OGC® DOCUMENT: 18-062R3

External identifier of this OGC® document: http://www.opengis.net/doc/IS/ogcapi-processes-1/2.0

Open Geospatial Consortium

OGC API - PROCESSES - PART 1: CORE

STANDARD
Implementation

DRAFT

Version: 2.0
Submission Date: 2022-01-01
Approval Date: 2022-01-01
Publication Date: 2024-01-01
Editor: Benjamin Pross, Panagiotis (Peter) A. Vretanos

CONTENTS

LIST OF TABLES

LIST OF FIGURES

LIST OF RECOMMENDATIONS

I ABSTRACT

This document defines the OGC API — Processes Standard. This standard builds on the OGC Web Processing Service (WPS) 2.0 Standard and defines the processing interface to communicate over a RESTful protocol using JSON encodings.

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 OGC Web Processing Service (WPS) Interface Standard provides a standard interface that simplifies the task of making simple or complex computational geospatial processing services accessible via web services. Such services include well-known processes found in GIS software as well as specialized processes for spatiotemporal modeling and simulation. While the OGC WPS standard was designed with spatial processing in mind, the standard could also be used to readily insert non-spatial processing tasks into a web services environment. The WPS standard provides a robust, interoperable, and versatile protocol for process execution on web services. WPS supports both immediate processing for computational tasks that take little time and asynchronous processing for more complex and time-consuming tasks. Moreover, the WPS standard defines a general process model that is designed to provide an interoperable description of processing functions. WPS is intended to support process cataloguing and discovery in a distributed environment.

This API is a newer and more modern way of programming and interacting with resources over the web while allowing better integration into existing software packages.

The resources that are provided by a server implementing the OGC API — Processes 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.

This following table provide an overview of resources, applicable HTTP methods and links to the related document sections.

Table 1 — Requirements class `Core`

RESOURCE	PATH	HTTP METHOD	PARAMETI	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/{processID}	GET	processID (in path)	Clause 7.10
Process execution	/processes/{processID}/execution	POST	processID (in path),	Clause 7.11

RESOURCE	PATH	HTTP METHOD	PARAMETI	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 7.13.2
Result	`/jobs/{jobID}/results/{outputID}`	GET	jobID (in path), outputID (in path)	Clause 7.13.1

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`

RESOURCE	PATH	HTTP METHOD	PARAMETI	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`

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

II KEYWORDS

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

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

III PREFACE

The OGC API — Processes Standard defines how a client application can request the execution of a process, how the inputs to that process can be provided, and how the output from the process is handled. The specification allows for the wrapping of computational tasks into an executable process that can 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.

IV 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

IV.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 METHOD	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 IV.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 METHOD	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.

IV.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 METHOD	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	
STATEMENT	Servers implementing the conformance class 'Job List' SHOULD have an access control in place for the /jobs endpoint to prevent misuse of job-ids.

IV.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.

V 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

VI SUBMITTERS

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

Table — Table of 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). https://docs.ogc.org/is/14-065/14-065r0.html.

T. Dierks, C. Allen: IETF RFC 2246, *The TLS Protocol Version 1.0*. RFC Publisher (1999). https://www.rfc-editor.org/info/rfc2246.

R. Fielding, J. Gettys, J. Mogul, H. Frystyk, L. Masinter, P. Leach, T. Berners-Lee: IETF RFC 2616, *Hypertext Transfer Protocol — HTTP/1.1*. RFC Publisher (1999). https://www.rfc-editor.org/info/rfc2616.

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*. RFC Publisher (1999). https://www.rfc-editor.org/info/rfc2617.

E. Levinson: IETF RFC 2387, *The MIME Multipart/Related Content-type*. RFC Publisher (1998). https://www.rfc-editor.org/info/rfc2387.

E. Rescorla: IETF RFC 2818, *HTTP Over TLS*. RFC Publisher (2000). https://www.rfc-editor.org/info/rfc2818.

T. Berners-Lee, R. Fielding, L. Masinter: IETF RFC 3986, *Uniform Resource Identifier (URI): Generic Syntax*. RFC Publisher (2005). https://www.rfc-editor.org/info/rfc3986.

A. Phillips, M. Davis: IETF RFC 4646, *Tags for Identifying Languages*. RFC Publisher (2006). https://www.rfc-editor.org/info/rfc4646.

R. Fielding, J. Reschke (eds.): IETF RFC 7231, *Hypertext Transfer Protocol (HTTP/1.1): Semantics and Content*. RFC Publisher (2014). https://www.rfc-editor.org/info/rfc7231.

J. Snell: IETF RFC 7240, *Prefer Header for HTTP*. RFC Publisher (2014). https://www.rfc-editor.org/info/rfc7240.

M. Nottingham: IETF RFC 8288, *Web Linking*. RFC Publisher (2017). https://www.rfc-editor.org/info/rfc8288.

T. Bray (ed.): IETF RFC 8259, *The JavaScript Object Notation (JSON) Data Interchange Format*. RFC Publisher (2017). https://www.rfc-editor.org/info/rfc8259.

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 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	
IDENTIFIER	`http://www.opengis.net/spec/ogcapi-processes-1/1.0/req/core`
OBLIGATION	requirement
TARGET TYPE	Web API
PREREQUISITES	http://www.opengis.net/spec/ogcapi_common-1/1.0/req/core RFC 2616 (HTTP/1.1) RFC 2818 (HTTP over TLS) 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.

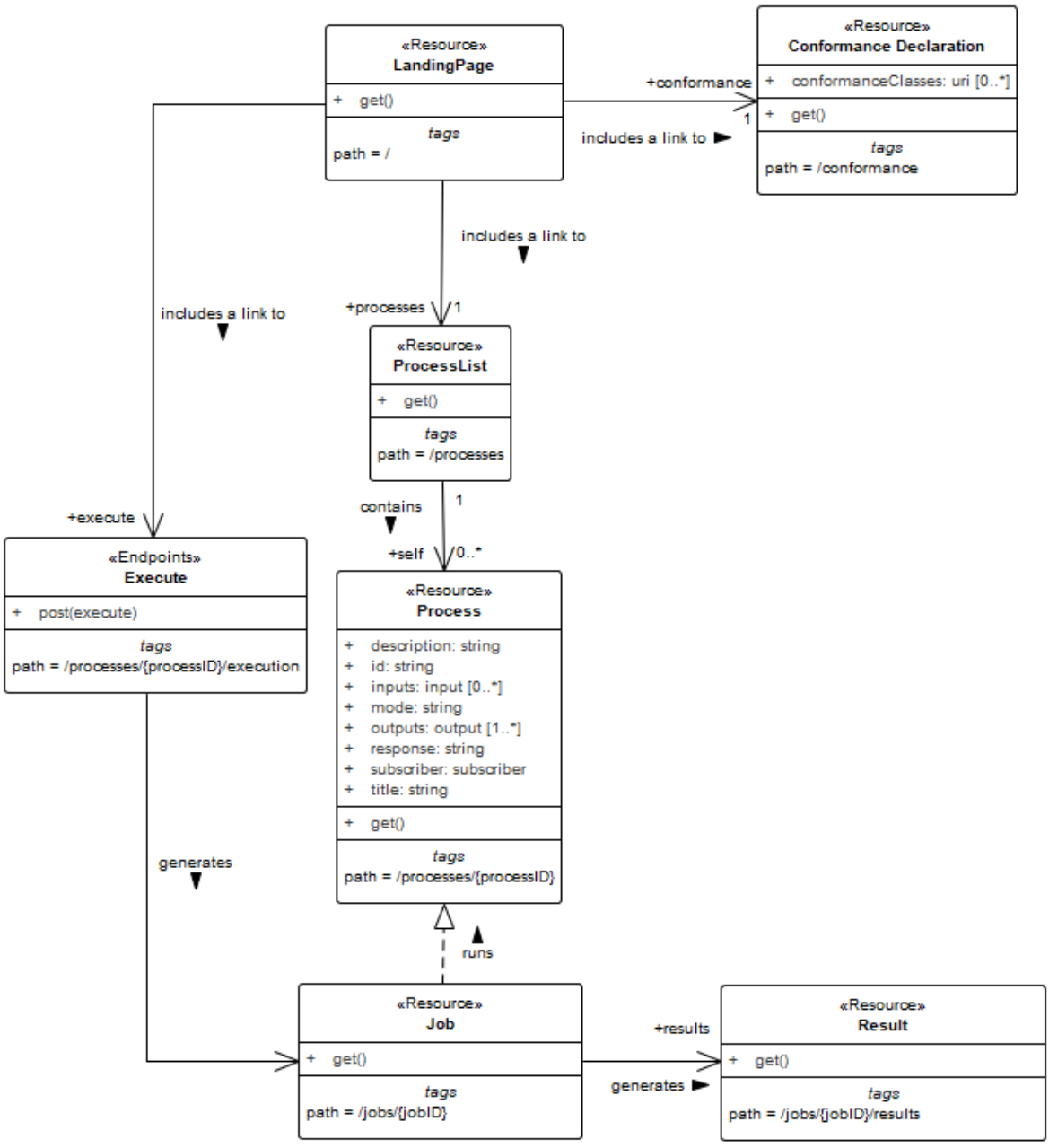

Figure 1 — Resources in the Core requirements class

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

7.2.2. Response

REQUIREMENT 2	
IDENTIFIER	`/req/core/landingpage-success`

REQUIREMENT 2	
STATEMENT	A successful execution of the operation SHALL be reported as a response with a HTTP status code `200`. The content of that response SHALL be based upon the OpenAPI 3.0 schema landingPage.yaml and include at least links to the following resources: • the API definition (relation type 'service-desc' or 'service-doc') • `/conformance` (relation type '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
  attribution:
    type: string
    title: attribution for the API
    description: The `attribution` should be short and intended for
presentation to a user, for example, in a corner of a map. Parts of the text
can be links to other resources if additional information is needed. The
string can include HTML markup.
  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	
IDENTIFIER	`The URIs of all API definitions referenced from the landing page SHALL support the HTTP GET method./req/core/api-definition-op The URIs of all API definitions referenced from the landing page SHALL support the HTTP GET method.`

PERMISSION 1	
LABEL	/per/core/api-definition-uri
STATEMENT	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	
IDENTIFIER	`A successful execution of the operation to get the API definition document SHALL be reported as a response with a HTTP status code /req/core/api-`

REQUIREMENT 4	
	`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`.
STATEMENT	The server SHALL return an API definition document.

RECOMMENDATION 2	
STATEMENT	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	
IDENTIFIER	`/req/core/conformance-op`
STATEMENT	The server SHALL support the HTTP GET operation at the path `/conformance`.

7.4.2. Response

REQUIREMENT 6	
IDENTIFIER	`/req/core/conformance-success`
STATEMENT	A successful execution of the operation SHALL be reported as a response with a HTTP status code `200`. The content of that response SHALL be based upon the OpenAPI 3.0 schema req-classes.yaml and list all OGC API — Processes requirements classes that the server conforms to.

```
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 requirements 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	
IDENTIFIER	`/req/core/http`
STATEMENT	The server SHALL conform to HTTP 1.1. If the server supports HTTPS, the server SHALL also conform to HTTP over TLS.

This include the correct use of status codes, headers. etc.

RECOMMENDATION 3	
STATEMENT	The server SHOULD support the HTTP 1.1 method HEAD for all resources that support the method GET.

Supporting the method HEAD in addition to GET can be useful for clients and is simple to implement. In particular, the HEAD method is useful in determine the size of a potential response.

RECOMMENDATION 4	
STATEMENT	The server SHOULD include the HTTP 1.1 `Content-Length` header in all responses.

Servers implementing CORS will implement the method OPTIONS, too.

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	
LABEL	/per/core/additional-status-codes
STATEMENT	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 5	
STATEMENT	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 6	
STATEMENT	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 7	
STATEMENT	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 SHOULD contain the value `location` to enable the browser to access the `location` header of the response.

RECOMMENDATION 8	
STATEMENT	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 9	
A	Links included in payload of responses SHOULD also be included as `Link` headers in the HTTP response according to RFC 8288, Clause 3.
B	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	
IDENTIFIER	`/req/core/process-list`

REQUIREMENT 8

STATEMENT	The server SHALL support the HTTP GET operation at the path `/processes`.

7.9.1.2. Parameter `limit`

REQUIREMENT 9

IDENTIFIER	`/req/core/pl-limit-definition`

STATEMENT	The operation SHALL support a parameter `limit` with the following characteristics (using an Open API 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

LABEL	/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

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

LABEL	/per/core/limit-response

PERMISSION 4	
A	The server MAY return fewer process summaries than requested (but not more).

7.9.2. Response

REQUIREMENT 11	
IDENTIFIER	/req/core/process-list-success
STATEMENT	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: "../common-core/link.yaml"
```

Figure 6 — 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: "../common-core/link.yaml"
```

Figure 7 — 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 8 — 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 9 — 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	
IDENTIFIER	`/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 10	
STATEMENT	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 11	
STATEMENT	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 12	
STATEMENT	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	
LABEL	/per/core/prev
A	A response to dereferencing a next page link (relation: `next`) MAY include a prev 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	
IDENTIFIER	`/req/core/process-description`
STATEMENT	The server SHALL support the HTTP GET operation at the path `/processes/{processID}`.

7.10.2. Response

REQUIREMENT 14	
IDENTIFIER	`/req/core/process-description-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 13	
STATEMENT	Implementations SHOULD consider supporting the OGC process description.

7.10.3. Error situations

See Clause 7.5.1 for general guidance.

REQUIREMENT 15	
IDENTIFIER	`/req/core/process-exception/no-such-process`
STATEMENT	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	
IDENTIFIER	`/req/core/process-execute-op`
STATEMENT	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	
IDENTIFIER	`/req/core/process-execute-request`
STATEMENT	The content of a request the request body SHALL be based upon the OpenAPI 3.0 schema execute.yaml.

