OGC® DOCUMENT: 21-009

**External identifier of this OGC® document:** http://www.opengis.net/doc/is/ogcapi-processes-3/0.0

Open Geospatial Consortium

# OGC API - PROCESSES - PART 3: WORKFLOWS AND CHAINING

## STANDARD
## Implementation

**CANDIDATE SWG DRAFT**

**Version:** 1.0
**Submission Date:** 2029-03-30
**Approval Date:** 2029-03-30
**Publication Date:** 2029-03-30
**Editor:** Jérôme Jacovella-St-Louis, Panagiotis (Peter) A. Vretanos

**Notice for Drafts:** This document is not an OGC Standard. This document is distributed for review and comment. This document is subject to change without notice and may not be referred to as an OGC Standard.
Recipients of this document are invited to submit, with their comments, notification of any relevant patent rights of which they are aware and to provide supporting documentation.

**License Agreement**

>Use of this document is subject to the license agreement at https://www.ogc.org/license

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

**Copyright notice**

Copyright © 2023 Open Geospatial Consortium
To obtain additional rights of use, visit https://www.ogc.org/legal

**Note**

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

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

# CONTENTS


I. ABSTRACT ..... vii

II. KEYWORDS ..... vii

III. PREFACE ..... viii

IV. SECURITY CONSIDERATIONS ..... ix

V. SUBMITTING ORGANIZATIONS ..... x

VI. SUBMITTERS ..... x

1. SCOPE ..... 2

2. CONFORMANCE ..... 4
   2.1. Requirements classes extending *Part 1: Core* execution requests ..... 4
   2.2. Requirements class specifying a new on-demand, client-driven execution mechanism ..... 5
   2.3. Requirements classes specifying alternative ad-hoc workflow definitions ..... 6
   2.4. Declaration of conformance ..... 6

3. NORMATIVE REFERENCES ..... 9

4. TERMS AND DEFINITIONS ..... 11

5. CONVENTIONS ..... 13
   5.1. Identifiers ..... 13

6. OVERVIEW ..... 15
   6.1. Architecture Diagram ..... 15
   6.2. Design goals ..... 17

7. NESTED PROCESSES AND REMOTE PROCESSES CONFORMANCE CLASSES ..... 21
   7.1. Requirement Class *Nested Processes* ..... 21
   7.2. Requirement Class *Remote Core Processes* ..... 22

8. COLLECTION INPUT AND REMOTE COLLECTIONS CONFORMANCE CLASSES ..... 25
   8.1. Requirement Class *Collection Input* ..... 25
   8.2. Requirement Class *Remote Collections* ..... 26

9. INPUT/OUTPUT FIELDS MODIFERS CONFORMANCE CLASSES ..... 29
9.1. Requirement Class *Input Fields Modifers* ..... 29
9.2. Requirement Class *Output Fields Modifers* ..... 31

10. DEPLOYABLE WORKFLOWS CONFORMANCE CLASS ..... 35
10.1. Requirement Class *Deployable Workflows* ..... 35

11. COLLECTION OUTPUT CONFORMANCE CLASS ..... 39
11.1. Requirement Class *Collection Output* ..... 39

12. COMMON WORKFLOW LANGUAGE DEFINITIONS CONFORMANCE CLASS ..... 44
12.1. Requirement Class *Common Workflow Language Definitions* ..... 44

13. OPENEO PROCESS GRAPHS WORKFLOW CONFORMANCE CLASS ..... 47
13.1. Requirement Class *OpenEO Process Graphs Workflow* ..... 47

14. MEDIA TYPES ..... 50

ANNEX A (INFORMATIVE) CONFORMANCE CLASS ABSTRACT TEST SUITE (NORMATIVE) ..... 52
A.1. Conformance Class A ..... 52

ANNEX B (INFORMATIVE) COASTAL EROSION SUSCEPTIBILITY EXAMPLE WORKFLOW ..... 54

ANNEX C (INFORMATIVE) REVISION HISTORY ..... 58

BIBLIOGRAPHY ..... 60

# LIST OF TABLES

Table 1 — Conformance class URIs ..... 7
Table C.1 — Revision history ..... 58

# LIST OF FIGURES

Figure 1 — Workflows extension architecture diagram ..... 16
Figure B.1 — Coastal Erosion Example Workflow Diagram ..... 55
Figure B.2 — Coastal Erosion Example Workflow Exectution Request ..... 56

# LIST OF RECOMMENDATIONS


REQUIREMENTS CLASS 1 ..... 21
REQUIREMENTS CLASS 2 ..... 22
REQUIREMENTS CLASS 3 ..... 25
REQUIREMENTS CLASS 4 ..... 26
REQUIREMENTS CLASS 5 ..... 29
REQUIREMENTS CLASS 6 ..... 31
REQUIREMENTS CLASS 7 ..... 35
REQUIREMENTS CLASS 8 ..... 39
REQUIREMENTS CLASS 9 ..... 44
REQUIREMENTS CLASS 10 ..... 47
REQUIREMENT 1 ..... 21
REQUIREMENT 2 ..... 22
REQUIREMENT 3 ..... 23
REQUIREMENT 4 ..... 23
REQUIREMENT 5 ..... 25
REQUIREMENT 6 ..... 26
REQUIREMENT 7 ..... 27
REQUIREMENT 8 ..... 27
REQUIREMENT 9 ..... 30
REQUIREMENT 10 ..... 30
REQUIREMENT 11 ..... 31
REQUIREMENT 12 ..... 32
REQUIREMENT 13 ..... 32
REQUIREMENT 14 ..... 33
REQUIREMENT 15 ..... 36
REQUIREMENT 16 ..... 36
REQUIREMENT 17 ..... 37
REQUIREMENT 18 ..... 40
REQUIREMENT 19 ..... 40
REQUIREMENT 20 ..... 41
REQUIREMENT 21 ..... 42

REQUIREMENT 22 ..... 44
REQUIREMENT 23 ..... 47
PERMISSION 1 ..... 41
REQUIREMENT A.1 ..... 52

# I ABSTRACT

This specification extends the core capabilities specified in *OGC API — Processes — Part 1: Core* with the ability to chain nested processes, refer to both local and external processes and collections of data accessible via OGC API standards as inputs to a process, and trigger execution of processes through OGC API data delivery specifications such as *OGC API — Tiles*, *DGGS*, *Coverages*, *Features*, *EDR* and *Maps*. Conformance classes defined in this specification enable clients to integrate discovered data and processes as a workflow in an ad-hoc manner (without first requiring to deploy the workflow as a process), then drive the execution on-demand for a particular area, time and resolution of interest using a "pull" mechanism (pulling the final required output from the client, as opposed to a batch execution starting from source inputs of the first processing step). The workflows defined with this specification can also be deployed as processes using the *OGC API — Processes — Part 2: Deploy*, *Replace*, *Undeploy* extension.

# II KEYWORDS

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

ogcdoc, OGC document, API, openapi, html, ogcapi

# III PREFACE

OGC API standards define modular API building blocks to spatially enable Web APIs in a consistent way. The OpenAPI specification is used to define the API building blocks.

*OGC API – Processes* provides API building blocks to describe, execute, monitor and retrieve results of Web-accessible processes. *OGC API – Processes* is comprised of multiple parts, each of them is a separate standard. The conformance classes defined in this *Part 3* document leverage conformance classes defined in *Part 1: Core*, and in some cases conformance classes defined in *Part 2: Deploy, Replace, Undeploy*.

The development of this extension has been financially supported by GeoConnections, a national collaborative initiative led by Natural Resources Canada. GeoConnections supports the integration and use of the Canadian Geospatial Data Infrastructure (CGDI), an on-line resource that improves the sharing, access and use of open geospatial information.

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

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

# IV SECURITY CONSIDERATIONS

No security considerations have been made for this document.

# V SUBMITTING ORGANIZATIONS

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

- Ecere Corporation
- CubeWerx Inc.
- GeoLabs
- University of Calgary
- Pangaea Innovations Pty. Ltd.
- Natural Resources Canada
- EOX IT Services GmbH
- Universitat Autònoma de Barcelona (CREAF)
- Centre de Recherche en Informatique de Montréal (CRIM)
- 52°North Spatial Information Research GmbH
- European Space Agency (ESA)
- US Army Geospatial Center (AGC)
- Hexagon
- Haute École d'Ingénierie et de Gestion du Canton de Vaud (HEIG-VD)
- Laboratoire des Sciences et Techniques de l'information de la Communication et de la Connaissance / Centre national de la recherche scientifique (Lab-STICC CNRS)

# VI SUBMITTERS

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

| Name | Affiliation |
|---|---|
| Jérôme Jacovella-St-Louis (editor) | Ecere Corporation |
| Panagiotis (Peter) A. Vretanos (editor) | CubeWerx Inc. |

| | |
|---|---|
| Gérald Fenoy | GeoLabs |
| Perry Peterson | University of Calgary |
| Marta Padilla Ruiz | University of Calgary |
| Matthew B.J. Purss | Pangaea Innovations Pty. Ltd. |
| Ryan Ahola | Natural Resources Canada |
| Stephan Meißl | EOX IT Services GmbH |
| Joan Masó Pau | Universitat Autònoma de Barcelona (CREAF) |
| Francis Charette Migneault | Centre de Recherche en Informatique de Montréal (CRIM) |
| Benjamin Proß | 52°North Spatial Information Research GmbH |
| Claudio Iacopino | European Space Agency (ESA) |
| Jeff Harrison | US Army Geospatial Center (AGC) |
| Stan Tillman | Hexagon |
| Olivier Ertz | Haute École d'Ingénierie et de Gestion du Canton de Vaud (HEIG-VD) |
| Erwan Bocher | Laboratoire des Sciences et Techniques de l'information de la Communication et de la Connaissance / Centre national de la recherche scientifique (Lab-STICC CNRS) |

# 1 SCOPE

# 1 SCOPE

This document specifies an extension to *OGC API — Processes — Part 1: Core* that defines the behavior of an implementation that supports the ability to define ad-hoc workflows, chain nested processes, refer to both local and external processes and collections of data accessible via OGC API standards as inputs to a process, and trigger execution of processes through OGC API data delivery specifications such as *OGC API — Tiles*, *DGGS*, *Coverages*, *Features*, *EDR* and *Maps*. This extension also defines the behavior of an implementation that supports the ability to deploy a workflow defined using the mechanisms it describes through the *OGC API — Processes — Part 2: Deploy, Replace, Undeploy* extension.

