspection, perform the following: 1. Validate that the landing page includes a “service-desc” and/or “service-doc” link to an API Definition. 2. Validate that the landing page includes a “http://www.opengis.net/def/rel/ogc/1.0/conformance” link to the conformance class declaration. 3. Validate that the landing page includes a “processes” link to the list of processes.
The landing page may be retrieved in a number of different formats. The following table identifies the applicable schema document for each format and the test to be used to validate the landing page against that schema. All supported formats should be exercised.	

Table A.1 — Schema and Tests for Landing Pages

FORMAT	SCHEMA DOCUMENT	TEST ID
HTML	landingPage.yaml	/conf/html/content
JSON	landingPage.yaml	/conf/geojson/content

A.2.2. API Definition /api

ABSTRACT TEST A.3	
/conf/core/api-definition-op	
Requirement	/req/core/api-definition-op
Test purpose	Validate that the API Definition document can be retrieved from the expected location.
Test method	1. Construct a path for the API Definition document that ends with `/api`. 2. Issue a HTTP GET request on that path 3. Validate the contents of the returned document using test / conf/core/api-definition-success.

ABSTRACT TEST A.4
/conf/core/api-definition-success

ABSTRACT TEST A.4	
Requirement	/req/core/api-definition-success
Test purpose	Validate that the API Definition complies with the required structure and contents.
Test method	1. Validate that a document was returned with a status code 200 2. Validate the API Definition document against an appropriate schema document.

A.2.3. Conformance Path /conformance

ABSTRACT TEST A.5	
/conf/core/conformance-op	
Requirement	/req/core/conformance-op
Test purpose	Validate that a Conformance Declaration can be retrieved from the expected location.
Test method	1. Construct a path for each “rel=http://www.opengis.net/def/rel/ogc/1.0/conformance” link on the landing page as well as for the {root}/conformance path. 2. Issue an HTTP GET request on each path 3. Validate the contents of the returned document using test /conf/core/conformance-success.

ABSTRACT TEST A.6	
/conf/core/conformance-success	
Requirement	/req/core/conformance-success
Test purpose	Validate that the Conformance Declaration response complies with the required structure and contents.
Test method	1. Validate that a document was returned with an HTTP status code of 200. 2. Validate the response document against OpenAPI 3.0 schema link: confClasses.yaml 3. Validate that the document includes the conformance class “http://www.opengis.net/spec/ogcapi-processes-1/1.0/conf/core”

ABSTRACT TEST A.6	
	4. Validate that the document list all OGC API conformance classes that the API implements.

A.2.4. HTTP 1.1

ABSTRACT TEST A.7	
/conf/core/http	
Requirement	/req/core/http
Test purpose	Validate that the resource paths advertised through the API conform with HTTP 1.1 and, where appropriate, TLS.
Test method	1. All compliance tests SHALL be configured to use the HTTP 1.1 protocol exclusively. 2. For APIs which support HTTPS, all compliance tests SHALL be configured to use HTTP over TLS (RFC 2818) with their HTTP 1.1 protocol.

A.2.5. Processes /processes

A.2.5.1. Process list

ABSTRACT TEST A.8	
/conf/core/process-list	
Requirement	/req/core/process-list
Test purpose	Validate that information about the processes can be retrieved from the expected location.
Test method	1. Issue an HTTP GET request to the URL {root}/processes 2. Validate the contents of the returned document using test /conf/core/process-list-success.

ABSTRACT TEST A.9	
/conf/core/pl-limit-definition	
Requirement	/req/core/pl-limit-definition

ABSTRACT TEST A.9	
Test purpose	Validate that the `limit` query parameter is constructed correctly.
Test method	1. Verify that the `limit` query parameter complies with its definition in requirement /req/core/pl-limit-definition. Note that the API can define different values for “minimum”, “maximum” and “default”.

ABSTRACT TEST A.10	
/conf/core/process-list-success	
Requirement	/req/core/process-list-success
Test purpose	Validate that the process list content complies with the required structure and contents.
Test method	1. Validate that a document was returned with an HTTP status code of 200. 2. Validate the process list content for all supported media types using the resources and tests identified in Table A.2
The process list may be retrieved in a number of different formats. The following table identifies the applicable schema document for each format and the test to be used to validate the against that schema. All supported formats should be exercised.	

Table A.2 — Schema and Tests for Lists content

FORMAT	SCHEMA DOCUMENT	TEST ID
HTML	processList.yaml	/conf/html/content
JSON	processList.yaml	/conf/json/content

ABSTRACT TEST A.11	
/conf/core/pl-links	
Requirement	/req/core/pl-links
Test purpose	Validate that the proper links are included in a response.
Test method	1. Get a list of process summaries as per test /conf/core/process-list-op.

ABSTRACT TEST A.11	
	2. Verify that the `links` section of the response contains a link with `rel=self`. 3. Verify that the `links` section of the response contains a link with `rel=alternate` for each response representation the service claims to support in its conformance document.

ABSTRACT TEST A.12	
/conf/core/pl-limit-response	
Requirement	/req/core/pl-limit-response
Test purpose	Validate that the `limit` query parameter is processed correctly.
Test method	1. Get a list of processes as per test /conf/core/process-list-op and append the `limit` query parameter to the request. 2. Count the number of process summaries listed in the response. 3. Verify that this count is not greater than the value specified by the `limit` parameter. 4. If the API definition specifies a maximum value for `limit` parameter, verify that the count does not exceed this maximum value.

A.2.5.2. Process description /processes/{processID}

ABSTRACT TEST A.13	
/conf/core/process	
Requirement	/req/core/process
Test purpose	Validate that a process description can be retrieved from the expected location.
Test method	1. For every Process described in the process list content, issue an HTTP GET request to the URL `/processes/{process ID}` where `{processID}` is the `id` property for the process. 1. Validate the response using the test /conf/core/process-success.

ABSTRACT TEST A.14
/conf/core/process-success

ABSTRACT TEST A.14	
Requirement	/req/core/process-success
Test purpose	Validate that the content complies with the required structure and contents.
Test method	1. Validate that a document was returned with an HTTP status code of 200. 2. Verify that the content of the response is valid description of the interface of the process for all supported process description models.
The interface of a process may be describing using a number of different models or process description languages. The following table identifies the applicable schema document for each process description model described in this standard.	

Table A.3 — Schema and Tests for Process Description Models

MODEL	SCHEMA DOCUMENT	TEST ID
OGC Process Description JSON	process.yaml	/conf/ogc-process-description/json-encoding

A.2.5.3. Process exception

ABSTRACT TEST A.15	
/conf/core/process-exception-no-such-process	
Requirement	/req/core/process-exception-no-such-process
Test purpose	Validate that an invalid process identifier is handled correctly.
Test method	1. Issue an HTTP GET request to a URL that includes the `{processID}` as a path element using a non-existent process identifier. 2. Validate that the document was returned with a 404. 3. Validate that the document contains the exception `type` "http://www.opengis.net/def/exceptions/ogcapi-processes-1/1.0/no-such-process". 4. Validate the document for all supported media types using the resources and tests identified in Table A.4
An exception response caused by the use of an invalid process identifier may be retrieved in a number of different formats. The following table identifies the applicable schema document	

ABSTRACT TEST A.15
for each format and the test to be used to validate the response. All supported formats should be exercised.

Table A.4 — Schema and Tests for Non-existent Process

FORMAT	SCHEMA DOCUMENT	TEST ID
HTML	exception.yaml	/conf/html/content
JSON	exception.yaml	/conf/json/content

A.2.6. Jobs

A.2.6.1. Job creation /processes/{processID}/execution

ABSTRACT TEST A.16	
/conf/core/job-creation-op	
Requirement	/req/core/job-creation-op
Test purpose	Validate the creation of a new job.
Test method	1. Issue an HTTP POST request to the URL '/processes/{processID}/execution' for each execution mode according to requirements /conf/core/job-creation-default-execution-mode or /conf/core/job-creation-auto-execution-mode. 2. Validate the contents of the POST request using the test /conf/core/job-creation-request. 3. Validate the creation of the job according the requirements /req/core/job-creation-default-execution-mode, /req/core/job-creation-auto-execution-mode.

ABSTRACT TEST A.17	
/conf/core/job-creation-auto-execution-mode	
Requirement	/req/core/job-creation-op
Test purpose	Validate that the server correctly handles the execution mode for a process.

ABSTRACT TEST A.17

Test method	1. Get a description of each process offered by the server using test /conf/core/process. 2. Inspect the description of each process and take note of the job control options for the process. 3. Setting the HTTP `Prefer` header to include the `respond-sync` preference, construct an execute request according to test /conf/core/job-creation-request. 4. For processes that are supposed to execute asynchronously according to the /req/core/job-creation-auto-execution-mode requirement, verify the successful execution according to the /conf/core/job-creation-success-async test. 5. For processes that are supposed to execute synchronously according to the /req/core/job-creation-auto-execution-mode requirement, verify the successful execution according to the relevant requirement based on the combination of execute parameters. 6. For processes that may execute either synchronously or asynchronously according to the /req/core/job-creation-auto-execution-mode requirement, verify that successful execution according to the relevant requirement based on the combination of execute parameters.

ABSTRACT TEST A.18

/conf/core/job-creation-default-execution-mode

Requirement	/req/core/job-creation-op
Test purpose	Validate that the server correctly handles the default execution mode for a process.
Test method	1. Get a description of each process offered by the server using test /conf/core/process. 2. Inspect the description of each process and take note of the job control options for the process. 3. Without setting the HTTP `Prefer` header, construct an execute request according to test /conf/core/job-creation-request. 4. For processes that are supposed to execute asynchronously according to the /req/core/job-creation-default-execution-mode requirement, verify the successful execution according to the /conf/core/job-creation-success-async test.

ABSTRACT TEST A.18	
	5. For processes that are supposed to execute synchronously according to the /req/core/job-creation-auto-execution-mode requirement, verify the successful synchronous execution according to the relevant requirement based on the combination of execute parameters.

ABSTRACT TEST A.19	
/conf/core/job-creation-request	
Requirement	/req/core/job-creation-request
Test purpose	Validate that the body of a job creation operation complies with the required structure and contents.
Test method	1. Verify the contents of the request body against the Open API 3.0 schema execute.yaml.

ABSTRACT TEST A.20	
/conf/core/job-creation-inputs	
Requirement	/req/core/job-creation-inputs
Test purpose	Validate that servers can accept input values both inline and by reference.
Test method	1. Verify that the server passes tests /conf/core/job-creation-input-inline and /conf/core/job-creation-input-ref .

ABSTRACT TEST A.21	
/conf/core/job-creation-input-inline	
Requirement	/req/core/job-creation-input-inline
Test purpose	Validate in-line process input values are validated against the corresponding schema from the process description.
Test method	1. Get a description of each process offered by the server using test /conf/core/process. 2. For each process construct an execute request according to test /conf/core/job-creation-request taking care to encode process inputs in-line with the execute request according to the requirement /req/core/job-creation-input-inline. 3. Verify that each process executes successfully according to the relevant requirement based on the combination of execute parameters.