```
type: object
properties:
  inputs:
    additionalProperties:
      $ref: "input.yaml"
  outputs:
    additionalProperties:
      $ref: "output.yaml"
  subscriber:
    $ref: "subscriber.yaml"
```

Figure 10 — Schema for execute

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

```
oneOf:
  - $ref: "inputValueNoObject.yaml"
  - $ref: "qualifiedInputValue.yaml"
  - $ref: "../common-core/link.yaml"
```

Figure 11 — 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"
```

Figure 12 — Schema for a process output

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

```
type: object
properties:
  mediaType:
    type: string
  encoding:
    type: string
```

```
schema:
  oneOf:
    - type: string
      format: url
    - type: object
```

Figure 13 — Schema for a format qualifier

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

REQUIREMENT 18	
IDENTIFIER	`/req/core/process-execute-inputs`
STATEMENT	The server SHALL support process input values specified in-line in an execute request (i.e. by value).
A	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
```

```
    items: {}
  - $ref: "binaryInputValue.yaml"
  - $ref: "bbox.yaml"
```

Figure 14 — 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	
IDENTIFIER	`/req/core/process-execute-input-array`
STATEMENT	The process input is defined in the process description as having a maximum cardinality of greater than one (maxOccurs>1).

REQUIREMENT 19	
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.

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 15 — Schema for a qualified value

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

(see also: format.yaml)

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

Figure 16 — Schema of a process input value

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

(see also: inputValueNoObject.yaml)

REQUIREMENT 20	
IDENTIFIER	`/req/core/process-execute-input-inline-object`
STATEMENT	1. The process input value is specified in-line in an execute request. 2. 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 inputValue.yaml.

REQUIREMENT 21	
IDENTIFIER	`/req/core/process-execute-input-mixed-type`
STATEMENT	1. The process input value is specified in-line in an execute request. 2. 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.

REQUIREMENT 21	
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 or 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 17 — Schema for an in-line binary value

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

REQUIREMENT 22	
IDENTIFIER	`/req/core/process-execute-input-inline-binary`
STATEMENT	1. The process input value is specified in-line in an execute request. 2. 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": "image/tiff; application=geotiff"
}
```

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

```
"imageInput": "R0lGODdhNAHCAfcAAAcHDD+Gs4sLDQpDaqGFdaHE54dJPEoECUlGRteKgcdITgo
kG4hoVkpY\ngNzHwKKkqOLm7RRjlEgpHU9iZ44lHQYqVdmki6doVmhHOMOIeJG20HiDjCcKBglIea
```

```
dISrso\nJGooFNbN2d2qr8aljyklHwQJQkdvkWaKxIdrb442LidLeGhMTp6LkeP1+Kh3aiUuVAoUH
mlu\ngkcwNYdZRmkJDYGcsDFokElVYyk1NsWWhLEPDtmQldrUyoyFhrjo+Nna5d+4tMGstspoXgc4
\n...qgu7sSu7qbtCs2u7t6u6rLsrp4u7veu76eO6vyu8w0u8xWu8x4u8yau8shu8y+u8zwu90Su9
\n00u91Wu914u92au928u9whsQADs="
```

NOTE 6: 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": "image/tiff; application=geotiff"
    },
    {
      "type": "string",
      "contentEncoding": "binary",
      "contentMediaType": "image/jp2"
    }
  ]
}
```

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

```
{
  "value": "VBORw0KGgoAAAANSUhEUgAABvwAAAa4CAYAAABMB35kAAABhGlDQ1BJQ0MgcHJvZml
sZQAA\nKJF9kT1Iw0AcxV9TpSL1A+xQxCFDdbIgKuKoVShChVArtOpgcumH0KQhSXFxFFwLDn4sVh1
c\nnHV1cBUEwQ8QNzcnRRcp8X9JoUWMB8f9eHfvcfcOEOplplkdY4Cm22Y6mRCzuRUx9IogouhH\n
... \nj3Z5mX7/PCPVRJV92rpHK24xcJrzk20+tkeYlCPqcZNO3Lpni1OJWatPCcmgGDEqx7Om6lfa
\nppM4k4BTe9+bsn3L9/9/yWhA0PwQGW8ipCZsnZt9lsdrYEM8z/M8z/M8z/M8z/M8z/MzLWY1\nAAA
ACUlEQVQ871H6P6JI+TxS5Wn2AAAAAElFTkSuQmCC",
  "mediaType": "image/jp2"
}
```

Bounding box values:

A process input value instance can be a bounding box.

REQUIREMENT 23	
IDENTIFIER	/req/core/process-execute-input-inline-bbox
STATEMENT	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:
  anyOf:
    - type: string
      format: uri
      enum:
        - "http://www.opengis.net/def/crs/OGC/1.3/CRS84"
        - "http://www.opengis.net/def/crs/OGC/0/CRS84h"
    - type: string
      format: uri
  default: "http://www.opengis.net/def/crs/OGC/1.3/CRS84"
```

Figure 18 — Schema for a bounding box value

NOTE 7: 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	
LABEL	/per/core/process-execute-input-inline-bbox
STATEMENT	Servers MAY copy the contents of bbox.yaml into another file and adjust the `default` and `enum` values as required.

NOTE 8: The `crs` property has been defined using an `anyOf` of either an enumeration or a generic URI string to allow execution requests making use of a CRS other than CRS84 and CRS84h that may be supported by the server to validate with the canonical OGC schema.

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": "date-time"
      },
      "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	
IDENTIFIER	`/req/core/process-execute-input-validation`
STATEMENT	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.
A	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	
IDENTIFIER	`/req/core/process-execute-default-execution-mode`
STATEMENT	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	
IDENTIFIER	`/req/core/process-execute-auto-execution-mode`
STATEMENT	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.

REQUIREMENT 26	
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 14	
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 15	
STATEMENT	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 16	
STATEMENT	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. 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	
IDENTIFIER	`/req/core/process-execute-default-outputs`

REQUIREMENT 27

STATEMENT	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 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.

Output value size:

Since a priori knowledge of the size of any given output is generally not known, hints can be provided to the server to prevent the inadvertant transmission of large outputs.

Return preferences signal to a server how output values should be transmitted based on the size of the output and any accompanying `return` preferences as define defined in IETF RFC 7240.

RECOMMENDATION 17

STATEMENT	If an execute request **is accompanied** with the HTTP `Prefer` header asserting a `return` preference, then the server SHOULD honor that preference. See requirements: • /req/core/process-execute-sync-one See recommendations: • /rec/core/process-execute-sync-many-json-prefer-none • /rec/core/process-execute-sync-many-json-prefer-minimal • /rec/core/process-execute-sync-many-json-prefer-representation • /rec/core/job-results-async-many-json-prefer-none • /rec/core/job-results-async-many-json-prefer-minimal • /rec/core/job-results-async-many-json-prefer-representation

RECOMMENDATION 18

STATEMENT	A client that accompanies an execute request with the HTTP `Prefer` header asserting a `return` preference SHOULD be prepared to receive output values either: • in-line in the response, • or by reference via a hyperlink in the response.

7.11.3. Example

Example — An execute request.

```
http://www.someserver.com/processes/echo/execution?
stringInput=Value2&
measureInput={measurement=10.3,uom=m,referece=https://ucum.org/ucum-essence.
xml}&
dateInput=2021-03-06T07:21:00&
doubleInput=3.14159&
arrayInput=1,2,3,4,5,6&
complexObjectInput={...JSON or XML string...}&
geometryInput=<WKTstring>,<WKTstring>&
boundingBoxInput=51.9,7,52,7.1;bbox-crs=http://www.opengis.net/def/crs/OGC/1.3/
CRS84&
```

```
{
  "inputs": {
    "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": "image/tiff; application=geotiff"
      },
      {
         "value": "VBORw0KGgoAAAANSUhEUgAABvwAAAa4CAYAAABMB35kAAABhGlDQ1BJQ0Mg
cHJvZmlsZQAA\nKJF9kT1Iw0AcxV9TpSL1A+xQxCFDdbIgKuKoVShChVArtOpgcumH0KQhSXFxFFw
LDn4sVh1c\nnHV1cBUEwQ8QNzcnRRcp8X9JoUWMB8f9eHfvcfcOEOplplkdY4Cm22Y6mRCzuRUx9I
ogouhH\n ... \nj3Z5mX7/PCPVRJV92rpHK24xcJrzk20+tkeYlCPqcZNO3Lpni1OJWatPCcmgGD
Eqx7Om6lfa\nppM4k4BTe9+bsn3L9/9/yWhA0PwQGW8ipCZsnZt9lsdrYEM8z/M8z/M8z/M8z/
MzLWY1\nAAAACUlEQVQ871H6P6JI+TxS5Wn2AAAAAElFTkSuQmCC",
         "encoding": "base64",
         "mediaType": "image/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&#x26;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": "image/tiff; application=geotiff" },
```

```
      "transmissionMode": "value"
    },
    "featureCollectionOutput": {
      "transmissionMode": "value"
    }
  },
  "response": "document"
}
```

7.11.4. Response

7.11.4.1. Overview

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 number of outputs requested,
- the negotiated content type for the response (via the HTTP Accept header),
- and any negotiated client preferences (via the HTTP Prefer header).

The following table maps the possible responses based on the combinations of these execute parameters. The column headers denote:

1. **Negotiated execute mode**: see Clause 7.11.2.3
2. **Requested # outputs**: the number of output requested in the execute request
3. **Return preference**: the negotiated return preferences
4. **Response HTTP code**: the HTTP status code that the server should generate in this context
5. **Response headers**: any required HTTP headers that the server must include in this context
6. **Response media type**: the value of the HTTP `Content-Type` header that the server should generate in this context
7. **Response body**: a description of the content of the response body in this context
8. **Req./Rec./Perm.**: the corresponding requirements/recommendations/permission as per this specification

Table 11 — Execute responses based on number of requested outputs, request HTTP headers and the size of output values.

NEGOTIATED EXECUTE MODE	REQUESTED # OUTPUTS	RETURN PREFERENCE (`Prefer` HEADER)	RESPONSE HTTP CODE	RESPONSE HEADERS	RESPONSE MEDIA TYPE (`Content-Type` HEADER)	RESPONSE BODY	REQ./REC./ PERM.
sync	1	*n/a*	200		*as negotiated*	requested process output	See requirement: /req/core/ process-execute-sync-one
	N	*none*	200		application/ json	results. yaml	See Note 3., Note 4
		minimal					See Note 3., Note 5
		representation					See Note 3., Note 6
	none	*n/a*	204		*n/a*	*empty*	See requirement: /req/core/ job-results-param-outputs-empty
async	*n/a*	*n/a*	201	*Location* to newly created job	application/ json	statusInfo. yaml	See requirement: /req/core/ process-execute-success-async

NOTE 1:

- The value ***n/a*** in a cell means that the corresponding parameter is not applicable, not specified, not required or can be ignored in this context.
- The value ***as negotiated*** means that the output content type is as negotiated between the client and the server based on the value of the HTTP Accept header.

NOTE 2: 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, a server that only offers processes that can be executed synchronously does not need to implement any of the asynchronous requirements.

NOTE 3:

- See requirement: /req/core/process-execute-sync-many-json
- See permission: /per/core/process-execute-sync-many-other-formats

NOTE 4:
- See recommendation: /rec/core/process-execute-sync-many-json-prefer-none

NOTE 5:
- See recommendation: /rec/core/process-execute-sync-many-json-prefer-minimal

NOTE 6:
- See recommendation: /rec/core/process-execute-sync-many-json-prefer-representation

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

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

NOTE: 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.

7.11.4.2. Response requesting a single processing output

When a process is executed synchronously and a single processing output is requested, the following requirements and recommendations apply:

REQUIREMENT 28	
IDENTIFIER	`/req/core/process-execute-sync-one`
STATEMENT	1. The negotiated execution mode is synchronous, 2. The number of requested outputs is 1.
A	A successful execution of the the operation SHALL be reported with a response with a HTTP status code `200`.
B	The media type of the response SHALL be as negotiated as per the HTTP content negotiation rules.
C	The content of response body SHALL be the requested process output value in the negotiated output format.

REQUIREMENT 29	
IDENTIFIER	`/req/core/process-execute-sync-one-default-content`
STATEMENT	1. The negotiated execution mode is synchronous,

REQUIREMENT 29	
	2. The number of requested outputs is 1. 3. No content negotiation has been sepcified using either HTTP headers or other methods.
A	The media type of the response SHALL be the default media type for the requested processes output as specified in the process description.
B	The content of response body SHALL be the requested process output value in the default output format.

NOTE 1: For servers that support the OGC Process Description conformance class see requirements /req/ogc-process-description/output-def and /req/ogc-process-description/output-mixed-type for determining the default format for a process output.

NOTE 2: The size of a response may be determined without actually transmitting the entire response by using the HTTP HEAD method to retrieve the `Content-Length` header.

7.11.4.3. Response requesting multiple processing outputs

When a process is executed synchronously and multiple processing outputs are requested, the following requirements and recommendations apply:

REQUIREMENT 30	
IDENTIFIER	`/req/core/process-execute-sync-many-json`
STATEMENT	1. The negotiated execution mode is synchronous. 2. The number of requested outputs is 2 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 19	
CONDITIONS	1. The negotiated execution mode is synchronous. 2. The number of requested outputs is 2 or more. 3. No return preference accompanies the request.
A	If the server deems that the size of an output value is *small*, that value SHOULD be included in-line in the response.

RECOMMENDATION 19

B	If the server deems that the size of an output value is *large*, that value SHOULD be included by reference via hyperlink in the response.

RECOMMENDATION 20

CONDITIONS	1. The negotiated execution mode is synchronous. 2. The number of requested outputs is 2 or more. 3. The negotiated return preference is `minimal`.
A	If the server deems that the size of an output value is *small*, that value SHOULD be included in-line in the response.
B	If the server deems that the size of an output value is *large*, that value SHOULD be included by reference via hyperlink in the response.

RECOMMENDATION 21

CONDITIONS	1. The negotiated execution mode is synchronous. 2. The number of requested outputs is 2 or more. 3. The negotiated return preference is `representation`.
A	Each requested output value SHOULD be included in-line in the response.

PERMISSION 7

LABEL	/per/core/process-execute-sync-many-other-formats	
	1. The negotiated execution mode is synchronous. 2. The number of requested outputs is 2 or more.	
STATEMENT	part	Servers MAY support other response formats or encodings (e.g. ZIP or `multipart/*`) that do not conform to results.yaml.
	part	This standard does not provide any guidance on these other formats or encodings.

7.11.4.4. Job creation on synchronous process execution

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 8	
LABEL	/per/core/process-execute-sync-job
STATEMENT	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 31	
IDENTIFIER	`/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.
B	When resolving the `rel=monitor` link, the content type of response SHALL be `application/json`.
C	When resolving the `rel=monitor` link, the body of the response SHALL be based upon the JSON schema fragment statusInfo.yaml.

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

7.11.4.5. Response for asynchronous processes execution

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

REQUIREMENT 32	
IDENTIFIER	`/req/core/process-execute-success-async`
STATEMENT	The negotiated execution mode is asynchronous.

REQUIREMENT 32	
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 33	
IDENTIFIER	`/req/core/job`
STATEMENT	The server SHALL support the HTTP GET operation at the path `/jobs/{jobID}`.

