OGC API - Processes - Part 1

Core

Open Geospatial Consortium

Submission Date: <yyyy-mm-dd>

Approval Date: <yyyy-mm-dd>

Publication Date: <yyyy-mm-dd>

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

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

Version: 1.0-draft.6-SNAPSHOT

Category: OGC® Implementation Specification

Editor: Benjamin Pross

OGC API - Processes - Part 1: Core

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

Table of Contents

i. Abstract

This standard defines the OGC API - Processes API 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 are listed in Table 1 below and include information about the server, the list of available processes (Process list and Process description), jobs (running processes) and results of process executions.

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

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

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

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

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

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

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

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

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

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

ii. Keywords

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

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

iii. Preface

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

iv. Submitting organizations

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

- 52°North GmbH
- Hexagon
- CubeWerx Inc.
- Ecere Corporation
- Terradue Srl
- European Space Agency (ESA)
- Spacebel

v. Submitters

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

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

Chapter 1. Scope

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

Chapter 2. Conformance

This standard defines seven requirements / conformance classes.

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

The main requirements class is:

- Core.

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

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

- JSON, and
- HTML.

The JSON encoding is mandatory.

The *Core* does not mandate any encoding or format for the formal definition of the API. 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 or instead of an OpenAPI 3.0 definition. Examples for alternative API definitions: OpenAPI 2.0 (Swagger), future versions of the OpenAPI specification, an OWS Common 2.0 capabilities document or WSDL.

The *Core* is intended to be a minimal useful API for the execution of processes 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 4. Conformance class URIs

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

Chapter 3. References

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

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

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

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

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

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

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

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

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

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

IETF RFC 7231: Hypertext Transfer Protocol (HTTP/1.1): Semantics and Content. https://tools.ietf.org/html/rfc7231

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

Chapter 4. Terms and Definitions

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

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

4.1. Process

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

```
p: X → Y
```

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

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.

4.2. Job

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

4.3. JSON

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

4.4. Process description

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

4.5. Process execution

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

4.6. Process input

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

4.7. Process offering

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

4.8. Process output

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

4.9. REST or RESTful

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

Chapter 5. Conventions

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

5.1. Identifiers

The normative provisions in this specification are denoted by the URI

http://www.opengis.net/spec/ogcapi-processes-1/1.0

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

5.2. Link relations

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

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

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

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

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

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

5.3. Abbreviated Terms

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

5.4. Use of 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.

5.5. HTTP URIs

This document does not restrict the lexical space of URIs used in the API beyond the requirements of the HTTP and URI Syntax IETF RFCs. If URIs include reserved characters that are delimiters in the URI subcomponent, these have to be percent-encoded. See Clause 2 of RFC 3986 (URI Syntax) for details.

Chapter 6. Overview

The OGC API - Processes builds on the WPS 2.0 standard and is modularized. 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.

Chapter 7. Requirements Class "Core"

The following section describes the core requirements class.

7.1. Overview

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

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

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

- The `APIDefinition` (no fixed path),
- The `Conformance` statements (path `/conformance`),
- The `processes` metadata (path `/processes`).doss
- the `execute` endpoint (path `/jobs`).

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

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

The list of processes contains a summary of each process the OGC API - Processes offers, including the link to a more detailed description of the process.