ABSTRACT TEST A.22	
/conf/core/job-creation-input-ref	
Requirement	/req/core/job-creation-input-ref
Test purpose	Validate that input values specified by reference in an execute request are correctly processed.
Test method	1. Get a description of each process offered by the server using test /conf/core/process. 2. For each identified process construct an execute request according to test /conf/core/job-creation-request taking care to encode input values by reference according to requirement /req/core/job-creation-input-ref. 3. Verify that each process executes successfully according to the relevant requirement based on the combination of execute parameters.

ABSTRACT TEST A.23	
/conf/core/job-creation-input-array	
Requirement	/req/core/job-creation-input-array
Test purpose	Verify that the server correctly recognizes the encoding of parameter values for input parameters with a maximum cardinality greater than one.
Test method	1. Get a description of each process offered by the server using test /conf/core/process. 2. Inspect the description of each process and identify the list of processes that have inputs with a maximum cardinality greater that one. 3. For each identified process construct an execute request according to test /conf/core/job-creation-request taking care to encode the inputs with maximum cardinality > 1 according to the requirement /req/core/job-creation-input-array. 4. Verify that each process executes successfully according to the relevant requirement based on the combination of execute parameters.

ABSTRACT TEST A.24	
/conf/core/job-creation-input-inline-object	
Requirement	/req/core/job-creation-input-inline-object

ABSTRACT TEST A.24	
Test purpose	Validate that inputs with a complex object schema encoded in-line in an execute request are correctly processed.
Test method	1. Get a description of each process offered by the server using test /conf/core/process. 2. Inspect the description of each process and identify the subset of processes that have inputs with complex `object schemas (i.e. inputs of type `object`). 3. For each identified process construct an execute request according to test /conf/core/job-creation-request taking care to encode the identified object inputs in-line in the execute request according to requirement /req/core/job-creation-input-inline-object. 4. Verify that each process executes successfully according to the relevant requirement based on the combination of execute parameters.

ABSTRACT TEST A.25	
/conf/core/job-creation-input-inline-mixed	
Requirement	/req/core/job-creation-input-inline-mixed
Test purpose	Validate that inputs of mixed content encoded in-line in an execute request are correctly processed.
Test method	1. Get a description of each process offered by the server using test /conf/core/process. 2. Inspect the description of each process and identify the subset of processes that have inputs of mixed content using the `oneOf[]` JSON Schema construct to define several alternate input value schemas. 3. For each identified process construct an execute request according to test /conf/core/job-creation-request taking care to encode the identified mix-content inputs in-line in the execute request according to requirement /req/core/job-creation-input-inline-mixed. 4. Verify that each process executes successfully according to the relevant requirement based on the combination of execute parameters.

ABSTRACT TEST A.26
/conf/core/job-creation-input-inline-binary

ABSTRACT TEST A.26	
Requirement	/req/core/job-creation-input-binary
Test purpose	Validate that binary input values encoded as base-64 string in-line in an execute request are correctly processes.
Test method	1. Get a description of each process offered by the server using test /conf/core/process. 2. Inspect the description of each process and identify the subset of processes that have binary inputs. 3. For each identified process construct an execute request according to test /conf/core/job-creation-request taking care to encode binary input values in-line in the execute request according to requirement /req/core/job-creation-input-inline-binary. 4. Verify that each process executes successfully according to the relevant requirement based on the combination of execute parameters.

ABSTRACT TEST A.27	
/conf/core/job-creation-input-inline-bbox	
Requirement	/req/core/job-creation-input-inline-bbox
Test purpose	Validate that inputs with a bounding box schema encoded in-line in an execute request are correctly processed.
Test method	1. Get a description of each process offered by the server using test /conf/core/process. 2. Inspect the description of each process and identify the subset of processes that have bounding box inputs that are supposed to conform to the bbox.yaml schema. 3. For each identified process construct an execute request according to test /conf/core/job-creation-request taking care to encode values for the identified bounding box inputs in-line in the execute request. 4. Verify that each process executes successfully according to the relevant requirement based on the combination of execute parameters.

ABSTRACT TEST A.28	
/conf/core/job-creation-input-validation	
Requirement	/req/core/job-creation-input-validation

ABSTRACT TEST A.28	
Test purpose	Verify that the server correctly validates process input values according to the definition obtained from the process description.
Test method	1. Get a description of each process offered by the server using test /conf/core/process. 2. Inspect the description of each process taking note of the definition of each process input and specifically the schema of each process input. 3. For each process construct an execute request according to test /conf/core/job-creation-request taking care to encode the input values according to the schema from the definition of each input. 4. Verify that each process executes successfully according to the relevant requirement based on the combination of execute parameters. 5. For each process construct an execute request according to test /conf/core/job-creation-request taking care to encode some of the input values in violation of the schema from the definition of the selected input. 6. Verify that each process generates an exception report that identifies the improperly specified input value(s).

The response generated when executing a process depends on a number of parameters specified in the execute request. The following table enumerates the relevant requirement that needs to be satisfied based on the various execute parameter combinations.

MODE	RESPONSE	TX MODE	# OUT	REQUIREMENT
sync	raw	value	1	/req/core/job-creation-sync-raw-value-one
			*	/req/core/job-creation-sync-raw-value-multi
		ref	1	/req/core/job-creation-sync-raw-ref
			*	
	document	value	1	/req/core/job-creation-sync-document
			*	
		ref	1	

MODE	RESPONSE	TX MODE	# OUT	REQUIREMENT
			*	
async	*any*	*any*	*any*	/req/core/job-creation-success-async

NOTE: The value *any* in a cell means *"for any valid value of the execute parameter represented by that cell"*.

ABSTRACT TEST A.29	
/conf/core/job-creation-sync-raw-value-one	
Requirement	/req/core/job-creation-sync-raw-value-one
Test purpose	Validate that the server responds as expected when synchronous execution is negotiated, a single output value is requested, the response type is `raw` and the output transmission is `value`.
Test method	1. Get a description of each process offered by the server using test /conf/core/process. 2. Inspect the description of each process and identify the subset of processes that generate at least one output and support the `sync-execute` job control option and the `value` output transmission. 3. For each identified process construct an execute request according to test /conf/core/job-creation-request ensuring that synchronous execution is negotiated according to test /conf/core/job-creation-default-execution-mode, that only one output is requested, that the requested response type is `raw` (i.e. `"response": "raw"`) and that the output transmission is set to `value` (i.e. `"transmissionMode": "value"`) according to requirement /req/core/job-creation-sync-raw-value-one. 4. Verify that each process executes successfully according to requirement /req/core/job-creation-sync-raw-value-one.

ABSTRACT TEST A.30	
/conf/core/job-creation-sync-raw-value-multi	
Requirement	/req/core/job-creation-sync-raw-value-multi
Test purpose	Validate that the server responds as expected when synchronous execution is negotiated, the response type is `raw` and the output transmission is `value`.
Test method	1. Get a description of each process offered by the server using test /conf/core/process.

ABSTRACT TEST A.30	
	2. Inspect the description of each process and identify the subset of processes that generate more that one output, support the `sync-execute` job control option and the `value` output transmission. 3. For each identified process construct an execute request according to test /conf/core/job-creation-request ensuring that synchronous execution is negotiated according to test /conf/core/job-creation-default-execution-mode, that more than one output is requested, that the requested response type is `raw` (i.e. `"response": "raw"`) and the the transmission mode is set to `value` (i.e. `"transmissionMode": "value"`) according to requirement /req/core/job-creation-sync-raw-value-multi. 4. Verify that each process executes successfully according to requirement /req/core/job-creation-sync-raw-value-multi.

ABSTRACT TEST A.31	
/conf/core/job-creation-sync-raw-ref	
Requirement	/req/core/job-creation-sync-raw-ref
Test purpose	Validate that the server responds as expected when synchronous execution is negotiated, the response type is `raw` and the transmission mode is `ref`.
Test method	1. Get a description of each process offered by the server using test /conf/core/process. 2. Inspect the description of each process and identify the subset of processes that support the `sync-execute` job control option and the `reference` output transmission. 3. For each identified process construct an execute request according to test /conf/core/job-creation-request ensuring that synchronous execution is negotiated according to test /conf/core/job-creation-default-execution-mode, that the requested response type is `raw` (i.e. `"response": "raw"`) and the transmission mode is set to `ref` (i.e. `"transmissionMode": "ref"`) according to requirement /req/core/job-creation-sync-raw-ref. 4. Verify that each process executes successfully according to requirement /req/core/job-creation-sync-raw-ref.

ABSTRACT TEST A.32
/conf/core/job-creation-sync-raw-mixed-multi

ABSTRACT TEST A.32	
Requirement	/req/core/job-creation-sync-raw-mixed-multi
Test purpose	Validate that the server responds as expected when synchronous execution is negotiated, the response type is `raw` and the output transmission is a mix of `value` and `reference`.
Test method	1. Get a description of each process offered by the server using test /conf/core/process. 2. Inspect the description of each process and identify the subset of processes that generate more that one output and support the `sync-execute` job control option. 3. For each identified process construct an execute request according to test /conf/core/job-creation-request ensuring that synchronous execution is negotiated according to test /conf/core/job-creation-default-execution-mode, that more than one output is requested, that the requested response type is `raw` (i.e. `"response": "raw"`) and the the transmission mode is a mix of `value` (i.e. `"transmissionMode": "value"`) and reference (i.e. `"transmissionMode": "reference"`) according to requirement /req/core/job-creation-sync-raw-mixed-multi. 4. Verify that each process executes successfully according to requirement /req/core/job-creation-sync-raw-mixed-multi. 5. For each output requested with `"transmissionMode": "value"` verify that the body of the corresponding part contains the output value. 6. For each output requested with `"transmissionMode": "reference"` verify that the body of the corresponding part is empty and the `Content-Location` header is included that points to the output value.

ABSTRACT TEST A.33	
/conf/core/job-creation-sync-document	
Requirement	/req/core/job-creation-sync-document
Test purpose	Validate that the server responds as expected when synchronous execution is negotiated and the response type is `document`.
Test method	1. Get a description of each process offered by the server using test /conf/core/process. 2. Inspect the description of each process and identify the subset of processes that support the `sync-execute` job control option.

ABSTRACT TEST A.33	
	3. For each identified process construct an execute request according to test /conf/core/job-creation-request ensuring that synchronous execution has been negotiated according to tests /conf/core/job-creation-default-execution-mode and the requested response type is `document` (i.e. `"response": "document"`) according to requirement /req /core/job-creation-sync-document. 4. Verify that each process executes successfully according to requirement /req/core/job-creation-sync-document.

ABSTRACT TEST A.34	
/conf/core/job-creation-success-async	
Requirement	/req/core/job-creation-success-async
Test purpose	Validate the results of a job that has been created using the `async` execution mode.
Test method	1. Validate that results of the job was returned with an HTTP status code 201. 2. Validate the HTTP headers of the results using the test /conf/core/job-creation-success-header-async.

A.2.6.2. Job status /jobs/{jobID}

ABSTRACT TEST A.35	
/conf/core/job-op	
Requirement	/req/core/job
Test purpose	Validate that the status info of a job can be retrieved.
Test method	1. Create a job as per /req/core/job-creation-op and note the {jobID} assigned to the job. 2. Issue an HTTP GET request to the URL '/jobs/{jobID}'. 3. Validate the contents of the returned document using the test /conf/core/job-success.