7.12.2. Response

REQUIREMENT 34	
IDENTIFIER	`/req/core/job-success`
STATEMENT	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.

```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: "../common-core/link.yaml"
```

Figure 20 — Schema for status info

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

```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 22	
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 35	
IDENTIFIER	`/req/core/job-exception-no-such-job`
STATEMENT	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

A process can generate zero or more outputs. This clause specifies how:

- processing results can be retrieved individually in a single request,
- a subset of processing results can be retrieved in a single request,
- or all processing results can be retrieved in a single request.

7.13.1. Retrieving results individually

7.13.1.1. Operation

REQUIREMENT 36	
IDENTIFIER	`/req/core/job-result`
STATEMENT	For each defined process output with identifier `outputId`, the server SHALL support the HTTP GET operation at the path `/jobs/{jobID}/results/{outputID}`.

NOTE: The size of a processing result may be determined without actually transmitting the entire result by using the HTTP HEAD method to retrieve the `Content-Length` header.

7.13.2. Retrieving multiple results

7.13.2.1. Operation

REQUIREMENT 37

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

7.13.2.2. Parameters

REQUIREMENT 38

IDENTIFIER	`/req/core/job-results-param-outputs`

STATEMENT	The operation at the `/jobs/{jobID}/results` endpoint SHALL support a parameter `outputs` with the following characteristics (using an OpenAPI Specification 3.0 fragment):
	`name: outputs`
	`in: query`
	`required: false`
	`schema:`
	`  type: array`
	`  items:`
	`  type:string`
	`style: form`
	`explode: false`

REQUIREMENT 39

IDENTIFIER	`/req/core/job-results-param-outputs-response`
STATEMENT	Only the outputs with identifers specified by the optional `outputs` parameter SHALL be included in the response.

REQUIREMENT 40

IDENTIFIER	`/req/core/job-results-param-outputs-omit`
STATEMENT	Omitting the `outputs` parameter indicates that all processing results SHALL be included in the response.

REQUIREMENT 41

IDENTIFIER	`/req/core/job-results-param-outputs-empty`

REQUIREMENT 41	
STATEMENT	If the `outputs` parameter is empty in an execute request, this SHALL cause the server to respond with a HTTP status code 204 and an empty response body.

7.13.2.3. Overview

The manner in which a server responds when retrieving job results depends on:

- the number of outputs requested,
- the negotiated content type for the response (via the HTTP Accept header) or lack thereof,
- and any negotiated client preferences (via the HTTP Prefer header).

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

The column headers in the following table denote:

1. **Requested # outputs**: the number of outputs requested in the execute request
2. **Access path**: the API endpoint from which the result(s) can be retrieved
3. **Return preference**: the negotiated return preferences
4. **Response HTTP code**: the HTTP status code that the server should generate in this context
5. **Response media type**: the value of the HTTP `Content-Type` header that the server should generate in this context
6. **Response body**: a description of the content of the response body in this context
7. **Req./Rec./Perm.**: the corresponding requirements/recommendations/permission as per this specification

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

REQUESTED # OUTPUTS	ACCESS PATH	RETURN PREFERENCE (`Prefer` HEADER)	RESPONSE HTTP CODE	RESPONSE MEDIA TYPE (`Content-Type` HEADER)	RESPONSE BODY	REQ./REC./ PERM.
1	/jobs/{job ID}/results/{outputID}	*n/a*	200	*as negotiated*	requested process output value	See requirement: /req/core/job-results-async-one

REQUESTED # OUTPUTS	ACCESS PATH	RETURN PREFERENCE (`Prefer` HEADER)	RESPONSE HTTP CODE	RESPONSE MEDIA TYPE (`Content-Type` HEADER)	RESPONSE BODY	REQ./REC./ PERM.
N	/jobs/{job ID}/results	*none*	200	application/json	results.yaml	See Note 2,Note 3
		minimal				See Note 2,Note 4
		representation				See Note 2,Note 5
none	*n/a*	*n/a*	204	*n/a*	*n/a*	See requirement: /req/core/job-results-param-outputs-empty

NOTE 1:

- The value ***n/a*** in a cell means value of the corresponding parameter is not applicable, not specified, not required or can be ignored in this context.
- The value ***as negotiated*** means that the output content type is as negotiated between the client and the server based on the value of the HTTP Accept header.

NOTE 2:

- See requirement: /req/core/job-results-async-many
- See permission: /per/core/job-results-async-many-other-formats

NOTE 3:

- See recommendation: /rec/core/job-results-async-many-json-prefer-none

NOTE 4:

- See recommendation: /rec/core/job-results-async-many-json-prefer-minimal

NOTE 5:

- See recommendation: /rec/core/job-results-async-many-json-prefer-representation

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

7.13.2.4. Retrieving results individually

REQUIREMENT 42	
IDENTIFIER	`/req/core/job-results-async-one`
STATEMENT	1. The negotiated execution mode is asynchronous,

REQUIREMENT 42	
	2. The number of requested outputs is 1. 3. The result is retieved from the `/jobs/{jobID}/results/{outputID}` endpoint.
A	The server SHALL respond with an HTTP status code of 200.
B	The media type of the response SHALL be as negotiated as per the HTTP content negotiation rules.
C	The content of response body SHALL be the requested process output value in the negotiated output format.

7.13.2.5. Retrieving multiple results

REQUIREMENT 43	
IDENTIFIER	`/req/core/job-results-async-many`
STATEMENT	1. The negotiated execution mode is asynchronous. 2. The number of requested outputs is 2 or more. 3. The results are retrieved from the `/jobs/{jobID}/results` endpoint.
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 23	
CONDITIONS	A return preference is ***not*** specified via the HTTP `prefer` header.
A	If the server deems that the size of an output value is *small*, that value SHOULD be included in-line in the response.
B	If the server deems that the size of an output value is *large*, that value SHOULD be included by reference via hyperlink in the response.

RECOMMENDATION 24	
CONDITIONS	The negotiated return preference is `minimal`.

RECOMMENDATION 24	
A	If the server deems that the size of an output value is *small*, that value SHOULD be included in-line in the response.
B	If the server deems that the size of an output value is *large*, that value SHOULD be included by reference via hyperlink in the response.

RECOMMENDATION 25	
CONDITIONS	The negotiated return preference is `representation`.
A	Each requested output value SHOULD be included in-line in the response.

PERMISSION 9		
LABEL	/per/core/job-results-async-many-other-formats	
STATEMENT	1. The negotiated execution mode is asynchronous. 2. The number of requested outputs is 2 or more.	
	part	Servers MAY support other response formats or encodings (e.g. ZIP or `multipart/*`) that do not conform to results.yaml.
	part	This standard does not provide any guidance on these other formats or encodings.

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": "image/tiff; application=geotiff"
    },
    {
      "value": "VBORw0KGgoAAAANSUhEUgAABvwAAAa4CAYAAABMB35kAAABhGlDQ1BJQ0MgcHJv
ZmlsZQAA\nKJF9kT1Iw0AcxV9TpSL1A+xQxCFDdbIgKuKoVShChVArtOpgcumH0KQhSXFxFFwLDn4sV
h1c\nnHV1cBUEwQ8QNzcnRRcp8X9JoUWMB8f9eHfvcfcOEOplplkdY4Cm22Y6mRCzuRUx9IogouhH\n
 ... \nj3Z5mX7/PCPVRJV92rpHK24xcJrzk20+tkeYlCPqcZNO3Lpni1OJWatPCcmgGDEqx7Om6lfa
\nppM4k4BTe9+bsn3L9/9/yWhA0PwQGW8ipCZsnZt9lsdrYEM8z/M8z/M8z/M8z/MzLWY1\nAAA
ACUlEQVQ871H6P6JI+TxS5Wn2AAAAAElFTkSuQmCC",
      "encoding": "base64",
      "mediaType": "image/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&#x26;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	
IDENTIFIER	`/req/core/job-results-exception/no-such-job`
STATEMENT	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	
IDENTIFIER	`/req/core/job-results-exception/results-not-ready`
STATEMENT	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	
IDENTIFIER	`/req/core/job-results-failed`
STATEMENT	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	
IDENTIFIER	`http://www.opengis.net/spec/ogcapi-processes-1/1.0/req/ogc-process-description`
OBLIGATION	requirement
TARGET TYPE	Web API
PREREQUISITES	OGC API — Processes Core 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

IDENTIFIER	`/req/ogc-process-description/json-encoding`
STATEMENT	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 23 — 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

IDENTIFIER	`/req/ogc-process-description/inputs-def`
STATEMENT	Each process input definition SHALL be listed in the `inputs` section according to the JSON Schema: inputDescription.yaml.

REQUIREMENT 48	
A	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 24 — Schema for a process input

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

(see also: `descriptionType.yaml`).

REQUIREMENT 49	
IDENTIFIER	`/req/ogc-process-description/input-def`
STATEMENT	The schema of each process **input** value SHALL be specified using the `schema` parameter.
A	The value of the `schema` parameter SHALL be a JSON fragment that validates according to the JSON Schema: schema.yaml.
B	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	

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

REQUIREMENT 51	
IDENTIFIER	/req/ogc-process-description/input-mixed-type
STATEMENT	An input that can be of mixed type SHALL be defined using the oneOf JSON Schema keyword.
A	Each sub-schema SHALL be a JSON fragment that validates according to the JSON Schema: schema.yaml.
B	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": "image/tiff; application=geotiff"
      },
      {
        "type": "string",
        "contentEncoding": "binary",
        "contentMediaType": "image/jp2"
      }
    ]
  }
}
```

Figure 26 — Mixed type input example

RECOMMENDATION 26	
STATEMENT	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).
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 27

STATEMENT 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 28

STATEMENT 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 27 — Example of semantic hints using the format key

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

Figure 28 — Schema for a process output

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

(see also: `descriptionType.yaml`).

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

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

REQUIREMENT 54	
IDENTIFIER	`/req/ogc-process-description/output-mixed-type`
STATEMENT	An output that can be of mixed type SHALL be defined using the `oneOf` JSON Schema keyword.
A	Each sub-schema SHALL be a JSON fragment that validates according to the JSON Schema: schema.yaml.
B	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": [
          "measurement",
          "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": "date-time"
      }
    },
    "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": "date-time"
          },
          "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": "image/tiff; application=geotiff"
          },
          {
            "type": "string",
            "contentEncoding": "binary",
            "contentMediaType": "image/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": [
          "measurement",
          "uom"
        ],
        "properties": {
          "measurement": {
            "type": "number"
          },
          "uom": {
            "type": "string"
          },
          "reference": {
            "type": "string",
            "format": "uri"
          }
        }
      }
    },
    "dateOutput": {
      "schema": {
        "type": "string",
        "format": "date-time"
      }
    },
    "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": "date-time"
            },
            "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": "image/tiff; application=geotiff"
            },
            {
              "type": "string",
              "contentEncoding": "binary",
              "contentMediaType": "image/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	
IDENTIFIER	`http://www.opengis.net/spec/ogcapi-processes-1/1.0/req/json`
OBLIGATION	requirement
TARGET TYPE	Web API
PREREQUISITES	OGC API — Processes Core JSON

REQUIREMENT 55	
IDENTIFIER	`/req/json/definition`
STATEMENT	`200`-responses of the server SHALL support the following media type: • `application/json` for the following API endpoints: • `/`Clause 7.2, • `/conformance`Clause 7.4, • `/processes`Clause 7.9, • `/processes/{processID}`Clause 7.10, • `/jobs/{jobID}`Clause 7.12 and for the following API endpoint: • `/processes/{processID}/execution`Clause 7.11 when the negotiated response format using the HTTP Accept header or the `f`Table 14 parameter is JSON, or the number of requested outputs is greater than one or the negotiated execution mode is asynchronous.

9.3. Requirement Class “HTML”

This section defines the requirements class HTML.

REQUIREMENTS CLASS 4

IDENTIFIER	`http://www.opengis.net/spec/ogcapi-processes-1/1.0/req/html`
OBLIGATION	requirement
TARGET TYPE	Web API
PREREQUISITES	OGC API — Processes Core http://www.opengis.net/spec/ogcapi_common/1.0/req/html W3C HTML 5

REQUIREMENT 56

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

REQUIREMENT 57

IDENTIFIER	`/req/html/content`
STATEMENT	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	
IDENTIFIER	`http://www.opengis.net/spec/ogcapi-processes-1/1.0/req/oas30`
OBLIGATION	requirement
TARGET TYPE	Web API
PREREQUISITES	OGC API — Processes 1.0 Core http://www.opengis.net/spec/ogcapi_common-1/1.0/req/oas30 OpenAPI Specification 3.0.1

REQUIREMENT 58	
IDENTIFIER	`/req/oas30/oas-definition-1`
A	The content of the response of the HTTP GET operation at the landing page SHALL include the following links to the API definition: • relation type `service-desc` and content type `application/vnd.oai.openapi+json;version=3.0 • relation type `service-doc` and content type `text/html`

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

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

10.2. Complete definition

REQUIREMENT 61	
IDENTIFIER	`/req/oas30/completeness`
STATEMENT	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	
IDENTIFIER	`/req/oas30/exceptions-codes`
STATEMENT	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://raw.githubusercontent.com/opengeospatial/ogcapi-processes/
openapi/schemas/common-core/exception.yaml
  text/html:
    schema:
      type: string
```

10.4. Security

REQUIREMENT 63	
IDENTIFIER	`/req/oas30/security`
STATEMENT	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	
IDENTIFIER	`http://www.opengis.net/spec/ogcapi-processes-1/1.0/req/job-list`
OBLIGATION	requirement
TARGET TYPE	Web API
PREREQUISITE	OGC API — Processes Core

11.2. Operation

11.2.1. Job list

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

RECOMMENDATION 29	
STATEMENT	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

IDENTIFIER	`/req/job-list/type-definition`

STATEMENT	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

IDENTIFIER	`/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

IDENTIFIER	`/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

IDENTIFIER	`/req/job-list/processID-definition`

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

REQUIREMENT 68

```
required: false
schema:
  type: array
  items:
    type: string
```

REQUIREMENT 69

IDENTIFIER	`/req/job-list/processid-response`
STATEMENT	If the parameter is specified with the operation, only jobs that have a value for the `processID` 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

IDENTIFIER	`/req/job-list/status-definition`

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

REQUIREMENT 71

IDENTIFIER	`/req/job-list/status-response`
STATEMENT	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

IDENTIFIER	`/req/job-list/datetime-definition`

STATEMENT	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`

A	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`

B	The syntax of `date-time` is specified by RFC 3339, 5.6.
C	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

IDENTIFIER	`/req/job-list/datetime-response`
STATEMENT	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

IDENTIFIER	`/req/job-list/duration-definition`

STATEMENT	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

A	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

IDENTIFIER	`/req/job-list/duration-response`
STATEMENT	1. 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. 2. 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).
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

IDENTIFIER	`/req/job-list/limit-definition`

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

PERMISSION 10

IDENTIFIER	`/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

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

PERMISSION 11

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

11.3. Response

REQUIREMENT 78

IDENTIFIER	`/req/job-list/job-list-success`
STATEMENT	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: "../common-core/link.yaml"
```

Figure 37 — 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

IDENTIFIER	`/req/job-list/links`
STATEMENT	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 30

STATEMENT	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 31

STATEMENT	Dereferencing a next page link (relation: `next`) SHOULD return additional jobs from the set of selected jobs that have not yet been returned.