2

# CONFORMANCE

# 2 CONFORMANCE

This standard defines ten requirements conformance classes.

The standardization targets of all conformance classes are “Web APIs”.

The requirements classes are:

## 2.1. Requirements classes extending *Part 1: Core* execution requests

**Requirements Class “Nested Processes”** (http://www.opengis.net/spec/ogcapi-processes-3/0.0/req/nested-processes)

The *Nested Processes* conformance class extends the execution request of *OGC API — Processes — Part 1: Core* with the ability to specify another process as an input. A `"process"` key is used to define such an input.

**Requirements Class “Remote Core Processes”** (http://www.opengis.net/spec/ogcapi-processes-3/0.0/req/remote-core-processes)

The *Remote Core Processes* conformance class extends the capability defined in the *Nested Processes* conformance class to the execution of remote processes using the *Core* conformance class of *OGC API — Processes — Core: Part 1* in addition to local processes. Remote processes supporting the *Collection Output* conformance class can also be executed with support for the *Remote Collections* as an alternative to this conformance class.

**Requirements Class “Collection Input”** (http://www.opengis.net/spec/ogcapi-processes-3/0.0/req/collection-input)

The *Collection Input* conformance class extends the capability defined in *OGC API — Processes — Part 1: Core* with the ability to reference an OGC API collection as input. A `"collection"` key is used to define such an input.

**Requirements Class “Remote Collections”** (http://www.opengis.net/spec/ogcapi-processes-3/0.0/req/remote-collections)

The *Remote Collections* conformance class extends the capability defined in the *Collection Input* conformance class to remote collections in addition to local collections. If *Nested Processes* are also supported, this also extends to remote processes supporting the *Collection Output* conformance classes, specified using the `"process"` key.

**Requirements Class “Input Fields Modifiers”** (http://www.opengis.net/spec/ogcapi-processes-3/0.0/req/input-fields-modifiers)

The *Input Fields Modifiers* conformance class extends the execution request of *OGC API — Processes — Part 1: Core* with the ability to filter or sort data used as an input to a process according to field values, as well as to select specific fields from the data or derive new fields from existing ones, using an expression language such as the OGC Common Query Language 2.0 (CQL2). Those modifiers are defined by the `"filter"`, `"properties"` and `"sortBy"` keys.

**Requirements Class "Output Fields Modifiers"** (http://www.opengis.net/spec/ogcapi-processes-3/0.0/req/output-fields-modifiers)

The *Output Fields Modifiers* conformance class extends the execution request of *OGC API — Processes — Part 1: Core* with the ability to filter or sort data returned as an output from a process according to field values, as well as to select specific fields from the data or derive new fields from existing ones, using an expression language such as the OGC Common Query Language 2.0 (CQL2). Those modifiers are defined by the `"filter"`, `"properties"` and `"sortBy"` keys.

**Requirements Class "Deployable Workflows"** (http://www.opengis.net/spec/ogcapi-processes-3/0.0/req/deployable-workflows)

The *Deployable Workflows* conformance class extends the execution request of *OGC API — Processes — Part 1: Core* with the ability to define parameterized inputs and select outputs in the context of deploying the workflow as a new process using *OGC API — Processes — Part 2: Deploy, Replace, Undeploy*. An `"$input"` key is used to define an external input to the deployed process, and an `"$output"` key is used to select an output from the deployed process. The extended JSON execution request defining the workflow can either be the direct payload of deployment `POST` request, or the execution unit of an *OGC Application Package*, if the implementation supports the associated *Part 2* conformance class.

## 2.2. Requirements class specifying a new on-demand, client-driven execution mechanism

**Requirements Class "Collection Output"** (http://www.opengis.net/spec/ogcapi-processes-3/0.0/req/collection-output)

The *Collection Output* conformance class specifies a new execution mechanism allowing to trigger the execution of the process of workflow for a specific area, time and resolution of interest as a direct result of requests for output data using OGC API data access mechanisms such as those defined by *OGC API — Tiles*, *DGGS*, *Coverages*, *Features*, *EDR* or *Maps*, in a manner completely agnostic to process execution. With this execution mechanism, selected by a new query parameter, the implementation's initial response is either a collection description document (`response=collection`), or an OGC API landing page (`response=landingPage`).

## 2.3. Requirements classes specifying alternative ad-hoc workflow definitions

**Requirements Class "Common Workflow Language Definitions"** (http://www.opengis.net/spec/ogcapi-processes-3/0.0/req/cwl-workflows)

The *Common Workflow Language Definitions* conformance class specifies the ability to execute Common Workflow Language (CWL) workflows in an ad-hoc manner i.e., without first requiring to deploy the workflow as a process (see also the relevant *Part 2* conformance class to deploy CWL workflow as a process).

**Requirements Class "OpenEO Process Graphs Workflows"** (http://www.opengis.net/spec/ogcapi-processes-3/0.0/req/openeo-workflows)

The *OpenEO Process Graphs Definitions* conformance class specifies the ability to execute workflows defined using OpenEO Process Graphs in an ad-hoc manner i.e., without first requiring to deploy the workflow as a process (see also the relevant *Part 2* conformance class to deploy OpenEO process graphs as a process).

All these conformance classes act as building blocks that should be implemented in combination with *OGC API — Processes — Part 1: Core*, as well as with *Part 2: Deploy*, *Replace*, *Undeploy* and/or OGC API data access specifications such as *OGC API — Tiles*, *DGGS*, *Coverages*, *Features*, *EDR*, *Maps*, when applicable.

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

## 2.4. Declaration of conformance

In order to conform to this OGC® interface standard, a software implementation shall choose to implement any one of the conformance levels specified in Annex A (normative). Conformance with this standard shall be checked using all the relevant tests specified in Annex A (normative) of this document if the respective conformance URLs listed in Table 1 is present in the conformance response.

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 1** — Conformance class URIs

| CONFORMANCE CLASS | URI |
|---|---|
| Nested Processes | http://www.opengis.net/spec/ogcapi-processes-3/0.0/conf/nested-processes |
| Remote Core Processes | http://www.opengis.net/spec/ogcapi-processes-3/0.0/conf/remote-core-processes |
| Collection Input | http://www.opengis.net/spec/ogcapi-processes-3/0.0/conf/collection-input |
| Remote Collections | http://www.opengis.net/spec/ogcapi-processes-3/0.0/conf/remote-collections |
| Input Fields Modifiers | http://www.opengis.net/spec/ogcapi-processes-3/0.0/conf/input-fields-modifiers |
| Output Fields Modifiers | http://www.opengis.net/spec/ogcapi-processes-3/0.0/conf/output-fields-modifiers |
| Deployable Workflows | http://www.opengis.net/spec/ogcapi-processes-3/0.0/conf/deployable-workflows |
| Collection Output | http://www.opengis.net/spec/ogcapi-processes-3/0.0/conf/collection-output |
| Common Workflow Language Definitions | http://www.opengis.net/spec/ogcapi-processes-3/0.0/conf/cwl-workflows |
| OpenEO Process Graphs Definitions | http://www.opengis.net/spec/ogcapi-processes-3/0.0/conf/openeo-workflows |

3

# NORMATIVE REFERENCES

# 3 NORMATIVE REFERENCES

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


Benjamin Pross, Panagiotis (Peter) A. Vretanos: OGC 18-062r2, *OGC API — Processes — Part 1: Core*. Open Geospatial Consortium (2021). https://docs.ogc.org/is/18-062r2/18-062r2.html.

Joan Masó, Jérôme Jacovella-St-Louis: OGC 20-057, *OGC API — Tiles — Part 1: Core*. Open Geospatial Consortium (2022). https://docs.ogc.org/is/20-057/20-057.html.

Clemens Portele, Panagiotis (Peter) A. Vretanos, Charles Heazel: OGC 17-069r3, *OGC API — Features — Part 1: Core*. Open Geospatial Consortium (2019). https://docs.ogc.org/is/17-069r3/17-069r3.html.

Mark Burgoyne, David Blodgett, Charles Heazel, Chris Little: OGC 19-086r5, *OGC API — Environmental Data Retrieval Standard*. Open Geospatial Consortium (2022). https://docs.ogc.org/is/19-086r5/19-086r5.html.