ABSTRACT TEST A.36
/conf/core/job-success

ABSTRACT TEST A.36	
Requirement	/req/core/job-success
Test purpose	Validate that the job status info complies with the require structure and contents.
Test method	1. Validate that the document was returned with an HTTP status code of 200. 2. Validate the job status info for all supported media types using the resources and tests identified in Table A.5
The status info page for a job may be retrieved in a number of different formats. The following table identifies the applicable schema document for each format and the test to be used to validate the status info against that schema. All supported formats should be exercised.	

Table A.5 — Schema and Tests for the Job Status Info

FORMAT	SCHEMA DOCUMENT	TEST ID
HTML	statusInfo.yaml	/conf/html/content
JSON	statusInfo.yaml	/conf/json/content

ABSTRACT TEST A.37	
/conf/core/job-exception-no-such-job	
Requirement	/req/core/job-exception-no-such-job
Test purpose	Validate that an invalid job identifier is handled correctly.
Test method	1. Issue an HTTP GET request to the URL that includes the `{jobID}` as a path element using a non-existent job identifier. 2. Validate that the document was returned with a 404. 3. Validate that the document contains the exception type “http://www.opengis.net/def/exceptions/ogcapi-processes-1/1.0/no-such-job”. 4. Validate the document for all supported media types using the resources and tests identified in Table A.6
An exception response caused by the use of an invalid job identifier may be retrieved in a number of different formats. The following table identifies the applicable schema document for each format and the test to be used to validate the response. All supported formats should be exercised.	

Table A.6 — Schema and Tests for the Job Result for Non-existent Job

FORMAT	SCHEMA DOCUMENT	TEST ID
HTML	exception.yaml	/conf/html/content
JSON	exception.yaml	/conf/json/content

A.2.6.3. Job results /jobs/{jobID}/results

ABSTRACT TEST A.38	
/conf/core/job-results	
Requirement	/req/core/job-results
Test purpose	Validate that the results of a job can be retrieved.
Test method	1. Create a job as per /req/core/job-creation-op and note the {jobID} assigned to the job. 2. Issue an HTTP GET request to the URL '/jobs/{jobID}/results'. 3. Validate that the document was returned with a status code 200. 4. Validate the contents of the returned document using the test /conf/core/job-results-success.

Retrieving the results of an asynchronously executed process depends on a number of parameters specified in the execute request. The following table enumerates the relevant requirement that needs to be satisfied based on the various execute parameter combinations.

MODE	RESPONSE	TX MODE	# OUT	ABSTRACT TEST
sync	*any*	*any*	*any*	/conf/core/job-results-sync
async	raw	value	1	/conf/core/job-results-async-raw-value-one
			*	/conf/core/job-results-async-raw-value-multi
		ref	1	/conf/core/job-results-async-raw-ref
			*	

MODE	RESPONSE	TX MODE	# OUT	ABSTRACT TEST
	document	value	1	/conf/core/job-results-async-document
			*	
		ref	1	
			*	

NOTE: The value *any* in a cell means *"for any valid value of the execute parameter represented by that cell"*.

ABSTRACT TEST A.39	
/conf/core/job-results-sync	
Requirement	/req/core/job-results-sync
Test purpose	Validate that the server responds as expected when getting results from a job for a process that has been executed synchronously.
Test method	1. Get a description of each process offered by the server using test /conf/core/process. 2. Inspect the description of each process and identify the subset of processes that support the `sync-execute` job control option. 3. For each identified process construct an execute request according to test /conf/core/job-creation-request ensuring that synchronous execution is negotiated according to test /conf/core/job-creation-default-execution-mode. 4. Inspect the headers of the response and see if a `Link` header is included with `rel=monitor`. 5. If the link exists, get the job status as per test /conf/core/job-op and ensure that the job status is set to `successful`.

ABSTRACT TEST A.40	
/conf/core/job-results-async-raw-value-one	
Requirement	/req/core/job-results-async-raw-value-one
Test purpose	Validate that the server responds as expected when asynchronous execution is negotiated, one output is requested, the response type is `raw` and the output transmission is `value`.

ABSTRACT TEST A.40	
Test method	1. Get a description of each process offered by the server using test /conf/core/process. 2. Inspect the description of each process and identify the subset of processes that generate at least one output and that support the `async-execute` job control option and the `value` output transmission. 3. For each identified process construct an execute request according to test /conf/core/job-creation-request ensuring that asynchronous execution is negotiated according to test /conf/core/job-creation-auto-execution-mode, that the requested response type is `raw` (i.e. `"response": "raw"`) and that the output transmission is set to `value` (i.e. `"outputTransmission": "value"`) according to requirement /req/core/job-creation-async-raw-value-one. 4. If the server responds asynchronously, periodically retrieve the status of the asynchronously executed job as per test /conf/core/job-op. 5. When the job status is `successful`, get the results as per test /conf/core/job-results and verify that they conform to requirement /req/core/job-results-async-raw-value-one.

NOTE 1: In the case where a process supports both `async-execute` and `sync-execute` job control options there is a possibility that the server responds synchronously even though the `Prefer` headers asserts a `respond-async` preference. In this case, the following additional test should be performed.

Test method	1. Inspect the headers of the synchronous response and see if a `Link` header is included with `rel=monitor`. 2. If the link exists, get the job status as per test /conf/core/job-op and ensure that the job status is set to `successful`. 3. Get the results as per test /conf/core/job-results and verify that they conform to the test /conf/core/job-results-async-raw-value-multi. 4. If the link does not exist then verify that the synchronous response conforms to requirement /req/core/job-creation-sync-raw-value-one.

ABSTRACT TEST A.41	
/conf/core/job-results-async-raw-value-multi	
Requirement	/req/core/job-results-async-raw-value-multi

ABSTRACT TEST A.41	
Test purpose	Validate that the server responds as expected when asynchronous execution is negotiated, more than one output is requested, the response type is `raw` and the output transmission is `value`.
Test method	1. Get a description of each process offered by the server using test /conf/core/process. 2. Inspect the description of each process and identify the subset of processes that generate more than one output and that support the `async-execute` job control option and the `value` output transmission. 3. For each identified process construct an execute request according to test /conf/core/job-creation-request ensuring that asynchronous execution is negotiated according to test /conf/core/job-creation-auto-execution-mode, that the requested response type is `raw` (i.e. `"response": "raw"`) and that the output transmission is set to `value` (i.e. `"outputTransmission": "value"`) according to requirement /req/core/job-creation-async-raw-value-multi. 4. Periodically retrieve the status of the asynchronously execute job as per test /conf/core/job-op. 5. When the job status is `successful`, get the results as per test /conf/core/job-results and verify that they conform to requirement /conf/core/job-results-async-raw-value-multi.

NOTE 2: In the case where a process supports both `async-execute` and `sync-execute` job control options there is a possibility that the server responds synchronously even though the `Prefer` headers asserts a `respond-async` preference. In this case, the following additional test should be performed.

Test method	1. Inspect the headers of the synchronous response and see if a `Link` header is included with `rel=monitor`. 2. If the link exists, get the job status as per test /conf/core/job-op and ensure that the job status is set to `successful`. 3. Get the results as per test /conf/core/job-results and verify that they conform to the test /conf/core/job-results-async-raw-value-multi. 4. If the link does not exist then verify that the synchronous response conforms to requirement /req/core/job-creation-sync-raw-value-multi.

ABSTRACT TEST A.42	
/conf/core/job-results-async-raw-ref	
Requirement	/req/core/job-results-async-raw-ref
Test purpose	Validate that the server responds as expected when asynchronous execution is negotiated, the response type is `raw` and the output transmission is `reference`.
Test method	1. Get a description of each process offered by the server using test /conf/core/process. 2. Inspect the description of each process and identify the subset of processes that support the `async-execute` job control option and the `reference` output transmission. 3. For each identified process construct an execute request according to test /conf/core/job-creation-request ensuring that synchronous execution is negotiated according to test /conf/core/job-creation-auto-execution-mode, that the requested response type is `raw` (i.e. `"response": "raw"`) and that the output transmission is set to `reference` (i.e. `"outputTransmission": "reference"`) according to requirement /req/core/job-creation-async-raw-ref. 4. If the server responds asynchronously, periodically retrieve the status of the asynchronously executed job as per test /conf/core/job-op. 5. When the job status is `successful`, get the results as per test /conf/core/job-results and verify that they conform to requirement /req/core/job-results-async-ref.

NOTE 3: In the case where a process supports both `async-execute` and `sync-execute` job control options there is a possibility that the server responds synchronously even though the `Prefer` headers asserts a `respond-async` preference. In this case, the following additional test should be performed.

Test method	1. Inspect the headers of the synchronous response and see if a `Link` header is included with `rel=monitor`. 2. If the link exists, get the job status as per test /conf/core/job-op and ensure that the job status is set to `successful`. 3. Get the results as per test /conf/core/job-results and verify that they conform to the test /conf/core/job-results-async-document. 4. If the link does not exist then verify that the synchronous response conforms to requirement /req/core/job-creation-sync-raw-ref.

ABSTRACT TEST A.43	
/conf/core/job-results-async-raw-mixed-multi	
Requirement	/req/core/job-results-async-raw-mixed-multi
Test purpose	Validate that the server responds as expected when asynchronous execution is negotiated, more than one output is requested, the response type is `raw` and the output transmission is a mix of `value` and `reference`.
Test method	1. Get a description of each process offered by the server using test /conf/core/process. 2. Inspect the description of each process and identify the subset of processes that generate more than one output and that support the `async-execute` job control option. 3. For each identified process construct an execute request according to test /conf/core/job-creation-request ensuring that asynchronous execution is negotiated according to test /conf/core/job-creation-auto-execution-mode, that the requested response type is `raw` (i.e. `"response": "raw"`) and that the output transmission is set to a mix of `value` (i.e. `"outputTransmission": "value"`) and `reference` (i.e. `"outputTransmission": "reference"`) according to requirement /req/core/job-creation-async-raw-mixed-multi. 4. Periodically retrieve the status of the asynchronously execute job as per test /conf/core/job-op. 5. When the job status is `successful`, get the results as per test /conf/core/job-results and verify that they conform to requirement /conf/core/job-results-async-raw-mixed-multi. 6. For each output requested with `"transmissionMode": "value"` verify that the body of the corresponding part contains the output value. 7. For each output requested with `"transmissionMode": "reference"` verify that the body of the corresponding part is empty and the `Content-Location` header is included that points to the output value.

NOTE 4: In the case where a process supports both `async-execute` and `sync-execute` job control options there is a possibility that the server responds synchronously even though the `Prefer` headers asserts a `respond-async` preference. In this case, the following additional test should be performed.

Test method	1. Inspect the headers of the synchronous response and see if a `Link` header is included with `rel=monitor`.

ABSTRACT TEST A.43	
	2. If the link exists, get the job status as per test /conf/cor e/job-op and ensure that the job status is set to `successful`. 3. Get the results as per test /conf/core/job-results and verify that they conform to the test /conf/core/job-results-async-raw-mixed-multi. 4. If the link does not exist then verify that the synchronous response conforms to requirement /req/core/job-creation-sync-raw-mixed-multi.

