OGC® DOCUMENT: 21-044

External identifier of this OGC® document: http://www.opengis.net/doc/PER/t17-D045

Open Geospatial Consortium

OGC TESTBED 17: CITE ENGINEERING REPORT

ENGINEERING REPORT

PUBLISHED

Submission Date: 2021-11-19
Approval Date: 2021-12-09
Publication Date: 2022-04-08
Editor: Luis Bermudez

License Agreement

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

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

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

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

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

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

Copyright notice

Note

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

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

CONTENTS

LIST OF TABLES

LIST OF FIGURES

I ABSTRACT

This OGC Testbed 17 Engineering Report (ER) documents the result of the work performed in the CITE thread of the OGC Testbed-17 initiative. CITE is the Compliance Interoperability & Testing Evaluation Subcommittee that provides a forum for an open, consensus discussion regarding approaches and issues related to conformance and interoperability testing as part of the OGC standards process. This ER provides information about the development of a test suite for the *OGC API — Processes* Standard (OGC18-062r2) to be executed in the OGC Test Evaluation tool (TEAM Engine). The ER also documents an evaluation of an alternative environment for OGC compliance testing.

II EXECUTIVE SUMMARY

This Engineering Report (ER) captures the result of the work performed in the CITE thread as part of the OGC Testbed-17 initiative. The document provides information about the development of a test suite for the OGC API – Processes standard to be executed in the OGC Validator tool (which is implemented using open source TEAM Engine software product) and evaluation of an alternative environment for OGC testing. The work is done under the umbrella of the Compliance Interoperability & Testing Evaluation (CITE) Subcommittee (SC) that provides a forum for an open, consensus discussion regarding approaches and issues related to conformance and interoperability testing as part of the OGC Standardization process.

The content of this ER will inform OGC Standards Working Groups (SWGs) about how to structure and write compliance tests for the most recent OGC API Standards. The ER also documents the pros and cons of using a possible alternative testing environment. It will facilitate enhancement of interoperability by providing the infrastructure to test and improve more implementations that are seeking compliance certification towards OGC standards.

As at 2021, several OGC SWGs are developing API standards that enable easier interaction of modern clients with servers. The OGC API — Processes standard is designed to enable a client to explore and run processes available over the web. The work done in Testbed 17 provides both the TestNG scripts and an alternative environment to write the tests to check if implementations are compliant with the OGC API — Processes.

The OGC Validator is currently implemented using a Java-based application called TEAM Engine to perform testing of software that claim compliance with one or more OGC standards. TEAM Engine executes one or more tests written in either the Compliance Test Language (CTL) or implemented using the TestNG Framework. The Testbed-17 participants explored the challenges of using this testing framework for the most recent OGC APIs.

The work in this thread responded to the following research questions:

- What does a TEAM Engine test for *OGC API — Processes* looks like?

- What alternative test environment(s) should be used in the future and why?
- How do tests look like for this new test environment?
- Is it possible to automatically generate tests from the latest generation of OGC specifications? If it is possible, then what level of automatization is possible? Does a high level of automatization require a change to the format that the OGC standards are currently encoded?

An overview of findings and recommendations is as follows:

- Creating compliance tests for the new OGC API Standards is possible.
- Two environments were tested, following TestNG and a new environment based on the ETF test framework. The ETF framework is used in the European Union's INSPIRE Validator and uses NeoTL. Note that the INSPIRE Validator also uses OGC's TEAM Engine instance for some tests. The work in the CITE thread of Testbed-17 demonstrated that OGC compliance testing can be performed using this environment. However, more work is necessary to make this approach part of the OGC Testing tools, including improving performance.
- Java stubs can be generated automatically from a Standards Abstract Test Suite (ATS) in ASCII doc. This speeds up the process of developing the new tests.
- For tests that require inspection of the results (e.g. a process), providing a scenario where the request and responses are known is important.
- The current TEAM Engine needs further enhancements such as supporting content type "application/problem+json" and providing better feedback to the tester (e.g. header content being sent).

III KEYWORDS

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

ogcdoc, OGC document, CITE, API, compliance

IV PREFACE

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.

V SECURITY CONSIDERATIONS

No security considerations have been made for this document.

VI SUBMITTING ORGANIZATIONS

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

- GeoSolutions

VII SUBMITTERS

All questions regarding this document should be directed to the editor or the contributors:

NAME	ORGANIZATION	ROLE
Luis Bermudez	GeoSolutions	Editor
Peter Vretanos	CubeWerx	Contributor
Benjamin Pross	52°North	Contributor
Gérald Feony	Geolabs	Contributor
Jon, Herrmann	interactive instruments Gmb H	Contributor

1

NORMATIVE REFERENCES

1 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.

OGC 18-062r2, OGC API — Processes — Part 1: Core, Open Geospatial Consortium (2021)

2

TERMS AND DEFINITIONS

2 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.

2.1. **Abstract Test Suite (ATS)**

A set of testable assertions about the functionality of a standard, which an implementation must support in order to achieve compliance to the standard. ATS are based on the conformance clauses defined in the standard.

2.2. **Compliance**

A state of a specific software product, which implements an OGC Standard and has passed the Compliance Testing Evaluation.

2.3. **Executable Test Suite (ETS)**

A set of code (e.g. Java and CTL) that provides runtime tests for the assertions defined by the ATS. Test data required to do the tests are part of the ETS.

2.4. **Process**

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

$p: X \rightarrow Y$

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

The term process is one of the most used terms both in the information and geosciences domain. If not stated otherwise, this specification uses the term process as an umbrella term for any algorithm, calculation or model that either generates new data or transforms some input data into output data as defined in section 4.1 of the WPS 2.0 standard.

3

INTRODUCTION

3 INTRODUCTION

The Compliance Program provides the resources, procedures, and policies to certify products for compliance with one or more OGC standards. Amongst the resources provided by the program are executable test suites that enable developers to test whether their products implement OGC Standards correctly. This Engineering Report (ER) provides the following major sections:

- Considerations for implementing an Executable Test Suite for *OGC API — Processes*
- New test environment and comparison with the current TEAM Engine
- Description of an ETS for *OGC API — Processes*
- Future work
- Conclusions

4

EXECUTABLE TEST SUITE

4 EXECUTABLE TEST SUITE

This section provides information about the development of the Executable Test Suite (ETS) for OGC API — Processes. The executable test suite was developed as a module for deployment into TEAM Engine software. The following standards development organizations are known to offer compliance testing using TEAM Engine:

- Open Geospatial Consortium (OGC)
- US Geospatial Intelligence Standards Working Group (GWG), with responsibility for standards of the US National System for Geospatial Intelligence (NSG)
- Defence Geospatial Information Working Group (DGIWG)

The European Union's INSPIRE Validator is also known to use the OGC's TEAM Engine instance for some INSPIRE validation tests.

4.1. TestNG Test

4.1.1. Overview

The executable test for OGC API — Processes 1.0 is available at the OGC GitHub repository:

https://github.com/opengeospatial/ets-ogcapi-processes10

The test has been developed in TestNG[https://testng.org/doc/], the current test framework used by the OGC Compliance Program.

The following standards have been used:

- OGC API — Processes — Part 1: Core (OGC 18-062r2)

Each conformance class is represented as a TestNG Java class within its own package. The package structure is shown in the following:

```
|
|_ org.opengis.cite.ogcapiprocesses10
|_ org.opengis.cite.ogcapiprocesses10.conformance
|_ org.opengis.cite.ogcapiprocesses10.general
|_ org.opengis.cite.ogcapiprocesses10.jobs
|_ org.opengis.cite.ogcapiprocesses10.landingpage
|_ org.opengis.cite.ogcapiprocesses10.openapi3
|_ org.opengis.cite.ogcapiprocesses10.process
|_ org.opengis.cite.ogcapiprocesses10.processlist
```

```
|_ org.opengis.cite.ogcapiprocesses10.util
```

Figure 1

The following conformance classes were implemented:

- Landing Page `/`
- API Definition `/api`
- Conformance Path `/conformance`
- HTTP 1.1
- Processes `/processes`
- Jobs `/jobs`

The following conformance classes were not implemented:

- Joblist
- Dismiss
- Callback

The method stubs and code comments (JavaDoc) for the Java classes were created out of the abstract test suite using an automated script.

A demonstration instance of the test suite is available here:

https://17.testbed.dev.52north.org/teamengine/

For the validation of the JSON requests/responses the OpenAPI4J library was used.

The following image shows the start screen of the user interface. The user can specify the endpoint of the landing page of the *OGC API – Processes* implementation, as well as the identifier of a testable process.

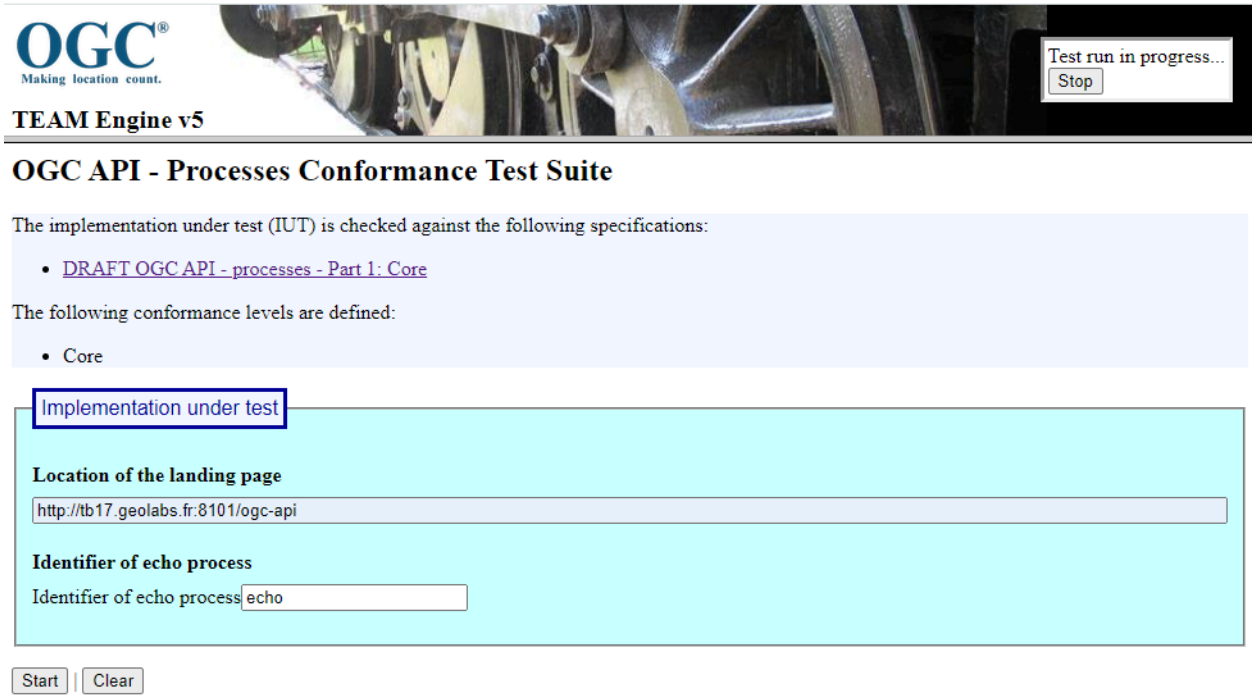

Figure 2 — Start screen of the TEAM Engine tests for the OGC API - Processes - Part 1 standard