Panagiotis (Peter) A. Vretanos: OGC API — Processes — Part 2: Deploy, Replace, Undeploy (Draft). OGC 20-044, Open Geospatial Consortium, https://github.com/opengeospatial/ogcapi-processes/tree/master/extensions/deploy_replace_undeploy/standard

Charles Heazel: OGC 19-072, *OGC API — Common — Part 1: Core*. Open Geospatial Consortium (2023). https://docs.ogc.org/is/19-072/19-072.html.

Charles Heazel: OGC API — Common — Part 2: Geospatial Data (Draft). OGC 20-024, Open Geospatial Consortium, http://docs.ogc.org/DRAFTS/20-024.html

Panagiotis (Peter) A. Vretanos: OGC Common Query Language (CQL2) (Draft). OGC 21-065, Open Geospatial Consortium, http://docs.ogc.org/DRAFTS/21-065.html

Panagiotis (Peter) A. Vretano, Clemens Portele: OGC API — Features — Part 3: Filtering (Draft). OGC 19-079r1, Open Geospatial Consortium, http://docs.ogc.org/DRAFTS/19-079r1.html

Charles Heazel, Jérôme Jacovella-St-Louis: OGC API — Coverages — Part 1: Core (Draft). OGC 19-087, Open Geospatial Consortium, http://docs.ogc.org/DRAFTS/19-087.html

Joan Masó, Jérôme Jacovella-St-Louis: OGC API — Maps — Part 1: Core (Draft). OGC 19-058, Open Geospatial Consortium, http://docs.ogc.org/DRAFTS/20-058.html

Matthew B.J. Purss, Jérôme Jacovella-St-Louis: OGC API — Discrete Global Grid Systems — Part 1: Core (Draft). OGC 21-038, Open Geospatial Consortium, https://opengeospatial.github.io/ogcna-auto-review/21-038.html

4

# TERMS AND DEFINITIONS

# 4 TERMS AND DEFINITIONS

This document uses the terms defined in OGC Policy Directive 49, which is based on the ISO/IEC Directives, Part 2, Rules for the structure and drafting of International Standards. In particular, the word “shall” (not “must”) is the verb form used to indicate a requirement to be strictly followed to conform to this document and OGC documents do not use the equivalent phrases in the ISO/IEC Directives, Part 2.

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

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

## 4.1. example term

term used for exemplary purposes

**Note 1 to entry:** An example note.

Example Here’s an example of an example term.

[**SOURCE:** ]

5

# CONVENTIONS

# 5 CONVENTIONS

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

## 5.1. Identifiers

The normative provisions in this standard are denoted by the URI

`http://www.opengis.net/spec/ogcapi-processes-3/0.0`

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

6

# OVERVIEW

# 6 OVERVIEW

As an extension to *OGC API — Processes — Part 1: Core*, this specification defines conformance classes for:

- extending execution requests submitted to `/processes/{processId}/execution`
  - referencing local and remote nested processes as inputs (`"process"`) (Section 7),
  - referencing local and remote OGC API collections as inputs (`"collection"`) (Section 8),
  - modifying data accessed as inputs and returned as outputs (filtering with `"filter"`, selecting and deriving fields with `"properties"`, sorting with `"sortBy"`) (Section 9),
  - deploying workflows defined using such execution request as processes using *OGC API — Processes — Part 2: Deploy, Replace, Undeploy*, with the ability to paremeterize inputs (`"$input"`) and select outputs (`"$output"`) (Section 10),
- requesting output data from OGC API data collections to trigger processing execution (Section 11),
- ad-hoc execution of workflows defined using the Common Workflow Language (Section 12), and
- ad-hoc execution of workflows defined using the OpenEO Process Graphs (Section 13).

## 6.1. Architecture Diagram

The following diagram illustrates how the conformance classes defined in this specification extending *OGC API — Processes — Part 1: Core* execution request as well as describing a new “Collection Output” execution mechanism work together with OGC API data access specifications.

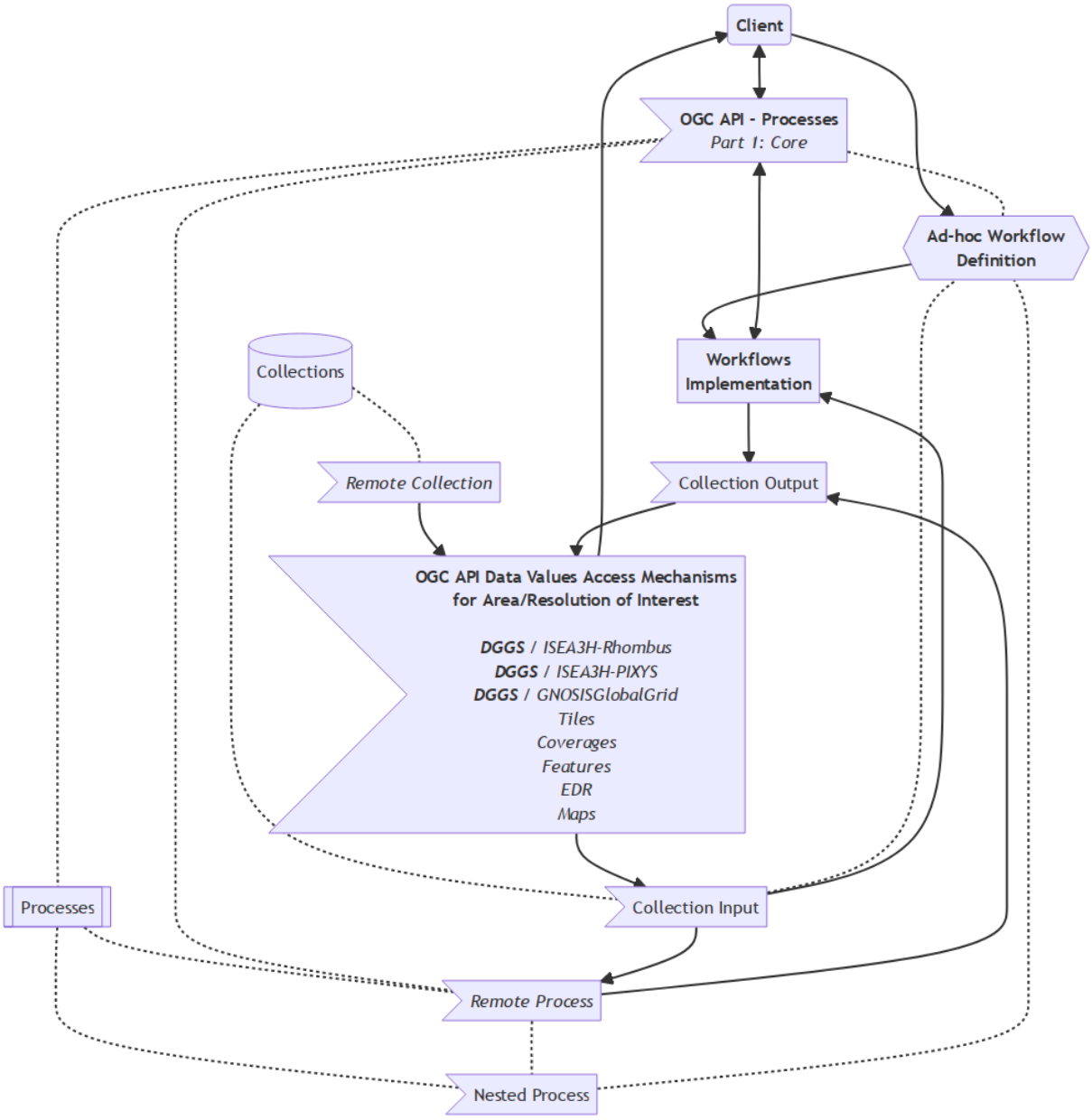


**Figure 1** — Workflows extension architecture diagram

## 6.2. Design goals

### 6.2.1. Objective

This specification aims to enable instant integration of geospatial data and processing capabilities available from anywhere for visualization and analysis.

### 6.2.2. Obstacles addressed

#### 6.2.2.1. Long batch processing times

The conformance classes defined in this specification aim to avoid long batch processing times, because they present several drawbacks:

- long feedback loops (configuring and adjusting parameters, retrieving and preparing input data, processing, visualizing / validating output for entire large datasets) slow down research and products deployment,
- processing results may be outdated by the time they are ready,
- batch processing entire datasets makes it difficult to prioritize more important processing (e.g., higher priority clients, emergency response) over less important requests (e.g., flooding attacks, high volume of requests from low priority clients),
- inefficient use of storage, bandwidth, computing, time, and therefore financial resources (large portions of datasets may be processed without the results ever getting used).

#### 6.2.2.2. Integration of diverse data and processes

The conformance classes defined in this specification aim to address the following challenges bringing together datasets and processes from different origins:

- the suitability and compatility of data sources and processing capabilities,
- the added complexity of deploying processes or workflow definitions in order to execute them, and associated authentication requirements,
- the interoperability issues associated with specific expectations in terms of format or API as part of a workflow definition,
- the possibility of re-using a workflow definition with similar datasets, and multiple areas, times or regions of interest.