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

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

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

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

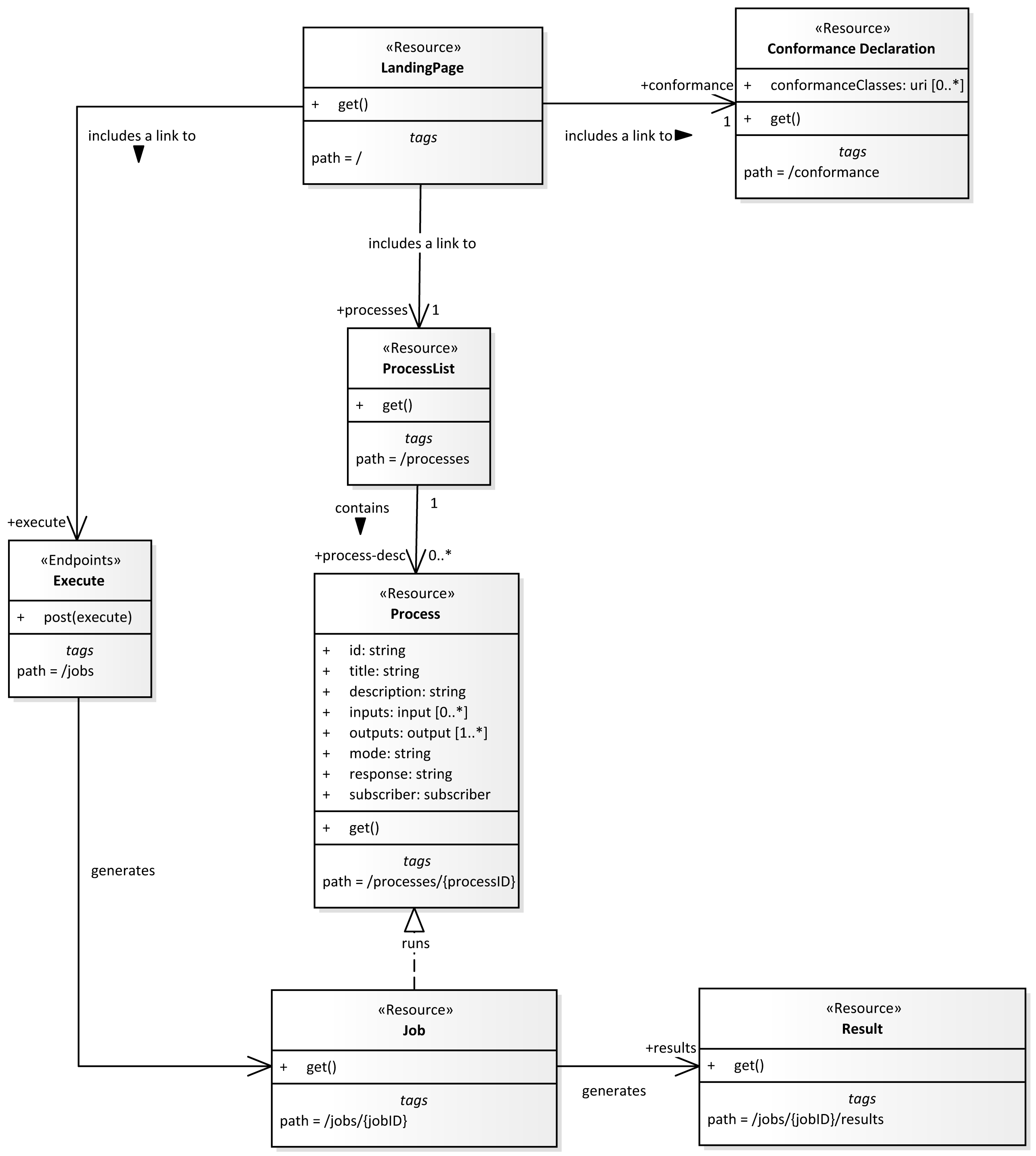

Figure 1. Resources in the Core requirements class

The OGC API - Processes standard utilizes elements of <draft> the OGC API-Common standard. Table 5 Identifies the API-Common Requirements Classes which are applicable to each section of this standard.

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

API - Processes Section	API-Common Requirements Class
API Landing Page	http://www.opengis.net/spec/ogcapi_common-1/1.0/req/core
API Definition	http://www.opengis.net/spec/ogcapi_common-1/1.0/req/core

Declaration of Conformance Classes	http://www.opengis.net/spec/ogcapi_common-1/1.0/req/core
OpenAPI 3.0	http://www.opengis.net/spec/ogcapi_common-1/1.0/req/oas30
HTML	http://www.opengis.net/spec/ogcapi_common-1/1.0/req/html

7.2. Retrieve the API landing page

The following section defines the requirements to retrieve an API landing page.

7.2.1. Operation

Requirement 1	/req/core/landingpage-op The server SHALL support the HTTP GET operation at the path `/`.

7.2.2. Response

Requirement 2	/req/core/landingpage-success A successful execution of the operation SHALL be reported as a response with a HTTP status code `200`. The content of that response SHALL be based upon the OpenAPI 3.0 schema landingPage.yaml and include at least links to the following resources: • the API definition (relation type 'service-desc' or 'service-doc') • `/conformance` (relation type 'conformance') • `/processes` (relation type 'processes')

The following is the schema for the landing page

```
type: object
required:
  - links
properties:
  title:
    type: string
    example: Example processing server
  description:
    type: string
    example: Example server implementing the OGC API - Processes 1.0
  links:
    type: array
    items:
      $ref: link.yaml
```

Example 1. Landing page response document

```
{
    "links": [{
        "href": "http://processing.example.org/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": "conformance",
        "type": "application/json",
        "title": "OGC API - Processes conformance classes implemented by this
server"
    },
    {
        "href": "http://processing.example.org/oapi-p/processes",
        "rel": "processes",
        "type": "application/json",
        "title": "Metadata about the processes"
    },
    {
        "href": "http://processing.example.org/oapi-p/jobs",
        "rel": "execute",
        "title": "The Execute endpoint"
    }]
}
```

7.2.3. Error situations

See HTTP status codes 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 OGC API - Processes provides an API definition that describes the capabilities of the server. This definition is used by developers to understand the API, by software clients to connect to the server, or by development tools to support the implementation of servers and clients.

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

7.3.2. Response

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

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

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

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

7.3.3. Error situations

See HTTP status codes for general guidance.

7.4. Declaration of conformance classes

7.4.1. Operation

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

Requirement 5	/req/core/conformance-op The server SHALL support the HTTP GET operation at the path `/conformance`.

7.4.2. Response

Requirement 6	/req/core/conformance-success 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.

The following is the schema for the list of requirements classes

```
type: object
required:
  - conformsTo
properties:
  conformsTo:
    type: array
    items:
      type: string
      example: "http://www.opengis.net/spec/ogcapi_processes/1.0/req/core"
```

Example 2. Requirements class response document

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

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

7.4.3. Error situations

See HTTP status codes for general guidance.

7.5. Use of HTTP 1.1

Requirement 7	/req/core/http The server SHALL conform to HTTP 1.1. If the server supports HTTPS, the server SHALL also conform to HTTP over TLS.

7.5.1. HTTP status codes

Table 6 lists the main HTTP status codes that clients should be prepared to receive.