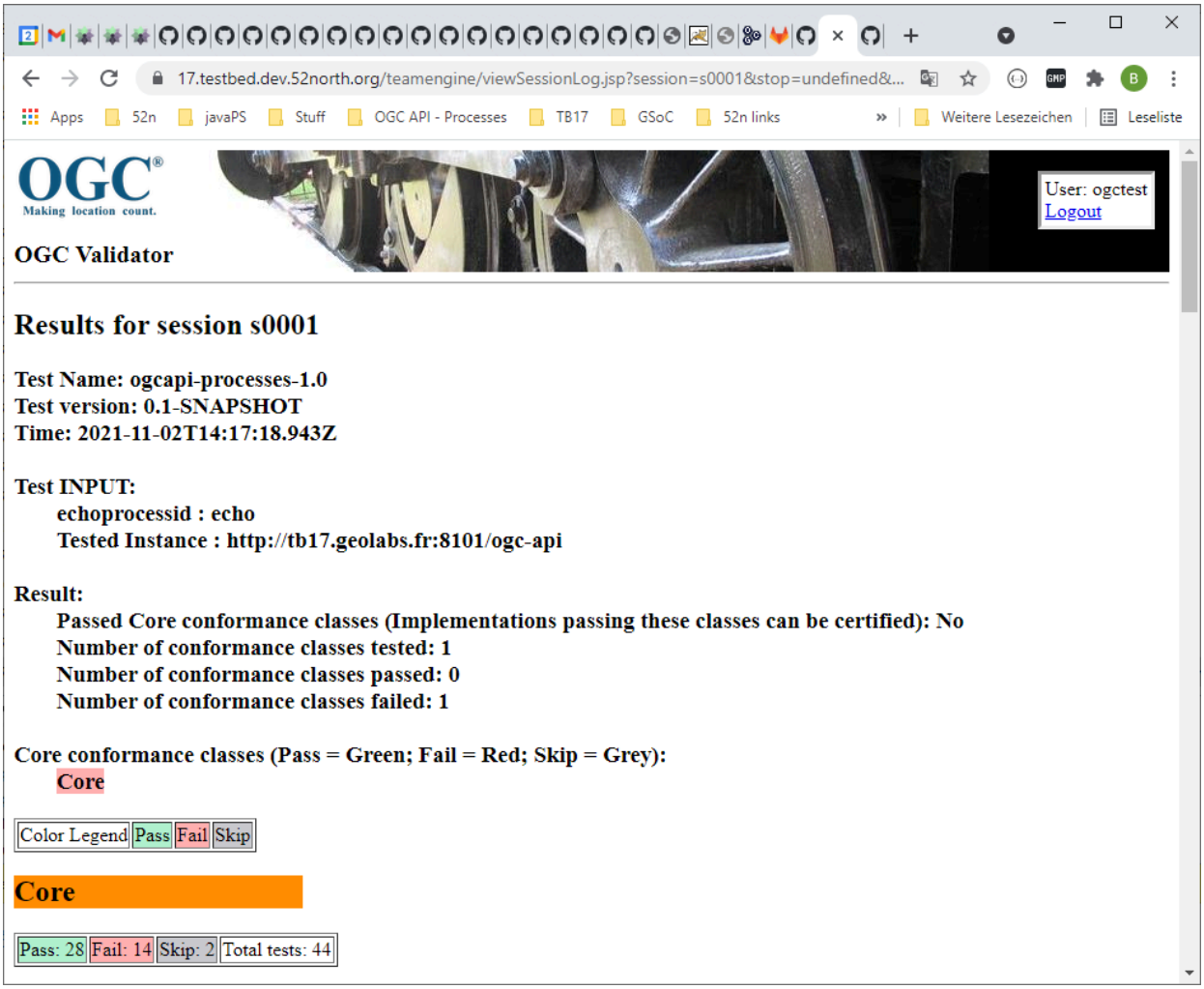

Figure 3 — Result screen of the TEAM Engine tests for the OGC API - Processes - Part 1 standard

During the testbed, 30 tests of the *OGC API — Processes* standard were implemented. An unresolved issue with the validation of the schema for results prevents the completion of a number of tests. The following table shows the test results using the *OGC API — Processes* instance running at: http://tb17.geolabs.fr:8101/ogc-api using the process published by the server that has the identifier “echo”:

Table 1

TEST	RESULT	REASON
test Job Results Sync	Skipped	Did not find Link with value rel=monitor, skipping test.
test Job Results Async Document	Passed	- =
test Job Creation Input Inline Binary	Failed	Not implemented yet.

TEST	RESULT	REASON
test Job Creation Sync Raw Mixed Multi	Failed	Not implemented yet.
test Job Results No Such Job	Passed	-
test Job Results Exception Results Not Ready	Failed	Not implemented yet.
validate Conformance Operation And Response	Passed	-
test Job Creation Input Inline	Passed	-
test Job Creation Sync Raw Value One	Failed	Got unexpected status code: 500
test Job Exception No Such Job	Passed	-
test Job Creation Inputs	Passed	-
test Job Creation Input Ref	Skipped	No input with href detected.
test Job Creation Request	Passed	-
test Process Success	Passed	-
test Job Op	Passed	-
test Process Exception No Such Process	Passed	-
test Job Results	Failed	body: Type expected 'string', found 'object'. (code: 1027) (...)
test Job Results Failed	Failed	body: Field 'type' is required. (code: 1026) From: body. <required>
test Job Success	Passed	-
test Process List Success	Passed	-
landing Page Validation	Passed	-
test Process	Passed	-
test Job Creation Success Async	Passed	-

TEST	RESULT	REASON
test Job Creation Input Array	Failed	Not implemented yet.
test Pl Links	Passed	-
test Job Creation Input Inline Mixed	Failed	Not implemented yet.
test Job Creation Auto Execution Mode	Passed	-
test Job Creation Input Validation	Failed	expected [400] but found [200]
test Job Creation Default Outputs	Passed	-
test Pl Limit Response	Passed	-
test Job Creation Sync Document	Passed	-
test Job Results Async Raw Ref	Passed	-
test Job Creation Input Inline Bbox	Passed	-
test Job Creation Input Inline Object	Passed	-
test Process List	Passed	-
test Job Results Async Raw Value One	Failed	Java exception
test Job Results Async Raw Mixed Multi	Failed	Not implemented yet.
test Job Creation Sync Raw Value Multi	Failed	Not implemented yet.
test Job Results Async Raw Value Multi	Failed	Not implemented yet.
test Pl Limit Definition	Passed	-
landing Page Retrieval	Passed	-
test Job Creation Sync Raw Ref	Failed	Not implemented yet.
test Job Creation Default Execution Mode	Passed	-
test Job Creation Op	Passed	-

4.1.2. Recommendations

- The TestNG framework works well with the executable test suite for the OGC API — Processes — Part 1: Core. Except for the validation issue for the result schema no critical issues were detected. The OpenAPI4J library reliably validates the JSON requests/responses.
- OGC API — Processes — Part 1: Core consists of several conformance classes with currently 44 tests. The tests are listed directly beneath each other. This way, it can be hard to get an overview of the passing/failing tests. Thus, a possibility for better structuring of the tests is recommended.
- The creation of new tests requires to execute the test repeatedly. Currently, single tests cannot be run, only the complete test suite, which can take a considerable amount of time. It is therefore recommended to investigate how single tests can be run.
- A large number of execute requests are sent to the implementation under test to evaluate various combinations of parameters. It is recommended to investigate whether responses to execute requests can be reused to cover different test cases.
- A number of the tests that fail seem to be a cascaded effect of the fact that the TestNG-based ETS does not recognize the specification relation types. For example, the test "landing Page Validation" fails because the ETS is looking for a link with relation "processes" but the correct link relation according to the specification is "http://www.opengis.net/def/rel/ogc/1.0/processes". The ETS should be modified to address this. The problem has been reported in the GitHub Issues log.
- The TestNG-based ETS does not seem to recognize the content type `application/problem+json` as a valid content type for an exception response. The *OGC API — Processes* specification, however, cites RFC 7807 where it specifies this as the correct MIME type for error report. A GitHub issue has been recorded for implementing support for the MIME type specified in RFC 7807.
- Improve feedback to the users. Good feedback enables a pseudo-interactive engagement with the OGC Validator supporting incremental refinement of the server being tested. The feedback should include the headers passed to the server, the URL that was accessed, and the content of the body sent to the server under test, if applicable. It is recommended to include this level of feedback on the TestNG-based ETS.

5

ALTERNATIVE TEST ENVIRONMENT

5 ALTERNATIVE TEST ENVIRONMENT

An alternative test environment based on the ETF test framework was used to validate implementations towards OGC standards, in particular focusing on the new OGC API standards. The standard selected was *OGC API – Process*.

5.1. ETF test framework

The ETF test framework is an open-source application framework that can be used by the tester to execute tests through a web interface. The ETF uses a modular software-architecture and is designed to deliver user-friendly, self-explanatory test reports.

It is successfully used as a basis for the European INSPIRE Validator and in several projects for German mapping agencies.

5.2. NeoTL DSL

For web service testing, ETF has leveraged SoapUI[https://www.soapui.org/], but for various reasons a new solution was evaluated in 2020. A new ETF test driver for geoservice and Web API testing was implemented in a prototype based on a new domain specific language (DSL). The DSL was named NeoTL.

The goal was to simplify the definition of tests with the DSL as much as possible and thus make it maintainable and accessible to subject matter experts without knowledge of a specific programming language. Language concepts were adapted for Testbed 17 and the special requirements for creating OGC API Processes tests.

To lower the entry barrier cloud-based tool support was leveraged. The test cases were developed in the browser, a local installation was not required.

Another goal was to simplify the communication between the various experts; the SWG members, test authors and implementers by using a DSL.

5.3. Comparison with CTL

Prior to 2014, executable test suites in the OGC Validator used the OGC Compliance Test Language (CTL). Due to limitations of CTL, since 2014 all new executable test suites developed for the OGC Validator are built using the TestNG framework instead of CTL. This section presents a comparison between CTL and NeoTL, however, it should be noted that CTL is a

legacy technology that is no longer used to develop new executable test suites in the OGC Compliance Program.

Domain Specific Languages are distinguished between **internal** and **external** DSLs.

Examples of internal DSL include domain-specific UML profiles, domain-specific XML Schema, the Gradle DSL of the Gradle build tool and also the CTL.

Internal DSL essentially use the language concepts of their host language. With Gradle these language concepts are based on Groovy[https://groovy-lang.org/], with CTL the concepts are based on XML elements and various XML schema. This means that the grammar is restricted to the syntax of the host language and it adopts all the nice but also the less pleasant syntactic features, as the following CTL test illustrates:

```
<test name="wfs:wfs-1.1.0-Basic-GetCapabilities-tc1">
    <param name="wfs.GetCapabilities.get.url"/>
    <assertion>The GET method request must be supported (using HTTP GET).
</assertion>
    <comment>GetCapabilities by GET. Pass if all of the following conditions
are true:
        (1) the response is schema valid;
        (2) the root document is an wfs:WFS_Capabilities document.
    </comment>
    <link title="wfs-1.1.0-Basic-GetCapabilities-tc1">http://cite.
opengeospatial.org/te2/about/wfs/1.1.0/site/ats-wfs11-basic-cc/GetCapabilities/
GET/BasicGetCapabilities-GET-tc1.html</link>
    <link>OGC 04-094, 13.1, p.79</link>
    <code>

        <xsl:variable name="request1">
            <request>
                <url>
                    <xsl:value-of select="$wfs.GetCapabilities.get.url"/>
                </url>
                <method>get</method>
                <param name="service">WFS</param>
                <param name="version">1.1.0</param>
                <param name="request">GetCapabilities</param>
                <p:XMLValidatingParser.GMLSF1/>
            </request>
        </xsl:variable>

        <xsl:choose>
            <xsl:when test="not($request1/*)">
                <ctl:message>FAILURE: Missing or invalid response entity.</ctl:
message>
                <ctl:fail/>
            </xsl:when>
            <xsl:otherwise>
                <xsl:variable name="expression">//wfs:WFS_Capabilities</xsl:
variable>
                <ctl:call-test name="ctl:assert-xpath">
                    <ctl:with-param name="expr" select="$expression"/>
                    <ctl:with-param name="doc" select="$request1"/>
                </ctl:call-test>
            </xsl:otherwise>
        </xsl:choose>

    </code>
```

```
</test>
```