ABSTRACT TEST A.44	
/conf/core/job-results-async-document	
Requirement	/req/core/job-results-async-document
Test purpose	Validate that the server responds as expected when the asynchronous execution is negotiated and the response type is `document`.
Test method	1. Get a description of each process offered by the server using test /conf/core/process. 2. Inspect the description of each process and identify the subset of processes that support the `async-execute` job control option. 3. For each identified process construct an execute request according to test /conf/core/job-creation-request ensuring that asynchronous execution is negotiated according to test /conf/core/job-creation-auto-execution-mode and that the requested response type is `document` (i.e. `"response": "document"`) according to requirement /req/core/job-creation-async-document. 4. If the server responds asynchronously periodically retrieve the status of the asynchronously execute job as per test /conf/core/job-op. 5. When the job status is `successful`, get the results as per test /conf/core/job-results and verify that they conform to requirement /req/core/job-results-async-document.

NOTE 5: In the case where a process supports both `async-execute` and `sync-execute` job control options there is a possibility that the server responds synchronously even though the `Prefer` headers asserts a `respond-async` preference. In this case, the following additional test should be performed:

ABSTRACT TEST A.44	
Test method	1. Inspect the headers of the synchronous response and see if a `Link` header is included with `rel=monitor`. 2. If the link exists, get the job status as per test /conf/cor e/ job-op and ensure that the job status is set to `successful`. 3. Get the results as per test /conf/core/job-results and verify that they conform to the test /conf/core/job-results-async-document. 4. If the link does not exist then verify that the synchronous response conforms to the requirement /req/core/job-creation-sync-documen.

ABSTRACT TEST A.45	
/conf/core/job-results-failed	
Requirement	/req/core/job-results-exception-no-such-job
Test purpose	Validate that the job results retrieved using an invalid job identifier complies with the require structure and contents.
Test method	1. Issue an HTTP GET request to the URL '/jobs/{jobID}/ results' using an invalid {jobID}. 2. Validate that the document was returned with a 404. 3. Validate that the document contains the exception type "http://www.opengis.net/def/exceptions/ogcapi-processes-1/1.0/no-such-job". 4. Validate the document for all supported media types using the resources and tests identified in Table A.7
The job results page for a job may be retrieved in a number of different formats. The following table identifies the applicable schema document for each format and the test to be used to validate the job results for a non-existent job against that schema. All supported formats should be exercised.	

Table A.7 — Schema and Tests for the Job Result for Non-existent Job

FORMAT	SCHEMA DOCUMENT	TEST ID
HTML	exception.yaml	/conf/html/content
JSON	exception.yaml	/conf/json/content

ABSTRACT TEST A.46	
/conf/core/job-results-exception-results-not-ready	
Requirement	/req/core/job-results-exception-results-not-ready
Test purpose	Validate that the job results retrieved for an incomplete job complies with the require structure and contents.
Test method	1. Create a job as per /req/core/job-creation-op and note the {jobID} assigned to the job; ensure that the job is long-running. 2. Issue an HTTP GET request to the URL '/jobs/{jobID}/results' before the job completes execution. 3. Validate that the document was returned with a 404. 4. Validate that the document contains the exception `type` "http://www.opengis.net/def/exceptions/ogcapi-processes-1/1.0/result-not-ready". 5. Validate the document for all supported media types using the resources and tests identified in Table A.8
The job results page for a job may be retrieved in a number of different formats. The following table identifies the applicable schema document for each format and the test to be used to validate the job results for an incomplete job against that schema. All supported formats should be exercised.	

Table A.8 — Schema and Tests for the Job Result for an Incomplete Job

FORMAT	SCHEMA DOCUMENT	TEST ID
HTML	exception.yaml	/conf/html/content
JSON	exception.yaml	/conf/json/content

ABSTRACT TEST A.47	
/conf/core/job-results-failed	
Requirement	/req/core/job-results-failed
Test purpose	Validate that the job results for a failed job complies with the require structure and contents.
Test method	1. Create a job as per /req/core/job-creation-op but arrange a priori that the job will fail; note the {jobID} assigned to the job.

ABSTRACT TEST A.47	
	2. Ensure that the failed job will not result in an HTTP error code of 404. 3. Issue an HTTP GET request to the URL '/jobs/{jobID}/results'. 4. Validate that the document was returned with a HTTP error code (4XX or 5XX). 5. Validate that the document contains an exception `type` that corresponds to the reason the job failed (e.g. InvalidParameterValue for invalid input data). 6. Validate the document for all supported media types using the resources and tests identified in Table A.9
The job results page for a job may be retrieved in a number of different formats. The following table identifies the applicable schema document for each format and the test to be used to validate the job results for a failed job against that schema. All supported formats should be exercised.	

Table A.9 — Schema and Tests for the Job Result for a Failed Job

FORMAT	SCHEMA DOCUMENT	TEST ID
HTML	exception.yaml	/conf/html/content
JSON	exception.yaml	/conf/json/content

A.3. Conformance Class OGC Process Description

CONFORMANCE CLASS A.2	
http://www.opengis.net/spec/ogcapi-processes-1/1.0/conf/ogc-process-description	
Requirements class	Requirements Class "OGC Process Description"
Target type	Web API

ABSTRACT TEST A.48	
/conf/ogc-process-description/json-encoding	
Requirement	/req/ogc-process-description/json-encoding

ABSTRACT TEST A.48	
Test purpose	Verify that a JSON-encoded OGC Process Description complies with the required structure and contents.
Test method	1. Retrieve a description of each process according to test /conf/core/process. 2. For each process, verify the contents of the response body validate against the JSON Schema: process.yaml.

ABSTRACT TEST A.49	
/conf/ogc-process-description/inputs-def	
Requirement	/req/ogc-process-description/inputs-def
Test purpose	Verify that the definition of `inputs` for each process complies with the required structure and contents.
Test method	1. Retrieve a description of each process according to test /conf/core/process. 2. For each process, verify that the definition of the inputs conforms to the JSON Schema: inputDescription.yaml.

ABSTRACT TEST A.50	
/conf/ogc-process-description/input-def	
Requirement	/req/ogc-process-description/input-def
Test purpose	Verify that the definition of each input for each process complies with the required structure and contents.
Test method	1. For each input identified according to the test /conf/ogc-process-description/inputs-def verify that the value of the `schema` key, that defines the input, validates according to the JSON Schema: schema.yaml.

ABSTRACT TEST A.51	
/conf/ogc-process-description/input-mixed-type	
Requirement	/req/ogc-process-description/input-mixed-type
Test purpose	Validate that each input of mixed type complies with the required structure and contents.
Test method	1. Retrieve a description of each process according to test /conf/core/process.

ABSTRACT TEST A.51	
	2. For each process identify if the process has one or more inputs of mixed type. 3. For each sub-schema of each identified input, verify that the definition validates according to the JSON Schema: schema.yaml.

ABSTRACT TEST A.52	
/conf/ogc-process-description/outputs-def	
Requirement	/req/ogc-process-description/outputs-def
Test purpose	Verify that the definition of `outputs` for each process complies with the required structure and contents.
Test method	1. Retrieve a description of each process according to test /conf/core/process. 2. For each process, verify that the definition of the outputs conforms to the JSON Schema: outputDescription.yaml.

ABSTRACT TEST A.53	
/conf/ogc-process-description/output-def	
Requirement	/req/ogc-process-description/output-def
Test purpose	Verify that the definition of each output for each process complies with the required structure and contents.
Test method	1. For each output identified according to the test /conf/ogc-process-description/outputs-def verify that the value of the `schema` key, that defines the output, validates according to the JSON Schema: schema.yaml.

ABSTRACT TEST A.54	
/conf/ogc-process-description/output-mixed-type	
Requirement	/req/ogc-process-description/output-mixed-type
Test purpose	Validate that each output of mixed type complies with the required structure and contents.
Test method	1. Retrieve a description of each process according to test /conf/core/process.

ABSTRACT TEST A.54	
	2. For each process identify if the process has one or more output of mixed type denoted by the use of the `oneOf` JSON Schema keyword. 3. For each sub-schema or each identified output, verify that the definition validates according to the JSON Schema: schema.yaml.

A.4. Conformance Class JSON

CONFORMANCE CLASS A.3	
http://www.opengis.net/spec/ogcapi-processes-1/1.0/conf/json	
Requirements class	Requirements Class "Core"
Target type	Web API

ABSTRACT TEST A.55	
/conf/json/definition	
Requirement	/req/json/definition
Test purpose	Verify support for JSON.
Test method	1. A resource is requested with response media type of `application/json`. 2. All `200` responses SHALL support the following media types: `application/json` for all resources.

A.5. Conformance Class HTML

CONFORMANCE CLASS A.4	
http://www.opengis.net/spec/ogcapi-processes-1/1.0/conf/dismiss	
Requirements class	Requirements Class "Core"
Dependency	Conformance Class "Core"

CONFORMANCE CLASS A.4	
Target type	Web API

ABSTRACT TEST A.56	
/conf/html/content	
Requirement	/req/html/content
Test purpose	Verify the content of an HTML document given an input document and schema.
Test method	1. Validate that the document is an W3C HTML 5 document 2. Manually inspect the document and verify that the HTML body contains: 1. all information in the schemas of the Response Object in the HTML `<body/>` 2. all links in HTML `<a/>` elements in the HTML `<body/>`.

ABSTRACT TEST A.57	
/conf/html/definition	
Requirement	/req/html/definition
Test purpose	Verify support for HTML
Test method	1. Verify that every `200` response of every operation of the API where HTML was requested is of media type `text/html`.

A.6. Conformance Class OpenAPI 3.0

CONFORMANCE CLASS A.5	
http://www.opengis.net/spec/ogcapi-processes-1/1.0/conf/oas30	
Requirements class	Requirements Class “OpenAPI Specification 3.0”
Dependency	Conformance Class “Core”
Target type	Web API

ABSTRACT TEST A.58	
/conf/oas30/completeness	
Requirement	/req/oas30/completeness
Test purpose	Verify the completeness of an OpenAPI document.
Test method	1. Verify that for each operation, the OpenAPI document describes all HTTP Status Codes and Response Objects that the API uses in responses.

ABSTRACT TEST A.59	
/conf/oas30/exceptions-codes	
Requirement	/req/oas30/exceptions-codes
Test purpose	Verify that the OpenAPI document fully describes potential exception codes.
Test method	1. Verify that for each operation, the OpenAPI document describes all HTTP Status Codes that may be generated.

ABSTRACT TEST A.60	
/conf/oas30/oas-definition-1	
Requirement	/req/oas30/oas-definition-1
Test purpose	Verify that JSON and HTML versions of the OpenAPI document are available.
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 A.61	
/conf/oas30/oas-definition-2	
Requirement	/req/oas30/oas-definition-2
Test purpose	Verify that the OpenAPI document is valid JSON.
Test method	1. Verify that the JSON representation conforms to the Open API Specification, version 3.0.