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

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

Table 6. Typical HTTP status codes

Status code	Description
`200`	A successful request.
`201`	The request was successful and one or more new resources have being created.
`400`	The server cannot or will not process the request due to an apparent client error. For example, a query parameter had an incorrect value.
`401`	The request requires user authentication. The response includes a `WWW-Authenticate` header field containing a challenge applicable to the requested resource.

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

7.6. Support for cross-origin requests

Access to content from a HTML page is by default prohibited for security reasons if the content is located on another host than the webpage ("same-origin policy"). 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 2	/rec/core/cross-origin If the server is intended to be accessed from the browser, cross-origin requests SHOULD be supported. Note that support can also be added in a proxy layer on top of the server.

Two common mechanisms to support cross-origin requests are:

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

Recommendation 3	/rec/core/access-control-expose-headers If the server is intended to be accessed from the browser and if Cross-origin resource sharing is supported, the `Access-Control-Expose-Headers` header SHOULD be used and the header SHOULD contain the value `location` to enable the browser to access the `location` header of the response.

Recommendation 4	/rec/core/html To support browsing an 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. Retrieve a process list

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

7.7.1. Operation

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

7.7.2. Response

Requirement 9	/req/core/process-list-success A successful execution of the *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.

The following is the schema for the process list

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

The following is the schema for the process summary

```
allOf:
  - $ref: "descriptionType.yaml"
  - type: object
    required:
      - version
    properties:
      version:
        type: string
      jobControlOptions:
        type: array
        items:
          $ref: "jobControlOptions.yaml"
      outputTransmission:
        type: array
        items:
          $ref: "transmissionMode.yaml"
      links:
        type: array
        items:
          $ref: "link.yaml"
```

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

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

The following is an example of Process list encoded as JSON.

```json
[
    {
        "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": "process-desc",
            "title": "process description"
          }
        ]
    }
]
```

7.7.3. Error situations

See HTTP status codes for general guidance.

7.8. Retrieve a process description

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

7.8.1. Operation

Requirement 10	/req/core/process The server SHALL support the HTTP GET operation at the path `/processes/{processID}`.

7.8.2. Response

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

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

Recommendation 5	/rec/core/ogc-process-description Implementations SHOULD consider supporting the OGC process description.

7.8.3. Error situations

See HTTP status codes for general guidance.

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

7.9. Create a new job

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

7.9.1. Operation

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

7.9.2. Request body

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

The following is the schema for execute

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

A process can be either synchronous or asynchronous.

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

The following is an example of an execute request

```
{
    "id": "EchoProcess",
    "inputs": {
        "complexInputId": {
            "format": {
                "mediaType": "application/xml"
            },
            "value": "<test/>"
        },
        "complexInputsId": [
            {
                "format": {
                    "mediaType": "text/plain",
                    "encoding": "UTF-8"
                },
                "value": "test"
            },
            {
                "format": {
                    "mediaType": "text/plain",
                    "encoding": "UTF-8"
                },
                "href": "https://test.data/test.txt"
            }
        ],
        "literalInputId": {
            "dataType": {
                "name": "double"
            },
            "value": "0.05"
        },
        "boundingboxInputId": {
            "bbox": [
                51.9,
                7,
                52,
                7.1
            ],
            "crs": "EPSG:4326"
        }
    },
    "outputs": {
        "literalOutputId": {
            "transmissionMode": "value"
        },
        "boundingboxOutput": {
            "transmissionMode": "value"
        },
        "complexOutputId": {
            "format": {
                "mediaType": "application/xml"
            },
            "transmissionMode": "value"
        },
        "complexOutputsId": {
            "format": {
                "mediaType": "text/plain"
            },
            "transmissionMode": "reference"
        }
    },
    "response": "document",
    "mode": "async"
}
```

7.9.3. Response

In the case of asynchronous execution, the requirements below apply:

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

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

For a synchronous execution, the following requirement applies:

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

7.9.4. Error situations

See HTTP status codes for general guidance.

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

7.10. Retrieve status information about a job

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

7.10.1. Operation

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

7.10.2. Response

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

The following is the schema for status info

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

The following is an example of 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
```

The following is an example of a job encoded as JSON.

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

7.10.3. Error situations

See HTTP status codes for general guidance.

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

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

7.11. Retrieve job results

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

7.11.1. Operation

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

7.11.2. Response

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

The following is the schema for the result of a job

```
additionalProperties:
    oneOf:
        - $ref: "inlineOrRefData.yaml"
        - $ref: "boundingBoxData.yaml"
        - type: array
          items:
              oneOf:
                  - $ref: "inlineOrRefData.yaml"
                  - $ref: "boundingBoxData.yaml"
```

The schema defines a map using the respective id of the output as key. The value of the output can be returned as inline value or as reference.

The following is an example of HTTP GET request for retrieving the result a job encoded as JSON.

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

The following is an example of a result encoded as JSON.

