OGC® DOCUMENT: 23-057R1

External identifier of this OGC® document: http://www.opengis.net/doc/IS/ogcapi-edr-2/1.0

Open Geospatial Consortium

OGC API - ENVIRONMENTAL DATA RETRIEVAL - PART 2: PUBLISH-SUBSCRIBE WORKFLOW

STANDARD
Extension

CANDIDATE PUBLIC RFC DRAFT

Version: 1.0
Submission Date: 2024-02-16
Approval Date: 2024-02-16
Publication Date: 2024-02-16
Editor: Tom Kralidis, Chris Little, Mark Burgoyne, Steve Olson, Shane Mill

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 © 2024 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 vi

II. KEYWORDS vi

III. PREFACE vii

IV. SECURITY CONSIDERATIONS viii

V. SUBMITTING ORGANIZATIONS ix

VI. SUBMITTERS ix

1. SCOPE 2

2. CONFORMANCE 4

3. NORMATIVE REFERENCES 6

4. TERMS, DEFINITIONS AND ABBREVIATED TERMS 8
   4.1. Terms and definitions 8
   4.2. Abbreviated terms 9

5. KEYWORDS 12

6. CONVENTIONS 14
   6.1. Identifiers 14

7. OVERVIEW 16

8. REQUIREMENTS CLASS PUBLISH-SUBSCRIBE (PUB/SUB) 18
   8.1. Overview 18
   8.2. OGC API Considerations 19

9. REQUIREMENTS CLASS PUBLISH-SUBSCRIBE (PUB/SUB) CHANNELS 23
   9.1. Overview 23

10. REQUIREMENTS CLASS PUBLISH-SUBSCRIBE (PUB/SUB) MESSAGE PAYLOADS 26
   10.1. Overview 26

ANNEX A (NORMATIVE) CONFORMANCE CLASS ABSTRACT TEST SUITE 31

A.1. Conformance Class Publish-Subscribe (Pub/Sub) 31
A.2. Conformance Class Publish-Subscribe (Pub/Sub) Channels 32
A.3. Conformance Class Publish-Subscribe (Pub/Sub) Message Payloads 33

ANNEX B (INFORMATIVE) EXAMPLES 37
B.1. Pub/Sub API Description Example 37

ANNEX C (INFORMATIVE) PUB/SUB MESSAGE PAYLOAD EXAMPLES 45
C.1. Pub/Sub Message Payload Example 45
C.2. Pub/Sub Message Payload Schema 46

ANNEX D (INFORMATIVE) USE CASES 49
D.1. Earth System Prediction model run and data granules notification 49

ANNEX E (INFORMATIVE) REVISION HISTORY 51

BIBLIOGRAPHY 53

LIST OF TABLES

Table E.1 51

LIST OF FIGURES

Figure 1 — Example of Publish-Subscribe workflow using OGC APIs 16
Figure 2 — OGC API landing page example link to an AsyncAPI document 19
Figure 3 — OGC API Pub/Sub link example to new collection notifications 20
Figure 4 — OGC API - Features example linking to a data payload channel 20
Figure 5 — OGC API - EDR example linking to a data payload channel 20
Figure 6 — Example id property 27
Figure 7 — Example pubtime property 28
Figure 8 — Example operation property 28

LIST OF RECOMMENDATIONS

REQUIREMENTS CLASS 1: REQUIREMENTS CLASS 'PUBLISH-SUBSCRIBE (PUB/SUB)' 18

REQUIREMENTS CLASS 2: REQUIREMENTS CLASS 'PUBLISH-SUBSCRIBE (PUB/SUB) CHANNELS' 23
REQUIREMENTS CLASS 3: REQUIREMENTS CLASS 'PUBLISH-SUBSCRIBE (PUB/SUB) MESSAGE PAYLOADS' 26
REQUIREMENT 1 19
REQUIREMENT 2 24
REQUIREMENT 3 27
REQUIREMENT 4 27
REQUIREMENT 5 28
REQUIREMENT 6 28
RECOMMENDATION 1 24
RECOMMENDATION 2 27
PERMISSION 1 18
PERMISSION 2 21
PERMISSION 3 28
CONFORMANCE CLASS A.1 31
CONFORMANCE CLASS A.2 32
CONFORMANCE CLASS A.3 33

I ABSTRACT

OGC APIs provide Web based capabilities which are typically based on polling for collection resource updates (new features, records, items, coverages, maps, etc.). Depending on a collection's temporal resolution or frequency of updates, an event-driven / Publish-Subscribe architecture provides a timely, efficient and low latency approach for the delivery of data updates or notifications of updates. The OGC API — Environmental Data Retrieval — Part 2: Publish-Subscribe Workflow provides recommendations on applying Publish-Subscribe architectural patterns to implementations of one or more OGC APIs.