ABSTRACT TEST A.62	
/conf/oas30/oas-impl	
Requirement	/req/oas30/oas-impl
Test purpose	Verify that all capabilities specified in the OpenAPI definition are implemented by the API.
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 OGC API — Processes standard.

ABSTRACT TEST A.63	
/conf/oas30/security	
Requirement	/req/oas30/security
Test purpose	Verify that any authentication protocols implemented by the API are documented in the OpenAPI document.
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.7. Conformance Class Job list

CONFORMANCE CLASS A.6	
http://www.opengis.net/spec/ogcapi-processes-1/1.0/conf/job-list	
Requirements class	Requirements Class “Core”
Target type	Web API

ABSTRACT TEST A.64	
/conf/job-list/job-list-op	
Requirement	/req/job-list/job-list-op

ABSTRACT TEST A.64	
Test purpose	Validate that information about jobs can be retrieved from the expected location.
Test method	1. Issue an HTTP GET request to the URL /jobs. 2. Validate the contents of the returned document using test /conf/job-list/job-list-success.

ABSTRACT TEST A.65	
/conf/job-list/type-definition	
Requirement	/req/job-list/type-definition
Test purpose	Validate that the `type` query parameter is constructed correctly.
Test method	1. Verify that the `type` query parameter complies with its definition in requirement /req/job-list/type-definition.

ABSTRACT TEST A.66	
/conf/job-list/processID-definition	
Requirement	/req/job-list/processID-definition
Test purpose	Validate that the `processID` query parameter is constructed correctly.
Test method	1. Verify that the `processID` query parameter complies with its definition in requirement /req/job-list/processID-definition.

ABSTRACT TEST A.67	
/conf/job-list/status-definition	
Requirement	/req/job-list/status-definition
Test purpose	Validate that the `status` query parameter is constructed correctly.
Test method	1. Verify that the `status` query parameter complies with its definition in requirement /req/job-list/status-definition.

ABSTRACT TEST A.68	
/conf/job-list/datetime-definition	
Requirement	/req/job-list/datetime-definition

ABSTRACT TEST A.68	
Test purpose	Validate that the `datetime` query parameter is constructed correctly.
Test method	1. Verify that the `datetime` query parameter complies with its definition in requirement /req/job-list/datetime-definition.

ABSTRACT TEST A.69	
/conf/job-list/duration-definition	
Requirement	/req/job-list/duration-definition
Test purpose	Validate that the `minDuration` and `maxDuration` query parameter are constructed correctly.
Test method	1. Verify that the `minDuration` query parameter complies with its definition in requirement /req/job-list/duration-definition, A. 2. Verify that the `maxDuration` query parameter complies with its definition in requirement /req/job-list/duration-definition, B.

ABSTRACT TEST A.70	
/conf/job-list/limit-definition	
Requirement	/req/job-list/limit-definition
Test purpose	Validate that the `limit` query parameter is constructed correctly.
Test method	1. Verify that the `limit` query parameter complies with its definition in requirement /req/job-list/limit-definition. Note that the API can define different values for “minimum”, “maximum” and “default”.

ABSTRACT TEST A.71	
/conf/job-list/job-list-success	
Requirement	/req/job-list/job-list-success
Test purpose	Validate that the job list content complies with the required structure and contents.
Test method	1. Validate that a document was returned with an HTTP status code of 200. 2. Validate the job list content for all supported media types using the resources and tests identified in Table A.10

ABSTRACT TEST A.71
A job list may be retrieved in a number of different formats. The following table identifies the applicable schema document for each format and the test to be used to validate the against that schema. All supported formats should be exercised.

Table A.10 — Schema and Tests for Job List Content

FORMAT	SCHEMA DOCUMENT	TEST ID
HTML	jobList.yaml	/conf/html/content
JSON	jobList.yaml	/conf/json/content

ABSTRACT TEST A.72	
/conf/job-list/links	
Requirement	/req/job-list/links
Test purpose	Validate that the proper links are included in a response.
Test method	1. Get a list of jobs as per test /conf/job-list/job-list-op. 2. Verify that the `links` section of the response contains a link with `rel=self`. 3. Verify that the `links` section of the response contains a link with `rel=alternate` for each response representation the service claims to support in its conformance document.

ABSTRACT TEST A.73	
/conf/job-list/type-response	
Requirement	/req/job-list/type-response
Test purpose	Validate that the `type` query parameter is processed correctly.
Test method	1. Get a list of jobs as per test /conf/job-list/job-list-op and append the `type` query parameter to the request. 2. Inspect the value of the `type` property for each job listed in the response. 3. Verify that that value of the `type` property matches one of the values specified for the `type` query parameter.

ABSTRACT TEST A.74	
/conf/job-list/processID-mandatory	
Requirement	/req/job-list/processID-mandatory
Test purpose	Validate that the `processID` property is present in every job.
Test method	1. Get a list of jobs as per test /conf/job-list/job-list-op. 2. Verify that each job includes a `processID` property.

ABSTRACT TEST A.75	
/conf/job-list/processID-response	
Requirement	/req/job-list/processID-response
Test purpose	Validate that the `processID` query parameter is processed correctly.
Test method	1. Get a list of jobs as per test /conf/job-list/job-list-op and append the `processID` parameter to the request. 2. Inspect the value of the `processID` property for each job listed in the response. 3. Verify that that value of the `processID` property matches one of the values specified for the `processID` query parameter.

ABSTRACT TEST A.76	
/conf/job-list/status-response	
Requirement	/req/job-list/status-response
Test purpose	Validate that the `status` query parameter is processed correctly.
Test method	1. Get a list of jobs as per test /conf/job-list/job-list-op and append the `status` query parameter to the request. 2. Inspect the value of the `status` property (see: statusInfo.yaml) for each job listed in the response. 3. Verify that the value of the `status` property matches one of the values specified for the `status` query parameter.

ABSTRACT TEST A.77
/conf/job-list/datetime-response

ABSTRACT TEST A.77	
Requirement	/req/job-list/datetime-response
Test purpose	Validate that the `datetime` query parameter is processed correctly.
Test method	1. Get a list of jobs as per test /conf/job-list/job-list-op and append the `datetime` query parameter to the request. 2. Inspect the value of the `created` (see: statusInfo.yaml) property for each job listed in the response. 3. Verify that the value of the `created` temporally intersects with the value specified for the `datetime` query parameter.

ABSTRACT TEST A.78	
/conf/job-list/duration-response	
Requirement	/req/job-list/duration-response
Test purpose	Validate that the `minDuration` and `maxDuration` query parameter are processed correctly.
Test method	1. Get a list of jobs as per test /conf/job-list/job-list-op and append the `minDuration` query parameter to the request. 2. Compute the duration of each job listed in the response document as per requirements /req/job-list/status-response, E or F depending on the current status of the job. 3. Verify that the computed duration of each job listed in the response is at least as long as the specified value of the `minDuration` query parameter. 4. Get a list of jobs as per test /conf/job-list/job-list-op and append the `maxDuration` query parameter to the request. 5. Compute the duration of each job listed in the response document as per requirements /req/job-list/status-response, E or F depending on the current status of the job. 6. Verify that the computed duration of each job listed in the response is no longer than the specified value of the `maxDuration` query parameter. 7. Get a list of jobs as per test /conf/job-list/job-list-op and append the `minDuration` and `maxDuration` query parameters to the request. 8. Compute the duration of each job listed in the response document as per requirements /req/job-list/status-response, E or F depending on the current status of the job.

ABSTRACT TEST A.78	
	9. Verify that the computed duration of each job listed in the response is at least as long as the specified value of the `minDuration` query parameter AND no longer than the value of the `maxDuration` query parameter.

ABSTRACT TEST A.79	
/conf/job-list/limit-response	
Requirement	/req/job-list/limit-response
Test purpose	Validate that the `limit` query parameter is processed correctly.
Test method	1. Get a list of jobs as per test /conf/job-list/job-list-op and append the `limit` query parameter to the request. 2. Count the number of jobs listed in the response. 3. Verify that this count is not greater than the value specified by the `limit` parameter. 4. If the API definition specifies a maximum value for `limit` parameter, verify that the count does not exceed this maximum value.

A.8. Conformance Class Callback

CONFORMANCE CLASS A.7	
http://www.opengis.net/spec/ogcapi-processes-1/1.0/conf/callback	
Requirements class	Requirements Class “Core”
Target type	Web API

ABSTRACT TEST A.80	
/conf/callback/job-callback	
Requirement	/req/callback/job-callback
Test purpose	Validate the passing of a subscriber-URL in an execute request.
Test method	1. Configure a URL endpoint to accept message body from the server.

ABSTRACT TEST A.80	
	2. Create an asynchronous execute request that includes the optional `subscriber` key (see execute.yaml. 3. Execute the asynchronous job using test /conf/core/job-creation-request. 4. Validate the job results are received by the specified callback URL.

A.9. Conformance Class Dismiss

CONFORMANCE CLASS A.8	
http://www.opengis.net/spec/ogcapi-processes-1/1.0/conf/dismiss	
Requirements class	Requirements Class "Core"
Target type	Web API

ABSTRACT TEST A.81	
/conf/dismiss/job-dismiss-op	
Requirement	/req/dismiss/job-dismiss-op
Test purpose	Validate that a running job can be dismissed.
Test method	1. Create an asynchronous job as per test /conf/core/job-creation-op; not the job identifier, {jobID}, assigned to the job. 2. Issue an HTTP DELETE operation to the URL '/jobs/{jobID}'. 3. Validate the contents of the returned document using test /conf/dismiss/job-dismiss-success.

ABSTRACT TEST A.82	
/conf/dismiss/job-dismiss-success	
Requirement	/req/dismiss/job-dismiss-success
Test purpose	Validate that the content returned when dismissing a job complies with the required structure and contents.

ABSTRACT TEST A.82	
Test method	1. Validate that a document was returned with an HTTP status code of 200. 2. Validate that the status is the response is set to “dismissed”. 3. Validate the process list content for all supported media types using the resources and tests identified in Table A.11
The response to dismissing a job can be presented in a number of different formats. The following table identifies the applicable schema document for each format and the test to be used to validate the against that schema. All supported formats should be exercised.	