### 6.2.3. Framework

These challenges are addressed within the consistent framework of the OGC API family of standards, as defined by *OGC API — Common — Part 1: Core* and *Part 2: Geospatial data*, by extending the processing capabilities defined in *OGC API — Processes — Part 1: Core* and *Part 2: Deploy, Replace, Undeploy* with the ability to seamlessly and effortlessly integrate with the access capabilities defined in OGC API data access specifications (such as *OGC API — Tiles*, *DGGS*, *Coverages*, *Features*, *EDR*, *Maps*...).

### 6.2.4. Approach

The conformance classes defined in this specification allow to:

- ***easily connect data and processes from anywhere in a re-usable and interoperable manner by***
    - leveraging the consistency of the OGC API family of standards, making it easy to find relevant processes and data collections regardless of access mechanisms,
    - leveraging the concept of a *GeoDataClass*, to easily establish the compatibility of a given data as an input to a particular process,
    - allowing ad-hoc execution of workflows definitions, without first needing to deploy the workflow as a process, making it more accessible for integration,
    - select formats and access mechanisms outside workflow definition, negotiating at each hop of a workflow, therefore making them more interoperable and re-usable,
    - making workflows definitions agnostic of area, time and region of interest, therefore easier to re-use;

- ***trigger on-demand processing with OGC API data access requests***
    - making it easy for OGC API data access clients to readily support processing while requiring very minimal processing-specific code (performing a single POST request),
    - making it possible to prioritize processing based on client requests and their importance based on credentials,
    - ensuring the latest available data is always used,
    - enabling instant feedback allowing to validate and tweak input parameters based on initial small requests tailored to the area and resolution of interest of a given view (or preview) of the results;

- ***minimize data exchange bandwidth requirements by***
    - requesting subset and downsampled data focusing on area, time, resolution and fields/bands of interests,

- allowing to optimize usage of processing and bandwidth resources while decreasing wait times.

### 6.2.5. Considerations for collection input / output

An important design goal of this extension is to separate from the workflow definition the negotiation of formats, data transfer APIs, area, time and resolution of intererest, and selection of outputs, therefore greatly enhancing the re-usability of the workflows. This negotiation happens at each hop along a workflow chain, and the end-user client need only decide on these for the server it directly connnects to (immediate hop). For example, some hops may operate better using DGGS zone IDs, while other using subset requests, and yet others using tiles; or some hops may support and work best with some format or API, while others work best with another.

When setting up a new workflow for collection output our asynchronous execution, if possible, the API should fully validate the immediate process and all nested inputs prior to returning a result. A successful result indicates a fully validated workflow which should normally succeed for requests within the range for which data is produced. It is expected that clients may make a large number of requests for the same workflow e.g., using subset or tiles or DGGS zone ID to trigger processing of a specific Area of Interest or resolution, as a client pans and zooms around a map.

With *collection input / output*, formats, CRS, specific APIs, or specific APIs do not have to be specified and should not be, so as to maximize the re-usability of the workflow. e.g., the data requests triggering the processing would specify the output formats via content type negotiation, and tiles requests would provide the AoI and resolution for any specific request which would propagate upwards in the workflow request chain.

7

# NESTED PROCESSES AND REMOTE PROCESSES CONFORMANCE CLASSES

# 7 NESTED PROCESSES AND REMOTE PROCESSES CONFORMANCE CLASSES

## 7.1. Requirement Class *Nested Processes*

The *Nested Processes* conformance class allows to use the output of a process as an input to another process. Unless an implementation also conforms to the *Remote Processes* conformance class, only local processes are supported.

Specifying a full process URI at the top-level for a process execution request is optional, since the execution request will be posted to a specific `{processId}`, but allows the workflow definition to be self-describing and reusable, enabling GIS clients to easily load the workflow definition as a data source, or publish it as a new virtual collection to any OGC API supporting execution requests as a content type for a `POST` method to `/collections`.

| REQUIREMENTS CLASS 1 | |
|---|---|
| **TARGET TYPE** | Nested Processes |
| **PREREQUISITES** | http://www.opengis.net/spec/ogcapi-processes-1/1.0/req/core<br>http://www.opengis.net/spec/ogcapi-processes-3/0.0/req/input-fields-modifiers<br>http://www.opengis.net/spec/ogcapi-processes-3/0.0/req/output-fields-modifiers<br>http://www.opengis.net/spec/ogcapi-processes-3/0.0/req/collection-input<br>http://www.opengis.net/spec/ogcapi-processes-3/0.0/req/remote-collections |
| **LABEL** | http://www.opengis.net/spec/ogcapi-processes-3/0.0/req/nested-processes |

The following requirements describe how an execution request references an OGC API process and how an implementation executes that process to use its output as an input to another process.

### 7.1.1. Referencing a process

| REQUIREMENT 1 | |
|---|---|
| **LABEL** | /req/nested-processes/referenced-process |
| **STATEMENT** | For referencing a process as an input to another process: |

| REQUIREMENT 1 | |
|---|---|
| A | The Implementation SHALL support execution requests containing a `"process"` key and a corresponding URI or Relative-URI value identifying a process as an input to another process. |
| B | The Implementation SHALL interpret Relative-URIs as relative to the parent process within which the containing process object is nested. |

### 7.1.2. Executing a nested process

| REQUIREMENT 2 | |
|---|---|
| LABEL | /req/nested-processes/process-execution |
| STATEMENT | For executing workflow definitions referencing a nested process: |
| A | The Implementation SHALL support executing local processes which can be executed using *OGC API — Process — Part 1: Core* as a process nested as an input to another process. |
| B | The Implementation SHALL consider the selected output (using the “outputs” key), or default to the only or first output of the process as the input. |

**NOTE:** For local processes, the specification is agnostic to how data flows from one process to the other.

## 7.2. Requirement Class *Remote Core Processes*

The *Remote Core Processes* conformance class allows to reference and execute a remote OGC API process using the *Core* conformance class of *OGC API — Processes — Part 1: Core* in addition to any local process.

| REQUIREMENTS CLASS 2 | |
|---|---|
| TARGET TYPE | Remote Core Processes |
| PREREQUISITE | http://www.opengis.net/spec/ogcapi-processes-3/0.0/req/nested-processes |
| LABEL | http://www.opengis.net/spec/ogcapi-processes-3/0.0/req/remote-core-processes |

The following requirements describe how an execution request references a remote OGC API process and how an implementation executes that remote process to use as an input to another process.

### 7.2.1. Referencing a remote process

| REQUIREMENT 3 | |
|---|---|
| **LABEL** | /req/nested-processes/referenced-process |
| **STATEMENT** | For referencing a remote process as an input to another process: |
| **A** | The Implementation SHALL support execution requests containing a `"process"` key and a corresponding URI or Relative-URI value identifying an authorized remote process as an input to another process. |

**NOTE:** The specification is agnostic to how an implementation determines whether a remote process is authorized or not. This may be based e.g., on allow/disallow lists, credentials and/or federations.

### 7.2.2. Executing a nested remote process

| REQUIREMENT 4 | |
|---|---|
| **LABEL** | /req/nested-processes/process-execution |
| **STATEMENT** | For executing workflow definitions referencing a nested remote process: |
| **A** | The Implementation SHALL support executing remote processes which can be executed using *OGC API — Process — Part 1: Core* as a process nested as an input to another process. |
| **B** | The Implementation SHALL support executing remote processes and retrieving their output using the *Core* conformance class of the *OGC API — Processes — Part 1: Core*. |

**NOTE:** The ability to support nested remote processes whose services support the *Collection Output* conformance class can also be achieved without conformance to this requirements class, through the *Nested Processes* and *Remote Collections* conformance classes.

8

# COLLECTION INPUT AND REMOTE COLLECTIONS CONFORMANCE CLASSES

# 8 COLLECTION INPUT AND REMOTE COLLECTIONS CONFORMANCE CLASSES

## 8.1. Requirement Class *Collection Input*

The *Collection Input* conformance class allows to specify an OGC API collection available using one or more OGC API data access mechanism specification as an input to a process. Unless an implementation also conforms to the *Remote Collections* conformance class, only local collections are supported.

| REQUIREMENTS CLASS 3 | |
|---|---|
| **TARGET TYPE** | Collection Input |
| **PREREQUISITES** | http://www.opengis.net/spec/ogcapi-processes-1/1.0/req/core<br>http://www.opengis.net/spec/ogcapi-processes-3/0.0/req/input-fields-modifiers |
| **LABEL** | http://www.opengis.net/spec/ogcapi-processes-3/0.0/req/collection-input |

The following requirements describe how an execution request references an OGC API collection and how an implementation requests data from that collection to use as an input to a process.

### 8.1.1. Referencing a collection

| REQUIREMENT 5 | |
|---|---|
| **LABEL** | /req/collection-input/referenced-collection |
| **STATEMENT** | For referencing a collection as an input to a process: |
| **A** | The Implementation SHALL support execution requests containing a `"collection"` key and a corresponding URI or Relative-URI value identifying an OGC API collection as an input to a process. |
| **B** | The Implementation SHALL interpret Relative-URIs as relative to the parent process within which the containing collection object is nested. |

### 8.1.2. Accessing data from a collection

| REQUIREMENT 6 | |
|---|---|
| **LABEL** | /req/collection-input/collection-access |
| **STATEMENT** | For executing workflow definitions referencing a collection as an input to a process: |
| **A** | The Implementation SHALL support accessing local collections (as defined by *OGC API — Common — Part 2: Geospatial data*) accessible using at least one OGC API data access mechanisms (e.g., *OGC API — Tiles*, *Coverages*, *DGGS*, *Features*, *EDR*, *Maps*) as an input to a process. |

**NOTE:** For local collections, the specification is agnostic to how data flows from the collection to the process.

## 8.2. Requirement Class *Remote Collections*

The *Remote Collections* conformance class allows to reference a remote OGC API collection in addition to any local collections.