Figure 4 — CTL test case example

An external DSL is a language that's parsed independently of the host general purpose language. For example, regular expressions and CSS. The concrete syntax and the semantics are freely defined. This means that external DSLs can be more flexible and expressive. The CTL example would look something like this in NeoTL (the Request definition is externalized and referenced):

```
TestCase "The GET method request must be supported (using HTTP GET)" {
    id: wfs.1.1.0.Basic-GetCapabilities-tc1
    description: "GetCapabilities by GET. Pass if all of the following
        conditions are true:
            (1) the response is schema valid;
            (2) the root document is an wfs:WFS_Capabilities document."

    references:
        - "Abstract Test Case wfs-1.1.0-Basic-GetCapabilities-atc3, p.79"
            "https://portal.ogc.org/files/?artifact_id=8339"
            AbstractTestCase

    ValidationStep "GetCapabilities with GET method" {
        id: step
        description: "GetCapabilities with GET method and validate response
against
            GMLSF1 schema"

        given:
            - Service is "WFS 1.1.0"

        when: Request requests.wfs1.GetCapabilities executed

        then:
            - Assert XPath {
              /wfs:WFS_Capabilities/* exists
            }
            - Assert XmlSchema {
              schema "http://schemas.opengis.net/wfs/1.1.3/wfs.xsd"
              validates
            }
    }
}

GetRequest "GetCapabilities" {
    id: requests.wfs1.GetCapabilities

    query:
        - "service" = "WFS"
        - "version" = "1.1.0"
        - "request" = "GetCapabilities"
}
```

Figure 5 — NeoTL test case example

Furthermore, it is possible to tailor the IDE closely to the language. Thus, it is possible to perform semantic checks in addition to syntactic checks, syntax highlighting and to simplify the implementation of tests for the test developer with several other IDE services. Figure 6

shows the editor with some IDE services like the **Outline** and the **Problem** views as well as the syntactically highlighted test definition for an OGC API — Processes test case.

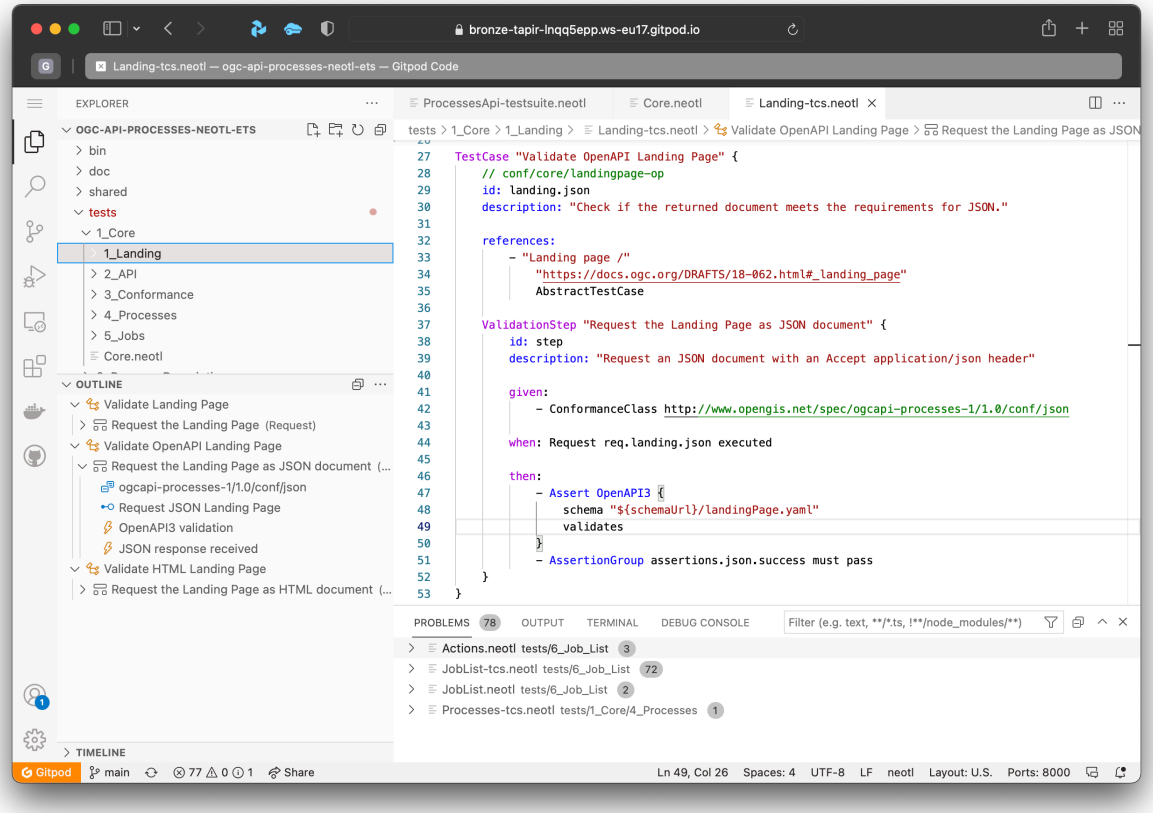

Figure 6 — NeoTL IDE example

All currently implemented IDE services are described in the language workbench documentation.

5.4. Structure of a NeoTL Test Case

The individual language concepts are described in detail in the documentation and additionally also displayed directly in the IDE as help. The section briefly describes how a NeoTL Test Case is structured and what concepts it is based on.

The basis for the structuring is the ISO 19105:2021 — Geographic information — Conformance and testing , and thus executable tests are composed of the following concepts:

- test suites
- test modules
- and test cases.

```
TestSuite "OGC API - Processes" {
    id: org.opengis.ets.ogcapi.processes
    version: 0.9.2-snapshot
    description: "Executable Test Suite for validating Web APIs that implement
            the 'OGC API - Processes - Part 1: Core' standard. The Test Suites
            are based on the normative Abstract Test Suites from Annex A of the
            OGC Implementation Specification draft 1.0-draft.7-SNAPSHOT."

   references:
        - "OGC Implementation Specification"
            "https://docs.ogc.org/DRAFTS/18-062.html"
            ImplementationSpecification

    executes:
        - oapi.processes.core
        - oapi.processes.joblist
        // ...

    defines:
        - URL $schemaUrl = "https://raw.githubusercontent.com/opengeospatial/
ogcapi-processes/master/core/openapi/schemas"
}
```

Figure 7 — Test Suite

```
TestModule "Job List" {
    id: oapi.processes.joblist

    description: "The Job list requirements class specifies how to retrieve a
job list from the API"

    references:
        - "Core ATS"
            "http://docs.ogc.org/DRAFTS/18-062.html#_conformance_class_core"
            AbstractTestSuite
        - "Conformance Class Job list"
            http://www.opengis.net/spec/ogcapi-processes-1/1.0/conf/job-list
            ConformanceClass

    executes:
        - joblist.list
        - joblist.list.links
        // ...
}
```

Figure 8 — Test Module

```
TestCase "Validate links section in job list" {
    id: joblist.list.links
    description: "Validate that the proper links are included in a response."

    references:
        - "Conformance Class Job list"
            "https://docs.ogc.org/DRAFTS/18-062.html#_conformance_class_job_
list"
            AbstractTestCase
```

```
    ValidationStep "Validate self link" {
        // ...
    }

    ValidationStep "Validate HTML link" {
        // ...
    }
}
```

Figure 9 — Test Case

The syntax for enumerations (see executes keyword in Figure 7) is influenced by YAML, but unlike YAML, the number of spaces before indentation is ignored.

A test case can contain one or multiple validation steps. Validation steps define the interaction with the implementation and the expected behavior. Their structure is based on the well-known Given-Then-When scheme used by many test frameworks.

```
ValidationStep "Request the Landing Page as JSON document" {
    id: step
    description: "Request an JSON document with an Accept application/json
header"

    given:
        - ConformanceClass http://www.opengis.net/spec/ogcapi-processes-1/1.0/
conf/json

    when: Request requests.landing.json executed

    then:
        - Assert OpenAPI3 {
            schema "${schemaUrl}/landingPage.yaml"
            validates
        }
        - Assert HTTP { statusCode "200" }
        - Assert HTTP { contentType "application/json" }
}
```

Figure 10 — Validation Step

Related literature [1] suggests that the three main sections of a Validation Step are:

- the **given** section which describes the precondition for the whole Validation Step
- the **when** section is the action that is executed to interact with the implementation
- the **then** section verifies the results of the previous action with assertions

5.4.1. Given

If all preconditions in the **given** section are met, then the action defined in the **when** section is executed. The result of the action must satisfy all assertions defined in the **then** section to result in a passed validation. If at least one precondition is not fulfilled, the result of the validation step is the status **skipped**, if a conformance class is prerequisite for the test case, which is not

implemented by the tested implementation, the status is **not_applicable**. Figure 11 shows how the status skipped is displayed in an ETF test report when a test case with required information has failed.

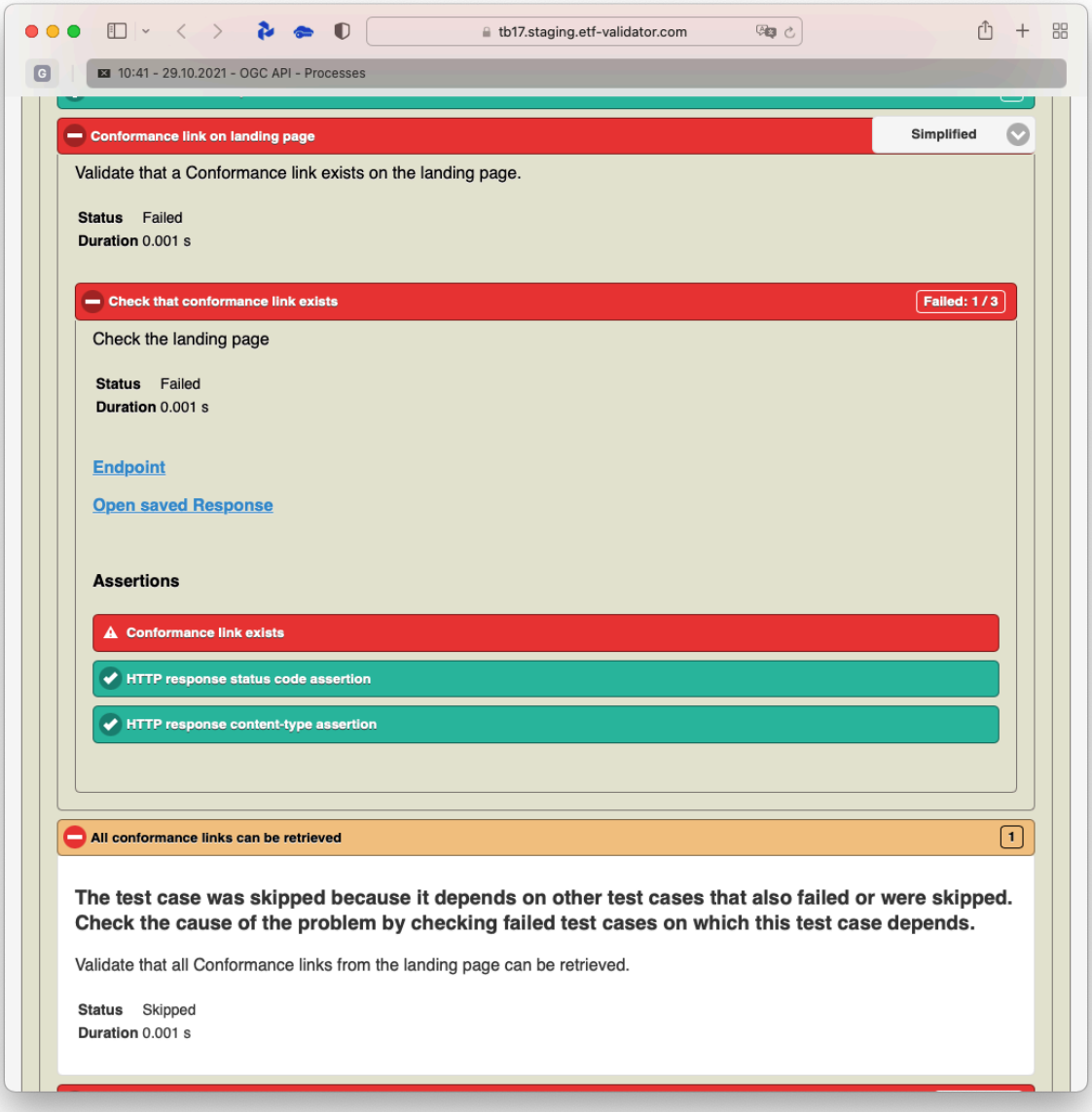

Figure 11 — ETF skipped status

5.4.2. When

In the When section, it can be specified which requests are sent to the implementation, whether information is extracted from a response, or whether further tests are performed at runtime based on a response. Figure 12 shows an example for a Post Request with a JSON payload.

```
PostRequest "Execute Echo Process" {
```

```
    id: requests.echo

    path: "processes/${echoId}/execution"

    headers:
        - "Accept" = "application/json"
        - "Prefer" = ${preferHeader}

    body {
        "inputs": ${inputs},
        "outputs": ${outputs},
        "response": "document"
    } as application/json
}
```

Figure 12 — Post request

Variables can be specified in the form **${variableName}** and will be replaced at execution time.