II KEYWORDS

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

OGC API, Pub/Sub, Publish, Subscribe, Publish-Subscribe, Event driven architecture, Asynchronous, OGC document, OGC

III PREFACE

The Environmental Data Retrieval — Part 2 Standard provides:

1. Requirements for Publish-Subscribe patterns specific to event driven data workflows and

2. Options for realizing Publish-Subscribe workflow in OGC APIs.

The Standard is based on the OGC Publish-Subscribe White Paper OGC 20-081, as well as the Discussion paper for Publish-Subscribe workflow in OGC APIs OGC 23-013. The goal of this Standard is to provide a basis for Publish-Subscribe implementation patterns within the OGC API ecosystem.

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 Standard.

V SUBMITTING ORGANIZATIONS

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

- Meteorological Service of Canada
- UK Met Office
- US National Weather Service

VI SUBMITTERS

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

NAME	AFFILIATION
Tom Kralidis	Meteorological Service of Canada
Chris Little	UK Met Office
Mark Burgoyne	UK Met Office
Steve Olson	NOAA/NWS
Shane Mill	NOAA/NWS

1

SCOPE

1 SCOPE

The OGC API — Environmental Data Retrieval — Part 2: Publish-Subscribe Workflows Standard defines building blocks that can be assembled to implement Publish-Subscribe workflows (discovery, topic structure, encoding) as part of OGC API — Environmental Data Retrieval — Part 1: Core. A topic structure is the structured information that a publisher makes available to allow subscribers to choose information of interest to them.

This Standard defines a discovery capability that contains a topic structure in support of binding to notifications for data access and retrieval.

This Standard defines a baseline message payload which can contain summary descriptive information in GeoJSON about a given notification for new data events (new granule, new model run, etc.).

2

CONFORMANCE

2 CONFORMANCE

This Standard defines Publish-Subscribe patterns specific to event driven data workflows, as well as options for realizing Publish-Subscribe workflows in implementations of OGC API Standards.

Requirements for two standardization target types are considered:

- API integration
- Pub/Sub channels, and
- Message payload

Conformance with this Standard shall be checked using all the relevant tests specified in Annex A (normative) of this document. The framework, concepts, and methodology for testing, and the criteria to be achieved to claim conformance are specified in the OGC Compliance Testing Policies and Procedures and the OGC Compliance Testing web site.

In order to conform to this Standard, a software implementation shall choose to implement:

- Any one of the conformance levels specified in Annex A (normative).

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

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.

Advanced Message Queueing Protocol (AMQP) v1.0) https://www.oasis-open.org/standard/amqp

MQTT Version 5.0 https://docs.oasis-open.org/mqtt/mqtt/v5.0/mqtt-v5.0.html

AsyncAPI Specification https://www.asyncapi.com/docs/reference/specification/v3.0.0

DRAFT WMO guidance on technical specifications of WIS 2.0 https://wmo-im.github.io/wis2-guide

DRAFT WMO WIS2 Notification Message https://github.com/wmo-im/wis2-notification-message

DRAFT WMO WIS2 Topic Hierarchy https://github.com/wmo-im/wis2-topic-hierarchy

WebSockets https://developer.mozilla.org/en-US/docs/Web/API/WebSockets_API

IANA Link Relations https://www.iana.org/assignments/link-relations/link-relations.xhtml

Mark Burgoyne, David Blodgett, Charles Heazel, Chris Little: OGC 19-086r5, *OGC API — Environmental Data Retrieval Standard*. Open Geospatial Consortium (2022). http://www.opengis.net/doc/IS/ogcapi-edr-1/1.0.0.

Clemens Portele, Panagiotis (Peter) A. Vretanos, Charles Heazel: OGC 17-069r4, *OGC API — Features — Part 1: Core corrigendum*. Open Geospatial Consortium (2022). http://www.opengis.net/doc/IS/ogcapi-features-1/1.0.1.

DRAFT OGC API — Records — Part 1: Core (2020)

DRAFT OGC Features and Geometries JSON — Part 1: Core (2021)

H. Butler, M. Daly, A. Doyle, S. Gillies, S. Hagen, T. Schaub: IETF RFC 7946, *The GeoJSON Format*. RFC Publisher (2016). https://www.rfc-editor.org/info/rfc7946.