If the *Nested Processes* conformance class is also supported, but *Remote Processes* is not, an implementation still provides the ability to substitute a collection by remote nested processes supporting the *Collection Output* conformance class as input to a process.

| REQUIREMENTS CLASS 4 | |
|---|---|
| **TARGET TYPE** | Remote Collections |
| **PREREQUISITES** | http://www.opengis.net/spec/ogcapi-processes-3/0.0/req/collection-input<br>http://www.opengis.net/spec/ogcapi-processes-3/0.0/req/nested-process |
| **LABEL** | http://www.opengis.net/spec/ogcapi-processes-3/0.0/req/remote-collections |

The following requirements describe how an execution request references a remote OGC API collection and how an implementation requests data from that collection to use as an input to a process.

### 8.2.1. Referencing a remote collection

| REQUIREMENT 7 | |
|---|---|
| **LABEL** | /req/remote-collections/referenced-collection |
| **STATEMENT** | For referencing a remote collection as an input to a process: |
| **A** | The Implementation SHALL support execution requests containing a `"collection"` key and a corresponding URI or Relative-URI value identifying an authorized OGC API collection as an input to a process. |
| **B** | If the Implementation conforms to the *Nested Process* conformance class, it SHALL also support execution requests containing a `"process"` key and a corresponding URI or Relative-URI value identifying an authorized remote process supporting the *Collection Output* conformance class as an input to another process. |
| **C** | The Implementation SHALL interpret Relative-URIs as relative to the parent process within which the containing collection or process object is nested. |

### 8.2.2. Accessing data from a remote collection

| REQUIREMENT 8 | |
|---|---|
| **LABEL** | /req/collection-input/collection-access |
| **STATEMENT** | For executing workflow definitions referencing a collection as an input to a process: |
| **A** | The Implementation SHALL support accessing a remote OGC API collection using one or more OGC API data access mechanisms (e.g., *OGC API — Tiles*, *Coverages*, *DGGS*, *Features*, *EDR*, *Maps*...) as an input to a process. |

9

# INPUT/OUTPUT FIELDS MODIFERS CONFORMANCE CLASSES

# 9 INPUT/OUTPUT FIELDS MODIFERS CONFORMANCE CLASSES

## 9.1. Requirement Class *Input Fields Modifers*

The *Input Fields Modifers* conformance class specifies how execution requests can modify an input (including those specified using the *Collection Input*, *Remote Collections*, *Nested Processes* and *Remote Processes* conformance classes) by filtering its data according to its fields (properties), selecting a subset of the data fields or deriving new fields from existing ones (e.g., by performing arithmetic), and/or sorting the data according to the value of fields. An expression language such as the OGC Common Query Language 2 (CQL2) can be used for this purpose. There are plans to extend CQL2 beyond boolean predicates to expressions that resolve to other types of values (e.g., numbers or geometry).

| REQUIREMENTS CLASS 5 | |
|---|---|
| **TARGET TYPE** | Input Fields Modifiers |
| **PREREQUISITES** | http://www.opengis.net/spec/ogcapi-processes-1/1.0/req/core<br>http://www.opengis.net/spec/ogcapi-features-3/0.0/req/features-filter |
| **LABEL** | http://www.opengis.net/spec/ogcapi-processes-3/0.0/req/input-fields-modifiers |

The following requirements describe the ability of an implementation to modify the data accessed as an input to a process by first applying a filter (`"filter"`), selecting a subset of the fields or deriving new fields (`"properties"`) or sorting the data (`"sortBy"`), before providing the data as input to the process. In an execution request, these additional keys are specified at the same level as the `"collection"` defined in the *Collection Input* conformance class, or the `"process"` defined in the *Nested Process* conformance class.

For an input generated by a nested process, this capability is equivalent to the *Output Fields Modifiers* conformance class from the perspective of that nested process. For a remote process (as defined in *Remote Processes* conformance class) that supports *Output Fields Modifiers*, or for a collection (or a process supporting *Collection Output*) with a data access mechanism providing the required functionality, it is preferrable to let the remote process or collection access handle the modification so as to lower bandwidth and processing requirements.

For an input from a *remote collection*, a data access mechanism for the collection may readily support some or all of these parameters (e.g., *OGC API — Features — Part 3: Filtering*). In this case, it is preferable for the Implementation to query the data with the relevant parameters.

This ability to pass on these modifiers to remote data and processing sources allows to easily bring a certain category of algorithms close to the data through simple standard expressions.

The extent to which an implementation support these field modifiers capabilities depends on its stated level of conformance to one or more expression language.

### 9.1.1. Filtering

| REQUIREMENT 9 | |
|---|---|
| **LABEL** | /req/input-fields-modifiers/filtering |
| **STATEMENT** | For filtering the data coming in as an input to a process: |
| **A** | The Implementation SHALL support specifying a `"filter"` key and a corresponding filter predicate as part of an input to a process. |
| **B** | The Implementation SHALL support one or more expression languages to define the filter, such as the OGC Common Query Language (CQL2). |

### 9.1.2. Selecting and Deriving Fields

| REQUIREMENT 10 | |
|---|---|
| **LABEL** | /req/input-fields-modifiers/derived-fields |
| **STATEMENT** | For selecting and/or deriving new fields from the input data to a process: |
| **A** | As part of an input to a process, the Implementation SHALL support specifying a `"properties"` key and a corresponding set of selected fields as an array of field names. |
| **B** | As part of an input to a process, the Implementation SHALL support specifying a `"properties"` key and a corresponding set of derived fields expressions as a key / value pair object mapping expressions to field names. |
| **C** | The Implementation SHALL support one or more expression languages to define the derived fields, such as the OGC Common Query Language (CQL2). |

### 9.1.3. Sorting

| REQUIREMENT 11 | |
|---|---|
| LABEL | /req/input-fields-modifiers/sorting |
| STATEMENT | For sorting the input data to a process according to its data fields: |
| A | The Implementation SHALL support specifying a `"sortBy"` key and a corresponding comma-separated sequence of expressions defining the order in which to sort the data coming in as an input to a process. |
| B | The Implementation SHALL support one or more expression languages to define the sorting expressions, such as the OGC Common Query Language (CQL2). |

## 9.2. Requirement Class *Output Fields Modifers*

The *Output Fields Modifers* conformance class specifies how execution requests can modify the data output from a process by filtering its data according to its fields (properties), selecting a subset of the data fields or deriving new fields from existing ones (e.g., by performing arithmetic), and/or sorting the data according to the value of fields. An expression language such as the OGC Common Query Language 2 (CQL2) can be used for this purpose. There are plans to extend CQL2 beyond boolean predicates to expressions that resolve to other types of values (e.g., numbers or geometry).

This capability is available whether using the *Collection Output* conformance class, or the synchronous or asynchronous execution capabilities of *OGC API — Processes — Part 1: Core*.

| REQUIREMENTS CLASS 6 | |
|---|---|
| TARGET TYPE | Output Fields Modifiers |
| PREREQUISITE | http://www.opengis.net/spec/ogcapi-processes-1/1.0/req/core |
| LABEL | http://www.opengis.net/spec/ogcapi-processes-3/0.0/req/output-fields-modifiers |

The following requirements describe the ability of an implementation to modify the data output from a process by applying a filter (`"filter"`), selecting a subset of the fields or deriving new fields (`"properties"`) or sorting the data (`"sortBy"`), before returning the data as an output of a process. In an execution request, these additional keys are specified at the same level as the `"process"` defined in the *Nested Process* conformance class, or at the top-level of an execution request as defined in *OGC API — Processes — Part 1: Core*.

When a nested process is used as an input to another process, this capability is equivalent to the *Input Fields Modifiers* conformance class from the perspective of the parent process receiving the output of the nested process. For a remote process (as defined in *Remote Processes* conformance

class) that supports *Output Fields Modifiers*, or supports *Collection Output* with a data access mechanism providing the required functionality, it is preferrable to let the remote process handle the modification so as to lower bandwidth and processing requirements.

The extent to which an implementation support these capabilities depends on its stated level of conformance to one or more expression language.