5.4.3. Then

Querying JSON properties in the **then** section is done using the JSONPath syntax. The results of the queries are checked with certain functions, which in turn are expressed in the DSL:

```
- Assert JSON {
    $.jobs[?(@.type == 'process')] exists
    or $.jobs empty
    otherwise FAIL with "No job entry exists with the
        requested process type 'process'."
}
```

Figure 13 — JSON assertion example exists or empty

```
- Assert JSON {
    $.jobs count == 1
    otherwise FAIL with "Expected exactly one entry in the Job List"
}
```

Figure 14 — JSON assertion example count

```
- Assert JSON {
    $.conformsTo contains some
        "http://www.opengis.net/spec/ogcapi-processes-1/1.0/conf/core"
    otherwise FAIL with "Expected an 'conformsTo' array containing
        the value
        'http://www.opengis.net/spec/ogcapi-processes-1/1.0/conf/core'."
}
```

Figure 15 — JSON assertion example contains

5.5. Dynamic tests in NeoTL

In addition to the declaration of conformance classes as precondition, the results of other validation steps can also be specified as preconditions. During the interaction with the implementation, further tests can be created based on the results. The concept of generators is used for this purpose, shown in Figure 16.

```
TestCase "Process descriptions" {
    id: process.description
    description: "Validate that a process description can be retrieved from
the expected location."

    ValidationStep "Generate Requests" {
            id: process.desc.generator
            description: "Generate Process Description Requests"

            given:
                - ConformanceClass http://www.opengis.net/spec/ogcapi-
processes-1/1.0/conf/ogc-process-description
                - Response from process.list.encoding.json
            when: Generator generators.process.ids executed
            then:
                - Assert JSON {
                    ${processIds} not empty
                    otherwise FAIL with "No IDs in process list found"
                }
    }

    ValidationStep "Validate Process Description" {
        id: process.desc
        description: "Validated responses of the generated process
descriptions requests"

        given:
            - One ${processId} of ${processIds} from process.desc.generator
        when: Request requests.processes.description executed
        then:
            - Assert HTTP { statusCode "200" }
            - Assert HTTP { contentType "application/json" }
            - Assert OpenAPI3 {
                schema
                "${schemaUrl}/process.yaml"
                validates
            }
            - Assert JSON {
                $.id equals ${processId}
                otherwise FAIL with
                "The id in the returned response does not match the requested
id"
            }
    }
}

Generator "Generate Process IDs" {
    id: generators.process.ids
```

```
    from RESPONSE
        as ${processIds} query $..id
}

GetRequest "Get Process Description" {
    id: requests.processes.description

    path: "processes/${processId}"

    headers:
        - "Accept" = "application/json"
}
```

Figure 16 — Generators

The generator is specified within the **when** section. With the generator, a set of values is extracted and a validation step is created and executed for each value in the set. The result in the report can be seen in Figure 17.

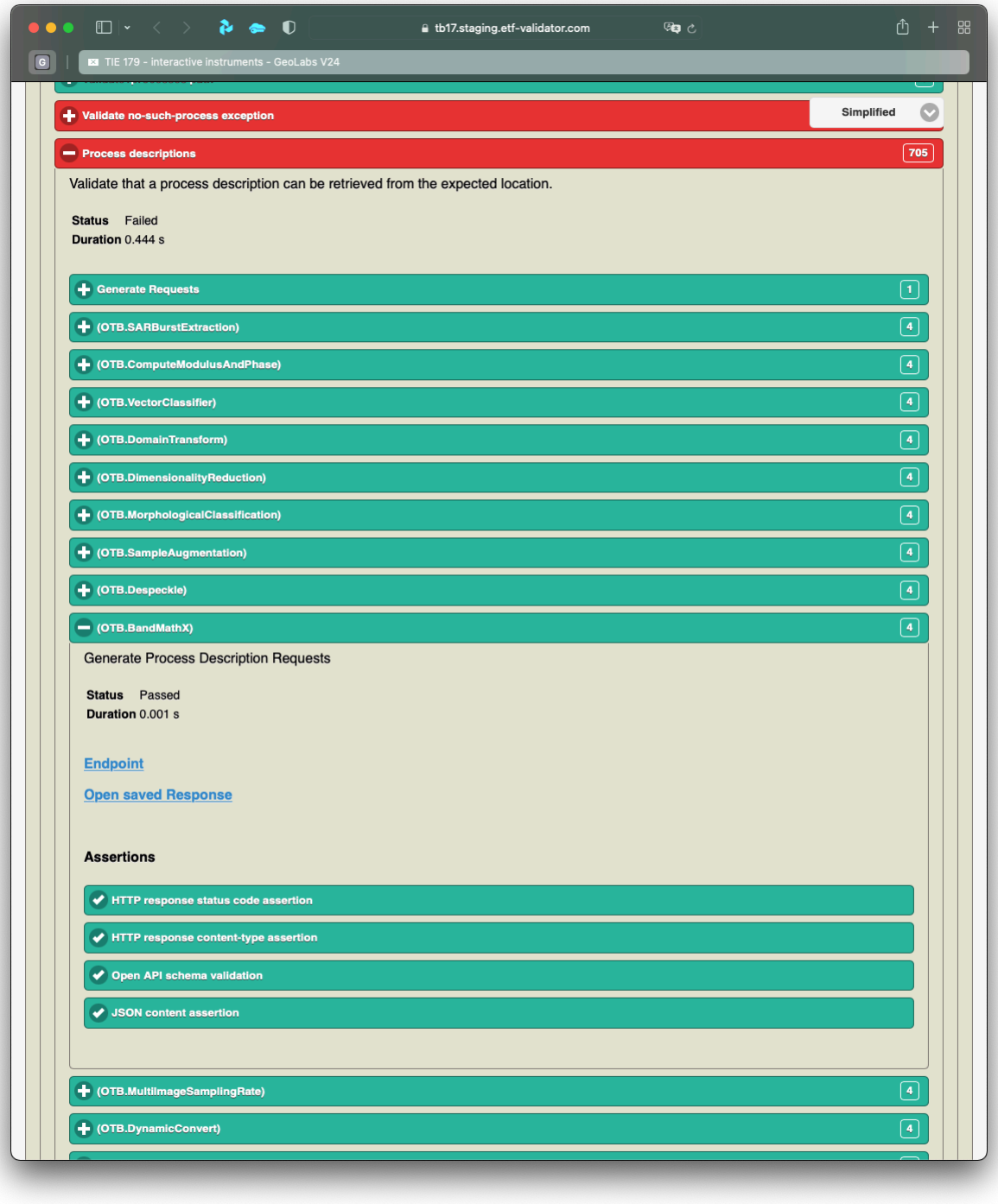

Figure 17 — NeoTL Generator report

If only one test is to be executed, an extractor must be used.

```
TestCase "Check for an echo process" {
    // /conf/core/job-creation-op
    id: jobs.echo
```

```
    description: "Check that an echo process with the id 'echo' or
'EchoProcess' exists"
    references:
    - "Processes /processes"
        "https://docs.ogc.org/DRAFTS/18-062.html#_jobs"
        AbstractTestCase

    ValidationStep "Extract EchoProcess ID" {
        id: extract
        description: "Check that an echo process with the id 'echo' or
'EchoProcess' exists"

        given:
            - Response from process.list.encoding.json

        when: Extractor extractors.echo.id executed

        then:
            - Assert JSON {
                ${echoId} exists
                otherwise FAIL with "No echo ID could be extracted"
              }
    }
}

Extractor "Extract Echo Process ID" {
    id: extractors.echo.id

    from RESPONSE
        as ${echoId} query $.*[?(@.id == 'echo' || @.id == 'EchoProcess')].id
}

TestCase "Request the process description of the the echo process" {
    id: jobs.echo.description
    description: "Check that the echo process description can be retrieved"
    references:
    - "Processes /processes"
        "https://docs.ogc.org/DRAFTS/18-062.html#_jobs"
        AbstractTestCase

    ValidationStep "Request EchoProcess description" {
        id: request
        description: "Check that the echo process description can be retrieved"

        given:
            - Value ${echoId} from jobs.echo.extract

        when: Request requests.process.description executed

        then:
            - Assert OpenAPI3 {
                schema
                "${schemaUrl}/process.yaml"
                validates
              }
    }
}
```

Figure 18 — Extractors

In the first test case, the ID of the echo process is queried and it is checked that such a process exists. If the validation step of the first test case does not fail, the validation step in the second test case jobs.echo.description is executed, which uses the extracted job ID in a request.

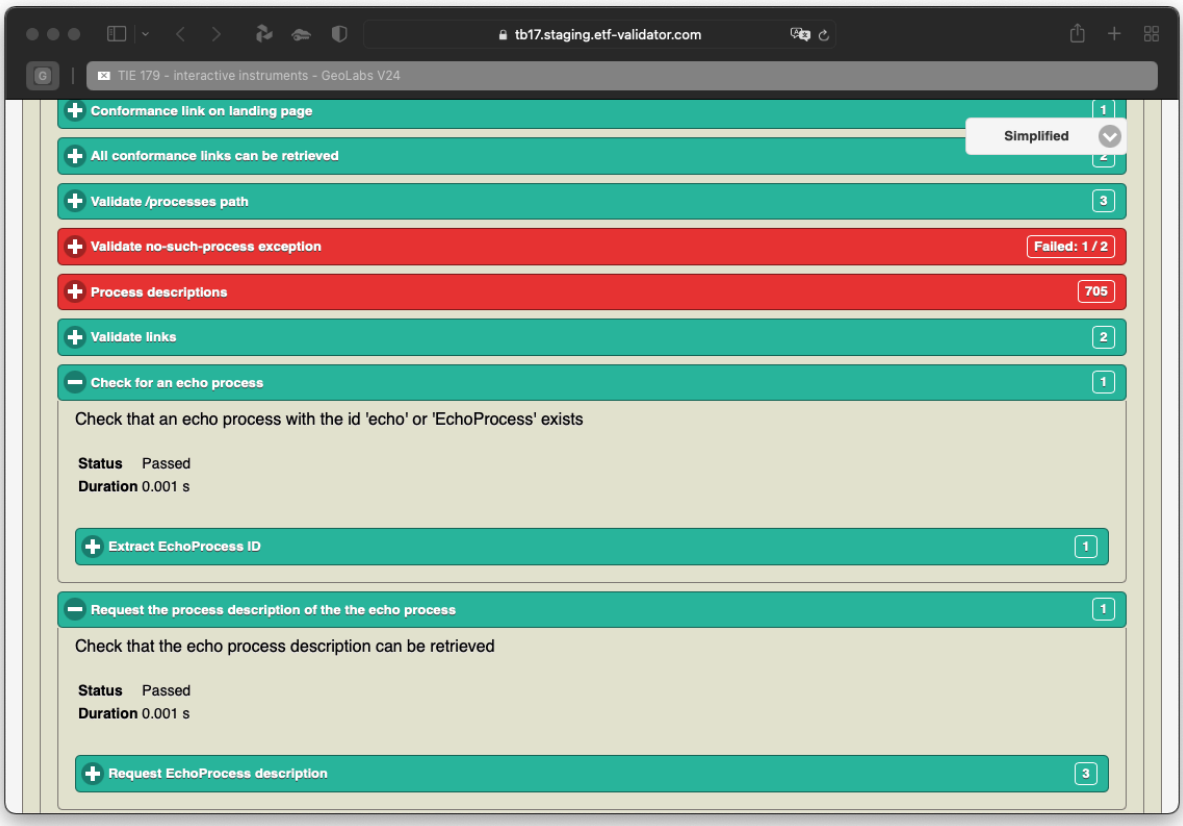

Figure 19 — NeoTL Extractor report

5.6. Extension Points

A DSL usually describes what should happen, but not exactly how it should happen. Up to a certain level, the NeoTL DSL allows to define assertions, such as:

- Does an element exist?
- Does there exist at least / exactly one element in a sequence / array?
- Does a property possess the given value?
- Does a JSON object have properties / is an array empty?
- How does it compare with other elements?
- Does a response validate against a schema?