```json
{
    "literalOutputId": {
        "dataType": {
            "name": "double"
        },
        "value": 0.05
    },
    "boundingboxOutputId": {
        "value": {
            "bbox": [
                51.9,
                7,
                52,
                7.1
            ],
            "crs": "EPSG:4326"
        }
    },
    "complexOutputId": {
        "format": {
            "mediaType": "application/xml"
        },
        "value": "<test/>"
    },
    "complexOutputsId": [
        {
            "format": {
                "mediaType": "text/plain"
            },
            "href": "https://processing.example.org/outputdata/faf7c1b6-3c47-4bab-
a8a0-9ba9fce5378b.txt"
        },
        {
            "format": {
                "mediaType": "text/plain"
            },
            "href": "https://processing.example.org/outputdata/ae8eeda3-d408-4852-
b442-de7d0b2d8c48.txt"
        }
    ]
}
```

7.11.3. Error situations

See HTTP status codes for general guidance.

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

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

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

Chapter 8. Requirements Class "OGC Process Description"

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

8.1. Overview

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

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

The process description allows the following information to be specified:

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

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

8.2. OGC process description

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

Schema for a process (process.yaml)

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

The schema imports the elements from the process summary and specifies two arrays for input and output descriptions.

The following is an example of HTTP GET request for retrieving the list of offered processes encoded as JSON.

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

The following is an example of a process encoded as JSON.

```
{
    "id": "EchoProcess",
    "title": "EchoProcess",
    "version": "1.0.0",
    "jobControlOptions": ["async-execute", "sync-execute"],
    "outputTransmission": ["value", "reference"],
    "inputs": [{
        "id": "boundingboxInput",
        "title": "boundingboxInput",
        "input": {
            "supportedCRS": [{
                "default": true,
                "crs": "EPSG:4326"
            }]
        },
        "minOccurs": 1,
        "maxOccurs": 1
    },
    {
        "id": "literalInput",
        "title": "literalInput",
        "input": {
            "literalDataDomain": {
                "dataType": {
                  "name": "double"
                },
                "valueDefinition": {
                    "anyValue": true
                }
            }
        },
        "minOccurs": 1,
        "maxOccurs": 1
    },
    {
        "id": "complexInput",
        "title": "complexInput",
        "input": {
            "formats": [{
                "default": true,
                "mediaType": "application/xml"
            },
            {
                "mediaType": "application/xml"
            },
            {
                "mediaType": "text/xml"
            }]
        },
        "minOccurs": 1,
        "maxOccurs": 1
    }],
    "outputs": [{
        "id": "boundingboxOutput",
        "title": "boundingboxOutput",
        "output": {
            "supportedCRS": [{
                "default": true,
                "crs": "EPSG:4326"
            }]
        }
    },
    {
        "id": "literalOutput",
        "title": "literalOutput",
        "output": {
            "literalDataDomain": {
                "dataType": {
                  "name": "double"
                },
                "valueDefinition": {
                    "anyValue": true
                }
            }
        }
    },
    {
        "id": "complexOutput",
        "title": "complexOutput",
        "output": {
            "formats": [{
                "default": true,
                "mediaType": "application/xml"
            },
            {
                "mediaType": "application/xml"
            },
            {
                "mediaType": "text/xml"
            }]
        }
    }],
    "links": [
      {
        "href": "https://processing.example.org/oapi-p/jobs",
        "rel": "execute",
        "title": "Execute endpoint"
      }
    ]
}
```

Chapter 9. Security Considerations

The OGC API - Processes specifies a Web API that enables the execution of computing processes, the retrieval of metadata describing their purpose and functionality and the retrieval of the results of the process execution. 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 7 gives an overview of the classification of HTTP the methods used in this standard:

Table 7. 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 method and can contain sensitive information:

Requirements class "Core":

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

Requirements class "Job list"

- Job 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

9.1. Operations using HTTP GET

Most of the operations defined in this standard are use the safe HTTP GET method. However, the resources that are returned by these operations contain information that could be used to exploit the API. Table 8 gives an overview of the resources specified in this standard and what kind of information they contain.

Table 8. 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 Execute operation 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 9. 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.

9.2. 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 10. 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.

9.3. 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 canceled. 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.

Chapter 10. Requirements classes for encodings

10.1. Overview

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

- JSON
- HTML

The JSON encoding is mandatory.

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

10.2. Requirement Class "JSON"

This section defines the requirements class JSON.

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

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

10.3. Requirement Class "HTML"

This section defines the requirements class HTML.

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

Dependency	API - Common HTML
Dependency	HTML5

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

Requirement 30	/req/html/content Every `200`-response of the server with the media type "text/html" SHALL be a HTML 5 document that includes the following information in the HTML body: • all information identified in the schemas of the Response Object in the HTML `<body/>`, and • all links in HTML `<a/>` elements in the HTML `<body/>`.

Chapter 11. Requirements Class "OpenAPI 3.0"

11.1. Basic requirements

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

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

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

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

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

11.2. Complete definition

Requirement 34	/req/oas30/completeness The OpenAPI definition SHALL specify for each operation all HTTP Status Codes and Response Objects that the server uses in responses. This includes the successful execution of an operation as well as all error situations that originate from the server.

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

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

11.3. Exceptions

Requirement 35	/req/oas30/exceptions-codes For error situations that originate from the server, the API definition SHALL cover all applicable HTTP Status Codes.

Example 3. An exception response object definition

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

11.4. Security

Requirement 36	/req/oas30/security For cases, where the operations of the server are access-controlled, the security scheme(s) SHALL be documented in the OpenAPI definition.

The OpenAPI specification currently supports the following security schemes:

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

Chapter 12. Requirements Class "Job list"

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

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

12.1. Operation

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

12.2. Response

Requirement 38	/req/job-list/job-list-success A successful execution of the operation SHALL be reported as a response with a HTTP status code 200. The content of that response SHALL be based upon the OpenAPI 3.0 schema jobList.yaml.

The following is the schema for the job list

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

The schema defines an array of status info elements.

The following is an example of an HTTP GET request for retrieving a list of jobs encoded as JSON.

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

The following is an example of a job list encoded as JSON.