### 9.2.1. Filtering

| REQUIREMENT 12 | |
|---|---|
| **LABEL** | /req/output-fields-modifiers/filtering |
| **STATEMENT** | For filtering the data returned as a data output from a process: |
| **A** | The Implementation SHALL support specifying a `"filter"` key and a corresponding filter predicate as part of a nested process or top-level execution request. |
| **B** | The Implementation SHALL support one or more expression languages to define the filter, such as the OGC Common Query Language (CQL2). |

### 9.2.2. Selecting and Deriving Fields

| REQUIREMENT 13 | |
|---|---|
| **LABEL** | /req/output-fields-modifiers/derived-fields |
| **STATEMENT** | For selecting and/or deriving new fields returned as a data output from a process: |
| **A** | As part of a nested process or top-level execution request object, the Implementation SHALL support specifying a `"properties"` key and a corresponding set of selected fields as an array of field names. |
| **B** | As part of a nested process or top-level execution request object, the Implementation SHALL support specifying a `"properties"` key and a corresponding set of derived fields expressions as a key / value pair object mapping expressions to field names. |
| **C** | The Implementation SHALL support one or more expression languages to define the derived fields, such as the OGC Common Query Language (CQL2). |

### 9.2.3. Sorting

| REQUIREMENT 14 | |
|---|---|
| **LABEL** | /req/output-fields-modifiers/sorting |
| **STATEMENT** | For sorting the data returned as an output from a process according to its data fields: |
| **A** | The Implementation SHALL support specifying a `"sortBy"` key and a corresponding comma-separated sequence of expressions defining the order in which to sort the data returned as part of a nested process or top-level execution request. |
| **B** | The Implementation SHALL support one or more expression languages to defined the sorting expressions, such as the OGC Common Query Language (CQL2). |

10

# DEPLOYABLE WORKFLOWS CONFORMANCE CLASS

# 10 DEPLOYABLE WORKFLOWS CONFORMANCE CLASS

## 10.1. Requirement Class *Deployable Workflows*

The *Deployable Workflows* conformance class specifies how a workflow execution request as defined in *OGC API — Processes — Part 1: Core*, with or without the capabilities defined in other conformance classes of this *Part 3* extension, can be used as an application package payload to deploy new processes using *OGC API — Processes — Part 2: Deploy, Replace, Undeploy*.

It also specifies how to parameterize inputs to the workflow as inputs to the deployed process, and how to select specific outputs from the processes involved to be returned as outputs of the deployed process.

| REQUIREMENTS CLASS 7 | |
|---|---|
| **TARGET TYPE** | Deployable Workflows |
| **PREREQUISITES** | http://www.opengis.net/spec/ogcapi-processes-2/0.0/req/deploy-replace-undeploy<br>http://www.opengis.net/spec/ogcapi-processes-2/0.0/req/ogcapppkg<br>http://www.opengis.net/spec/ogcapi-processes-3/0.0/req/nested-processes<br>http://www.opengis.net/spec/ogcapi-processes-3/0.0/req/remote-processes<br>http://www.opengis.net/spec/ogcapi-processes-3/0.0/req/collection-input<br>http://www.opengis.net/spec/ogcapi-processes-3/0.0/req/remote-collections<br>http://www.opengis.net/spec/ogcapi-processes-3/0.0/req/input-fields-modifiers<br>http://www.opengis.net/spec/ogcapi-processes-3/0.0/req/output-fields-modifiers |
| **LABEL** | http://www.opengis.net/spec/ogcapi-processes-3/0.0/req/deployable-workflows |

The following requirements describe the ability of an implementation to deploy a new process defined as an *OGC API — Processes — Part 1: Core* execution request with *OGC API — Processes — Part 2: Deploy, Replace, Undeploy*, including parameterizing inputs as external inputs to the deployed process using `{ "$input" : "{input name}" }` as input objects (in `"inputs"`), and selecting specific outputs to be returned from the deployed process using `{ "$output" : "{output name}" } as output objects (in `"outputs"`). If no `"$output"` object is used within a deployed workflow definition, the deployed process is assumed to return all outputs from the top-level process with the same names as that process.

A minimal but adequate process description for the resulting process can automatically be generated based on the context where the `"$input"` and `"$output"` objects appear within the

workflow definition, allowing to describe all inputs to the process and its outputs by inferrence from the source process descriptions and/or input data.

The workflow definition could either directly be the payload of the `POST` request to `/processes` (using `Content-Type: application/json`), or be part of the `executionUnit` of an OGC Application Package as defined in Part 2's *OGC Application Package* conformance class (using `Content-Type: application/ogcapppkg+json`).

**NOTE:** The `application/ogcapppkg+json` media type needs to be registered with IANA.

### 10.1.1. Parameterized inputs

| REQUIREMENT 15 | |
|---|---|
| **LABEL** | /req/deployable-workflows/parameterized-inputs |
| **STATEMENT** | For deploying workflow definitions defined as an execution request: |
| **A** | The Implementation SHALL support defining a parameterizable input using the `"$input"` key and a corresponding input name value as part of an input to a process (within the `inputs` of an execution request or nested input process). |
| **B** | The deployed process SHALL have a corresponding input for each input selected using the `"$input"` using the specified name. |
| **C** | Each input of the deployed process SHALL inherit the expected data type of the input in the context where the `"$input"` key is specified. |

### 10.1.2. Selected outputs

| REQUIREMENT 16 | |
|---|---|
| **LABEL** | /req/deployable-workflows/parameterized-outputs |
| **STATEMENT** | For deploying workflow definitions defined as an execution request: |
| **A** | The Implementation SHALL support selecting specific outputs of processess involved in workflow definition using the `"$output"` key and a corresponding output name value as part of an output of a process (within the `outputs` of an execution request or nested input process). |
| **B** | The deployed process SHALL have a corresponding output for each output selected using the `"$output"` using the specified name. |
| **C** | Each output of the deployed process SHALL inherit the expected data type of the output in the context where the `"$output"` key is specified. |

| REQUIREMENT 16 | |
|---|---|
| D | If no “$output” key appears in a workflow definition, the Implementation SHALL assume that all outputs from the top-level process are returned, using the same name as specified in the description of that top-level process. |

### 10.1.3. Deploying workflow as process

| REQUIREMENT 17 | |
|---|---|
| LABEL | /req/deployable-workflows/deploying-workflow |
| STATEMENT | For deploying workflow definitions defined as an execution request: |
| A | The Implementation SHALL support deploying a workflow definition as a process using the POST method mechanism described in *OGC API — Processes — Part 2: Deploy, Replace, Undeploy* with a payload consisting of an *OGC API — Processes — Part 1: Core* execution request (`Content-Type: application/json`, unless a specific media type execution requests is allocated), with support for any any extensions defined in this specification for which conformance is declared. |
| B | If the *OGC Application Package* conformance class is supported, the Implementation SHALL also support deploying a workflow definition as a process using the POST method mechanism described in *OGC API — Processes — Part 2: Deploy, Replace, Undeploy* with the `executionUnit` of the OGC Application Package payload (`Content-Tyep: application/ogcapppkg+json`) consisting of an *OGC API — Processes — Part 1: Core* execution request, with support for any any extensions defined in this specification for which conformance is declared. |

11

# COLLECTION OUTPUT CONFORMANCE CLASS

# 11 COLLECTION OUTPUT CONFORMANCE CLASS

## 11.1. Requirement Class *Collection Output*

The *Collection Output* conformance class specifies how clients can trigger the execution of a process or workflow for a specific area, time and resolution of interest as a result of requesting output data using OGC API data access mechanisms, such as *OGC API — Tiles*, *DGGS*, *Coverages*, *Features*, *EDR*, or *Maps*.

This provides an alternative to the synchronous and asynchronous execution mechanisms defined in *OGC API — Processes — Part 1: Core* for which area, time and resolution of interest are hardcoded in the execution request. With *Core*, when lengthy batch processing is not desirable, completely new execution requests returning a partial output must continuously be re-submitted.

This conformance class allows to submit an execution request once and then access the output using mechanisms widely supported by clients in a manner completely agnostic of process execution.

Compared to separate execution requests, this mechanism makes it easier for implementations to optimize for scenarios where clients will stream small partial outputs from the same pre-established processing pipeline. Parsing the execution request, validating inputs, and handshakes with remote components of a workflow only need to be performed once. An implementation could also preempt future requests based on past requests, offsetting the latency of a complex distributed workflow.

A collection document for the result generated from a workflow MAY provide a link to the source definition of that workflow (link relation type: `http://www.opengis.net/def/rel/ogc/1.0/workflow`).

A service MAY also implement the capability to deploy a workflow as a persistent collection e.g., by supporting a `POST` method for the `/collections` end-point accepting a JSON workflow definition (execution request) as payload.

| REQUIREMENTS CLASS 8 | |
|---|---|
| **TARGET TYPE** | Collection Output |
| **PREREQUISITES** | http://www.opengis.net/spec/ogcapi-processes-1/1.0/req/core<br>http://www.opengis.net/spec/ogcapi-common-2/1.0/req/collections<br>http://www.opengis.net/spec/ogcapi-tiles-1/1.0/conf/geodata-tilesets<br>http://www.opengis.net/spec/ogcapi-features-1/1.0/conf/core |

| REQUIREMENTS CLASS 8 | |
|---|---|
| | http://www.opengis.net/spec/ogcapi-coverages-1/0.0/conf/core<br>http://www.opengis.net/spec/ogcapi-coverages-1/0.0/conf/coverage-scaling<br>http://www.opengis.net/spec/ogcapi-coverages-1/0.0/conf/coverage-subset<br>http://www.opengis.net/spec/ogcapi-coverages-1/0.0/conf/coverage-rangesubset<br>http://www.opengis.net/spec/ogcapi-dggs-1/0.0/conf/data-retrieval<br>http://www.opengis.net/spec/ogcapi-dggs-1/0.0/conf/zone-query<br>http://www.opengis.net/spec/ogcapi-edr-1/1.0/conf/core<br>http://www.opengis.net/spec/ogcapi-edr-1/1.0/conf/collections<br>http://www.opengis.net/spec/ogcapi-maps-1/0.0/conf/core<br>http://www.opengis.net/spec/ogcapi-maps-1/0.0/conf/spatial-subset<br>http://www.opengis.net/spec/ogcapi-maps-1/0.0/conf/datetime<br>http://www.opengis.net/spec/ogcapi-maps-1/0.0/conf/scaling |
| **LABEL** | http://www.opengis.net/spec/ogcapi-processes-3/0.0/req/collection-output |

The following requirements describe an execution mechanism where a client is redirected (using a `303` *See Other* HTTP status code and a `Location:` header) to an OGC API landing page or collection description document, which provides links to request output data using standard OGC API data access mechanisms, triggering on-demand processing for a particular area, time and resolution of interest.