RECOMMENDATION 32

STATEMENT	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 12

IDENTIFIER	`/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 Requirement 15: `/req/core/process-exception/no-such-process`).

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	
IDENTIFIER	`http://www.opengis.net/spec/ogcapi-processes-1/1.0/req/callback`
OBLIGATION	requirement
TARGET TYPE	Web API
PREREQUISITE	OGC API — Processes Core

REQUIREMENT 80	
IDENTIFIER	`/req/callback/job-callback`
STATEMENT	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	
IDENTIFIER	`http://www.opengis.net/spec/ogcapi-processes-1/1.0/req/dismiss`
OBLIGATION	requirement
TARGET TYPE	Web API
PREREQUISITE	OGC API — Processes Core

13.1. Operation

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

13.2. Response

REQUIREMENT 82	
IDENTIFIER	`/req/dismiss/job-dismiss-success`
STATEMENT	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

REQUIREMENTS CLASS "KVP-ENCODED EXECUTE"

14 REQUIREMENTS CLASS "KVP-ENCODED EXECUTE"

14.1. Overview

REQUIREMENTS CLASS 9	
IDENTIFIER	`http://www.opengis.net/spec/ogcapi-processes-1/1.0/req/kvp-execute`
OBLIGATION	requirement
TARGET TYPE	Web API
PREREQUISITE	Requirements class 1: `http://www.opengis.net/spec/ogcapi-processes-1/1.0/req/core`

The "Execute a process" clause describes how to invoke a process by sending a JSON-encoded execute request to the execution endpoint of a server using the HTTP POST method. This invocation pattern does not, however, allow dynamic execute requests to be embedded in other documents. This clause describes a process invocation method that encodes an execute request as a URL with query parameters.

The following examples illustrate KVP-encoded process execution requests.

Example — Examples

```
https://www.someserver.com/processes/area:retrieve/execution?
collections=observations&
bbox=5.8,47.2,15.1,55.1&
datetime=2019-08-09&
variables=TMAX,TMIN,PRCP

http://www.someserver.com/processes/echo/execution?
stringInput=Value2&
measureInput=%7B%22value%22%3A%7B%22measurement%22%3A10.3%2C%22uom%22%3A%22m%22
%2C%22reference%22%3A%22https%3A%2F%2Fucum.org%2Fucum-essence.xml%22%7D%7D
dateInput=2021-03-06T07:21:00&
doubleInput=3.14159&
arrayInput=%5B1%2C2%2C3%2C4%2C5%2C6%5D&
complexObjectInput=%22value%22%3A%7B%22property1%22%3A%22value1%22%2C%22propert
y2%22%3A%22value2%22%2C%22property5%22%3Atrue%7D%7D&
geometryInput=%5B%7B%22value%22%3A%22%3Cgml%3APolygongml%3Aid%3D%5C%22GID1
%5C%22srsName%3D%5C%22urn%3Aogc%3Adef%3Acrs%3AOGC%3A%3ACRS84%5C%22%3E%3Cgml
%3Aexterior%3E%3Cgml%3ALinearRing%3E%3Cgml%3AposList%3E-77.02451938.810529-
77.02463538.810973-77.02470438.810962-77.02477638.811239-77.02495738.81121-
77.02490538.811012-77.02490538.811012-77.02486538.810857-77.02502438.810832-
```

```
77.02507138.811012-77.02520338.810992-77.0250638.810444-77.02451938.810529%3
C%2Fgml%3AposList%3E%3C%2Fgml%3ALinearRing%3E%3C%2Fgml%3Aexterior%3E%3C%2Fgml
%3APolygon%3E%22%2C%22mediaType%22%3A%22application%2Fgml%2Bxml%3Bversion%3D3
.2%22%7D%2C%7B%22value%22%3A%7B%22type%22%3A%22Polygon%22%2C%22coordinates%22
%3A%5B%5B%5B-176.5814819%2C-44.10896301%5D%2C%5B-176.5818024%2C-44.10964584%5D
%2C%5B-176.5844116%2C-44.11236572%5D%2C%5B-176.5935974%2C-44.11021805%5D
%2C%5B-176.5973511%2C-44.10743332%5D%2C%5B-176.5950928%2C-44.10562134%5D%2C
%5B-176.5858459%2C-44.1043396%5D%2C%5B-176.5811157%2C-44.10667801%5D%2C%5B-
176.5814819%2C-44.10896301%5D%5D%5D%7D%7D%5D&
imagesInput=%5B%7B%22href%22%3A%22https%3A%2F%2Fwww.someserver.com%2Fogcapi%2FD
araa%2Fcollections%2FDaraa_DTED%2Fstyles%2FTopographic%2Fcoverage%3F...%22%2C%2
2type%22%3A%22image%2Ftiff%3Bapplication%3Dgeotiff%22%7D%2C%7B%22value%22%3A%22
VBORw0KGgoAAAANSUhEUgAABvwAAAa4CAYAAABMB35kAAABhGlDQ1BJQ0MgcHJvZmlsZQAA%5CnKJF9
kT1Iw0AcxV9TpSL1A%2BxQxCFDdbIgKuKoVShChVArtOpgcumH0KQhSXFxFFwLDn4sVh1c%5CnnHV1c
BUEwQ8QNzcnRRcp8X9JoUWMB8f9eHfvcfcOEOplplkdY4Cm22Y6mRCzuRUx9IogouhH%5Cn...%5Cnj
3Z5mX7%2FPCPVRJV92rpHK24xcJrzk20%2BtkeYlCPqcZNO3Lpni1OJWatPCcmgGDEqx7Om6lfa%5Cn
ppM4k4BTe9%2Bbsn3L9%2F9%2FyWhA0PwQGW8ipCZsnZt9lsdrYEM8z%2FM8z%2FM8z%2FM8z%2FM8z
%2FMzLWY1%5CnAAAACUlEQVQ871H6P6JI%2BTxS5Wn2AAAAAElFTkSuQmCC%22%2C%22encoding%22
%3A%22base64%22%2C%22mediaType%22%3A%22image%2Fjp2%22%7D%5D
boundingBoxInput=51.9,7,52,7.1&
boundingBoxInput-crs=http://www.opengis.net/def/crs/OGC/1.3/CRS84&
featureCollectionInput=%7B%22value%22%3A%22%3C%3Fxmlversion%3D%5C%221.0%5C%22
encoding%3D%5C%22UTF-8%5C%22%3F%3E%3CFeatureCollectionxmlns%3D%5C%22http%3A%2F
%2Fschemas.myserver.com%2Fnamespaces%2Fnull%5C%22xmlns%3Agml%3D%5C%22http%3A%2F
%2Fwww.opengis.net%2Fgml%2F3.2%5C%22xmlns%3Axsi%3D%5C%22http%3A%2F%2Fwww.w3.org
%2F2001%2FXMLSchema-instance%5C%22xsi%3AschemaLocation%3D%5C%22http%3A%2F%2Fsc
hemas.myserver.com%2Fnamespaces%2Fnullhttps%3A%2F%2Fwww.pvretano.com%2Fmyserver
%2Fogcapi%2Fdaraa%2Fschema%3Ff%3DGML32%26amp%3Bcollectionids%3DTransportationGr
oundCrvhttp%3A%2F%2Fwww.opengis.net%2Fgml%2F3.2http%3A%2F%2Fschemas.opengis.net
%2Fschemas%2Fgml%2F3.2.1%2Fgml.xsd%5C%22%3E...%22%2C%22mediaType%22%3A%22applic
ation%2Fgml%2Bxml%3Bversion%3D3.2%22%7D&
stringOutput[include]=true&
measureOutput[include]=true&
dateOutput[include]=true&
doubleOutput[include]=true&
arrayOutput[include]=true&
complexObjectOutput[include]=true&
geometryOutput[include]=true&
boundingBoxOutput[include]=true&
imageOutput[include]=true&
featureCollectionOutput[include]=true
```

NOTE: Add more examples!

14.2. Execute a process

This section describes the requirements for invoking a process with an execute request encoded as a URL with query parameters.

14.2.1. Operation

REQUIREMENT 83	
IDENTIFIER	`/req/kvp-execute/process-execute-op`
STATEMENT	The server SHALL support the HTTP GET operation at the path `/processes/{processID}/execution`.

14.2.2. Parameters

14.2.2.1. Format parameter

Table 14

Requirement 1	/req/kvp-execute/f-definition

A	The operation SHALL support a parameter `f` with the following characteristics:
	`name: f`
	`in: query`
	`required: false`
	`schema:`
	`  type: string`
	`  format: uri`
	`style: form`
	`explode: false`

Table 15

Requirement 2	/req/kvp-execute/f-response
A	The `f` parameter SHALL behave in the same way that the HTTP `Accept` header behaves.

Example — Negotiating a respone format.: This is an execution example where an input image is provided, along with other parameters, to the `myProcess` process and the response format is requested to be either a PNG or JPEG image (in that order of preference).

```
https://www.someserver.com/processes/myProcess/execution?
inputImage=s3://mybucker/myInputImage.tif&
bbox=5.8,47.2,15.1,55.1&
datetime=2019-08-09&
output=image&
f=image%2Fjpeg%3B%20q%3D0.5%2Cimage%2Fpng%20q%3D0.8
```

14.2.2.2. Prefer parameter

Table 16

Requirement 3	/req/kvp-execute/prefer-definition

A	The operation SHALL support a parameter `prefer` with the following characteristics:
	`name: prefer`
	`in: query`
	`required: false`
	`schema:`
	`  type: string`
	`  format: uri`
	`style: form`
	`explode: false`

Table 17

Requirement 4	/req/kvp-execute/f-response
A	The `prefer` parameter SHALL behave in the same way that the HTTP `Prefer` header behaves.

Example — Asynchronous execution.: In this example, the `prefer` parameter is used to indicate that asynchronous execution is prefered.

```
https://www.someserver.com/processes/myProcess/execution?
inputImage=s3://mybucker/myInputImage.tif&
bbox=5.8,47.2,15.1,55.1&
datetime=2019-08-09&
output=image&
f=image%2Fjpeg%3B%20q%3D0.5%2Cimage%2Fpng%20q%3D0.8&
prefer=handling%3Dlenient%2C%20wait%3D100%2C%20respond-async
```

14.2.2.3. Process inputs

14.2.2.3.1. Overview

REQUIREMENT 84	
IDENTIFIER	`/req/kvp-execute/input-query-parameters`
A	Each input defined in a process description SHALL map to a URI query parameter on the process execution endpoint.
B	The name of this URI query parameter SHALL be the identifier of the corresponding process input as defined in the process description.
C	The name of this URI query parameter SHALL be case in-sensitive. (??)

REQUIREMENT 85	
IDENTIFIER	`/req/kvp-execute/input-query-parameter-values`
STATEMENT	The server SHALL support process input values specified in-line as the value of the corresponding URI query parameter in the KVP-encoded execute request (i.e. by value).
A	The server SHALL support process input values specified by reference using a link as the value of the corresponding URI query parameter in the KVP-encoded execute request (i.e. by reference).

The value of an input URI query parameter in a KVP-encoded execute request can be:

- a simple value,
- a complex value,
- a binary value,
- a bounding box value,
- an array of values,
- or a reference to a value.

14.2.2.3.2. Simple string input

REQUIREMENT 86	
IDENTIFIER	`/req/kvp-execute/string-input-value`
STATEMENT	1. The process input value is specified in-line in an execute request. 2. A process input, with identifier `{input-name}`, is defined as type `string` in the process description.

A	The input parameter SHALL have the following characteristics:
	`name: {input-name}`
	`in: query`
	`required: false`
	`schema:`
	`  type: string`
	`style: form`
	`explode: false`
	where the token `{input-name}` represents the identifier of the input.

PERMISSION 13	
IDENTIFIER	`/per/kvp-execute/string-input-value`
A	The characteristics of a string-valued input parameter MAY include additional validation keywords (i.e. `maxLength`, `minLength`, `pattern`).
B	The characteristics of a string-valued input parameter MAY include the `format` validation keyword.
C	The characteristics of a string-valued input parameter MAY include additional keywords for string-encoded data (i.e. `contentEncoding`, `contentMediaType`, `contentSchema`).
D	The characteristics of a string-valued input parameter MAY include additional validation keywords (i.e. `enum`, `const`) that apply to any type.
E	The characteristics of a string-valued input parameter MAY include basic meta-data annotations (i.e. `title` and `description`, `default`, `deprecated`, `readOnly`, `writeOnly` and `examples`.

Example — Simple string input examples.: A string literal:

```
stringInput=String+value
```

A date string:

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

14.2.2.3.3. Simple numeric input

REQUIREMENT 87	
IDENTIFIER	`/req/kvp-execute/numeric-input-value`
STATEMENT	1. The process input value is specified in-line in an execute request. 2. A process input, with identifier `{input-name}`, is defined as a numeric type input in the process description.

A	The input parameter SHALL have the following characteristics:
	`name: {input-name}`
	`in: query`
	`required: false`
	`schema:`
	`  type: {numeric-type}`
	`style: form`
	`explode: false`
	where the token `{input-name}` represented the identifier of the input and the token {numeric-type} is the value `number` or `integer`.

PERMISSION 14	
IDENTIFIER	`/per/kvp-execute/numeric-input-value`
A	The characteristics of a numeric input parameter MAY include additional validation keywords (i.e. `multipleOf`, `maximum`, `exclusiveMaximum`, `minimum`, `exclusiveMinimum`).
B	The characteristics of a numeric input parameter MAY include additional validation keywords (i.e. `enum`, `const`) that apply to any type.
C	The characteristics of a numeric input parameter MAY include basic meta-data annotations (i.e. `title`, `description`, `default`, `deprecated`, `readOnly`, `writeOnly` and `examples`).

A number:

```
numberInput=3.14159
```

Figure 42

An integer:

```
integerInput=10
```

Figure 43

14.2.2.3.4. Simple boolean input

REQUIREMENT 88	
IDENTIFIER	`/req/kvp-execute/boolean-input-value`
STATEMENT	1. The process input value is specified in-line in an execute request. 2. A process input, with identifier `{input-name}` is defined as type `boolean` in the process description.

A	The input parameter SHALL have the following characteristics:
	`name: {input-name}`
	`in: query`
	`required: false`
	`schema:`
	`  type: boolean`
	`style: form`
	`explode: false`
	where the token `{input-name}` represented the identifier of the input.