- Is a specific content type returned?
- Is a specific status code returned?
- Does an HTML message validate against a W3C markup validator?

For many tests, these assertions are sufficient and they help the test developer to stay on an abstract level. For more complex queries, a general purpose language, such as Java, is inevitably required.

During the implementation of executable test cases, there were two areas where general purpose languages had to be used:

1. A Java library was integrated to generate input data from a JSON schema definition
2. A special case had to be considered for the generation of inputs for **Qualified Values**. This was implemented in XQuery 3.1. Although XQuery is originally designed for processing XML, it has a generic data model. It can be used to process both XML and JSON, enabling processing of new OGC API services as well as services that serve XML.

The listing in Figure 20 shows the second case with the XQuery extension.

```
(:~
 : Check if schema defines a Qualified Value.
 :
 : see https://docs.ogc.org/DRAFTS/18-062.html#req_core_process-execute-input-
inline-object
 :)
declare %private function oapip:mustBeQualified($schema) {
  exists($schema[type='object' and not(properties/bbox)]) or
exists($schema[oneOf or anyOf or allOf])
};

(:~
 : Wraps each qualified value into a 'value' JSON object
 :
 : see https://docs.ogc.org/DRAFTS/18-062.html#req_core_process-execute-input-
inline-object
 :)
declare function oapip:qualifiedSampleValues($nodes, $genFct as
function(item()) as item()* ) {
  for $n in $nodes/*
      return element { $n/name() } {
        let $generated := $genFct($n/schema)
        return if (oapip:mustBeQualified($n/schema)) then
          element { "value" } {
                (: qualified value :)
                $generated
          }
        else
          $generated
      }
```

```
};
```

Figure 20 — XQuery extension

The defined function is called an extractor.

```
Extractor "Extract the input and output values from the schema object" {
    id: extractors.inputs

    from RESPONSE
        as ${inputs} query {
            oapip:qualifiedSampleValues( $.inputs, etf:sampleFromSchema )
        }
}
```

Figure 21 — Extractor calling XQuery

The etf:sampleFromSchema function is an XQuery wrapper for the Java library mentioned in the first case and is not described here.

5.7. Modularization

All definitions are stored in files. For better overview, higher level structure definitions are only allowed once per file, otherwise the test developer is free to split the definitions into multiple files.

Assertions can also be externalized, combined into assertion groups and used in multiple validation steps:

```
AssertionGroup "JSON response received" {
    id: assertions.json.success

    assertions:
        - Assert HTTP { statusCode "200" }
        - Assert HTTP { contentType "application/json" }
}

AssertionGroup "Processes Conformance Class in Json Array" {
    id: assertions.processes.cc.json.array

    assertions:
        - AssertionGroup assertions.json.success must pass
        - Assert OpenAPI3 {
            schema
            "${schemaUrl}/confClasses.yaml"
            validates
        }
}
```

Figure 22 — Assertion Groups

If assertions in another file are referenced, this must be specified via an import statement.

```
import "Assertions.neotl"
```

Figure 23 — Import statement

The following directory structure was created for the Executable Test Suite. The content is described after the directory or file name.

```
.
├── ProcessesApi-testsuite.neotl # the Test Suite definition
├── extensions # the extension libraries that are used in this test suite
│   └── SampleGenerator.xqm
├── shared # definitions that are used in several Validation Steps
│   └── Assertions.neotl
├── tests
│   ├── 1_Core
│   │   ├── 1_Landing
│   │   │   ├── Actions.neotl
│   │   │   └── Landing-tcs.neotl # Test Case definitions
│   │   ├── 2_API
│   │   │   ├── Api-tcs.neotl
│   │   │   └── Actions.neotl
│   │   ├── 3_Conformance
│   │   │   ├── Actions.neotl # externalized Actions
│   │   │   ├── Assertions.neotl # externalized Assertions
│   │   │   └── Conformance-tcs.neotl # Test Case definitions
│   │   ├── 4_Processes
│   │   │   ├── Actions.neotl
│   │   │   ├── Assertions.neotl
│   │   │   └── Processes-tcs.neotl
│   │   ├── 5_Jobs
│   │   │   ├── Actions.neotl
│   │   │   └── Jobs-tcs.neotl
│   │   └── Core.neotl # the Test Module definition
│   ├── 2_Process_Description
# etc.
```

Figure 24 — File structure

Currently, Java libraries can also be placed as JAR files in the extension folder and be imported. They are loaded at test runtime with a separate Java ClassLoader to avoid conflicts between different ETSs.

5.8. Summary of executable Test developed

The tests developed in NeoTL have been published in a public GitHub repository [2]

At the beginning of the implementation, the definitions of the abstract test cases were parsed and templates were generated for the DSL. Since some abstract test cases describe the

validation of several aspects at once, they could not be implemented one-to-one, but had to be adapted to the DSL.

A further degree of automation could have been achieved if:

- the dependencies between tests would have been hyperlinked in a structured way
- the interaction with the implementation would have been separated from the validation of the results

Some abstract test cases could not be implemented because they require a sequence of events with certain test data or require internal control of the implementation under test's behavior that cannot be realized with a black box test (see Abstract Test Case /conf/core/job-results-failed or /conf/dismiss/job-dismiss-op).

Certain tests require the test framework to receive and analyze certain test data from the application (e.g. Server) being tested, via HTTP Post or HTTP Get Request (see /conf/callback/job-callback or /conf/core/job-results-async-raw-ref). These far-reaching framework functionalities were also not implemented.

The following table contains an overview of all test cases implemented during Testbed 17. The mapping of the abstract test case identifiers to the NeoTL executable test case identifiers can be seen in the second and third columns.

Table 2

ABSTRAC TEST CASE NO.	ABSTRACT TEST CASE IDENTIFIER	NEOTL EXECUTABLE TEST CASE IDENTIFIERS	STATUS	COMMENT	ABSTRACT TEST PURPOSE
1	/conf/core/landingpage-op	core.landing.any, core.landing.json, core.landing.html	implemented		Validate that a landing page can be retrieved from the expected location.
2	/conf/core/landingpage-success	core.landing.any, core.landing.json, core.landing.html	implemented		Validate that the landing page complies with the require structure and contents.
3	/conf/core/api-definition-op	core.api.json	implemented		Validate that the API Definition document can be retrieved from the expected location.
4	/conf/core/api-definition-success	core.api.json	implemented		Validate that the API Definition complies with the

ABSTRAC TEST CASE NO.	ABSTRACT TEST CASE IDENTIFIER	NEOTL EXECUTABLE TEST CASE IDENTIFIERS	STATUS	COMMENT	ABSTRACT TEST PURPOSE
					required structure and contents.
5	/conf/core/conformance-op	core. conformance, core. conformance. link,core. conformance. links	implemented		Validate that a Conformance Declaration can be retrieved from the expected location.
6	/conf/core/conformance-success	core. conformance. links	implemented		Validate that the Conformance Declaration response complies with the required structure and contents.
7	/conf/core/http	-	-	not a test case but a general precondition	Validate that the resource paths advertised through the API conform with HTTP 1. 1 and, where appropriate, TLS.
8	/conf/core/process-list	core.process.list	implemented		Validate that information about the processes can be retrieved from the expected location.
9	/conf/core/pl-limit-definition	core.process. list.limit	implemented		Validate that the limit query parameter is constructed correctly.
10	/conf/core/process-list-success	core.process.list	implemented		Validate that the process list content complies with the required structure and contents.
11	/conf/core/pl-links	core.process. list.links	implemented		Validate that the proper links

ABSTRAC TEST CASE NO.	ABSTRACT TEST CASE IDENTIFIER	NEOTL EXECUTABLE TEST CASE IDENTIFIERS	STATUS	COMMENT	ABSTRACT TEST PURPOSE
					are included in a response.
12	/conf/core/pl-limit-response	core.process.list.limit	implemented		Validate that the limit query parameter is processed correctly.
13	/conf/core/process	core.process.description	implemented		Validate that a process description can be retrieved from the expected location.
14	/conf/core/process-success	core.process.description	implemented		Validate that the content complies with the required structure and contents.
15	/conf/core/process-exception-no-such-process	core.process.exception.no.such.process	implemented		Validate that an invalid process identifier is handled correctly.
16	/conf/core/job-creation-op	core.jobs.echo	implemented	abstract description	Validate the creation of a new job.
17	/conf/core/job-creation-auto-execution-mode	core.jobs.echo	implemented		Validate that the server correctly handles the execution mode for a process.
18	/conf/core/job-creation-default-execution-mode	core.jobs.echo	implemented		Validate that the server correctly handles the default execution mode for a process.
19	/conf/core/job-creation-request	core.jobs.echo	implemented		Validate that the body of a job creation operation complies with the required structure and contents.

ABSTRAC TEST CASE NO.	ABSTRACT TEST CASE IDENTIFIER	NEOTL EXECUTABLE TEST CASE IDENTIFIERS	STATUS	COMMENT	ABSTRACT TEST PURPOSE
20	/conf/core/job-creation-inputs	core.jobs.echo	implemented		Validate that servers can accept input values both inline and by reference.
21	/conf/core/job-creation-input-inline	core.jobs.echo	implemented		Validate in-line process input values are validated against the corresponding schema from the process description.
22	/conf/core/job-creation-input-ref	core.jobs.echo	implemented		Validate that input values specified by reference in an execute request are correctly processed.
23	/conf/core/job-creation-input-array	core.jobs.echo	implemented		Verify that the server correctly recognizes the encoding of parameter values for input parameters with a maximum cardinality greater than one.
24	/conf/core/job-creation-input-inline-object	core.jobs.echo	implemented		Validate that inputs with a complex object schema encoded in-line in an execute request are correctly processed.
25	/conf/core/job-creation-input-inline-mixed	core.jobs.echo	implemented		Validate that inputs of mixed content encoded in-line in an execute request

ABSTRAC TEST CASE NO.	ABSTRACT TEST CASE IDENTIFIER	NEOTL EXECUTABLE TEST CASE IDENTIFIERS	STATUS	COMMENT	ABSTRACT TEST PURPOSE
					are correctly processed.
26	/conf/core/job-creation-input-inline-binary	core.jobs.echo	implemented		Validate that binary input values encoded as base-64 string in-line in an execute request are correctly processes.
27	/conf/core/job-creation-input-inline-bbox	core.jobs.echo	implemented		Validate that inputs with a bounding box schema encoded in-line in an execute request are correctly processed.
28	/conf/core/job-creation-input-validation	core.jobs.echo	implemented		Verify that the server correctly validates process input values according to the definition obtained from the process description.
29	/conf/core/job-creation-sync-raw-value-one		not implemented	no implementation for testing sync mode was available	Validate that the server responds as expected when synchronous execution is negotiated, a single output value is requested, the response type is raw and the output transmission is value.
30	/conf/core/job-creation-sync-raw-value-multi		not implemented	no implementation for testing sync mode was available	Validate that the server responds as expected when synchronous execution is

ABSTRAC TEST CASE NO.	ABSTRACT TEST CASE IDENTIFIER	NEOTL EXECUTABLE TEST CASE IDENTIFIERS	STATUS	COMMENT	ABSTRACT TEST PURPOSE
					negotiated, the response type is raw and the output transmission is value.
31	/conf/core/job-creation-sync-raw-ref		not implemented	no implementation for testing sync mode was available	Validate that the server responds as expected when synchronous execution is negotiated, the response type is raw and the transmission mode is ref.
32	/conf/core/job-creation-sync-raw-mixed-multi		not implemented	no implementation for testing sync mode was available	Validate that the server responds as expected when synchronous execution is negotiated, the response type is raw and the output transmission is a mix of value and reference.
33	/conf/core/job-creation-sync-document	core.jobs.echo	implemented		Validate that the server responds as expected when synchronous execution is negotiated and the response type is document.
34	/conf/core/job-creation-success-async	core.jobs.echo	implemented		Validate the results of a job that has been created using the async execution mode.