### 11.1.1. `response` query parameter

| REQUIREMENT 18 | |
|---|---|
| **LABEL** | /req/collection-output/response-query-parameter |
| **STATEMENT** | For initiating the execution of a workflow and retrieving its output as an OGC API collection: |
| **A** | The Implementation SHALL support the `response` query parameter with `collection` and `landingPage` as possible values. |

### 11.1.2. Collection Response

A request for a collection response is only valid for an execution request generating or selecting a single output which can be retrieved using an OGC API data access mechanism.

| REQUIREMENT 19 | |
|---|---|
| **LABEL** | /req/collection-output/collection-response |

| REQUIREMENT 19 | |
|---|---|
| STATEMENT | For initiating the execution of a workflow and retrieving its output as an OGC API collection: |
| A | For requests generating or selecting a single output, the Implementation SHALL return a 303 *See Other* HTTP status code together with a *Location:* response header redirecting to a collection description document as defined in *OGC API — Common — Part 2: Geospatial data* as a response to a request specifying a `response=collection` query parameter. |
| B | For requests generating multiple outputs which do not select a particular output, the Implementation SHALL return a 400 status error as a response to a request specifying a `response=collection` query parameter. |
| C | The collection description towards which the client is redirected SHALL link to at least one OGC API data access mechanism such as *OGC API — Tiles*, *DGGS*, *Coverages*, *Features*, *EDR*, *Maps*… |

### 11.1.3. Landing Page Response

A request for a landing page response is valid for an execution request generating one or more outputs which can be retrieved using an OGC API data access mechanism.

| REQUIREMENT 20 | |
|---|---|
| LABEL | /req/collection-output/landing-page-response |
| STATEMENT | For initiating the execution of a workflow and retrieving its output as an OGC API dataset landing page: |
| A | The Implementation SHALL return a 303 *See Other* HTTP status code together with a *Location:* response header redirecting to an OGC API dataset landing page as defined in *OGC API — Common — Part 1: Core* as a response to a request specifying a `response=landingPage` query parameter. |
| B | The landing page towards which the client is redirected SHALL contain at link to a list of available collections at `../collections` as defined in *OGC API — Processes — Part 2: Geospatial data*. |
| C | The collections linked from the landing page towards which the client is redirected SHALL link to at least one OGC API data access mechanism such as *OGC API — Tiles*, *DGGS*, *Coverages*, *Features*, *EDR*, *Maps*… |

### 11.1.4. Collection Output expiry

| PERMISSION 1 | |
|---|---|
| LABEL | /per/collection-output/expiry |

| PERMISSION 1 | |
|---|---|
| **STATEMENT** | Collection output resources are ephemeral in nature: |
| **A** | An Implementation MAY return a *410 Gone* HTTP status code for resources associated with a previous *Collection Output* response (i.e., a landing page, a collection, or any resources linked from either) if a significant period of time has elapsed since both the original request and any further associated data access request from the client. |
| **B** | An Implementation MAY redirect a client re-posting an identical or equivalent execution request for *Collection Output* (e.g., after having received a 410 gone status) to the same location as a previous execution (expired and reinstated, or still valid), or to a new location. |

### 11.1.5. Data Access Mechanisms

| REQUIREMENT 21 | |
|---|---|
| **LABEL** | /req/collection-output/data-access-mechanisms |
| **STATEMENT** | For triggering processing and returning resulting output data as a response to data access requests: |
| **A** | The Implementation SHALL support at least one OGC API data access mechanisms (e.g., *OGC API — Tiles*, *Coverages*, *DGGS*, *Features*, *EDR*, *Maps*) corresponding to the links returned from a a `response=collection` or `response=landingPage` request. |
| **B** | The Implementation SHALL trigger the processing required to respond to the request with considerations for the area, time and/or resolution of interest associated with the request. |

12

# COMMON WORKFLOW LANGUAGE DEFINITIONS CONFORMANCE CLASS

# 12 COMMON WORKFLOW LANGUAGE DEFINITIONS CONFORMANCE CLASS

## 12.1. Requirement Class *Common Workflow Language Definitions*

The *Common Workflow Language Definitions* conformance class specifies the use of the Common Workflow Language (CWL) Workflow Description as an alternative to an extended *OGC API — Processes — Part 1: Core* execution request to define an ad-hoc workflow (a workflow that is not first required to be deployed as a process before being executed).

The capability to deploy a CWL workflow as a process using *OGC API — Processes — Part 2: Deploy, Replace, Undeploy* is already covered by a conformance class of Part 2.

**TODO**: Verify that a CWL conformance class already exists or is planned for *Part 2*.

**TODO**: Verify and describe which (if any) of the other conformance classes defined in this extension applies to ad-hoc CWL workflows.

| REQUIREMENTS CLASS 9 | |
|---|---|
| **TARGET TYPE** | Common Workflow Language Definitions |
| **PREREQUISITES** | http://www.opengis.net/spec/ogcapi-processes-1/1.0/req/core<br>Common Workflow Language [1] |
| **LABEL** | http://www.opengis.net/spec/ogcapi-processes-3/0.0/req/cwl-workflows |

The following requirements describe the ability of an implementation to execute a CWL workflow in an ad-hoc manner and return its output, without requiring a prior deployment.

### 12.1.1. Ad-hoc Common Workflow Language Execution

| REQUIREMENT 22 | |
|---|---|
| **LABEL** | /req/cwl-workflows/adhoc-cwl-execution |

## REQUIREMENT 22

| | |
|---|---|
| **STATEMENT** | For executing a workflow defined using the Common Workflow Language (CWL) definition: |
| **A** | The Implementation SHALL support ad-hoc execution of a CWL workflow definition (`Content-Type: application/cwl+json`) POSTed to `/processes/CWL/execution`. |

13

# OPENEO PROCESS GRAPHS WORKFLOW CONFORMANCE CLASS

# 13 OPENEO PROCESS GRAPHS WORKFLOW CONFORMANCE CLASS

## 13.1. Requirement Class *OpenEO Process Graphs Workflow*

The *OpenEO Process Graphs Workflow* conformance class specifies the use of OpenEO Process Graphs to define workflows as an alternative to an extended *OGC API — Processes — Part 1: Core* execution request, to be executed in an ad-hoc manner (a workflow that is not first required to be deployed as a process before being executed).

The capability to deploy an OpenEO workflow definition as a process using *OGC API — Processes — Part 2: Deploy, Replace, Undeploy* is already covered by a conformance class of Part 2.

**TODO**: Verify that an OpenEO conformance class already exists or is planned for *Part 2*.

**TODO**: Verify and describe which (if any) of the other conformance classes defined in this extension applies to ad-hoc OpenEO workflows.

| REQUIREMENTS CLASS 10 | |
|---|---|
| **TARGET TYPE** | OpenEO Process Graphs Workflows |
| **PREREQUISITES** | http://www.opengis.net/spec/ogcapi-processes-1/1.0/req/core<br>OpenEO Process Graphs [2] |
| **LABEL** | http://www.opengis.net/spec/ogcapi-processes-3/0.0/req/openeo-workflows |

The following requirements describe the ability of an implementation to execute a CWL workflow in an ad-hoc manner and return its output, without requiring a prior deployment.

### 13.1.1. Ad-hoc OpenEO Workflow Execution

| REQUIREMENT 23 | |
|---|---|
| **LABEL** | /req/openeo-workflows/adhoc-openeo-execution |
| **STATEMENT** | For executing a workflow defined as an OpenEO Process Graph: |

## REQUIREMENT 23

| | |
|---|---|
| A | The Implementation SHALL support ad-hoc execution of a workflow defined using an OpenEO Process Graph POSTed to `/processes/OpenEO/execution` |

14

# MEDIA TYPES

# 14 MEDIA TYPES

The following follow media types are referenced by this specification.

- `application/json`: As with *OGC API — Processes — Part 1: Core*, used for the `Content-Type` of execution requests posted (`POST`) to `/processes/{processId}/execution`, as well as to deploy workflow definitions using *OGC API — Processes — Part 2: Deploy, Replace, Undeploy* by posting (`POST`) to `/processes`.

  - A new media type (`application/ogcmoaw+json` — Modular OGC API Workflow JSON, `application/ogcexec+json`, `application/ogcexecreq+json`?) could be registered with IANA to specifically reference *OGC API — Processes* execution requests.

- `application/ogcapppkg+json`: For deploying workflows for which an execution request is embedded in the executionUnit of an OGC Application Package as defined in the relevant class of *OGC API — Processes — Part 2: Deploy, Replace, Undeploy*.

- `application/cwl+json`: Common Workflow Language JSON workflow definition, for ad-hoc execution of CWL.

In addition, media types referenced by other OGC specifications normatively referenced by this specification are also relevant.