PERMISSION 15	
IDENTIFIER	`/per/kvp-execute/boolean-input-value`

PERMISSION 15	
A	The characteristics of a boolean-valued input parameter MAY include additional validation keywords (i.e. `enum`, `const`) that apply to any type.
B	The characteristics definition of a boolean-valued input parameter MAY include basic meta-data annotations (i.e. `title` and `description`, `default`, `deprecated`, `readOnly`, `writeOnly` and `examples`.

A Boolean:

```
booleanInput=true
```

Figure 45

14.2.2.3.5. Complex-valued input

A complex value is a value with structure that is defined using a schema. The schema of a complex process input value is defined in the process description and can be specified using JSON-Schema.

REQUIREMENT 89	
IDENTIFIER	`/req/kvp-execute/complex-input-value`
STATEMENT	1. The process input value is specified in-line in an execute request. 2. A process input, with identifier `{input-name}`, is defined as type `object` in the process description.
A	Complex-valued inputs SHALL be encoded as URL-encoded string-data.

B	The input parameter SHALL have the following characteristics:
	`name: {input-name}`
	`in: query`
	`required: false`
	`schema:`
	`  type: string`
	`style: form`
	`explode: false`
	where the token `{input-name}` represents the identifier of the input.

PERMISSION 16	
IDENTIFIER	`/per/kvp-execute/complex-input-value`
A	The characteristics of a complex-valued input parameter MAY include additional keywords for string-encoded data (i.e. `contentEncoding`, `contentMediaType`, `contentSchema`).

PERMISSION 16	
B	The characteristics of a complex-valued input parameter MAY include additional validation keywords (i.e. `enum`, `const`) that apply to any type.
C	The characteristics of a complex-valued input parameter MAY include basic meta-data annotations (i.e. `title` and `description`, `default`, `deprecated`, `readOnly`, `writeOnly` and `examples`.

A complex-valued input parameter can be encoded as JSON, XML or some other text encoding of complex values such as Well Know Text (WKT).

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

```
complexObjectInput=%22value%22%3A%7B%22property1%22%3A%22value1%22%2C%22propert
y2%22%3A%22value2%22%2C%22property3%22%3A%22value3%22%7D%7D
Decoded value is:
{"value":{"property1":"value1","property2":"value2","property3":"value3"}}
```

14.2.2.3.6. Array-valued input

REQUIREMENT 90	
IDENTIFIER	`/req/kvp-execute/array-input-value`
STATEMENT	1. The process input value is specified in-line in an execute request. 2. A process input, with identifier `{input-name}`, is defined as type `array` in the process description.
A	Array-valued inputs SHALL be encoded as URL-encoded string-data.

B	The input parameter SHALL have the following characteristics:
	`name: {input-name}`
	`in: query`
	`required: false`
	`schema:`
	`  type: string`
	`style: form`
	`explode: false`
	where the token `{input-name}` represents the identifier of the input.

The elements of an array input value can be:

- a simple value,
- a complex value,
- a binary value,
- a bounding box value,
- an embeded array,

- or references to values using links.

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

```
arrayOfSimpleValues=%5B1%2C2%2C4%2C10%2C7%5D
Decoded value is:
arrayOfSimpleValues=[1,2,4,10,7]
```

An array with a single simple value:

```
arrayOfSimpleValues=%5Ba%5D

Decoded value is:
arrayOfSimpleValues=[a]
```

An array of complex values:

```
arrayOfQualifiedValues=%5B%7B%22value%22%3A%7B%22measurement%22%3A10.3%2C%22uo
m%22%3A%22m%22%2C%22reference%22%3A%22https%3A%2F%2Fucum.org%2Fucum-essence.xm
l%22%7D%7D%2C%7B%22value%22%3A%7B%22measurement%22%3A10.5%2C%22uom%22%3A%22m%22
%2C%22reference%22%3A%22https%3A%2F%2Fucum.org%2Fucum-essence.xml%22%7D%7D%2C%7
B%22value%22%3A%7B%22measurement%22%3A10.9%2C%22uom%22%3A%22m%22%2C%22reference
%22%3A%22https%3A%2F%2Fucum.org%2Fucum-essence.xml%22%7D%7D%5D

Decoded value is:
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"
    }
  }
]
```

An array of bounding box values:

```
bboxes=%7B%22bbox%22%3A%5B-160.2871383684127%2C21.77618201427491%2C-160.052267
32350857%2C22.035461193553438%5D%7D%2C%7B%22bbox%22%3A%5B-159.8119271016866%2C2
1.868377883379342%2C-159.2728781199529%2C22.27253365936666%5D%7D%2C%7B%22bbox%2
2%3A%5B-158.2809447669924%2C21.25785072496643 5%2C-157.62449293560164%2C21.7571
9228424245%5D%7D%2C%7B%22bbox%22%3A%5B-157.3665805976117%2C21.056725789376443%2
C-156.68744814385997%2C21.207484214479813%5D%7D%2C%7B%22bbox%22%3A%5B-157.07214
66133514%2C20.73998198222469%2C-156.8016490422814 4%2C20.958125881743094%5D%7D%2
C%7B%22bbox%22%3A%5B-156.67924097602452%2C20.599995119588588%2C-155.92896964790
82%2C21.048822845802146%5D%7D%2C%7B%22bbox%22%3A%5B-156.72917922429528%2C20.492
```

```
09982653643%2C-156.53052477930564%2C20.635403336088483%5D%7D%2C%7B%22bbox%22%3A
%5B-156.07415148465623%2C18.89763704552276%2C-154.87280121907844%2C20.334219894
95902%5D%7D
Decoded value is:
bboxes=[
  {
    "bbox":[-160.2871383684127,21.77618201427491,
            -160.05226732350857,22.035461193553438]
  },
  {
    "bbox":[-159.8119271016866,21.868377883379342,
            -159.2728781199529,22.27253365936666]
  },
  {
    "bbox":[-158.2809447669924,21.257850724966435,
            -157.62449293560164,21.75719228424245]
  },
  {
    "bbox":[-157.3665805976117,21.056725789376443,
            -156.68744814385997,21.207484214479813]
  },
  {
    "bbox":[-157.0721466133514,20.73998198222469,
            -156.80164904228144,20.958125881743094]
  },
  {
    "bbox":[-156.67924097602452,20.599995119588588,
            -155.9289696479082,21.048822845802146]
  },
  {
    "bbox":[-156.72917922429528,20.49209982653643,
            -156.53052477930564,20.635403336088483]
  },
  {
    "bbox":[-156.07415148465623,18.89763704552276,
            -154.87280121907844,20.33421989495902]
  }
]
```

An array of arrays:

```
arrayOfArrays=%5B%5B1%2C2%2C3%2C4%5D%2C%5B%22a%22%2C%22b%22%2C%22c%22%2C%22d%22
%5D%5D
Decoded value is:
arrayOfArrays=[[1,2,3,4],["a","b","c","d"]]
```

An array of value references:

```
images=%5B%7B%22href%22%3A%22http%3A%2F%2Fwww.imagearchive.com%2Fimages
%2Fimage01.tif%22%2C%22type%22%3A%22image%2Ftiff%3Bapplcation%3Dgeotiff%22%7D%2
C%7B%22href%22%3A%22http%3A%2F%2Fwww.imagearchive.com%2Fimages%2Fimage19.jp2%2C
%22type%22%3A%22image%2Fjp2%22%7D%5D
Decoded value is:
images=[
  {
    "href":"http://www.imagearchive.com/images/image01.tif",
    "type":"image/tiff;applcation=geotiff"
  },
  {
    "href":"http://www.imagearchive.com/images/image19.jp2,
    "type":"image/jp2"
  }
```

```
]
```

14.2.2.3.7. Binary-valued input

NOTE: Not sure we should specify this. Binary input values should be by reference only in my opinion. It really makes no sense to specify a binary value by-value in a URL-encoded execute request. Does it? Something small like icons … maybe?

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.

REQUIREMENT 91

IDENTIFIER	`/req/kvp-execute/binary-input-value`
STATEMENT	1. The process input value is specified in-line in an execute request. 2. A process input, with identifier `{input-name}`, is defined as binary type in the process description. 3. The input value does not need to be qualified with a format specification.

A	The input parameter SHALL have the following characteristics:
	`name: {input-name}`
	`in: query`
	`required: false`
	`schema:`
	`  type: string`
	`  format: byte`
	`style: form`
	`explode: false`
	where the token `{input-name}` represents the identifier of the input.

A binary value can be optionally qualified with a format parameter. This is usually done to identify one of a number of possible input types for the specified input parameter.

REQUIREMENT 92

IDENTIFIER	`/req/kvp-execute/binary-input-value-qualified`
STATEMENT	1. The process input value is specified in-line in an execute request. 2. A process input, with identifier `{input-name}`, is defined as binary type in the process description. 3. The input value needs to qualified with a format specification.

A	The input parameter SHALL have the following characteristics:
	`name: {input-name}`
	`in: query`
	`required: false`
	`schema:`
	`  type: object`

REQUIREMENT 92

```
            required:
              - value
            properties:
              value:
                type: string
                format: byte
              mediaType:
                type: string
              encoding:
                type: string
              schema:
                oneOf:
                  - type: string
                    format: url
                  - type: object
          style: deepObject
          explode: true
```

where the token `{input-name}` represents the identifier of the input.

Example — 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": "image/tiff; application=geotiff"
}
```

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

```
imageInput="R0lGODdhNAHCAfcAAAcHDD+Gs4sLDQpDaqGFdaHE54dJPEoECUlGRteKgcdITgokG
4hoVkpY\ngNzHwKKkqOLm7RRjlEgpHU9iZ44lHQYqVdmki6doVmhHOMOIeJG20HiDjCcKBglIeadI
Srso\nJGooFNbN2d2qr8aljyklHwQJQkdvkWaKxIdrb442LidLeGhMTp6LkeP1+Kh3aiUuVAoUHm
lu\ngkcwNYdZRmkJDYGcsDFokElVYyk1NsWWhLEPDtmQldrUyoyFhrjo+Nna5d+4tMGstspoXgc4
\n...qgu7sSu7qbtCs2u7t6u6rLsrp4u7veu76eO6vyu8w0u8xWu8x4u8yau8shu8y+u8zwu90Su9
\n00u91Wu914u92au928u9whsQADs="
```

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

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

```
imageInput[value]=
VBORw0KGgoAAAANSUhEUgAABvwAAAa4CAYAAABMB35kAAABhGlDQ1BJQ0MgcHJvZmlsZQAA
\nKJF9kT1Iw0AcxV9TpSL1A+xQxCFDdbIgKuKoVShChVArtOpgcumH0KQhSXFxFFwLDn4sVh1c
\nnHV1cBUEwQ8QNzcnRRcp8X9JoUWMB8f9eHfvcfcOEOplplkdY4Cm22Y6mRCzuRUx9IogouhH\n
... \nj3Z5mX7/PCPVRJV92rpHK24xcJrzk20+tkeYlCPqcZNO3Lpni1OJWatPCcmgGDEqx7Om6lfa
\nppM4k4BTe9+bsn3L9/9/yWhA0PwQGW8ipCZsnZt9lsdrYEM8z/M8z/M8z/M8z/M8z/
MzLWY1\nAAAACUlEQVQ871H6P6JI+TxS5Wn2AAAAAElFTkSuQmCC&imageInput[mediaType]=
image/jp2
```

14.2.2.3.8. Bounding box-valued input

REQUIREMENT 93	
IDENTIFIER	/req/kvp-execute/bbox-input-value
STATEMENT	1. The process input value is specified in-line in an execute request. 2. The process input parameter is defined as a bbox in the process description.

A	A bbox input parameter SHALL have the following characteristics:
	`name: {bbox-parameter-name}`
	`in: query`
	`description: |-`
	`  Only features that have a geometry that intersects the bounding box are selected.`
	`  The bounding box is provided as four or six numbers, depending on whether the`
	`  coordinate reference system includes a vertical axis (height or depth):`
	
	`  * Lower left corner, coordinate axis 1`
	`  * Lower left corner, coordinate axis 2`
	`  * Minimum value, coordinate axis 3 (optional)`
	`  * Upper right corner, coordinate axis 1`
	`  * Upper right corner, coordinate axis 2`
	`  * Maximum value, coordinate axis 3 (optional)`
	
	`  If the value consists of four numbers, the coordinate reference system is`
	`  WGS 84 longitude/latitude (http://www.opengis.net/def/crs/OGC/1.3/CRS84)`
	`  unless a different coordinate reference system is specified in the parameter `bbox-crs`.`
	
	`  If the value consists of six numbers, the coordinate reference system is WGS 84`
	`  longitude/latitude/ellipsoidal height (http://www.opengis.net/def/crs/OGC/0/CRS84h)`
	`  unless a different coordinate reference system is specified in the parameter `bbox-crs`.`
	
	`  The query parameter `bbox-crs` is specified in OGC API - Features - Part 2: Coordinate`
	`  Reference Systems by Reference.`
	
	`  For WGS 84 longitude/latitude the values are in most cases the sequence of`
	`  minimum longitude, minimum latitude, maximum longitude and maximum latitude.`
	`  However, in cases where the box spans the antimeridian the first value`
	`  (west-most box edge) is larger than the third value (east-most box edge).`
	
	`  If the vertical axis is included, the third and the sixth number are the`

REQUIREMENT 93

```
    bottom and the top of the 3-dimensional bounding box.

    If a feature has multiple spatial geometry properties, it is the
  decision of the
    server whether only a single spatial geometry property is used to
  determine
    the extent or all relevant geometries.
  required: false
  schema:
    type: array
    oneOf:
    - minItems: 4
      maxItems: 4
    - minItems: 6
      maxItems: 6
    items:
      type: number
  style: form
  explode: false
```

where the token `{bbox-input-name}` represented the identifier of the input.

REQUIREMENT 94

IDENTIFIER	`/req/kvp-execute/bbox-crs-input-value`
STATEMENT	1. The server supports CRSs other than WGS84. 2. The process input value is specified in-line in an execute request. 3. The process defines an input parameter named `{bbox-input-name}`.

A	A bbox-crs input parameter SHALL have the following characteristics:
	`name: {bbox-parameter-name}-crs`
	`description: |-`
	`  Asserts the CRS used for the coordinate values of the bbox parameter.`
	`in: query`
	`required: false`
	`schema:`
	`  type: string`
	`  format: uri`
	`  enum:`
	`  - http://www.opengis.net/def/crs/OGC/1.3/CRS84`
	`  default:`
	`  - http://www.opengis.net/def/crs/OGC/1.3/CRS84`
	`style: form`
	`explode: false`
	where the token `{bbox-input-name}` represents the identifier of a bbox input parameter and the token `{bbox-input-name}-crs` represents the identifier of a corresponding input that indicates which CRS is being used for the coordinates of the `{bbox-input-name}` parameter.

Example — Bounding box input examples.: An example instance value for a bounding box input named `my_bbox` might be:

```
my_bbox=-79.63732855116733,43.570691463538644,-79.10227279076784,43.86582298161
152
```

This is the same example as above but in a different CRS. An input parameters names `my_bbox-crs` is used to convey the CRS of the `my_bbox` parameter.