ABSTRACT TEST CASE NO.	ABSTRACT TEST CASE IDENTIFIER	NEOTL EXECUTABLE TEST CASE IDENTIFIERS	STATUS	COMMENT	ABSTRACT TEST PURPOSE
35	/conf/core/job-op	core.jobs.echo	implemented		Validate that the status info of a job can be retrieved.
36	/conf/core/job-success	core.jobs.echo	implemented		Validate that the job status info complies with the require structure and contents.
37	/conf/core/job-exception-no-such-job	core.jobs.echo	implemented		Validate that an invalid job identifier is handled correctly.
38	/conf/core/job-results	core.jobs.echo	implemented		Validate that the results of a job can be retrieved.
39	/conf/core/job-results-sync	core.jobs.echo	implemented		Validate that the server responds as expected when getting results from a job for a process that has been executed synchronously.
40	/conf/core/job-results-async-raw-value-one		not implemented	Not implemented in favor of higher priority tests	Validate that the server responds as expected when asynchronous execution is negotiated, one output is requested, the response type is raw and the output transmission is value.
41	/conf/core/job-results-async-raw-value-multi		not implemented	Not implemented in favor of higher priority tests	Validate that the server responds as expected when asynchronous execution is negotiated, more

ABSTRACT TEST CASE NO.	ABSTRACT TEST CASE IDENTIFIER	NEOTL EXECUTABLE TEST CASE IDENTIFIERS	STATUS	COMMENT	ABSTRACT TEST PURPOSE
					than one output is requested, the response type is raw and the output transmission is value.
42	/conf/core/job-results-async-raw-ref		not implemented	Not implemented in favor of higher priority tests	Validate that the server responds as expected when asynchronous execution is negotiated, the response type is raw and the output transmission is reference.
43	/conf/core/job-results-async-raw-mixed-multi		not implemented	Not implemented in favor of higher priority tests	Validate that the server responds as expected when asynchronous execution is negotiated, more than one output is requested, the response type is raw and the output transmission is a mix of value and reference.
44	/conf/core/job-results-async-document	core.jobs.echo	implemented		Validate that the server responds as expected when the asynchronous execution is negotiated and the response type is document.
45	/conf/core/job-results-failed		not implemented	Requires a concrete, reproducible data-	Validate that the job results retrieved using

ABSTRAC TEST CASE NO.	ABSTRACT TEST CASE IDENTIFIER	NEOTL EXECUTABLE TEST CASE IDENTIFIERS	STATUS	COMMENT	ABSTRACT TEST PURPOSE
				driven scenario or white box test, duplicate identifier with ATC 47	an invalid job identifier complies with the require structure and contents.
46	/conf/core/job-results-exception-results-not-ready		not implemented	Requires a concrete, reproducible data-driven scenario or white box test	Validate that the job results retrieved for an incomplete job complies with the require structure and contents.
47	/conf/core/job-results-failed	core.jobs.results.failed	implemented		Validate that the job results for a failed job complies with the require structure and contents.
48	/conf/ogc-process-description/json-encoding	core tests with ogc-process-description CC	indirectly	expressed by dependencies on 'core'	Verify that a JSON-encoded OGC Process Description complies with the required structure and contents.
49	/conf/ogc-process-description/inputs-def	core tests with ogc-process-description CC	indirectly	expressed by dependencies on 'core'	Verify that the definition of inputs for each process complies with the required structure and contents.
50	/conf/ogc-process-description/input-def	core tests with ogc-process-description CC	indirectly	expressed by dependencies on 'core'	Verify that the definition of each input for each process complies with the required structure and contents.
51	/conf/ogc-process-description/input-mixed-type	core tests with ogc-process-description CC	indirectly	expressed by dependencies on 'core'	Validate that each input of mixed type complies with the required

ABSTRAC TEST CASE NO.	ABSTRACT TEST CASE IDENTIFIER	NEOTL EXECUTABLE TEST CASE IDENTIFIERS	STATUS	COMMENT	ABSTRACT TEST PURPOSE
					structure and contents.
52	/conf/ogc-process-description/outputs-def	core tests with ogc-process-description CC	indirectly	expressed by dependencies on 'core'	Verify that the definition of outputs for each process complies with the required structure and contents.
53	/conf/ogc-process-description/output-def	core tests with ogc-process-description CC	indirectly	expressed by dependencies on 'core'	Verify that the definition of each output for each process complies with the required structure and contents.
54	/conf/ogc-process-description/output-mixed-type	core tests with ogc-process-description CC	indirectly	expressed by dependencies on 'core'	Validate that each output of mixed type complies with the required structure and contents.
55	/conf/json/definition	core tests with json CC	indirectly	expressed by dependencies on 'core'	Verify support for JSON.
56	/conf/html/content	core tests with html CC	indirectly	expressed by dependencies on 'core'	Verify the content of an HTML document given an input document and schema.
57	/conf/html/definition	core tests with html CC	indirectly	expressed by dependencies on 'core'	Verify support for HTML
58	/conf/oas30/completeness	openapi3.definitions.openapi	implemented	Checked with OpenAPI3 Assertion	Verify the completeness of an OpenAPI document.
59	/conf/oas30/exceptions-codes	openapi3.definitions.openapi	implemented	Checked with OpenAPI3 Assertion	Verify that the OpenAPI document fully describes potential exception codes.

ABSTRAC TEST CASE NO.	ABSTRACT TEST CASE IDENTIFIER	NEOTL EXECUTABLE TEST CASE IDENTIFIERS	STATUS	COMMENT	ABSTRACT TEST PURPOSE
60	/conf/oas30/oas-definition-1	openapi3. definitions. openapi, openapi3. definitions.html	implemented		Verify that JSON and HTML versions of the OpenAPI document are available.
61	/conf/oas30/oas-definition-2	openapi3. definitions. openapi	implemented		Verify that the OpenAPI document is valid JSON.
62	/conf/oas30/oas-impl	-	implemented	The description is too abstract	Verify that all capabilities specified in the OpenAPI definition are implemented by the API.
63	/conf/oas30/security	openapi3. definitions. openapi	implemented	Checked with OpenAPI3 Assertion	Verify that any authentication protocols implemented by the API are documented in the OpenAPI document.
64	/conf/job-list/job-list-op	oapi.processes. joblist.list	implemented		Validate that information about jobs can be retrieved from the expected location.
65	/conf/job-list/type-definition	oapi.processes. joblist. parameter.type	implemented		Validate that the type query parameter is constructed correctly.
66	/conf/job-list/processID-definition	core.jobs.echo	indirectly	Indirectly through other test case	Validate that the processID query parameter is constructed correctly.
67	/conf/job-list/status-definition	oapi.processes. joblist.	implemented		Validate that the status query

ABSTRACT TEST CASE NO.	ABSTRACT TEST CASE IDENTIFIER	NEOTL EXECUTABLE TEST CASE IDENTIFIERS	STATUS	COMMENT	ABSTRACT TEST PURPOSE
		parameter. status			parameter is constructed correctly.
68	/conf/job-list/datetime-definition		not implemented	Not implemented in favor of higher priority tests	Validate that the datetime query parameter is constructed correctly.
69	/conf/job-list/duration-definition		not implemented	Not implemented in favor of higher priority tests	Validate that the minDuration and maxDuration query parameter are constructed correctly.
70	/conf/job-list/limit-definition	oapi.processes. joblist. parameter.limit	implemented		Validate that the limit query parameter is constructed correctly.
71	/conf/job-list/job-list-success	oapi.processes. joblist.list	implemented		Validate that the job list content complies with the required structure and contents.
72	/conf/job-list/links	oapi.processes. joblist.links	implemented		Validate that the proper links are included in a response.
73	/conf/job-list/type-response	oapi.processes. joblist.list	implemented		Validate that the type query parameter is processed correctly.
74	/conf/job-list/processID-mandatory	core.jobs.echo	indirectly	Indirectly through other test case	Validate that the processID property is present in every job.
75	/conf/job-list/processID-response	core.jobs.echo	indirectly	Indirectly through other test case	Validate that the processID query parameter is processed correctly.

ABSTRACT TEST CASE NO.	ABSTRACT TEST CASE IDENTIFIER	NEOTL EXECUTABLE TEST CASE IDENTIFIERS	STATUS	COMMENT	ABSTRACT TEST PURPOSE
76	/conf/job-list/status-response	oapi.processes.joblist.parameter.status	implemented		Validate that the status query parameter is processed correctly.
77	/conf/job-list/datetime-response		not implemented	Not implemented in favor of higher priority tests	Validate that the datetime query parameter is processed correctly.
78	/conf/job-list/duration-response		not implemented	Not implemented in favor of higher priority tests	Validate that the minDuration and maxDuration query parameter are processed correctly.
79	/conf/job-list/limit-response	oapi.processes.joblist.parameter.limit	implemented		Validate that the limit query parameter is processed correctly.
80	/conf/callback/job-callback	oapi.callback.echo	not implemented ...	Callback implemented in Framework, Concept in DSL, but not yet in the Execution Engine	Validate the passing of a subscriber-URL in an execute request.
81	/conf/dismiss/job-dismiss-op		not implemented	echo process should have a sleep parameter, which can be set by the framework	Validate that a running job can be dismissed.
82	/conf/dismiss/job-dismiss-success		not implemented	echo process should have a sleep parameter, which can be set by the framework	Validate that the content returned when dismissing a job complies with the required structure and contents.

5.9. TIE results

Out of the 82 abstract test cases, 65 executable test cases were implemented. The implemented test cases were run against the OGC API — Processes implementation of GeoLabs which passed 47 tests.

Other implementations were tested, but these did not implement the latest version of the OGC API — Processes standard and therefore stopped the test runs at a very early stage.

5.10. Findings and Recommendations

1. During the implementation phase of the tests, maintenance of the used open-source validation library OpenAPI4J was discontinued. As an alternative, the KaiZen OpenAPI parser was tested, which did not reliably return errors when validating more complex schemas, especially when combining schemas with oneOf, anyOf and allOf. The OpenAPI4j library provided verifiable results with the GeoLabs implementation and therefore continued to be used.
2. Since a large number of JSON libraries are written in JavaScript, one could consider supporting this language alongside Java and XQuery as an additional extension language for NeoTL.
3. Currently, Java libraries can only be stored as JAR files in an ETS folder. Referencing specific versions as known from Maven or Gradle could be considered. However, the consideration could also be influenced by the second question, whether JavaScript libraries should also be usable.
4. Since only one ETS was implemented, it was not possible to evaluate how well the reusability of definitions between different ETSs works.
5. The abstract test cases should be referenceable via an URL so that the URL can be embedded in the executable tests. This might be performed using the OGC definition server[https://www.ogc.org/def-server].
6. The abstract test cases could be made more machine-readable so that dependencies can be extracted and that a distinction can be made between interaction with the implementation and expected behavior. A fully automated transformation into tests is not be possible, but a number of information could be pre-filled in templates.
7. Certain language concepts can be improved or simplified. This would require further feedback from users of NeoTL.
8. The IDE currently lacks a function to execute the test cases or individual requests locally. For a test, the ETS must currently be deployed.