4

TERMS, DEFINITIONS AND ABBREVIATED TERMS

4 TERMS, DEFINITIONS AND ABBREVIATED TERMS

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

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

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

4.1. Terms and definitions

This document uses the terms defined in Sub-clause 5.3 of [OGC06-121r9], which is based on the ISO/IEC Directives, Part 2, Rules for the structure and drafting of International Standards. In particular, the word “shall” (not “must”) is the verb form used to indicate a requirement to be strictly followed to conform to this standard.

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

4.1.1. **Broker**

Intermediary between Subscribers and other Publishers which have been previously registered with the Broker. The Broker is not the original producer of Messages, but acts as an intermediary, (re-)publishing messages received from other Publishers and decoupling them from their Subscribers.

4.1.2. **Collection**

A geospatial resource that may be available as one or more sub-resource distributions that conform to one or more OGC API standards. (OGC 20-024)

4.1.3. Dataset

A collection of data, published or curated by a single agent, and available for access or download in one or more representations. (DCAT)

4.1.4. Distribution

A specific representation of a dataset. A dataset might be available in multiple serializations that may differ in various ways, including natural language, media-type or format, schematic organization, temporal and spatial resolution, level of detail or profiles (which might specify any or all of the above). (DCAT)

4.1.5. Subscriber

An entity that creates a subscription to a Publisher.

4.1.6. Message

A container within which data (such as JSON, XML, binary data, or other content) is transported. Messages may include additional information beyond data, including headers or other metadata used for routing or security purposes.

4.1.7. Channel

A term (string) used to filter messages from a Broker.

4.2. Abbreviated terms

AMQP	Advanced Message Queuing Protocol

AMQPS	Advanced Message Queuing Protocol Secure
API	Application Programming Interface
CORS	Cross-Origin Resource Sharing
GeoJSON	Geospatial JavaScript Object Notation (JSON)
HTTP	Hypertext Transfer Protocol
HTTPS	Hypertext Transfer Protocol Secure
IANA	Internet Assigned Numbers Authority
JSON	JavaScript Object Notation
MQP	Message Queuing Protocol
MQTT	Message Queuing Telemetry Transport
MQTTS	Message Queuing Telemetry Transport Secure
OASIS	Organization for the Advancement of Structured Information Standards
OGC	Open Geospatial Consortium
URI	Uniform Resource Identifier
WIS	WMO Information System
WMO	World Meteorological Organization
YAML	YAML Ain't Markup Language

5

KEYWORDS

5 KEYWORDS

6

CONVENTIONS

6 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.

6.1. Identifiers

The normative provisions in this standard are denoted by the URI:

`http://www.opengis.net/spec/ogcapi-environmental-data-retrieval-2/1.0`

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

7

OVERVIEW

7 OVERVIEW

Implementations of OGC API Standards provide Web based capabilities which are typically based on polling for collection resource updates (new features/records items, coverages, maps, etc.). Depending on a collection's temporal resolution or frequency of updates, an event-driven / Publish-Subscribe architecture provides a timely, efficient and low latency approach for delivery of data resource updates.

The following requirements and recommendations apply to Publish-Subscribe architectural patterns for use with implementations of OGC API Standards.

The Publish-Subscribe architecture assumes reasonable connectivity or favourable DDIL conditions. DDIL refers to Denied, Disrupted, Intermittent, or Limited communications.