```
[
    {
        "jobID": "8ca109b4-3b86-4a9c-a284-a6d50f91019e",
        "status": "running",
        "message": "Perform step 1/2",
        "progress": 50,
        "links": [
            {
                "href": "http://processing.example.org/oapi-p/jobs/8ca109b4-3b86-4a9c-
a284-a6d50f91019e",
                "rel": "status",
                "type": "application/json",
                "hreflang": "en",
                "title": "Job status"
            }
        ]
    },
    {
        "id": "0cf773a5-282a-4e23-96cc-f5dab18123e5",
        "infos": {
            "jobID": "0cf773a5-282a-4e23-96cc-f5dab18123e5",
            "status": "successful",
            "message": "EchoProcess job finished successful",
            "progress": 100,
            "links": [
                {
                    "href": "http://processing.example.org/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": "results",
                    "type": "application/json",
                    "hreflang": "en",
                    "title": "Job result"
                }
            ]
        }
    },
    {
        "id": "63aadd9c-c0e5-4a7f-80f0-228dbb158f09",
        "infos": {
            "jobID": "63aadd9c-c0e5-4a7f-80f0-228dbb158f09",
            "status": "failed",
            "message": "EchoProcess job failed",
            "progress": 100,
            "links": [
                {
                    "href": "http://processing.example.org/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": "exceptions",
                    "type": "application/json",
                    "hreflang": "en",
                    "title": "Job exception"
                }
            ]
        }
    }
]
```

12.3. Error situations

See HTTP status codes for general guidance.

If the process with the specified identifier does not exist on the server, the status code of the response SHALL be `404` (see [req_core_no-such-process]).

Chapter 13. Requirements Class "Callback"

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

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

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

The following is an example for a callback in the execute operation

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

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

Chapter 14. Requirements Class "Dismiss"

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

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

14.1. Operation

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

14.2. Response

Requirement 41	/req/dismiss/job-dismiss-success A successful execution of the operation SHALL be reported as a response with a HTTP status code 200. The content of that response SHALL be based upon the OpenAPI 3.0 schema statusInfo.yaml. The status SHALL be set to "dismissed".

The following is an example of a dismissed job encoded as JSON.

```
{
  "jobID" : "81574318-1eb1-4d7c-af61-4b3fbcf33c4f",
  "status": "dismissed",
  "message": "Job dismissed",
  "progress": 56,
  "links": [
    {
      "href": "http://processing.example.org/oapi-p/jobs",
      "rel": "up",
      "type": "application/json",
      "title": "The job list of this server"
    }
  ]
}
```

14.3. Error situations

See HTTP status codes 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-result-exception/no-such-job).

Chapter 15. Media Types

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

- `application/json` for all resources.

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

- `text/html`.

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

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

Chapter 16. Additional API Building Blocks

The core requirements classes of 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. 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 Retrieve an API definition). 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 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 2	/per/core/alternative-process-description Servers MAY support alternative means of describing the inputs and outputs of a process.

The alternative-process-description permission allows server implementations to describe a process, such as by defining the request and response body of a POST request to a process endpoint using the OpenAPI specification directly (see this example).

Permission 3	/per/core/alternative-process-paths 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:

```yaml
openapi: 3.0.2
info:
  title: Alternative OGC API - Processes
  description: This is an alternative OGC API - Processes
  contact:
    email: you@your-company.com
  license:
    name: Apache 2.0
    url: http://www.apache.org/licenses/LICENSE-2.0.html
  version: 1.0.0
paths:
  /buffer:
    post:
      summary: execute buffer process
      operationId: executeBuffer
      requestBody:
        description: buffer inputs
        content:
          application/json:
            schema:
              $ref: '#/components/schemas/bufferExecute'
      responses:
        "200":
          description: buffer created
          content:
            application/json:
              schema:
                $ref: '#/components/schemas/bufferResult'
        "400":
          description: invalid input
components:
  schemas:
    bufferExecute:
      required:
      - data
      - width
      type: object
      properties:
        data:
          maxItems: 10
          minItems: 1
          type: array
          description: this is possible to provide the abstract in here
          items:
            oneOf:
            - type: string
              format: application/geo+json
            - type: string
```