Table A.11 — Schema and Tests for Dismissing a Job

FORMAT	SCHEMA DOCUMENT	TEST ID
HTML	statusInfo.yaml	/conf/html/content
JSON	statusInfo.yaml	/conf/json/content

B

ANNEX B (INFORMATIVE) REVISION HISTORY

B

ANNEX B (INFORMATIVE) 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 indentation
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
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
2020-05-05	1.0-draft.3	Gérald Fenoy	12	Add Requirements Class 'Dismiss', fix includes and section headers

DATE	RELEASE	EDITOR	PRIMARY CLAUSES MODIFIED	DESCRIPTION
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.
2020-07-21	1.0-draft.4	Benjamin Pross	2,6,10, Annex A	Editorial fixes, incorporated comments from Carl Reed, updated example
2020-07-23	1.0-draft.4	Benjamin Pross	7,10,11	Add dependency to API Common
2020-07-27	1.0-draft.4	Benjamin Pross	9	Add security considerations section
2020-07-30	1.0-draft.4	Benjamin Pross	7,9	Add section about HTTP and HTTPS, fix links to RFCs, add additional guidance to security considerations section
2020-08-10	1.0-draft.4	Panagiotis (Peter) A. Vretanos	all	Add ATS, adjust links throughout the document
2020-08-13	1.0-draft.4	Benjamin Pross	9	Work on security considerations section
2020-09-02	1.0-draft.4	Benjamin Pross	9	Incorporated further comments from Andreas Matheus
2020-10-08	1.0-draft.5	Benjamin Pross	All	Tag version 1.0-draft.4, continue work on version 1.0-draft.5
2020-10-22	1.0-draft.5	Benjamin Pross	Annex A	Continued to rename collection to list
2020-11-02	1.0-draft.5	Benjamin Pross	7	Fix issue #100
2020-11-13	1.0-draft.5	Benjamin Pross	7	Fix issue #103
2021-01-15	1.0-draft.5	Benjamin Pross	7, 12	Move /jobs endpoint to root level, changes in execute and result schema

DATE	RELEASE	EDITOR	PRIMARY CLAUSES MODIFIED	DESCRIPTION
2021-01-19	1.0-draft.6	Benjamin Pross	-	Set version to 1.0-draft.6-SNAPSHOT
2021-01-19	1.0-draft.6	Benjamin Pross	7	Adjust example paths
2021-01-19	1.0-draft.6	Benjamin Pross	7	Part B.x
2021-01-25	1.0-draft.6	Benjamin Pross	7	Fix issue 3
2021-01-25	1.0-draft.6	Benjamin Pross	7	Adjust links and replace WPS 2.0 SWG with OGC API — Processes SWG
2021-01-25	1.0-draft.6	Benjamin Pross	7	Fix CNR3
2021-01-25	1.0-draft.6	Benjamin Pross	7	CNR13
2021-01-25	1.0-draft.6	Benjamin Pross	7	CNR19
2021-01-25	1.0-draft.6	Benjamin Pross	7	CNR21
2021-01-25	1.0-draft.6	Benjamin Pross	7	CNR23
2021-01-25	1.0-draft.6	Benjamin Pross	7	CNR24
2021-02-01	1.0-draft.6	Benjamin Pross	7	Fixes #87
2021-02-01	1.0-draft.6	Benjamin Pross	7	Fixes #118
2021-02-02	1.0-draft.6	Benjamin Pross	7	Adjust text for additional api building blocks
2021-02-02	1.0-draft.6	Benjamin Pross	7	CNR9
2021-02-02	1.0-draft.6	Benjamin Pross	7	Replace term Web Processing Service in core
2021-02-09	1.0-draft.6	Benjamin Pross	7	CNR7, CNR14
2021-02-09	1.0-draft.6	Benjamin Pross	7	CNR8
2021-02-09	1.0-draft.6	Benjamin Pross	7	CNR25

DATE	RELEASE	EDITOR	PRIMARY CLAUSES MODIFIED	DESCRIPTION
2021-02-09	1.0-draft.6	Benjamin Pross	7	CNR20]
2021-02-09	1.0-draft.6	Benjamin Pross	7	CNR26
2021-02-22	1.0-draft.6	Benjamin Pross	7	Editorial fixes
2021-02-22	1.0-draft.6	Benjamin Pross	7	Fixes #130
2021-03-01	1.0-draft.6	Benjamin Pross	7	Adjust texts to moved execute endpoint
2021-03-08	1.0-draft.6	Gérald Fenoy	10	Fix old syntaxes in JobList example used from the file: clause_10_job_list.adoc
2021-03-08	1.0-draft.6	Panagiotis (Peter) A. Vretanos	X	Modify process description to allow JSON-Schema to be used to describe inputs and outputs. As a result of this change, a lot of the current structures, boundingBoxData, complex Data, literalData, etc. can all be removed since these can be adequately described using JSON-Schema.
2021-03-11	1.0-draft.6	Benjamin Pross	X	Fix issue #143
2021-03-11	1.0-draft.6	Benjamin Pross	X	Fix links
2021-03-11	1.0-draft.6	Benjamin Pross	X	Fixes #148
2021-03-11	1.0-draft.6	Benjamin Pross	X	Fix #145
2021-03-17	1.0-draft.6	Panagiotis (Peter) A. Vretanos	X	Refine the use of JSON Schema to describe input and output process parameters.
2021-03-17	1.0-draft.6	Panagiotis (Peter) A. Vretanos	X	Update input/output description schema to convert the `inputs` and `outputs` keys in the process description from arrays to objects. Each key in the updated `inputs/outputs` object is the identified for the corresponding process input/output.
2021-03-19	1.0-draft.6	Panagiotis (Peter) A. Vretanos	X	Merge pull request #6 from opengeospatial/master

DATE	RELEASE	EDITOR	PRIMARY CLAUSES MODIFIED	DESCRIPTION
2021-03-24	1.0-draft.6	Benjamin Pross	-	Update UML
2021-03-24	1.0-draft.6	Benjamin Pross	-	Add eap and xmi files
2021-03-28	1.0-draft.6	Panagiotis (Peter) A. Vretanos	X	Remove the ability to infinitely nest inputs.
2021-03-29	1.0-draft.6	Panagiotis (Peter) A. Vretanos	X	* Remove unnecessary schemas that can now be defined using JSON Schema and propagate those changes to the other schemas. * Update some of the indentation in the yaml files so the yamllint does not complain. * Further refine the examples. * Update the text of the specification accordingly.
2021-03-29	1.0-draft.6	Panagiotis (Peter) A. Vretanos	X	Move additionalProperties from output.yaml to execute.yaml to be consistent with what was done with input.yaml.
2021-03-29	1.0-draft.6	Panagiotis (Peter) A. Vretanos	X	Allow simple values to be encoded directly. So, `"key": {"value":10}` becomes `"key": 10`.
2021-03-29	1.0-draft.6	Panagiotis (Peter) A. Vretanos	X	Add array, in additiona to string, number & boolean, to possible direct input types.
2021-04-09	1.0-draft.6	Panagiotis (Peter) A. Vretanos	X	Update example to use new, more compact form for specifying simple scalar values.
2021-04-09	1.0-draft.6	Panagiotis (Peter) A. Vretanos	X	1. Make mediateType optional 2. Modify the schema tag to be a reference to a schema or be an inline JSON schema. 3. Change name of “encoding” tag to “characterEncoding” to make more clear what it means.
2021-04-09	1.0-draft.6	Panagiotis (Peter) A. Vretanos	X	Add missing input type array.
2021-04-09	1.0-draft.6	Panagiotis (Peter) A. Vretanos	X	Patch merge inconsistency between issues #122, #152 and #155.
2021-04-09	1.0-draft.6	Panagiotis (Peter) A. Vretanos	X	Fix some spacing issues with the yaml files.

DATE	RELEASE	EDITOR	PRIMARY CLAUSES MODIFIED	DESCRIPTION
2021-04-09	1.0-draft.6	Panagiotis (Peter) A. Vretanos	X	Patch dangling reference in result.yaml.
2021-04-12	1.0-draft.6	Benjamin Pross	X	This should fix #142
2021-04-12	1.0-draft.6	Benjamin Pross	X	Use upper case in bullet point list
2021-04-12	1.0-draft.6	Benjamin Pross	X	Add new requirement for inputs, this should fix #129
2021-04-12	1.0-draft.6	Benjamin Pross	X	Remove id from execute JSON schema
2021-04-12	1.0-draft.6	Benjamin Pross	X	Adjust requirement to new execute endpoint
2021-04-12	1.0-draft.6	Benjamin Pross	X	Adjust examples
2021-04-12	1.0-draft.6	Benjamin Pross	X	Adjust execute endpoint in ATS
2021-04-12	1.0-draft.6	Benjamin Pross	X	Add recommendation regarding access control for the /jobs endpoint
2021-04-13	1.0-draft.6	Gérald Fenoy	X	Update execute.yaml
2021-04-13	1.0-draft.6	Gérald Fenoy	X	Update format.yaml
2021-04-13	1.0-draft.6	Gérald Fenoy	X	Create referenceData.yaml
2021-04-13	1.0-draft.6	Panagiotis (Peter) A. Vretanos	X	Patch JSON schema fragments in some of the example inputs. All add a units of measure input example.
2021-04-13	1.0-draft.6	Panagiotis (Peter) A. Vretanos	X	A review after the merge of #122, #152 and #155 revealed an inconsistency in the input definition. Specifically the merge overwrote the change that allow direct input values (i. e. “key”: “value”). This commit fixes these inconsistencies.
2021-04-13	1.0-draft.6	Panagiotis (Peter) A. Vretanos	X	Remove include path fragment that appears in clause 7. For some reason it was commented out. I uncommented it and clean up the format of the permission.

DATE	RELEASE	EDITOR	PRIMARY CLAUSES MODIFIED	DESCRIPTION
2021-04-13	1.0-draft.6	Panagiotis (Peter) A. Vretanos	X	Add some additional requirements around process inputs. Specifically an input can be specified inline or by reference. It it is specified inline than it shall conform to its schema in the process description. If by reference then a link. yaml link shall be used.
2021-04-14	1.0-draft.6	Panagiotis (Peter) A. Vretanos	X	Add requirements for input cardinality and for inlining or referencing input values.
2021-04-14	1.0-draft.6	Panagiotis (Peter) A. Vretanos	X	Add the schema for a standard bbox definition that process descriptions can reference. This was everyone can uses the same bbox definition.
2021-04-14	1.0-draft.6	Panagiotis (Peter) A. Vretanos	X	Update the bbox schema to enforce either 4 or 6 items (i.e. 5 is not allowed).
2021-04-14	1.0-draft.6	Panagiotis (Peter) A. Vretanos	X	Add a description indicating how this file can be used.
2021-04-15	1.0-draft.6	Benjamin Pross	X	Remove unnecessary oneOf
2021-04-15	1.0-draft.6	Benjamin Pross	X	Remove dash
2021-04-15	1.0-draft.6	Benjamin Pross	X	Use additionalProperties instead of pattern Properties
2021-04-15	1.0-draft.6	Panagiotis (Peter) A. Vretanos	X	Remove observedProperty as per SWG resolution of 29MAR2021. The observed Property is useful for certain domains but seems out of scope for the core.
2021-04-15	1.0-draft.6	Panagiotis (Peter) A. Vretanos	X	Remove file that does not seem to be referenced anywhere.
2021-04-15	1.0-draft.6	Panagiotis (Peter) A. Vretanos	X	Patch reference to input and output descriptions.
2021-04-15	1.0-draft.6	Panagiotis (Peter) A. Vretanos	X	Add an additional requirement that if a value is specified by reference then its value type must match the type or types specified in the process description. I suppose that an allOf could be used to constrain the type property of the link but that seem a bit heavy.