```
my_bbox=43.570691463538644,-79.63732855116733,43.86582298161152,-
79.10227279076784&my_bbox-crs=https://www.opengis.net/def/crs/EPSG/0/4326
```

14.2.2.3.9. Input parameters value by reference

REQUIREMENT 95	
IDENTIFIER	`/req/kvp-execute/input-by-reference`
STATEMENT	1. The process input value is specified by reference in an execute request.

A	If the input cardinality is 1 then the input parameter SHALL have the following characteristics:
	`name: {input-name}`
	`in: query`
	`required: false`
	`schema:`
	`  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`
	`style: deepObject`
	`explode: true`
	where the token `{input-name}` represented the identifier of the input.

B	If the input cardinality is greater than 1 then the value of the input parameter SHALL be encoded as a link object and appropriately URL-encoded to a string.

Example — Input-by-reference example

```
imageInput[href]=http://www.someserver.com/image.tiff&#x26;imageInput[type]=
image/tiff;application=geotiff
```

14.2.2.4. Input cardinality

REQUIREMENT 96

IDENTIFIER	`/req/kvp-execute/input-cardinality`
STATEMENT	1. The process input value is specified in-line in an execute request. 2. A process input, with identifier `{input-name}`, is defined as having a cardinality greater than one (i.e. `minOccurs` > 1) in the process description. 3. The number of input values specified for the `{input-name}` process input is greater than one.
A	A process input having more that one value SHALL be encoded as a list of values according to the query parameter serialization rules of the OpenAPI Specification v3.0.3. The default list value separator is the comma (“,”) but other values are possible too.

B	The input parameter SHALL have the following characteristics:
	`name: {input-name}`
	`in: query`
	`required: false`
	`schema:`
	`  type: array`
	`  items:`
	`  oneOf:`
	`  - type: string`
	`  - type: number`
	`  - type: integer`
	`  - type: boolean`
	`  - type: null`
	`style: form`
	`explode: false`
	where the token `{input-name}` represents the identifier of the input.

C	List elements that are objects SHALL be encoded as URL-encoded strings.
D	List elements that are arrays SHALL be encoded as URL-encoded strings.

Example — Input value with cardinality greater than 1: An example of a string-value input with cardinality greater than 1.

```
...&color=blue,green,red,yellow&...
```

An example of a object-valued input with cardinality greater than 1. In this case, two geometries are specified as input. The first geometry is encoded as GML object and the second geometry is encoded as GeoJSON object.

```
...&geometryInput=%7B%22value%22%3A%22%3Cgml%3APolygon%20gml%3Aid%3D%5
C%22GID1%5C%22%20srsName%3D%5C%22urn%3Aogc%3Adef%3Acrs%3AOGC%3A%3ACRS84
%5C%22%3E%3Cgml%3Aexterior%3E%3Cgml%3ALinearRing%3E%3Cgml%3AposList%3E-
77.024519%2038.810529%20-77.024635%2038.810973%20-77.024704%2038.810962%20-
77.024776%2038.811239%20-77.024957%2038.81121%20-77.024905%2038.811012%20-
77.024905%2038.811012%20-77.024865%2038.810857%20-77.025024%2038.810832%20-
77.025071%2038.811012%20-77.025203%2038.810992%20-77.02506%2038.810444%20-
77.024519%2038.810529%3C%2Fgml%3AposList%3E%3C%2Fgml%3ALinearRing%3E%3C%2F
gml%3Aexterior%3E%3C%2Fgml%3APolygon%3E%22%2C%22mediaType%22%3A%22applicat
ion%2Fgml%2Bxml%3B%20version%3D3.2%22%7D,%7B%22value%22%3A%7B%22type%22%3A
%22Polygon%22%2C%22coordinates%22%3A%5B%5B%5B-176.5814819%2C-44.10896301%5D
%2C%5B-176.5818024%2C-44.10964584%5D%2C%5B-176.5844116%2C-44.11236572%5D
%2C%5B-176.5935974%2C-44.11021805%5D%2C%5B-176.5973511%2C-44.10743332%5D%2C
%5B-176.5950928%2C-44.10562134%5D%2C%5B-176.5858459%2C-44.1043396%5D%2C%5B-
```

```
176.5811157%2C-44.10667801%5D%2C%5B-176.5814819%2C-44.10896301%5D%5D%5D%7D%2C%2
2mediaType%22%3A%22application%2Fgeo%2Bjson%22%7D%0A&....

Decoded value is:
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"
  }
```

An example of an array-value input with cardinality greater than 1.

```
...&param1=%5B1%2C2%2C3%5D,%5B6%2C7%2C8%5D&...

Decoded value is:
param1=[1,2,3],[6,7,8]
```

14.2.2.5. Process outputs

REQUIREMENT 97

IDENTIFIER	`/req/kvp-execute/output`
STATEMENT	1. A process output, with identifier `{output-name}`, is defined in the process description.

A	The output parameter SHALL have the following characteristics:
	`name: {output-name}`
	`in: query`
	`required: false`
	`schema:`
	`  type: object`
	`  required:`
	`  - include`
	`  properties:`
	`  include`
	`  type: boolean`
	`  mediaType:`
	`  type: string`
	`  encoding:`
	`  type: string`
	`  schema:`

REQUIREMENT 97

```
                oneOf:
                  - type: string
                    format: url
                  - type: object
          style: deepObject
          explode: true
```

where the token `{output-name}` represents the identifier of the output.

Example — Output examples