```
              format: application/gml+xml
        width:
          maximum: 100
          minimum: 1
          type: integer
          default: 20
    bufferResult:
      type: object
      properties:
        outputs:
          type: array
          items:
            oneOf:
            - type: string
              format: application/geo+json
            - type: string
              format: application/gml+xml
```

The goals of these additional API building blocks are:

- Enabling 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.

Annex A: Abstract Test Suite (Normative)

A.1. Introduction

OGC Web APIs are not a Web Services in the traditional sense. Rather, they define the behavior and content of a set of Resources exposed through a Web Application Programing Interface (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.

A.2. Conformance Class Core

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

A.2.1. Landing Page /

Abstract Test 1	**/conf/core/landingpage-op**
Test Purpose	Validate that a landing page can be retrieved from the expected location.
Requirement	/req/core/landingpage-op
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 2	**/conf/core/landingpage-success**
Test Purpose	Validate that the landing page complies with the require structure and contents.
Requirement	/req/core/landingpage-success

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 Schema and Tests for Landing Pages 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 "conformance" link to the conformance class declaration. c. Validate that the landing page includes a "data" link to the Feature contents.

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 11. Schema and Tests for Landing Pages

Format	Schema Document	Test ID
HTML	landingPage.yaml	/conf/html/content
JSON	landingPage.yaml	/conf/geojson/content

A.2.2. API Definition /api

Abstract Test 3	**/conf/core/api-definition-op**
Test Purpose	Validate that the API Definition document can be retrieved from the expected location.
Requirement	/req/core/api-definition-op
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 4	**/conf/core/api-definition-success**

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

A.2.3. Conformance Path /conformance

Abstract Test 5	**/conf/core/conformance-op**
Test Purpose	Validate that a Conformance Declaration can be retrieved from the expected location.
Requirement	/req/core/conformance-op
Test Method	1. Construct a path for each "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 6	**/conf/core/conformance-success**
Test Purpose	Validate that the Conformance Declaration response complies with the required structure and contents.
Requirement	/req/core/conformance-success
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.2.4. HTTP 1.1

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

A.2.5. Processes /processes

A.2.5.1. Process list

Abstract Test 8	**/conf/core/process-list**
Test Purpose	Validate that information about the processes can be retrieved from the expected location.
Requirement	/req/core/process-list
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 9	**/conf/core/process-list-success**
Test Purpose	Validate that the process list content complies with the required structure and contents.
Requirement	/req/core/process-list-success
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 Schema and Tests for Lists content

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 12. Schema and Tests for Lists content

Format	Schema Document	Test ID
HTML	processList.yaml	/conf/html/content
JSON	processList.yaml	/conf/json/content

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

Abstract Test 10	**/conf/core/process**
Test Purpose	Validate that a process description can be retrieved from the expected location.
Requirement	/req/core/process
Test Method	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. . Validate the response using the test /conf/core/process-success.

Abstract Test 11	**/conf/core/process-success**
Test Purpose	Validate that the content complies with the required structure and contents.
Requirement	/req/core/process-success
Test Method	1. Validate that a document was returned with an HTTP status code of 200. 2. Verify that the content of the response is valid description of the interface of the process for all supported process description models.

The interface of a process may be describing using a number of different models or process description languages. The following table identifies the applicable schema document for each process description model described in this standard.

Table 13. Schema and Tests for Process Description Models

Model	Schema Document	Test ID
OGC Process Description JSON	process.yaml	/conf/ogc-process-description/json-encoding

A.2.5.3. Process exception

Abstract Test 12	**/conf/core/process-exception-no-such-process**
Test Purpose	Validate that an invalid process identifier is handled correctly.
Requirement	/req/core/process-exception-no-such-process
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 code "NoSuchProcess". 4. Validate the document for all supported media types using the resources and tests identified in Schema and Tests for Non-existent Process

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 14. Schema and Tests for Non-existent Process

Format	Schema Document	Test ID
HTML	exception.yaml	/conf/html/content
JSON	exception.yaml	/conf/json/content

A.2.6. Jobs

A.2.6.1. Job creation /jobs

Abstract Test 13	**/conf/core/job-creation-op**
Test Purpose	Validate the creation of a new job.
Requirement	/req/core/job-creation-op

Test Method	1. Issue an HTTP POST request to the URL '/jobs' for each execution mode according to the test /conf/core/job-creation-mode. 2. Validate the contents of the POST request using the test /conf/core/job-creation-request. 3. Validate the creation of the job according to the execution mode using test /conf/core/job-creation-mode.

Abstract Test 14	**/conf/core/job-creation-request**
Test Purpose	Validate that the body of a job creation operation complies with the required structure and contents.
Requirement	/req/core/job-creation-request
Test Method	Verify the contents of the request body against the OpenAPI 3.0 schema execute.yaml.

Abstract Test 15	**/conf/core/job-creation-mode**
Test Purpose	Validate the creation of a new job according to its execution mode.
Requirement	/req/core/job-creation-mode
Test Method	1. Create a job for each execution mode according to the test /req/core/job-creation-op. 2. Validate the creation of the job according to the execution mode using the resource and tests identified in Schema and Tests for Job Creation.

A job may be executed in one of the three modes; `sync`, `async` or `auto`. The following tables identified the applicable test to check based on the execution mode.

Table 15. Schema and Tests for Job Creation