DATE	RELEASE	EDITOR	PRIMARY CLAUSES MODIFIED	DESCRIPTION
2021-04-15	1.0-draft.6	Panagiotis (Peter) A. Vretanos	X	Rename the file name of the ATS so that it matched the requirement file name.
2021-04-15	1.0-draft.6	Panagiotis (Peter) A. Vretanos	X	Update the description example.
2021-04-15	1.0-draft.6	Panagiotis (Peter) A. Vretanos	X	Patch the $ref.
2021-04-15	1.0-draft.6	Panagiotis (Peter) A. Vretanos	X	Clarify the language of the requirement a bit (I think).
2021-04-15	1.0-draft.6	Panagiotis (Peter) A. Vretanos	X	Split the /req/core/job-creation-input-cardinality requirement into two requirements to make it easier to test in the ATS.
2021-04-15	1.0-draft.6	Panagiotis (Peter) A. Vretanos	X	Add tests for input cardinality handing.
2021-04-16	1.0-draft.6	Panagiotis (Peter) A. Vretanos	X	Clarify the text of the requirements and the ATS about input multiplicity (i.e. issue #129).
2021-04-16	1.0-draft.6	Panagiotis (Peter) A. Vretanos	X	Remove obsolete note.
2021-04-16	1.0-draft.6	Panagiotis (Peter) A. Vretanos	X	Fix formatting.
2021-04-19	1.0-draft.6	Benjamin Pross	X	Add requirement and recommendation for testing. Should fix #157
2021-04-19	1.0-draft.6	Benjamin Pross	X	Adjust wording
2021-04-19	1.0-draft.6	Panagiotis (Peter) A. Vretanos	X	- Get rid on minOccurs/maxOccurs and rely instead on JSON Schema structures to define the cardinality of a process input. — The schema object in the process description is too generic so add three levels of JSON Schema conformance ranging from very simple to full JSON schema.
2021-04-19	1.0-draft.6	Panagiotis (Peter) A. Vretanos	X	Patch small $ref issues.

DATE	RELEASE	EDITOR	PRIMARY CLAUSES MODIFIED	DESCRIPTION
2021-04-20	1.0-draft.6	Benjamin Pross	X	Adjust path of execution endpoint
2021-04-20	1.0-draft.6	Benjamin Pross	X	Remove unused schema, fixes #173
2021-04-20	1.0-draft.6	Benjamin Pross	X	Remove link to execute endpoint from landing page
2021-04-20	1.0-draft.6	Benjamin Pross	X	Add recommendation to add link to job monitoring endpoint to the landing page
2021-04-25	1.0-draft.6	Panagiotis (Peter) A. Vretanos	X	Remove the patternProperties key that allow JSON Schema extensions keys that begin with “x-”. Two point about this extension mechanism… (1) it breaks compatablity with swagger which is bad; (2) I can’t really think of a good reason right now that we would want to extend the syntax of JSON Schema using this mechanism and so I think removing it is OK.
2021-04-26	1.0-draft.6	Panagiotis (Peter) A. Vretanos	X	- Update ATS to handle JSON Schema compliance levels. — Update examples to add a multi-type feature collection input. — Add a general inline value structure (qualified Value.yaml) that allows selection of a specified input type of a multi-type input.
2021-04-26	1.0-draft.6	Panagiotis (Peter) A. Vretanos	X	Remove duplicate facet definitions.
2021-04-29	1.0-draft.6	Panagiotis (Peter) A. Vretanos	X	Remove the various schema levels and only support the full OpenApi 3.0 compatible version of JSON Schema (formerly called schemaLevel3.yaml).
2021-05-03	1.0-draft.6	Benjamin Pross	X	Merge pull request #172 from pvretano/issue-170
2021-05-03	1.0-draft.6	Panagiotis (Peter) A. Vretanos	X	Patch invalid references the schemaFull.yaml/ schemaLevel3.yaml. All should be references to schema.yaml.
2021-05-05	1.0-draft.6	Gérald Fenoy	X	Fix typo
2021-05-05	1.0-draft.6	Gérald Fenoy	X	Use relative urls.
2021-05-05	1.0-draft.6	Gérald Fenoy	X	Ue correct reference for bbox

DATE	RELEASE	EDITOR	PRIMARY CLAUSES MODIFIED	DESCRIPTION
2021-05-05	1.0-draft.6	Gérald Fenoy	X	Fix typo
2021-05-05	1.0-draft.6	Gérald Fenoy	X	Few typo
2021-05-05	1.0-draft.6	Gérald Fenoy	X	Fix typo
2021-05-06	1.0-draft.6	Gérald Fenoy	X	Remove link.yaml references when schema.yaml is already referenced.
2021-05-06	1.0-draft.6	Gérald Fenoy	X	Get back enum items, default and, example.
2021-05-06	1.0-draft.6	Gérald Fenoy	X	Keep only items.
2021-05-06	1.0-draft.6	Gérald Fenoy	X	Fix typo
2021-05-06	1.0-draft.6	Gérald Fenoy	X	Go bak
2021-05-06	1.0-draft.6	Ubuntu	X	Make Swagger-UI working again and the api able to validate.
2021-05-10	1.0-draft.6	Panagiotis (Peter) A. Vretanos	X	Remove the concept of Level 0,1,2,3 JSON schema and simply use what was called Level 3 which is the full JSON Schema.
2021-05-10	1.0-draft.6	Panagiotis (Peter) A. Vretanos	X	Make the mode on execute options with the default being specified in the process description.
2021-05-10	1.0-draft.6	Ubuntu	X	Remove uneeded yaml file.
2021-05-11	1.0-draft.6	Ubuntu	X	Get the not, allOf, oneOf, anyOf, items and contentSchema available in the meta-schema.
2021-05-11	1.0-draft.6	Gérald Fenoy	X	Reset example despite warnings messages.
2021-05-11	1.0-draft.6	Gérald Fenoy	X	Reset additionalProperties in schema.yaml
2021-05-11	1.0-draft.6	Gérald Fenoy	X	Fix indentation
2021-05-11	1.0-draft.6	Gérald Fenoy	X	Reset properties/additionalProperties
2021-05-11	1.0-draft.6	Gérald Fenoy	X	Remove schema.yaml references from schema.yaml

DATE	RELEASE	EDITOR	PRIMARY CLAUSES MODIFIED	DESCRIPTION
2021-05-11	1.0-draft.6	Panagiotis (Peter) A. Vretanos	X	Add optional date-time fields that track milestones in the lifecycle of a job.
2021-05-11	1.0-draft.6	Panagiotis (Peter) A. Vretanos	X	Fix small inconsistencies in the sequence diagrams.
2021-05-11	1.0-draft.6	Panagiotis (Peter) A. Vretanos	X	Make sure result/results is consistently applied everywhere. The schemas and the resource endpoints should be 'results' (plural).
2021-05-12	1.0-draft.6	Gérald Fenoy	X	Fix typo in example definition for Process Description
2021-05-12	1.0-draft.6	Panagiotis (Peter) A. Vretanos	X	Clarify some requirements that where flagged as ambiguous in issue 178.
2021-05-13	1.0-draft.6	Gérald Fenoy	X	Add schema_swagger.yaml for a minimal schema definition to be used from swagger-ui and schema.yaml for the full featured schema.
2021-05-14	1.0-draft.6	Gérald Fenoy	X	Add swagger relevant files for giving the opportunity to use the schema_swagger.yaml finally and be able to using your API from swagger-ui
2021-05-17	1.0-draft.6	Panagiotis (Peter) A. Vretanos	X	Remove the ambiguity introduced by allowing process input values to be any object type. If the process input schema is similar to one of the builtin schemas (link.yaml, qualified Value.yaml, etc.) a server may not be able ti disambiguate the input intent.
2021-05-18	1.0-draft.6	Panagiotis (Peter) A. Vretanos	X	Update Execute.json
2021-05-18	1.0-draft.6	Panagiotis (Peter) A. Vretanos	X	Update Result.json
2021-05-19	1.0-draft.6	Panagiotis (Peter) A. Vretanos	X	Clarify the behavior for all the combinations of mode/response/transmissionMode/# of outputs.
2021-05-19	1.0-draft.6	Panagiotis (Peter) A. Vretanos	Annex A	Align ATS with all the changes made for issue #178.

DATE	RELEASE	EDITOR	PRIMARY CLAUSES MODIFIED	DESCRIPTION
2021-05-20	1.0-draft.6	Panagiotis (Peter) A. Vretanos	7	Update clause_7_core.adoc
2021-05-22	1.0-draft.6	Panagiotis (Peter) A. Vretanos	X	Change the job status “completed” to “successful”. The job status “completed” is not a value status.
2021-05-22	1.0-draft.6	Panagiotis (Peter) A. Vretanos	X	Change the job status “completed” to “successful”. The job status “completed” is not a valid job status.
2021-05-22	1.0-draft.6	Panagiotis (Peter) A. Vretanos	X	Update exception reporting to align with common which uses RFC 7807.
2021-05-22	1.0-draft.6	Panagiotis (Peter) A. Vretanos	X	Add OpenAPI example. I following the pattern used in OGG API Features for the example OpenAPI files found there.
2021-05-22	1.0-draft.6	Panagiotis (Peter) A. Vretanos	X	Simplify the response tables, for sync and async execution, by collapsing similarly responding paths into fewer rows.
2021-05-22	1.0-draft.6	Panagiotis (Peter) A. Vretanos	X	Update the exception status codes referenced in the ATS to be the URIs defined as a result of RFC 7807.
2021-05-22	1.0-draft.6	Panagiotis (Peter) A. Vretanos	X	Clarify that server must implement support for both in-line process input values and process input values specified by reference.
2021-05-22	1.0-draft.6	Panagiotis (Peter) A. Vretanos	X	Add abstract tests for verifying that a server can handle inputs by value and by reference.
2021-05-25	1.0-draft.6	Panagiotis (Peter) A. Vretanos	X	Refactor the schemas execute.yaml, inline OrRefData.yaml and qualifiedValue.yaml to better emphasize the validation relationship between the definition of a process input in the process description and an process input value in an execute request. This, of course, cascaded into a whole bunch of other related clarifications.
2021-05-25	1.0-draft.6	Panagiotis (Peter) A. Vretanos	X	Lint all the yaml and json files.
2021-05-26	1.0-draft.6	Benjamin Pross	X	Add Panagiotis (Peter) A. Vretanos as editor