```
...&out1[include]=true&out2[include]true&out2[include]true&...
```

```
...&out1[include]=true&out1[mediaType]=application/geo+tiff&...
```

14.3. Response

Whether a processes is invoked using a JSON-encoded request that is HTTP POST'ed to the execution endpoint or using a URL-encoded request, the behaviour of the server with regard to the response is the same. The details of the response can be found in Clause 7.11.4.

15

MEDIA TYPES

15 MEDIA TYPES

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

- `application/json` for all resources.

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

- `text/html`.

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

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

16

ADDITIONAL API BUILDING BLOCKS

16 ADDITIONAL API BUILDING BLOCKS

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

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

This workflow is useful for generic 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 17	
LABEL	/per/core/alternative-process-description
STATEMENT	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 18	
LABEL	/per/core/alternative-process-paths
STATEMENT	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 55

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	
IDENTIFIER	`/req/core/test-process`
STATEMENT	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 Recommendation A.1 for further guidance.

RECOMMENDATION A.1	
STATEMENT	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: 1. An Echo process that returns any input that is provided, without any actual processing. 2. 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: CORE	
IDENTIFIER	`http://www.opengis.net/spec/ogcapi-processes-1/1.0/conf/core`
SUBJECT	http://www.opengis.net/spec/ogcapi-processes-1/1.0/conf/core
TARGET TYPE	Web API
CONFORMANCE TESTS	Requirement A.2: `/conf/core` Abstract test A.3: `/conf/core/api-definition-op` Abstract test A.4: `/conf/core/api-definition-success` Abstract test A.5: `/conf/core/conformance-op` Abstract test A.6: `/conf/core/conformance-success` Abstract test A.7: `/conf/core/http` Abstract test A.35: `/conf/core/job-exception-no-such-job` Abstract test A.33: `/conf/core/job-op` Abstract test A.36: `/conf/core/job-result` Abstract test A.37: `/conf/core/job-results` Abstract test A.43: `/conf/core/job-results-async-many` Abstract test A.42: `/conf/core/job-results-async-one` Abstract test A.44: `/conf/core/job-results-exception-no-such-job` Abstract test A.45: `/conf/core/job-results-exception-results-not-ready` Abstract test A.46: `/conf/core/job-results-failed` Abstract test A.38: `/conf/core/job-results-param-outputs` Abstract test A.41: `/conf/core/job-results-param-outputs-empty` Abstract test A.40: `/conf/core/job-results-param-outputs-omit` Abstract test A.39: `/conf/core/job-results-param-outputs-response` Abstract test A.31: `/conf/core/job-results-success-sync` Abstract test A.34: `/conf/core/job-success` Abstract test A.1: `/conf/core/landingpage-op` Abstract test A.2: `/conf/core/landingpage-success` Abstract test A.9: `/conf/core/pl-limit-definition` Abstract test A.10: `/conf/core/pl-limit-response` Abstract test A.12: `/conf/core/pl-links` Abstract test A.13: `/conf/core/process-description` Abstract test A.14: `/conf/core/process-description-success` Abstract test A.15: `/conf/core/process-exception-no-such-process` Abstract test A.26: `/conf/core/process-execute-auto-execution-mode` Abstract test A.25: `/conf/core/process-execute-default-execution-mode` Abstract test A.27: `/conf/core/process-execute-default-outputs` Abstract test A.19: `/conf/core/process-execute-input-array` Abstract test A.23: `/conf/core/process-execute-input-inline-bbox` Abstract test A.22: `/conf/core/process-execute-input-inline-binary`

CONFORMANCE CLASS A.1: CORE	
	Abstract test A.21: /conf/core/process-execute-input-inline-mixed Abstract test A.20: /conf/core/process-execute-input-inline-object Abstract test A.18: /conf/core/process-execute-inputs Abstract test A.24: /conf/core/process-execute-input-validation Abstract test A.16: /conf/core/process-execute-op Abstract test A.17: /conf/core/process-execute-request Abstract test A.32: /conf/core/process-execute-success-async Abstract test A.30: /conf/core/process-execute-sync-many-json Abstract test A.28: /conf/core/process-execute-sync-one Abstract test A.29: /conf/core/process-execute-sync-one-default-content Abstract test A.8: /conf/core/process-list Abstract test A.11: /conf/core/process-list-success

A.3. Abstract test suite

REQUIREMENT A.2	
IDENTIFIER	/conf/core
TARGET	Requirement A.1: /req/core/test-process
INCLUDED IN	Conformance class A.1: http://www.opengis.net/spec/ogcapi-processes-1/1.0/conf/core
TEST PURPOSE	Ensure that a testable process is offered by the server being tested.
TEST METHOD	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 Recommendation A.1 for further guidance.

A.4. Retrieve the API landing page

ABSTRACT TEST A.1	
IDENTIFIER	/conf/core/landingpage-op

ABSTRACT TEST A.1	
REQUIREMENT	Requirement 1: `/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	
IDENTIFIER	`/conf/core/landingpage-success`
REQUIREMENT	Requirement 2: `/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 3. For formats that require manual inspection, perform the following: a) Validate that the landing page includes a “service-desc” and/or “service-doc” link to an API Definition. b) Validate that the landing page includes a “http://www.opengis.net/def/rel/ogc/1.0/conformance” link to the conformance class declaration. c) Validate that the landing page includes a “http://www.opengis.net/def/rel/ogc/1.0/processes” link to the list of processes.

NOTE: 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/json/content

A.5. Retrieve an API definition

ABSTRACT TEST A.3	
IDENTIFIER	`/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	
IDENTIFIER	`/conf/core/api-definition-success`
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.6. Declaration of conformance classes

ABSTRACT TEST A.5	
IDENTIFIER	`/conf/core/conformance-op`
REQUIREMENT	Requirement 5: `/req/core/conformance-op`
TEST PURPOSE	Validate that a Conformance Declaration can be retrieved from the expected location.

ABSTRACT TEST A.5	
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	
IDENTIFIER	`/conf/core/conformance-success`
REQUIREMENT	Requirement 6: `/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” 4. Validate that the document list all OGC API conformance classes that the API implements.

A.7. Use of HTTP 1.1

ABSTRACT TEST A.7	
IDENTIFIER	`/conf/core/http`
REQUIREMENT	Requirement 7: `/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.8. Retrieve a process list

ABSTRACT TEST A.8	
IDENTIFIER	`/conf/core/process-list`
REQUIREMENT	Requirement 8: `/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	
IDENTIFIER	`/conf/core/pl-limit-definition`
REQUIREMENT	Requirement 9: `/req/core/pl-limit-definition`
TEST PURPOSE	Validate that the `limit` query parameter is constructed correctly.
TEST METHOD	Verify that the `limit` query parameter complies with its definition in requirement /req/core/pl-limit-definition.

NOTE 1: The API can define different values for “minimum”, “maximum” and “default”.

ABSTRACT TEST A.10	
IDENTIFIER	`/conf/core/pl-limit-response`
REQUIREMENT	Requirement 10: `/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.

ABSTRACT TEST A.10	
	4. If the API definition specifies a maximum value for `limit` parameter, verify that the count does not exceed this maximum value.

ABSTRACT TEST A.11	
IDENTIFIER	`/conf/core/process-list-success`
REQUIREMENT	Requirement 11: `/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

NOTE 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.12	
IDENTIFIER	`/conf/core/pl-links`
REQUIREMENT	Requirement 12: `/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. 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.

A.9. Retrieve a process description

ABSTRACT TEST A.13	
IDENTIFIER	`/conf/core/process-description`
REQUIREMENT	Requirement 13: `/req/core/process-description`
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/{processID}` where `{processID}` is the `id` property for the process. 2. Validate the response using the test /conf/core/process-description-success.

ABSTRACT TEST A.14	
IDENTIFIER	`/conf/core/process-description-success`
REQUIREMENT	Requirement 14: `/req/core/process-description-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.

NOTE: 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.10. Process exception

ABSTRACT TEST A.15	
IDENTIFIER	`/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

NOTE: 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 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.11. Process execution /processes/{processID}/execution

ABSTRACT TEST A.16

IDENTIFIER	`/conf/core/process-execute-op`
REQUIREMENT	Requirement 16: `/req/core/process-execute-op`
TEST PURPOSE	Validate the execution of a process.
TEST METHOD	1. Issue an HTTP POST request to the URL '/processes/{processID}/execution' for each execution mode according to requirements /conf/core/process-execute-default-execution-mode or /conf/core/process-execute-auto-execution-mode. 2. Validate that the server supports a POST request for that operation according to the test /conf/core/process-execute-request. 3. Validate the execution of a process according to the requirements /req/core/process-execute-default-execution-mode, /req/core/process-execute-auto-execution-mode.

ABSTRACT TEST A.17

IDENTIFIER	`/conf/core/process-execute-request`
REQUIREMENT	Requirement 17: `/req/core/process-execute-request`
TEST PURPOSE	Validate that the server supports a process execution operation complying with the required structure and contents.
TEST METHOD	Verify that the server supports a process execution request whose body conforms to the OpenAPI 3.0 schema execute.yaml.

ABSTRACT TEST A.18

IDENTIFIER	`/conf/core/process-execute-inputs`
REQUIREMENT	Requirement 18: `/req/core/process-execute-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/process-execute-input-inline-binary, /conf/core/process-execute-input-inline-mixed, /conf/core/process-execute-input-inline-object, /conf/core/process-execute-input-inline-bbox . 2. For each test, specify the input value in-line in the execute request. 3. For each test, specify the input value by reference using a link in the execute request.

ABSTRACT TEST A.19

IDENTIFIER	`/conf/core/process-execute-input-array`
REQUIREMENT	Requirement 19: `/req/core/process-execute-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-description. 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/process-execute-request taking care to encode the inputs with maximum cardinality > 1 according to the requirement /req/core/process-execute-input-array. 4. Verify that each process executes successfully according to the relevant requirement (one of: /req/core/process-execute-success-async, /req/core/process-execute-sync-many-json, /req/core/process-execute-sync-one, /req/core/process-execute-sync-one-default-content) based on the combination of execute parameters.

ABSTRACT TEST A.20

IDENTIFIER	`/conf/core/process-execute-input-inline-object`
REQUIREMENT	Requirement 20: `/req/core/process-execute-input-inline-object`
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-description. 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/process-execute-request taking care to encode the identified object inputs in-line in the execute request according to requirement /req/core/process-execute-input-inline-object. 4. Verify that each process execute successfulle according to the relevant requirement (one of: /req/core/process-execute-success-async, /req/core/process-execute-sync-many-json, /req/core/process-execute-sync-one, /req/core/process-execute-sync-one-default-content) based on the combination of execute parameters.

ABSTRACT TEST A.21

IDENTIFIER	`/conf/core/process-execute-input-inline-mixed`

ABSTRACT TEST A.21

REQUIREMENT	/req/core/process-execute-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-description. 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/process-execute-request taking care to encode the identified mix-content inputs in-line in the execute request according to requirement /req/core/process-execute-input-inline-mixed. 4. Verify that each process execute successfully according to the relevant requirement (one of: /req/core/process-execute-success-async, /req/core/process-execute-sync-many-json, /req/core/process-execute-sync-one, /req/core/process-execute-sync-one-default-content) base on the combination of execute parameters.

ABSTRACT TEST A.22

IDENTIFIER	`/conf/core/process-execute-input-inline-binary`
REQUIREMENT	Requirement 22: `/req/core/process-execute-input-inline-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-description. 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/process-execute-request taking care to encode binary input values in-line in the execute request according to requirement /req/core/process-execute-input-inline-binary. 4. Verify that each process execute successfully according to the relevant requirement (one of: /req/core/process-execute-success-async, /req/core/process-execute-sync-many-json, /req/core/process-execute-sync-one, /req/core/process-execute-sync-one-default-content) base on the combination of execute parameters.

ABSTRACT TEST A.23

IDENTIFIER	`/conf/core/process-execute-input-inline-bbox`
REQUIREMENT	Requirement 23: `/req/core/process-execute-input-inline-bbox`

ABSTRACT TEST A.23

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-description. 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/process-execute-request taking care to encode values for the identified bounding box inputs in-line in the execute request. 4. Verify that each process execute successfulle according to the relevant requirement (one of: /req/core/process-execute-success-async, /req/core/process-execute-sync-many-json, /req/core/process-execute-sync-one, /req/core/process-execute-sync-one-default-content) based on the combination of execute parameters.

ABSTRACT TEST A.24

IDENTIFIER	`/conf/core/process-execute-input-validation`
REQUIREMENT	Requirement 24: `/req/core/process-execute-input-validation`
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-description. 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/process-execute-request taking care to encode the input values according to the schema from the definition of each input. 4. Verify that each process execute successfully according to the relevant requirement (one of: /req/core/process-execute-success-async, /req/core/process-execute-sync-many-json, /req/core/process-execute-sync-one, /req/core/process-execute-sync-one-default-content) base on the combination of execute parameters. 5. For each process construct an execute request according to test /conf/core/process-execute-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).

ABSTRACT TEST A.25

IDENTIFIER	`/conf/core/process-execute-default-execution-mode`

ABSTRACT TEST A.25

REQUIREMENT	Requirement 16: `/req/core/process-execute-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-description. 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/process-execute-request. 4. For processes that are supposed to execute asynchronously according to the /req/core/process-execute-default-execution-mode requirement, verify the successful execution according to the /conf/core/process-execute-success-async test. 5. For processes that are supposed to execute synchronously according to the /req/core/process-execute-auto-execution-mode requirement, verify the successful synchronous execution according to the relevant requirement (one of: /req/core/process-execute-sync-many-json, /req/core/process-execute-sync-one, /req/core/process-execute-sync-one-default-content) base on the combination of execute parameters.

ABSTRACT TEST A.26

IDENTIFIER	`/conf/core/process-execute-auto-execution-mode`
REQUIREMENT	Requirement 16: `/req/core/process-execute-op`
TEST PURPOSE	Validate that the server correctly handles the execution mode for a process.
TEST METHOD	1. Get a description of each process offered by the server using test /conf/core/process-description. 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/process-execute-request. 4. For processes that are supposed to execute asynchronously according to the /req/core/process-execute-auto-execution-mode requirement, verify the successful execution according to the /req/core/process-execute-success-async test. 5. For processes that are supposed to execute synchronously according to the /req/core/process-execute-auto-execution-mode requirement, verify the successful execution according to the relevant requirement (one of: /req/core/process-execute-sync-many-json, /req/core/process-execute-sync-one, /req/core/process-execute-sync-one-default-content) based on the combination of execute parameters. 6. For processes that may execute either synchronously or asynchronously according to the /req/core/process-execute-auto-execution-mode requirement, verify the successful execution according to the relevant requirement (one of: /req/core/process-execute-success-async, /req/core/process-execute-sync-many-json, /req/core/process-execute-

ABSTRACT TEST A.26

	sync-one, /req/core/process-execute-sync-one-default-content) based on the combination of execute parameters.

ABSTRACT TEST A.27

IDENTIFIER	`/conf/core/process-execute-default-outputs`
REQUIREMENT	Requirement 16: `/req/core/process-execute-op`
TEST PURPOSE	Validate that the server correctly handles the case where no `outputs` parameter is specified on an execute request.
TEST METHOD	1. Get a description of each process offered by the server using test /conf/core/process-description. 2. Inspect the description of each process taking note of the definition of each process output. 3. For each process construct an execute request according to test /conf/core/process-execute-request taking care to omit the `outputs` parameter. 4. Verify that each process executes successfully according to the relevant requirement (one of: /req/core/process-execute-success-async, /req/core/process-execute-sync-many-json, /req/core/process-execute-sync-one, /req/core/process-execute-sync-one-default-content) based on the combination of execute parameters. 5. Verify that each process includes all the outputs, as defined in the process description, in the response.

ABSTRACT TEST A.28

IDENTIFIER	`/conf/core/process-execute-sync-one`
REQUIREMENT	Requirement 28: `/req/core/process-execute-sync-one`
TEST PURPOSE	Validate that the correct output is generated upon process execution when a single output is requested.
TEST METHOD	1. Negotiate synchronous process execution as per test /req/core/process-execute-op. 2. Request a single output with identifier `{outputID}` as per test /req/core/job-results-param-outputs. 3. Negotiate an output format that is supported for the requested output according to the process' description that has been retrieved as per test /req/core/process-description. 4. Validate that the response conforms to the requirements of /req/core/process-execute-sync-one.

ABSTRACT TEST A.29

IDENTIFIER	`/conf/core/process-execute-sync-one-default-content`
REQUIREMENT	Requirement 29: `/req/core/process-execute-sync-one-default-content`
TEST PURPOSE	Validate that the default output format is generated upon process execution when a single output is requested but no specific supported format is negotiated.
TEST METHOD	1. Negotiate synchronous process execution as per test /req/core/process-execute-op. 2. Request a single output with identifier `{outputID}` as per test /req/core/job-results-param-outputs. 3. Do not negotiate a specific format for the requested output. 4. Validate that the response conforms to the requirements of /req/core/process-execute-sync-one-default-content.

ABSTRACT TEST A.30

IDENTIFIER	`/conf/core/process-execute-sync-many-json`
REQUIREMENT	Requirement 30: `/req/core/process-execute-sync-many-json`
TEST PURPOSE	Validate that the correct output with the correct content type is generated when a process generating multiple responses is synchronously executed.
TEST METHOD	1. Negotiate synchronous process execution as per test /req/core/process-execute-op. 2. Request at least 2 process outputs as per test /req/core/job-results-param-outputs. 3. Validate that the response conforms to the requirements of /req/core/process-execute-sync-many-json.

ABSTRACT TEST A.31

IDENTIFIER	`/conf/core/job-results-success-sync`
REQUIREMENT	Requirement 31: `/req/core/job-results-success-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-description. 2. Inspect the description of each process and identify the subset of processes that support the `sync-execute` job control option.

ABSTRACT TEST A.31	
	3. For each identified process construct an execute request according to test /conf/core/process-execute-request ensuring that synchronous execution is negotiated according to test /conf/core/process-execute-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.32	
IDENTIFIER	`/conf/core/process-execute-success-async`
REQUIREMENT	Requirement 32: `/req/core/process-execute-success-async`
TEST PURPOSE	Validate the results of a job that has been created using the `async` execution mode.
TEST METHOD	1. Validate that the response returned with an HTTP status code of 201. 2. Validate that the response includes the HTTP `Location` header and that the link points to the newly created job. 3. Validate that the response body conforms to the JSON Schema fragment statusInfo.yaml.