Mode	**Test ID**
sync	/conf/core/job-creation-success-sync
async	/conf/core/job-creation-success-async
auto	/conf/core/job-creation-success-auto

Abstract Test 16	/conf/core/job-creation-success-async
Test Purpose	Validate the result of a job that has been created using the `async` execution mode.
Requirement	/req/core/job-creation-success-async
Test Method	1. Validate that result of the job was returned with an HTTP status code 201. 2. Validate the HTTP headers of the result using the test /conf/core/job-creation-success-header-async.

Abstract Test 17	/conf/core/job-creation-success-header-async
Test Purpose	Validate the HTTP header for an asynchronously executed job.
Requirement	/req/core/job-creation-success-header-async
Test Method	1. Validate that the response contains the 'Location' header. 2. Issue an HTTP GET request to the URL that is the value of the 'Location' header. 3. Validate the result of resolving the 'Location' header URL using the test /conf/core/job-result-op.

Abstract Test 18	/conf/core/job-creation-success-sync
Test Purpose	Validate the result of a job that has been created using the `sync` execution mode.
Requirement	/req/core/job-creation-success-sync
Test Method	1. Validate that result of the job was returned with a status code 200. 2. Validate the content of the result using the resource and tests identified in Schema and Tests for the Response of a Synchronously Executed Job.

The type of response a job generates is determined by the value of the `response` attribute. The value of the `response` attribute may be `document` or `raw`. The following table identified the applicable test to check based on the value of the `response` attribute.

Table 16. Schema and Tests for the Response of a Synchronously Executed Job

Response Type	Schema	Test ID
document	results.yaml	/conf/core/job-result-success
raw	N/A	/conf/core/job-creation-success-sync-raw

Abstract Test 19	**/conf/core/job-creation-success-sync-raw**
Test Purpose	Validate the job result when the `response` attribute is set to `raw`.
Requirement	/req/core/job-creation-success-sync-raw
Test Method	1. Validate that the result of the job was returned with an HTTP status code 200. 2. Get a description of the executed process using test /conf/core/process. 3. From the process description, note the expected media type(s) for the output that was specified in the execute request body. 4. Verify that the response is of the expected media type. 5. If the response has an associated schema, validate the response against that schema.

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

Abstract Test 20	**/conf/core/fc-op**
Test Purpose	Validate that the status info of a job can be retrieved.
Requirement	/req/core/fc-op
Test Method	1. Create a job as per /req/core/job-creation-op and note the {jobID} assigned to the job. 2. Issue an HTTP GET request to the URL '/jobs/{jobID}'. 3. Validate the contents of the returned document using the test /conf/core/job-success.

Abstract Test 21	**/conf/core/job-success**
Test Purpose	Validate that the job status info complies with the require structure and contents.

Requirement	/req/core/job-success
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 Schema and Tests for the Job Status Info

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 17. 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 22	**/conf/core/job-exception-no-such-job**
Test Purpose	Validate that an invalid job identifier is handled correctly.
Requirement	/req/core/job-exception-no-such-job
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 code "NoSuchJob". 4. Validate the document for all supported media types using the resources and tests identified in Schema and Tests for the Job Result for Non-existent Job

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 18. Schema and Tests for the Job Result for Non-existent Job

Format	Schema Document	Test ID
HTML	exception.yaml	/conf/html/content
JSON	exception.yaml	/conf/json/content

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

Abstract Test 23	**/conf/core/job-result**
Test Purpose	Validate that the results of a job can be retrieved.
Requirement	/req/core/job-result
Test Method	1. Create a job as per /req/core/job-creation-op and note the {jobID} assigned to the job. 2. Issue an HTTP GET request to the URL '/jobs/{jobID}/results'. 3. Validate that the document was returned with a status code 200. 4. Validate the contents of the returned document using the test /conf/core/job-result-success.

Abstract Test 24	**/conf/core/job-result-success**
Test Purpose	Validate that the job result complies with the require structure and contents.
Requirement	/req/core/job-result-success
Test Method	Validate the job result for all supported media types using the resources and tests identified in Schema and Tests for the Job Result

The job result 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 19. Schema and Tests for the Job Result

Format	**Schema Document**	**Test ID**
HTML	result.yaml	/conf/html/content
JSON	result.yaml	/conf/json/content

Abstract Test 25	**/conf/core/job-result-failed**
Test Purpose	Validate that the job result retrieved using an invalid job identifier complies with the require structure and contents.
Requirement	/req/core/job-result-exception-no-such-job

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 code "NoSuchJob". 4. Validate the document for all supported media types using the resources and tests identified in Schema and Tests for the Job Result for Non-existent Job

The job result 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 result for a non-existent job against that schema. All supported formats should be exercised.

Table 20. 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 26	**/conf/core/job-result-exception-result-not-ready**
Test Purpose	Validate that the job result retrieved for an incomplete job complies with the require structure and contents.
Requirement	/req/core/job-result-exception-result-not-ready
Test Method	1. Create a job as per /req/core/job-creation-op and note the {jobID} assigned to the job; ensure that the job is long-running. 2. Issue an HTTP GET request to the URL '/jobs/{jobID}/results' before the job completes execution. 3. Validate that the document was returned with a 404. 4. Validate that the document contains the exception code "ResultNotReady". 5. Validate the document for all supported media types using the resources and tests identified in Schema and Tests for the Job Result for an Incomplete Job

The job result 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 21. 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 27	**/conf/core/job-result-failed**
Test Purpose	Validate that the job result for a failed job complies with the require structure and contents.
Requirement	/req/core/job-result-failed