9. In case of missing language concepts, the following approach has proven successful during development: in the first step, the extension is developed in a general purpose language. If it is used more often and could be useful for other executable test suites, one can think about including it directly as a language concept in the DSL.

10. The performance of the IDE can be improved, especially by caching constraint checks.

11. With a high number of generated validation steps, saving the responses, generating the report and viewing it in the browser takes a relatively long time. A test run with the GeoLabs implementation involves the execution of over 700 generated requests and the validation of over 2800 assertions. The runtime of the tests takes less than 40 seconds, while the whole report is available after almost 1.5 minutes. This could be optimized in the framework.

6

IMPLEMENTATION CONSIDERATIONS

6 IMPLEMENTATION CONSIDERATIONS

This section provides information about implementation considerations to the implementers of implementations as well developers of OGC Compliance tests.

6.1. Automatization of Tests from OGC Standards

6.1.1. TestNG Test Suite

The Java method stubs for the TestNG test suite were automatically generated.

The Java doc and method stub are automatically generated from the ATS, which source code is written in the AsciiDoc format.

The code to do the transformation of the tests was written by 52°North. Given that new OGC standards are written in AsciiDoc, a common library for the transformation of Abstract Test Suites to code in different programming languages would be useful.

6.2. Echo Process

In some cases, certain requirements of the specification (e.g. jobs conformance class) requires a special mechanism for testing. For example, when a process has to run asynchronously for some time. The test engine should monitor the execution and the intermediate status messages. But, if the process is not known, the implementation of the test execution can become a challenge.

If the echo process is implemented, it will provide a scenario that a test engine can follow to validate part of the test.

Here is an example definition of the echo process:

```
{
   "id":"EchoProcess",
   "title":"Echo Process",
   "description":"This process accepts and number of input and simply echoes
each input as an output.",
   "version":"1.0.0",
   "jobControlOptions":[
      "async-execute",
      "sync-execute"
   ],
   "outputTransmission":[
      "value",
```

```
      "reference"
   ],
   "inputs":{
      "pause":{
         "title":"Number of Pause Seconds",
         "description":"The number of seconds the EchoProcess process should
pause execution, to simulate actually doing something, and thus give the test
engine time to run the async execution tests.  For example, get statuses from
the server, etc.  If the server does not implement async-execute then this
parameter shall have no effect on the execution of the EchoProcess process and
test engines should simply ignore it.",
         "minOccurs":0,
         "schema":{
            "type":"double",
            "default":4.0
         }
      },
      "stringInput":{
         "title":"String Literal Input Example",
         "description":"This is an example of a STRING literal input.",
         "schema":{
            "type":"string",
            "enum":[
               "Value1",
               "Value2",
               "Value3"
            ],
            "example":[
               "Value2"
            ]
         }
      },
      "measureInput":{
         "title":"Numerical Value with UOM Example",
         "description":"This is an example of a NUMERIC literal with an
associated unit of measure.",
         "schema":{
            "type":"object",
            "required":[
               "value",
               "uom"
            ],
            "properties":{
               "measurement":{
                  "type":"number"
               },
               "uom":{
                  "type":"string"
               },
               "reference":{
                  "type":"string",
                  "format":"uri"
               }
            },
            "examples":[
               {
                  "meansurement":10,
                  "uom":"m"
               }
            ]
         }
      },
      "dateInput":{
```

```
            "title":"Date Literal Input Example",
            "description":"This is an example of a DATE literal input.",
            "schema":{
               "type":"string",
               "format":"dateTime",
               "examples":[
                  "2021-10-31T23:59:59"
               ]
            }
         },
         "doubleInput":{
            "title":"Bounded Double Literal Input Example",
            "description":"This is an example of a DOUBLE literal input that is
bounded between a value greater than 0 and 10.  The default value is 5.",
            "schema":{
               "type":"number",
               "format":"double",
               "minimum":0,
               "maximum":10,
               "default":5,
               "exclusiveMinimum":true,
               "examples":[
                  3.14159
               ]
            }
         },
         "arrayInput":{
            "title":"Array Input Example",
            "description":"This is an example of a single process input that is
an array of values.  In this case, the input array would be interpreted as a
single value and not as individual inputs.",
            "schema":{
               "type":"array",
               "minItems":2,
               "maxItems":10,
               "items":{
                  "type":"integer"
               },
               "examples":[
                  [
                     1,
                     7
                  ],
                  [
                     2,
                     4,
                     6
                  ]
               ]
            }
         },
         "complexObjectInput":{
            "title":"Complex Object Input Example",
            "description":"This is an example of a complex object input.",
            "schema":{
               "type":"object",
               "required":[
                  "property1",
                  "property5"
               ],
               "properties":{
                  "property1":{
                     "type":"string"
```

```
                },
                "property2":{
                   "type":"string",
                   "format":"uri"
                },
                "property3":{
                   "type":"number"
                },
                "property4":{
                   "type":"string",
                   "format":"dateTime"
                },
                "property5":{
                   "type":"boolean"
                }
             },
             "examples":[
                {
                   "property1":"Some string.",
                   "property2":"http://www.opengis.org",
                   "property5":true
                }
             ]
          }
       }
    },
    "outputs":{
       "stringOutput":{
          "schema":{
             "type":"string",
             "enum":[
                "Value1",
                "Value2",
                "Value3"
             ]
          }
       },
       "measureOutput":{
          "schema":{
             "type":"object",
             "required":[
                "value",
                "uom"
             ],
             "properties":{
                "measurement":{
                   "type":"number"
                },
                "uom":{
                   "type":"string"
                },
                "reference":{
                   "type":"string",
                   "format":"uri"
                }
             }
          }
       },
       "dateOutput":{
          "schema":{
             "type":"string",
             "format":"dateTime"
          }
```

```
      },
      "doubleOutput":{
         "schema":{
            "type":"number",
            "format":"double",
            "minimum":0,
            "maximum":10,
            "default":5,
            "exclusiveMinimum":true
         }
      },
      "arrayOutput":{
         "schema":{
            "type":"array",
            "minItems":2,
            "maxItems":10,
            "items":{
               "type":"integer"
            }
         }
      },
      "complexObjectOutput":{
         "schema":{
            "type":"object",
            "required":[
               "property1",
               "property5"
            ],
            "properties":{
               "property1":{
                  "type":"string"
               },
               "property2":{
                  "type":"string",
                  "format":"uri"
               },
               "property3":{
                  "type":"number"
               },
               "property4":{
                  "type":"string",
                  "format":"dateTime"
               },
               "property5":{
                  "type":"boolean"
               }
            }
         }
      }
   },
   "links":[
      {
         "href":"https://processing.example.org/oapi-p/processes/EchoProcess/ex
ecution",
         "rel":"http://www.opengis.net/def/rel/ogc/1.0/execute",
         "title":"Execute endpoint"
      }
   ]
}
```


Figure 25 — Example definition of the echo process

Another example is as follows:


```
{
   "id":"echo",
   "title":"Echo input",
   "description":"Simply echo the value provided as input",
   "version":"2.0.0",
   "jobControlOptions":[
      "sync-execute",
      "async-execute",
      "dismiss"
   ],
   "outputTransmission":[
      "value",
      "reference"
   ],
   "links":[
      {
         "rel":"execute",
         "type":"application/json",
         "title":"Execute End Point",
         "href":"http://tb17.geolabs.fr:8108/ogc-api/processes/echo/execution"
      },
      {
         "rel":"alternate",
         "type":"text/html",
         "title":"Execute End Point",
         "href":"http://tb17.geolabs.fr:8108/ogc-api/processes/echo/execution.
html"
      }
   ],
   "inputs":{
      "a":{
         "title":"Literal Input (string)",
         "description":"An input string",
         "schema":{
            "type":"string",
            "default":"Any value"
         }
      },
      "b":{
         "title":"Complex Input",
         "description":"A complex input ",
         "schema":{
            "oneOf":[
               {
                  "type":"string",
                  "contentEncoding":"utf-8",
                  "contentMediaType":"text/xml"
               },
               {
                  "type":"object"
               }
            ]
         }
      },
      "c":{
         "title":"BoundingBox Input ",
         "description":"A boundingbox input ",
         "schema":{
            "type":"object",
```

```
                "required":[
                   "bbox",
                   "crs"
                ],
                "properties":{
                   "bbox":{
                      "type":"array",
                      "oneOf":[
                         {
                            "minItems":4,
                            "maxItems":4
                         },
                         {
                            "minItems":6,
                            "maxItems":6
                         }
                      ],
                      "items":{
                         "type":"number",
                         "format":"double"
                      }
                   },
                   "crs":{
                      "type":"string",
                      "format":"uri",
                      "default":"urn:ogc:def:crs:EPSG:6.6:4326",
                      "enum":[
                         "urn:ogc:def:crs:EPSG:6.6:4326",
                         "urn:ogc:def:crs:EPSG:6.6:3785"
                      ]
                   }
                }
             }
          },
          "pause":{
             "title":"Literal Input (double)",
             "description":"An optional input which can be used to specify the
number of seconds to pause the service before returning",
             "schema":{
                "type":"number",
                "default":10,
                "format":"double",
                "nullable":true
             }
          }
       },
       "outputs":{
          "a":{
             "title":"The output a",
             "description":"The output a returned",
             "schema":{
                "type":"string",
                "default":"Any value"
             }
          },
          "b":{
             "title":"The output b",
             "description":"The output b returned",
             "schema":{
                "oneOf":[
                   {
                      "type":"string",
                      "contentEncoding":"utf-8",
```

```
                    "contentMediaType":"text/xml"
                 },
                 {
                    "type":"object"
                 }
              ]
           }
        },
        "c":{
           "title":"BoundingBox output ",
           "description":"A boundingbox output ",
           "schema":{
              "type":"object",
              "required":[
                 "bbox",
                 "crs"
              ],
              "properties":{
                 "bbox":{
                    "type":"array",
                    "oneOf":[
                       {
                          "minItems":4,
                          "maxItems":4
                       },
                       {
                          "minItems":6,
                          "maxItems":6
                       }
                    ],
                    "items":{
                       "type":"number",
                       "format":"double"
                    }
                 },
                 "crs":{
                    "type":"string",
                    "format":"uri",
                    "default":"urn:ogc:def:crs:EPSG:6.6:4326",
                    "enum":[
                       "urn:ogc:def:crs:EPSG:6.6:4326",
                       "urn:ogc:def:crs:EPSG:6.6:3785"
                    ]
                 }
              }
           }
        }
     }
  }
```


Figure 26 — Another example definition of the echo process

At the exception of the pause input parameter, every input is returned with the same name as an output. In the second case three inputs are present to illustrate the use of the three data types that were available in the WPS specification:

- a: a simple string (LiteralData),
- b: a complex data (can be application/json or text/xml),
- c: a bounding box

The pause parameter has been added to give the opportunity to test long running jobs. This way, the job list can be tested during the execution, especially the status and progress changes but, also to dismiss the conformance class, if implemented.

Implementation guidance:

- Servers SHALL implement sync-execute.
- Servers SHOULD implement async-execute.
- Servers SHALL implement value for outputTransmission.
- Servers SHOULD implement reference for outputTransmission.
- If async-execute is not supported then the text engine should ignore the pause input. The execution endpoint (link with rel=ogc:execute) should be adjusted to reflect the server actual execution endpoint for the EchoProcess process.