A.12. Jobs

A.12.1. Job status /jobs/{jobID}

ABSTRACT TEST A.33	
IDENTIFIER	`/conf/core/job-op`
REQUIREMENT	/req/core/job-op
TEST PURPOSE	Validate that the status info of a job can be retrieved.
TEST METHOD	1. Create a job as per /req/core/process-execute-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.34	
IDENTIFIER	`/conf/core/job-success`
REQUIREMENT	Requirement 34: `/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

NOTE 1: 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.35	
IDENTIFIER	`/conf/core/job-exception-no-such-job`
REQUIREMENT	Requirement 35: `/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

NOTE 2: 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.12.2. Retrieve job results

ABSTRACT TEST A.36	
IDENTIFIER	`/conf/core/job-result`
REQUIREMENT	Requirement 36: `/req/core/job-result`
TEST PURPOSE	Validate that each process output with identifier `{outputID}` can be retrived from the `/jobs/{jobID}/results/{outputID}` endpoint.
TEST METHOD	1. Negotiate asynchronous process execution as per test /req/core/process-execute-op. 2. Validate that each process output can be retrived as per requirement /req/core/job-result.

ABSTRACT TEST A.37	
IDENTIFIER	`/conf/core/job-results`
REQUIREMENT	Requirement 37: `/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/process-execute-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. Depending on the number of outputs requested, the negotiated content type of the response and any client preferences, validate the contents of the returned document using the tests /conf/core/job-results-success-sync, /conf/core/job-results-async-one or /conf/core/job-results-async-many.

ABSTRACT TEST A.38

IDENTIFIER	`/conf/core/job-results-param-outputs`
REQUIREMENT	Requirement 38: `/req/core/job-results-param-outputs`
TEST PURPOSE	Validate that the `outputs` parameter is constructed correctly.
TEST METHOD	1. Verify that the `outputs` parameter complies with its definition in requirement req/core/job-results-param-outputs.

NOTE 1: The API can define different values for “minimum”, “maximum” and “default”.

ABSTRACT TEST A.39

IDENTIFIER	`/conf/core/job-results-param-outputs-response`
REQUIREMENT	Requirement 39: `/req/core/job-results-param-outputs-response`
TEST PURPOSE	Validate that only the requested processing results are included in the response when the `outputs` parameter is specified on an execution request.
TEST METHOD	Include the `outputs` parameter on a process execution request that enumerates a list of process outputs to include the response. Verify that the response includes the requested process outputs.

ABSTRACT TEST A.40

IDENTIFIER	`/conf/core/job-results-param-outputs-omit`
REQUIREMENT	Requirement 40: `/req/core/job-results-param-outputs-omit`
TEST PURPOSE	Validate that all processing results are included in the response when the `outputs` parameter is omitted from an execution request.
TEST METHOD	Omit the `outputs` parameter from a process execution request and verify that all processing results are included in the response.

ABSTRACT TEST A.41

IDENTIFIER	`/conf/core/job-results-param-outputs-empty`
REQUIREMENT	Requirement 41: `/req/core/job-results-param-outputs-empty`

ABSTRACT TEST A.41

TEST PURPOSE	Validate that no processing results are available when the `outputs` parameter is present in an execution request but is empty.
TEST METHOD	Verify that the server responds with a 204 HTTP status code and an empty response body when an output is requested.

ABSTRACT TEST A.42

IDENTIFIER	`/conf/core/job-results-async-one`
REQUIREMENT	Requirement 42: `/req/core/job-results-async-one`
TEST PURPOSE	Validate that only the requested processing result is included in the response when the `outputs` parameter requesting output with identifier `{outputID}` is specified on an execution request.
TEST METHOD	1. Negotiate asynchronous process execution as per test /req/core/process-execute-op. 2. Request a single process output with output identifier `{outputID}` as per test /req/core/job-results-param-outputs. 3. Retrieve the processing response from the `/jobs/{jobID}/results/{outputID}` endpoint. 4. Validate that the response conforms to the requirement /req/core/job-results-async-one.

ABSTRACT TEST A.43

IDENTIFIER	`/conf/core/job-results-async-many`
REQUIREMENT	Requirement 43: `/req/core/job-results-async-many`
TEST PURPOSE	Validate that only the requested processing results are included in the response when the `outputs` parameter is specified on an execution request.
TEST METHOD	1. Negotiate asynchronous process execution as per test /req/core/process-execute-op. 2. Request at least 2 process outputs as per test /req/core/job-results-param-outputs. 3. Validate that the processing results retrieved from the `/jobs/{jobID}/results` endpoint conform to requirement /req/core/job-results-async-many.

ABSTRACT TEST A.44

IDENTIFIER	`/conf/core/job-results-exception-no-such-job`
REQUIREMENT	/req/core/job-results-exception-no-such-job

ABSTRACT TEST A.44	
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

NOTE 2: 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.45	
IDENTIFIER	`/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/process-execute-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

NOTE 3: 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.46	
IDENTIFIER	`/conf/core/job-results-failed`
REQUIREMENT	Requirement 46: `/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/process-execute-op but arrange a priori that the job will fail; note the {jobID} assigned to the job. 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

NOTE 4: 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

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

A.13. Conformance Class OGC Process Description

CONFORMANCE CLASS A.2	
IDENTIFIER	`http://www.opengis.net/spec/ogcapi-processes-1/1.0/conf/ogc-process-description`
SUBJECT	http://www.opengis.net/spec/ogcapi-processes-1/1.0/conf/ogc-process-description
TARGET TYPE	Web API
CONFORMANCE TESTS	Abstract test A.49: `/conf/ogc-process-description/input-def` Abstract test A.50: `/conf/ogc-process-description/input-mixed-type` Abstract test A.48: `/conf/ogc-process-description/inputs-def` Abstract test A.47: `/conf/ogc-process-description/json-encoding` Abstract test A.52: `/conf/ogc-process-description/output-def` Abstract test A.53: `/conf/ogc-process-description/output-mixed-type` Abstract test A.51: `/conf/ogc-process-description/outputs-def`

ABSTRACT TEST A.47	
IDENTIFIER	`/conf/ogc-process-description/json-encoding`
REQUIREMENT	Requirement 47: `/req/ogc-process-description/json-encoding`
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-description. 2. For each process, verify the contents of the response body validate against the JSON Schema: process.yaml.

ABSTRACT TEST A.48	
IDENTIFIER	`/conf/ogc-process-description/inputs-def`

ABSTRACT TEST A.48

REQUIREMENT	Requirement 48: `/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-description. 2. For each process, verify that the definition of the inputs conforms to the JSON Schema: inputDescription.yaml.

ABSTRACT TEST A.49

IDENTIFIER	`/conf/ogc-process-description/input-def`
REQUIREMENT	Requirement 49: `/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	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.50

IDENTIFIER	`/conf/ogc-process-description/input-mixed-type`
REQUIREMENT	Requirement 51: `/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-description. 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.51

IDENTIFIER	`/conf/ogc-process-description/outputs-def`
REQUIREMENT	Requirement 52: `/req/ogc-process-description/outputs-def`

ABSTRACT TEST A.51	
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-description. 2. For each process, verify that the definition of the outputs conforms to the JSON Schema: outputDescription.yaml.

ABSTRACT TEST A.52	
IDENTIFIER	`/conf/ogc-process-description/output-def`
REQUIREMENT	Requirement 53: `/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	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.53	
IDENTIFIER	`/conf/ogc-process-description/output-mixed-type`
REQUIREMENT	Requirement 54: `/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-description. 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.14. Conformance Class JSON

CONFORMANCE CLASS A.3	
IDENTIFIER	`http://www.opengis.net/spec/ogcapi-processes-1/1.0/conf/json`
SUBJECT	http://www.opengis.net/spec/ogcapi-processes-1/1.0/conf/json
TARGET TYPE	Web API
CONFORMANCE TEST	Abstract test A.54: `/conf/json/definition`

ABSTRACT TEST A.54	
IDENTIFIER	`/conf/json/definition`
REQUIREMENT	Requirement 55: `/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.15. Conformance Class HTML

CONFORMANCE CLASS A.4	
IDENTIFIER	`http://www.opengis.net/spec/ogcapi-processes-1/1.0/conf/html`
SUBJECT	http://www.opengis.net/spec/ogcapi-processes-1/1.0/conf/html
PREREQUISITE	Conformance Class “Core”
TARGET TYPE	Web API
CONFORMANCE TESTS	Abstract test A.55: `/conf/html/content` Abstract test A.56: `/conf/html/definition`

ABSTRACT TEST A.55	
IDENTIFIER	`/conf/html/content`
REQUIREMENT	Requirement 57: `/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: a) all information in the schemas of the Response Object in the HTML `<body/>`, b) all links in HTML `<a/>` elements in the HTML `<body/>`.

ABSTRACT TEST A.56	
IDENTIFIER	`/conf/html/definition`
REQUIREMENT	Requirement 56: `/req/html/definition`
TEST PURPOSE	Verify support for HTML
TEST METHOD	Verify that every `200` response of every operation of the API where HTML was requested is of media type `text/html`.

A.16. Conformance Class OpenAPI 3.0

CONFORMANCE CLASS A.5	
IDENTIFIER	`http://www.opengis.net/spec/ogcapi-processes-1/1.0/conf/oas30`
SUBJECT	http://www.opengis.net/spec/ogcapi-processes-1/1.0/conf/oas30
PREREQUISITE	Conformance Class “Core”
TARGET TYPE	Web API
CONFORMANCE TESTS	Abstract test A.57: `/conf/oas30/completeness` Abstract test A.58: `/conf/oas30/exceptions-codes` Abstract test A.59: `/conf/oas30/oas-definition-1` Abstract test A.60: `/conf/oas30/oas-definition-2`

CONFORMANCE CLASS A.5

Abstract test A.61: `/conf/oas30/oas-impl`
Abstract test A.62: `/conf/oas30/security`

ABSTRACT TEST A.57

IDENTIFIER	`/conf/oas30/completeness`
REQUIREMENT	Requirement 61: `/req/oas30/completeness`
TEST PURPOSE	Verify the completeness of an OpenAPI document.
TEST METHOD	Verify that for each operation, the OpenAPI document describes all HTTP Status Codes and Response Objects that the API uses in responses.

ABSTRACT TEST A.58

IDENTIFIER	`/conf/oas30/exceptions-codes`
REQUIREMENT	Requirement 62: `/req/oas30/exceptions-codes`
TEST PURPOSE	Verify that the OpenAPI document fully describes potential exception codes.
TEST METHOD	Verify that for each operation, the OpenAPI document describes all HTTP Status Codes that may be generated.

ABSTRACT TEST A.59

IDENTIFIER	`/conf/oas30/oas-definition-1`
REQUIREMENT	Requirement 58: `/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.60	
IDENTIFIER	`/conf/oas30/oas-definition-2`
REQUIREMENT	Requirement 59: `/req/oas30/oas-definition-2`
TEST PURPOSE	Verify that the OpenAPI document is valid JSON.
TEST METHOD	Verify that the JSON representation conforms to the OpenAPI Specification, version 3.0.

ABSTRACT TEST A.61	
IDENTIFIER	`/conf/oas30/oas-impl`
REQUIREMENT	Requirement 60: `/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.62	
IDENTIFIER	`/conf/oas30/security`
REQUIREMENT	Requirement 63: `/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.17. Conformance Class Job list

CONFORMANCE CLASS A.6

IDENTIFIER	`http://www.opengis.net/spec/ogcapi-processes-1/1.0/conf/job-list`
SUBJECT	http://www.opengis.net/spec/ogcapi-processes-1/1.0/conf/job-list
TARGET TYPE	Web API
CONFORMANCE TESTS	Abstract test A.71: `/conf/job-list/datetime-definition` Abstract test A.72: `/conf/job-list/datetime-response` Abstract test A.73: `/conf/job-list/duration-definition` Abstract test A.74: `/conf/job-list/duration-response` Abstract test A.63: `/conf/job-list/job-list-op` Abstract test A.78: `/conf/job-list/job-list-success` Abstract test A.75: `/conf/job-list/limit-definition` Abstract test A.76: `/conf/job-list/limit-response` Abstract test A.77: `/conf/job-list/links` Abstract test A.67: `/conf/job-list/processID-definition` Abstract test A.66: `/conf/job-list/processID-mandatory` Abstract test A.68: `/conf/job-list/processID-response` Abstract test A.69: `/conf/job-list/status-definition` Abstract test A.70: `/conf/job-list/status-response` Abstract test A.64: `/conf/job-list/type-definition` Abstract test A.65: `/conf/job-list/type-response`

ABSTRACT TEST A.63

IDENTIFIER	`/conf/job-list/job-list-op`
REQUIREMENT	Requirement 64: `/req/job-list/job-list-op`
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.64

IDENTIFIER	`/conf/job-list/type-definition`
REQUIREMENT	Requirement 65: `/req/job-list/type-definition`
TEST PURPOSE	Validate that the `type` query parameter is constructed correctly.

ABSTRACT TEST A.64

TEST METHOD	Verify that the `type` query parameter complies with its definition in requirement /req/job-list/type-definition.

ABSTRACT TEST A.65

IDENTIFIER	`/conf/job-list/type-response`
REQUIREMENT	Requirement 66: `/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.66

IDENTIFIER	`/conf/job-list/processID-mandatory`
REQUIREMENT	Requirement 67: `/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.67

IDENTIFIER	`/conf/job-list/processID-definition`
REQUIREMENT	Requirement 68: `/req/job-list/processID-definition`
TEST PURPOSE	Validate that the `processID` query parameter is constructed correctly.
TEST METHOD	Verify that the `processID` query parameter complies with its definition in requirement /req/job-list/processID-definition.

ABSTRACT TEST A.68

IDENTIFIER	`/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.69

IDENTIFIER	`/conf/job-list/status-definition`
REQUIREMENT	Requirement 70: `/req/job-list/status-definition`
TEST PURPOSE	Validate that the `status` query parameter is constructed correctly.
TEST METHOD	Verify that the `status` query parameter complies with its definition in requirement /req/job-list/status-definition.

ABSTRACT TEST A.70

IDENTIFIER	`/conf/job-list/status-response`
REQUIREMENT	Requirement 71: `/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.71

IDENTIFIER	`/conf/job-list/datetime-definition`
REQUIREMENT	Requirement 72: `/req/job-list/datetime-definition`
TEST PURPOSE	Validate that the `datetime` query parameter is constructed correctly.
TEST METHOD	Verify that the `datetime` query parameter complies with its definition in requirement /req/job-list/datetime-definition.

ABSTRACT TEST A.72

IDENTIFIER	`/conf/job-list/datetime-response`
REQUIREMENT	Requirement 73: `/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.73

IDENTIFIER	`/conf/job-list/duration-definition`
REQUIREMENT	Requirement 74: `/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.74

IDENTIFIER	`/conf/job-list/duration-response`

ABSTRACT TEST A.74

REQUIREMENT	Requirement 75: `/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. 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.75

IDENTIFIER	`/conf/job-list/limit-definition`
REQUIREMENT	Requirement 76: `/req/job-list/limit-definition`
TEST PURPOSE	Validate that the `limit` query parameter is constructed correctly.
TEST METHOD	Verify that the `limit` query parameter complies with its definition in requirement /req/job-list/limit-definition.

NOTE 1: The API can define different values for “minimum”, “maximum” and “default”.

ABSTRACT TEST A.76

IDENTIFIER	`/conf/job-list/limit-response`

ABSTRACT TEST A.76	
REQUIREMENT	Requirement 77: `/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.

ABSTRACT TEST A.77	
IDENTIFIER	`/conf/job-list/links`
REQUIREMENT	Requirement 79: `/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.

NOTE 2: 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.78	
IDENTIFIER	`/conf/job-list/job-list-success`

ABSTRACT TEST A.78	
REQUIREMENT	Requirement 78: `/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

A.18. Conformance Class Callback

CONFORMANCE CLASS A.7	
IDENTIFIER	`http://www.opengis.net/spec/ogcapi-processes-1/1.0/conf/callback`
SUBJECT	http://www.opengis.net/spec/ogcapi-processes-1/1.0/conf/callback
TARGET TYPE	Web API
CONFORMANCE TEST	Abstract test A.79: `/conf/callback/job-callback`

ABSTRACT TEST A.79	
IDENTIFIER	`/conf/callback/job-callback`
REQUIREMENT	Requirement 80: `/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. 2. Create an asynchronous execute request that includes the optional `subscriber` key (see execute.yaml. 3. Execute the asynchronous job using test /conf/core/process-execute-request. 4. Validate the job results are received by the specified callback URL.

A.19. Conformance Class Dismiss

CONFORMANCE CLASS A.8	
IDENTIFIER	`http://www.opengis.net/spec/ogcapi-processes-1/1.0/conf/dismiss`
SUBJECT	http://www.opengis.net/spec/ogcapi-processes-1/1.0/conf/dismiss
TARGET TYPE	Web API
CONFORMANCE TESTS	Abstract test A.80: `/conf/dismiss/job-dismiss-op` Abstract test A.81: `/conf/dismiss/job-dismiss-success`

ABSTRACT TEST A.80	
IDENTIFIER	`/conf/dismiss/job-dismiss-op`
REQUIREMENT	Requirement 81: `/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/process-execute-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.81	
IDENTIFIER	`/conf/dismiss/job-dismiss-success`
REQUIREMENT	Requirement 82: `/req/dismiss/job-dismiss-success`
TEST PURPOSE	Validate that the content returned when dismissing a job 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 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.

NOTE: 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 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, complexData, 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
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

DATE	RELEASE	EDITOR	PRIMARY CLAUSES MODIFIED	DESCRIPTION
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.
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

DATE	RELEASE	EDITOR	PRIMARY CLAUSES MODIFIED	DESCRIPTION
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.
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.

DATE	RELEASE	EDITOR	PRIMARY CLAUSES MODIFIED	DESCRIPTION
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 patternProperties
2021-04-15	1.0-draft.6	Panagiotis (Peter) A. Vretanos	X	Remove observedProperty as per SWG resolution of 29MAR2021. The observedProperty 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.
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).

DATE	RELEASE	EDITOR	PRIMARY CLAUSES MODIFIED	DESCRIPTION
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.
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

DATE	RELEASE	EDITOR	PRIMARY CLAUSES MODIFIED	DESCRIPTION
				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 (qualifiedValue.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 Open Api 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
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

DATE	RELEASE	EDITOR	PRIMARY CLAUSES MODIFIED	DESCRIPTION
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
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 ProcessDescription
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.

DATE	RELEASE	EDITOR	PRIMARY CLAUSES MODIFIED	DESCRIPTION
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, qualifiedValue.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.
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.

DATE	RELEASE	EDITOR	PRIMARY CLAUSES MODIFIED	DESCRIPTION
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, inlineOrRefData.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
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 inputValueNoObject.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 Process Description.

DATE	RELEASE	EDITOR	PRIMARY CLAUSES MODIFIED	DESCRIPTION
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 inlineOrRefData.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 ProcessDescription
2021-06-03	1.0-draft.6	Gérald Fenoy	X	Fix issue with binaryInputValue.yaml
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 Open API 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.

DATE	RELEASE	EDITOR	PRIMARY CLAUSES MODIFIED	DESCRIPTION
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.
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.

DATE	RELEASE	EDITOR	PRIMARY CLAUSES MODIFIED	DESCRIPTION
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 “jobsList” 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.
2021-06-15	1.0-draft.6	Panagiotis (Peter) A. Vretanos	X	Add the contentMediaType facet to the GeoJSON 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

DATE	RELEASE	EDITOR	PRIMARY CLAUSES MODIFIED	DESCRIPTION
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
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 processIDPath Param.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

DATE	RELEASE	EDITOR	PRIMARY CLAUSES MODIFIED	DESCRIPTION
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
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] 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.

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

[3] https://www.w3.org/TR/html5/

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

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

[6] Hutton, B.: Relative JSON Pointers, https://json-schema.org/draft/2020-12/relative-json-pointer.html

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