A

# ANNEX A (INFORMATIVE) CONFORMANCE CLASS ABSTRACT TEST SUITE (NORMATIVE)

A

# ANNEX A (INFORMATIVE) CONFORMANCE CLASS ABSTRACT TEST SUITE (NORMATIVE)

**NOTE:** Ensure that there is a conformance class for each requirements class and a test for each requirement (identified by requirement name and number)

## A.1. Conformance Class A

### A.1.1. Requirement 1

| REQUIREMENT A.1 | |
|---|---|
| TEST PURPOSE | Verify that… |
| TEST METHOD | Inspect… |

### A.1.2. Requirement 2

B

# ANNEX B (INFORMATIVE) COASTAL EROSION SUSCEPTIBILITY EXAMPLE WORKFLOW

# B ANNEX B (INFORMATIVE) COASTAL EROSION SUSCEPTIBILITY EXAMPLE WORKFLOW

This annex presents a sample workflow predicting the susceptibility of coastal erosion from four data sources. This workflow (purely intended as a demonstrative example, not to be scientificaly accurate in any way), was assembled in collaboration with Perry Peterson and Marta Padilla Ruiz from the University of Calgary in the context of the OGC Federated Marine SDI Phase 3 Innovation Program initiative.

It demonstrates the use of the *Nested Processes*, *Collection Input*, *Input fields modifiers* and *Output fields modifiers* conformance classes.

The expressions used to derive field values (`"properties"`) are written using the CQL2 language extended with a ternary conditional operator as found in the C programming language and its derivatives, as well as with the ability to return not only boolean values, but also numeric values.

Note that in the case of remote processes, the derived fields (`"properties"`) specified for the *Slope* and Aspect processes could be using either the *Input* or *Output field modifiers* conformance class, depending on whether the *Slope* and *Aspect* processes generating the data, or the *PassThrough* process receiving the data, support the capability.

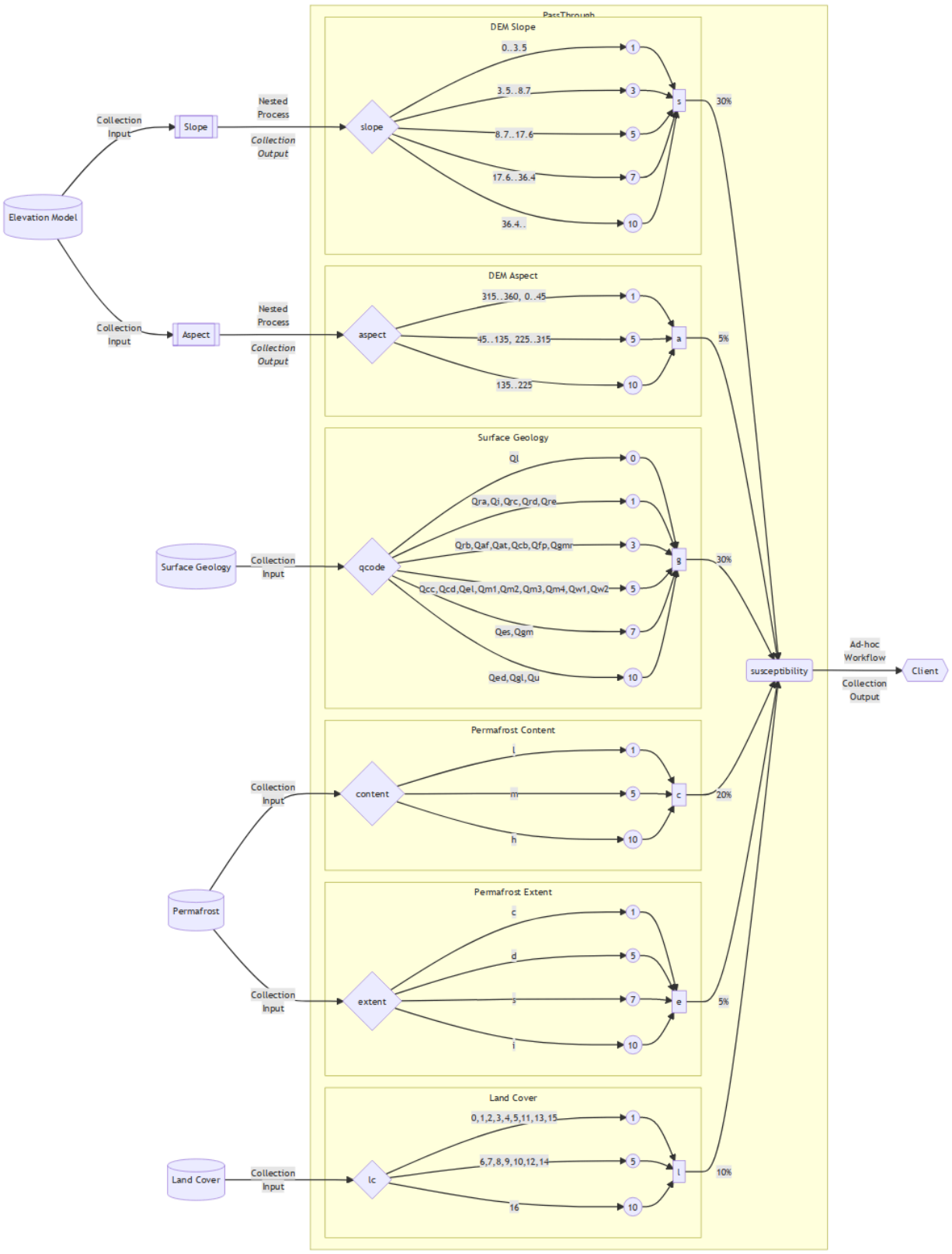


**Figure B.1** — Coastal Erosion Example Workflow Diagram

```
{
   "process": "https://example.com/ogcapi/processes/PassThrough",
   "inputs": {
      "data": [
         {
            "process": "https://example.com/ogcapi/processes/Slope",
            "inputs": {
               "dem": { "collection": "https://example.com/ogcapi/collections/
DEM" }
            },
            "properties": { "s" : "slope >= 36.4 ? 10 : slope >= 17.6 : 7 :
slope >= 8.7 ? 5 : slope >= 3.5 ? 3 : 1" }
         },
         {
            "process": "https://example.com/ogcapi/processes/Aspect",
            "inputs": {
               "dem": { "collection": "https://example.com/ogcapi/collections/
DEM" }
            },
            "properties": { "a" : "aspect >= 315 or aspect < 45 ? 1 : aspect >=
 225 or aspect < 135 : 5 : 10" }
         },
         {
            "collection": "https://example.com/ogcapi/collections/
ArcticPermafrost",
            "properties": {
               "e" : "extent = 'c' ? 1 : extent = 'd' ? 5 : extent = 's' ? 7 :
extent = 'i' ? 10 : null"
               "c" : "content = 'l' ? 1 : content = 'm' ? 5 : content = 'h' ?
10 : 0"
            }
         },
         {
            "collection": "https://example.com/ogcapi/collections/
Landsat7LandCover",
            "properties": { "l" : "lc in(0,1,2,3,4,5,11,13,15) ? 1 : lc
in(6,7,8,9,10,12,14) ? 5 : lc = 16 ? 10 : 0" }
         },
         {
            "collection": "https://example.com/ogcapi/collections/
AlaskaSurficialGeology",
            "properties": {
               "g" :
               "qcode = 'Ql' ? 0 : qcode in ('Qra','Qi','Qrc', 'Qrd', 'Qre')
? 1 : qcode in ('Qrb','Qaf', 'Qat', 'Qcb','Qfp','Qgmr') ? 3 : qcode in ('Qcc',
'Qcd','Qel','Qm1', 'Qm2','Qm3','Qm4','Qw1','Qw2') ? 5 : qcode in ('Qes','Qgm')
? 7 : qcode in ('Qed','Qgl','Qu') ? 10 : 0"
            }
         }
      ]
   },
   "properties": { "susceptibility" : "0.30 * s + 0.05 * a + 0.05 * e + 0.20 *
c + 0.10 * l + 0.30 * g" }
}
```

**Figure B.2 — Coastal Erosion Example Workflow Exectution Request**

C

# ANNEX C (INFORMATIVE) REVISION HISTORY

C

# ANNEX C (INFORMATIVE) REVISION HISTORY

**Table C.1** — Revision history

| DATE | RELEASE | EDITOR | PRIMARY CLAUSES MODIFIED | DESCRIPTION |
|---|---|---|---|---|
| 2020-10-19 | 0.1 | J. St-Louis | all | initial draft |
| 2022-10-19 | 0.2 | J. St-Louis | all | reorganized into multiple conformance classes, using metanorma template, with more detailed descriptions and requirements |

# BIBLIOGRAPHY

# BIBLIOGRAPHY


[1] Peter Amstutz, Michael R. Crusoe, Nebojša Tijanić (editors), Brad Chapman, John Chilton, Michael Heuer, Andrey Kartashov, Dan Leehr, Hervé Ménager, Maya Nedeljkovich, Matt Scales, Stian Soiland-Reyes, Luka Stojanovic (2020): Common Workflow Language, v1.2. Specification, Common Workflow Language working group. https://w3id.org/cwl/

[2] OpenEO: OpenEO Developers API Reference / Process Graphs. https://openeo.org/documentation/1.0/developers/api/reference.html#section/Processes/Process-Graphs