The test engine should then follow each branch in the following response summary tables: - https://docs.ogc.org/DRAFTS/18-062.html#sc_execute_response - https://docs.ogc.org/DRAFTS/18-062.html#_response_7 and verify that the server behaves accordingly. So, for example, in Table 7, the echo process is executed with:

- sync-execute
- response=document
- outputTransmission=reference
- and N outputs are returned

The server should respond with an HTTP status code of 200. The body of the response should be of type application/json and it should validate against the results.yaml schema. The test engine will also have to parse the response to verify that the value of each output in the response matches the input value specified by the engine.

6.3. Execution Process

When implementing a very trivial echo service and deploying it on the GeoLabs prototype Server Instance, the participants realized that it may be a great help for service developers to expose a fully-featured scenario that can be used for testing a service execution. The initial echo service accessible on the GeoLabs Server Instance was simply returning the output without potentially converting the input into another format or reprojecting it, depending on its type. In the sample implementation provided, 3 inputs are supported: one string, one complex, and one bounding box. So, depending on the service complexity or the level of development of a specific service, the Test Suite itself may have difficulty producing a relevant request automatically. In consequence, experiments were performed including the definition of multiple

paths corresponding to services' execution path with their associated request's body examples directly within the exposed OpenAPI.

So, we may consider adding one option to recommendation 24:

- Provide one or more example requests for a specific service in the exposed API.

Benefits of such an addition:

- Ease interactions with the published API using traditional OpenAPI tools, such as SwaggerUI;
- Ensure working scenarios for specific services;
- Advertise specific server implementation capabilities, such as automatic OGC Web Services publication;
- Give direct access to the Test Suites to a list of testable services without relying on fetching every single process description;
- Provide the Executable Test Suites with a set of pre-generated request bodies that can be used rather than having to generate the requests at run-time;
- Use external tools such as spectral to validate OpenAPI definition documents including the bodies of the requests as examples implying requests body's validation.

Identified issue:

- Introducing the requirement of testing external execute request body provided within the OpenAPI from the Test Suite;
- Keep the specification and the request bodies in sync;
- Increased OpenAPI definition size.

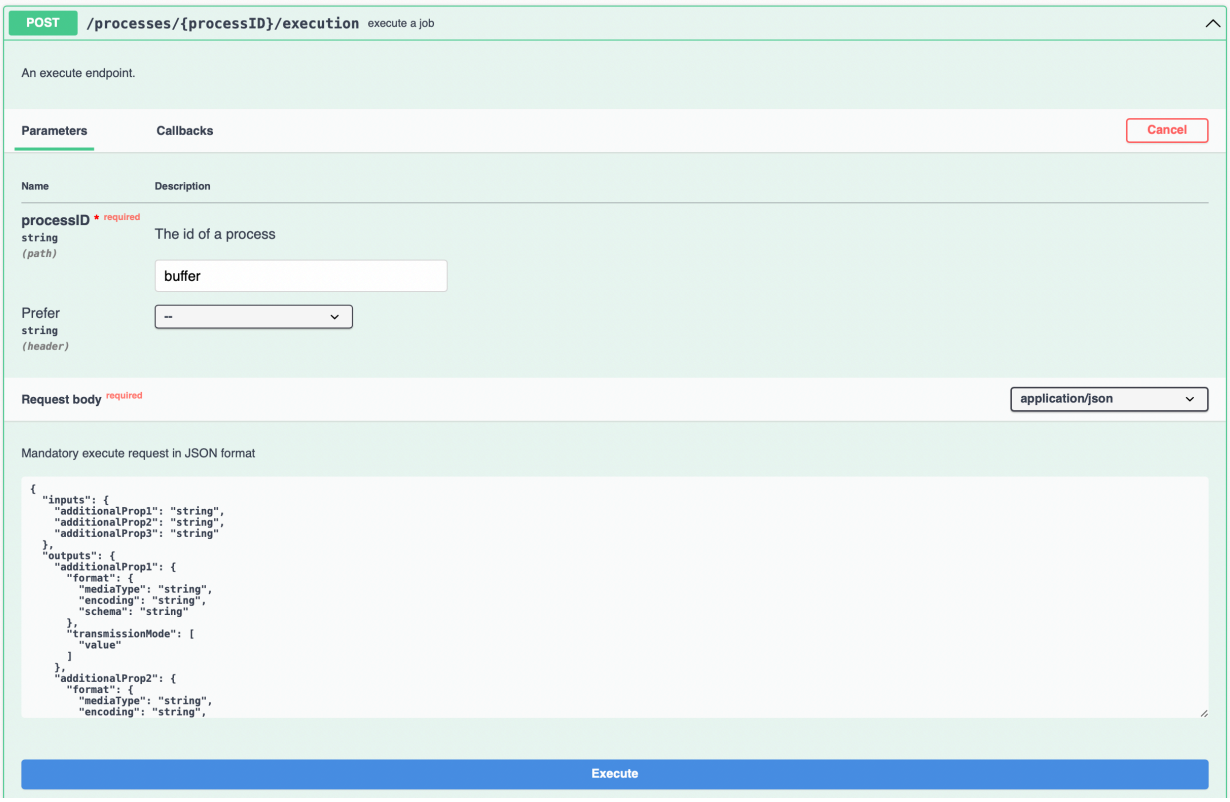

Figure 27 — SwaggerUI without examples

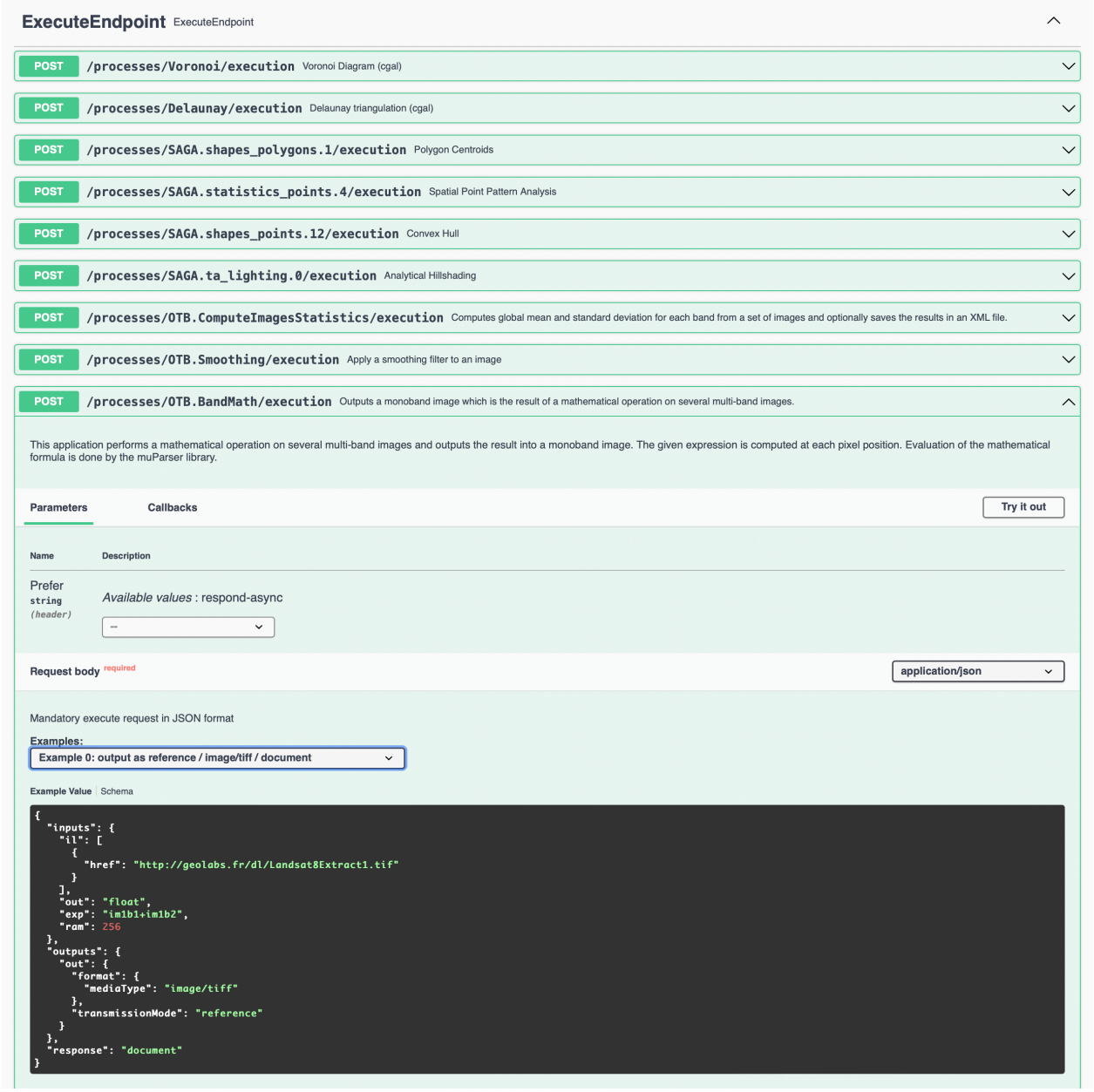

Figure 28 — SwaggerUI with additional paths and associated examples

7

CONCLUSIONS

7 CONCLUSIONS

The OGC Testbed 17 initiative provided a thread to advance testing of implementations of OGC Standards. The test for *OGC API – Processes — Part 1: Core* was developed using the TestNG framework (the current recommended OGC approach), and using NeoTL as an alternate approach.

The TestNG-based executable test suite needs to be improved to support better feedback and to support media types defined in RFC 7807. These enhancements to the TestNG-based executable test suite are planned for future implementation. NeoTL tests need to improve on performance and a mechanism to be able to test a sequence of events needs to be implemented. The NeoTL-based ETF framework also needs support for authentication and user profiles.

The Testbed-17 CITE thread also showed that part of the development of executable test scripts could be automated starting from an ATS in an AsciiDoc document and generating Java stubs that can then be implemented for TestNG and TEAM Engine. The generation of Java stubs from an ATS encoded in asciidoc could potentially leverage the metanorma toolchain.

8

FUTURE WORK

8 FUTURE WORK

The Testbed-17 CITE thread participants recommended that future work should include the following work items.

The ETS with NeoTL test framework can also be used to develop ETS test. Future work would require better integration with the OGC Validator, including performance, generating proper feedback to the user, and finding a mechanism to execute tests that are sequential.

The TestNG-based ETS requires further enhancements to improve OGC API testing, such as support for MIME types specified in RFC 7807 and provision of better feedback to the users (e.g. header content).

As mentioned in a previous section, the CITE SC noted that it is important that there be a single Executable Test Suite for each OGC Standard, to avoid ambiguity of test results. One suggestion was to enhance ETF towards supporting DevOps, while TEAM Engine continues to be the only tool used by the OGC Validator for compliance certification. This suggests that future testbeds should focus on implementing more TestNG-based Executable Test Suites for use in TEAM Engine.

OGC should consider making recommendations for tests that require inspection of the results such as when returning the results of a process. This could be done by creating a sample process that every implementer needs to implement or better creating an example documented scenario where the required responses are known.

A

ANNEX A (INFORMATIVE) REVISION HISTORY

A ANNEX A (INFORMATIVE) REVISION HISTORY

Table A.1 — Revision History

DATE	EDITOR	RELEASE	PRIMARY CLAUSES MODIFIED	DESCRIPTIONS
2021-11-19	L. Bermudez	.1	all	Draft submitted to OGC
2022-01-13	L. Bermudez	.2	all	Draft incorporating comments from OGC staff

BIBLIOGRAPHY

BIBLIOGRAPHY

1. Fowler, M.: GivenWhenThen, https://martinfowler.com/bliki/GivenWhenThen.html (2013)

2. Interactive Instruments: NeoTL Executable Test Suites for OGC API Processes on GitHub. https://github.com/interactive-instruments/ogc-api-processes-neotl-ets (2021)