Test Method	1. Create a job as per /req/core/job-creation-op but arrange a priori that the job will fail; note the {jobID} assigned to the job. 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 code 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 Schema and Tests for the Job Result for a Failed Job

The job result 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 result for a failed job against that schema. All supported formats should be exercised.

Table 22. Schema and Tests for the Job Result for a Failed Job

Format	Schema Document	Test ID
HTML	exception.yaml	/conf/html/content
JSON	exception.yaml	/conf/json/content

A.3. Conformance Class OGC Process Description

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

Requirements class	Requirements Class "OGC Process Description"

Abstract Test 28	**/conf/core/job-creation-request**
Test Purpose	Validate a JSON-encoded OGC process description complies with the required structure and contents.
Requirement	/req/ogc-process-description/json-encoding
Test Method	Verify the contents of the request body against the OpenAPI 3.0 schema process.yaml.

A.4. Conformance Class JSON

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

Abstract Test 29	**/conf/json/definition**
Test Purpose	Verify support for JSON.
Requirement	/req/json/definition
Test Method	1. A resource is requested with response media type of `application/json`. 2. All `200` responses SHALL support the following media types: `application/json` for all resources.

A.5. Conformance Class HTML

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

Abstract Test 30	**/conf/html/content**
Test Purpose	Verify the content of an HTML document given an input document and schema.
Requirement	/req/html/content
Test Method	1. Validate that the document is an HTML 5 document 2. Manually inspect the document and verify that the HTML body contains: ◦ all information in the schemas of the Response Object in the HTML `<body/>` ◦ all links in HTML `<a/>` elements in the HTML `<body/>`.

Abstract Test 31	**/conf/html/definition**
Test Purpose	Verify support for HTML
Requirement	/req/html/definition
Test Method	Verify that every `200` response of every operation of the API where HTML was requested is of media type `text/html`.

A.6. Conformance Class OpenAPI 3.0

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

Abstract Test 32	**/conf/oas30/completeness**
Test Purpose	Verify the completeness of an OpenAPI document.
Requirement	/req/oas30/completeness

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

Abstract Test 33	**/conf/oas30/exceptions-codes**
Test Purpose	Verify that the OpenAPI document fully describes potential exception codes.
Requirement	/req/oas30/exceptions-codes
Test Method	Verify that for each operation, the OpenAPI document describes all HTTP Status Codes that may be generated.

Abstract Test 34	**/conf/oas30/oas-definition-1**
Test Purpose	Verify that JSON and HTML versions of the OpenAPI document are available.
Requirement	/req/oas30/oas-definition-1
Test Method	1. Verify that an OpenAPI definition in JSON is available using the media type `application/vnd.oai.openapi+json;version=3.0` and link relation `service-desc` 2. Verify that an HTML version of the API definition is available using the media type `text/html` and link relation `service-doc`.

Abstract Test 35	**/conf/oas30/oas-definition-2**
Test Purpose	Verify that the OpenAPI document is valid JSON.
Requirement	/req/oas30/oas-definition-2
Test Method	Verify that the JSON representation conforms to the OpenAPI Specification, version 3.0.

Abstract Test 36	**/conf/oas30/oas-impl**
Test Purpose	Verify that all capabilities specified in the OpenAPI definition are implemented by the API.

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

Abstract Test 37	**/conf/oas30/security**
Test Purpose	Verify that any authentication protocols implemented by the API are documented in the OpenAPI document.
Requirement	/req/oas30/security
Test Method	1. Identify all authentication protocols supported by the API. 2. Validate that each authentication protocol is described in the OpenAPI document by a Security Schema Object and its use is specified by a Security Requirement Object.

A.7. Conformance Class Job list

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

Abstract Test 38	**/conf/job-list/job-list-op**
Test Purpose	Validate that information about jobs can be retrieved from the expected location.
Requirement	/req/job-list/job-list-op
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 39	**/conf/job-list/job-list-success**

Test Purpose	Validate that the job list content complies with the required structure and contents.
Requirement	/req/job-list/job-list-success
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 Schema and Tests for Job List Content

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 23. Schema and Tests for Job List Content

Format	Schema Document	Test ID
HTML	jobList.yaml	/conf/html/content
JSON	jobList.yaml	/conf/json/content

A.8. Conformance Class Callback

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

Abstract Test 40	**/conf/callback/job-callback**
Test Purpose	Validate the passing of a subscriber-URL in an execute request.
Requirement	/req/callback/job-callback

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/job-creation-request. 4. Validate the job result is received by the specified callback URL.

A.9. Conformance Class Dismiss

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

Abstract Test 41	**/conf/dismiss/job-dismiss-op**
Test Purpose	Validate that a running job can be dismissed.
Requirement	/req/dismiss/job-dismiss-op
Test Method	1. Create an asynchronous job as per test /conf/core/job-creation-request*; 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 42	**/conf/dismiss/job-dismiss-success**
Test Purpose	Validate that the content returned when dismissing a job complies with the required structure and contents.
Requirement	/req/dismiss/job-dismiss-success

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 Schema and Tests for Dismissing a Job

The response to dismissing a job can be presented in a number of different formats. The following table identifies the applicable schema document for each format and the test to be used to validate the against that schema. All supported formats should be exercised.

Table 24. Schema and Tests for Dismissing a Job

Format	Schema Document	Test ID
HTML	statusInfo.yaml	/conf/html/content
JSON	statusInfo.yaml	/conf/json/content

Annex B: Revision History

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

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

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

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

Date	Release	Editor	Primary clauses modified	Description
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