DATE	RELEASE	EDITOR	PRIMARY CLAUSES MODIFIED	DESCRIPTION
2021-05-26	1.0-draft.6	Panagiotis (Peter) A. Vretanos	X	Move bbox.yaml from inlineOrRefData.yaml to inputValue.yaml so that it is also a validation target.
2021-05-26	1.0-draft.6	Panagiotis (Peter) A. Vretanos	X	The intent was to add bbox.yaml to inputValue NoObject.yaml but not inputValue.yaml.
2021-05-26	1.0-draft.6	Panagiotis (Peter) A. Vretanos	X	Remove references to the now obsolete Level 0, Level 1, etc. schema conformance classes.
2021-05-28	1.0-draft.6	Panagiotis (Peter) A. Vretanos	X	Remove the mode parameter and instead rely on the HTTP Prefer header and defined default execution mode behavior.
2021-05-28	1.0-draft.6	Panagiotis (Peter) A. Vretanos	X	Add a recommendation to included the Preference-Applied header in the response if the request was accompanied with the HTTP Prefer header.
2021-06-02	1.0-draft.6	Jerome St-Louis	i. Abstract	Fixed mismatched sections in i. Abstract
2021-06-03	1.0-draft.6	Gérald Fenoy	X	Initial integration of files require for use with swagger-ui
2021-06-03	1.0-draft.6	Gérald Fenoy	X	Fix path for reference.yaml file
2021-06-03	1.0-draft.6	Gérald Fenoy	X	Fix typos in process and exception. Try fixing the example ProcessDescription.
2021-06-03	1.0-draft.6	Gérald Fenoy	X	Replace tabs with spaces, fix URLs for geometryGeoJSON schema which is available in yaml, add nullable and remove
2021-06-03	1.0-draft.6	Gérald Fenoy	X	Replace tabs with spaces.
2021-06-03	1.0-draft.6	Gérald Fenoy	X	Move ref to binaryInputValue.yaml from inline OrRefData.yaml to inputValueNoObject.yaml
2021-06-03	1.0-draft.6	Gérald Fenoy	X	Fix use of externalValue
2021-06-03	1.0-draft.6	Gérald Fenoy	X	Fix 2 use of externalValue
2021-06-03	1.0-draft.6	Gérald Fenoy	X	Try fixing issue with example Process Description
2021-06-03	1.0-draft.6	Gérald Fenoy	X	Fix issue with binaryInputValue.yaml

DATE	RELEASE	EDITOR	PRIMARY CLAUSES MODIFIED	DESCRIPTION
2021-06-03	1.0-draft.6	Gérald Fenoy	X	General fix in ogcapi-process-1.yaml. Fix responses/Results to use relative path.
2021-06-03	1.0-draft.6	Gérald Fenoy	X	Small fix in path.
2021-06-03	1.0-draft.6	Gérald Fenoy	X	Try fixing issue with ProcessDescription example
2021-06-03	1.0-draft.6	Gérald Fenoy	X	Try fixing issue with ProcessDescription example using allOf for value
2021-06-03	1.0-draft.6	Gérald Fenoy	X	Try fixing issue with ProcessDescription example using basic object and a ref
2021-06-03	1.0-draft.6	Gérald Fenoy	X	Fix the ProcessDescription example issue by using externalValue
2021-06-03	1.0-draft.6	Gérald Fenoy	X	Add the Preference-Applied header information's.
2021-06-08	1.0-draft.6	Panagiotis (Peter) A. Vretanos	X	Path invalid reference to component file.
2021-06-08	1.0-draft.6	Panagiotis (Peter) A. Vretanos	X	Patch type that is preventing swagger validation of example OpenAPI file.
2021-06-08	1.0-draft.6	Panagiotis (Peter) A. Vretanos	X	(1) Remove the consolidated building blocks YAML file. (2) Update the example OpenAPI definition file to reference each component individually from its corresponding schema file instead of referencing the component from the now-deleted building blocks YAML file.
2021-06-09	1.0-draft.6	Steve McDaniel	X	Indentation issue in process.yaml, outputs should be at the same level as inputs
2021-06-10	1.0-draft.6	Panagiotis (Peter) A. Vretanos	7	Minor typo.
2021-06-10	1.0-draft.6	Panagiotis (Peter) A. Vretanos	X	Add missing default value for `response` parameter. Should be `raw`.
2021-06-10	1.0-draft.6	Panagiotis (Peter) A. Vretanos	X	Add an informative statement about the default value for the `response` parameters. This is normatively defined in the schema.
2021-06-10	1.0-draft.6	Panagiotis (Peter) A. Vretanos	X	Update server URL to point to the correct endpoint.

DATE	RELEASE	EDITOR	PRIMARY CLAUSES MODIFIED	DESCRIPTION
2021-06-10	1.0-draft.6	Panagiotis (Peter) A. Vretanos	X	Fix invalid reference to transmissionMode=ref. Should be reference.
2021-06-10	1.0-draft.6	Panagiotis (Peter) A. Vretanos	X	Make explicit the fact that omitting the “outputs” parameter in an execute request means that all defined outputs are being requested.
2021-06-10	1.0-draft.6	Panagiotis (Peter) A. Vretanos	X	Remove file to conform to ATS file name pattern.
2021-06-10	1.0-draft.6	Panagiotis (Peter) A. Vretanos	X	Update all OAPIR-specific link relations to use the pattern http://www.opengis.net/def/rel/ogc/1.0/{rel}. Eventually there will be registered with the OGC-NA.
2021-06-10	1.0-draft.6	Panagiotis (Peter) A. Vretanos	X	Remove unused link relation.
2021-06-11	1.0-draft.6	Jerome St-Louis	X	results.yaml: Removed array (#219)
2021-06-13	1.0-draft.6	Panagiotis (Peter) A. Vretanos	X	Add a light-weight query capability to the jobs list. Add paging to the jobs list. Add paging to the process list.
2021-06-13	1.0-draft.6	Panagiotis (Peter) A. Vretanos	X	Add requirements and abstract tests to handle the case where the negotiated execution mode is sync or async, the response mode is raw, more than one output is requested and a mix of transmission modes (value or reference) are requested.
2021-06-13	1.0-draft.6	Panagiotis (Peter) A. Vretanos	X	Add requirements and abstract tests to handle the case where the negotiated execution mode is sync or async, the requested response is raw, more that 1 output is requested and a mix of transmission modes (value or reference) are requested.
2021-06-14	1.0-draft.6	Panagiotis (Peter) A. Vretanos	X	Change “processList” to “processes” and “jobs List” to “jobs” so that the key name matches the resource endpoint name.
2021-06-15	1.0-draft.6	Panagiotis (Peter) A. Vretanos	X	Remove default value for job control options. In the OGC process description the supported execution modes must be explicitly listed so there is no need for a default.

DATE	RELEASE	EDITOR	PRIMARY CLAUSES MODIFIED	DESCRIPTION
2021-06-15	1.0-draft.6	Panagiotis (Peter) A. Vretanos	X	Add the contentMediaType facet to the Geo JSON feature collection inputs/outputs. Although this is not strictly necessary is makes parsing and interpreting the input/output easier.
2021-06-16	1.0-draft.6	Panagiotis (Peter) A. Vretanos	X	Rel value should be 'job-list' not 'jobs-list'.
2021-06-17	1.0-draft.6	Panagiotis (Peter) A. Vretanos	X	Extend the list of "format" values to provide semantic hints inputs and outputs.
2021-06-17	1.0-draft.6	Panagiotis (Peter) A. Vretanos	X	Reword requirement for clarity.
2021-06-17	1.0-draft.6	Panagiotis (Peter) A. Vretanos	X	Fix missing allOf[] in one of the examples outputs.
2021-06-18	1.0-draft.6	Panagiotis (Peter) A. Vretanos	X	Oopsie! Forgot to make processID mandatory if the server supports the Job List conformance class.
2021-06-18	1.0-draft.7	Benjamin Pross	X	Adjust version
2021-06-22	1.0-draft.7	Panagiotis (Peter) A. Vretanos	X	Housekeeping.
2021-06-28	1.0-draft.7	Benjamin Pross	X	Merge pull request #235 from pvretano/housekeeping
2021-07-05	1.0-draft.7	Benjamin Pross	X	Revert "Adjust version"
2021-07-21	1.0-draft.7	Gérald Fenoy	X	Small fix about parameters
2021-07-21	1.0-draft.7	Gérald Fenoy	X	Fix title headers
2021-07-21	1.0-draft.7	Gérald Fenoy	-	Update clause_0_front_material.adoc
2021-07-28	1.0-draft.7	Benjamin Pross	12	Fix issue with empty chapter 12
2021-08-06	1.0-draft.7	Gérald Fenoy	X	Add enum to status.yaml

DATE	RELEASE	EDITOR	PRIMARY CLAUSES MODIFIED	DESCRIPTION
2021-08-06	1.0-draft.7	Gérald Fenoy	X	Add statuses.yaml in schema
2021-08-06	1.0-draft.7	Gérald Fenoy	X	Delete statuses.yaml
2021-08-06	1.0-draft.7	Gérald Fenoy	X	Create status.yaml
2021-08-06	1.0-draft.7	Gérald Fenoy	X	Add status.yaml
2021-08-06	1.0-draft.7	Gérald Fenoy	X	Try using status.yaml reference
2021-08-06	1.0-draft.7	Gérald Fenoy	X	Revert changes
2021-08-09	1.0-draft.7	Gérald Fenoy	X	Update status.yaml
2021-08-09	1.0-draft.7	Gérald Fenoy	X	Update statusInfo.yaml
2021-08-09	1.0-draft.7	Gérald Fenoy	X	Rename status.yaml to statusCode.yaml
2021-08-09	1.0-draft.7	Gérald Fenoy	X	Update status.yaml
2021-08-09	1.0-draft.7	Gérald Fenoy	X	Update statusInfo.yaml
2021-08-09	1.0-draft.7	Gérald Fenoy	X	Rename processId.yaml and processid.yaml to processIdPathParam.yaml and processIdQueryParam.yaml respectively
2021-08-09	1.0-draft.7	Gérald Fenoy	X	Add missing parameters to OpenAPI example
2021-08-19	1.0-draft.7	Gérald Fenoy	X	Add process value
2021-08-23	1.0-draft.7	Gérald Fenoy	X	Set format to date-time for more clarity
2021-08-23	1.0-draft.7	Gérald Fenoy	X	Update datetime.yaml
2021-08-24	1.0-draft.7	Benjamin Pross	X	Use HTTP GET method (instead of operation) throughout the document
2021-08-25	1.0-draft.7	Benjamin Pross	X	Add informative texts

DATE	RELEASE	EDITOR	PRIMARY CLAUSES MODIFIED	DESCRIPTION
2021-08-25	1.0-draft.7	Benjamin Pross	X	Merge branch ‘comments-emmanuel-devys’ into comments-amy-youmans
2021-08-25	1.0-draft.7	Benjamin Pross	X	Fix ordering of requirements
2021-08-26	1.0-draft.7	Benjamin Pross	7	Revert changes — replace GET method with GET operation
2021-09-03	1.0-draft.7	Benjamin Pross	7	Add informative text about execution paths

BIBLIOGRAPHY

BIBLIOGRAPHY

1. Panagiotis (Peter) A. Vretanos: OGC 09-025r2, *OGC® Web Feature Service 2.0 Interface Standard – With Corrigendum*. Open Geospatial Consortium (2014). http://docs.opengeospatial.org/is/09-025r2/09-025r2.html

2. Andreas Matheus: OGC 15-022, *OGC® Testbed 11 Engineering Report: Implementing Common Security Across the OGC Suite of Service Standards*. Open Geospatial Consortium (2015). https://portal.ogc.org/files/?artifact_id=63312

3. Hutton, B.: JSON Schema: A Media Type for Describing JSON Documents, https://json-schema.org/draft/2020-12/json-schema-core.html

4. Hutton, B.: JSON Schema Validation: A Vocabulary for Structural Validation of JSON, https://json-schema.org/draft/2020-12/json-schema-validation.html

5. https://www.w3.org/TR/html5/

6. Open API Initiative. OpenAPI Specification 3.0.2. Available at: https://github.com/OAI/OpenAPI-Specification/blob/master/versions/3.0.2.md.