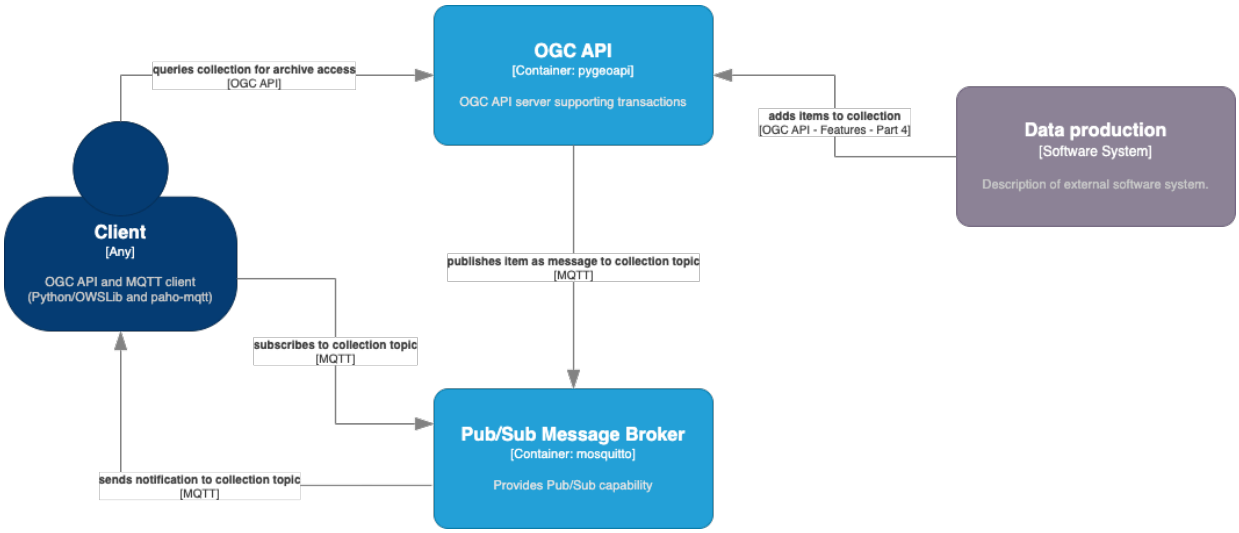

Figure 1 — Example of Publish-Subscribe workflow using OGC APIs

8

REQUIREMENTS CLASS PUBLISH-SUBSCRIBE (PUB/SUB)

8 REQUIREMENTS CLASS PUBLISH-SUBSCRIBE (PUB/SUB)

8.1. Overview

REQUIREMENTS CLASS 1: REQUIREMENTS CLASS 'PUBLISH-SUBSCRIBE (PUB/SUB)'	
IDENTIFIER	`http://www.opengis.net/spec/ogcapi-environmental-data-retrieval-2/1.0/req/pubsub`
TARGET TYPE	Pub/Sub
CONFORMANCE CLASS	Conformance class A.1: `http://www.opengis.net/spec/ogcapi-environmental-data-retrieval-2/1.0/conf/pubsub`
NORMATIVE STATEMENT	Requirement 1: `/req/pubsub/api`

Event-driven workflows provide Publish-Subscribe based capabilities as part of information systems and architectures. The Publish-Subscribe model also provides efficiencies in providing data "as it happens", thereby preventing potential clients from continuous polling to check on the availability of new data or resources.

The Open Geospatial Consortium (OGC) has conducted significant work on event-based models and architectures. The Publish-Subscribe model results in less network traffic and more timely responses to manage event-based models such as urgent, temporally unpredictable data (examples include, but are not limited to: traffic conditions, weather or hazard warnings, and real-time sensor data).

Building on the OGC Publish-Subscribe Interface Standard OGC 13-131r1, as well as the recommendations put forward in the OGC Pub/Sub White Paper [OGC 20-046] produced as part of OGC Testbed 12, as well as the Discussion paper for Publish-Subscribe workflow in OGC APIs [OGC 23-013], the EDR — Part 2: Publish-Subscribe Standard discusses approaches for integrating Publish-Subscribe architecture into the OGC API suite of Standards.

PERMISSION 1	
IDENTIFIER	`/per/pubsub/protocols`
A	A Publish-Subscribe MAY use the message queueing protocol of their choice and/or based on application requirements.

8.2. OGC API Considerations

The OGC API building block approach would typically be used for shared components in API implementations in support of a polling workflow. Using HTTP, this means that the client initiates and invokes requests and receives responses from the server. A key concept of the OGC API building blocks architecture is the service endpoint of the URL path specifying a resource and any similar sub-resources, which can be applied for Pub/Sub workflow as follows:

- Data producers: Messages are published to a broker, applied to a given channel (example: `collections/mycollection`).
- Broker provisioning: Published messages are sent to subscribers.
- Subscribers and data consumers: Messages are received by users subscribed to one or more channels (explicitly or using wildcards or filtering).

The above workflow requires adherence to a structure of information channels, auto-discovery of those channels, as well as processing of generic messages for broad interoperability by all components.

8.2.1. AsyncAPI

Based on research and testing, the Pub/Sub White Paper recommended the use of AsyncAPI. AsyncAPI provides an event-driven equivalent of what is provided by OpenAPI for OGC API Standards (description of protocols, channels, parameters, models, etc.). An implementation of the OGC API landing page requirements class can provide a link to an AsyncAPI document as follows:

```
{
    "rel": "service-desc",
    "type": "application/json",
    "title": "AsyncAPI document",
    "href": "https://example.org/asyncapi"
}
```

Figure 2 — OGC API landing page example link to an AsyncAPI document

REQUIREMENT 1	
IDENTIFIER	`/req/pubsub/api`
INCLUDED IN	Requirements class 1: `http://www.opengis.net/spec/ogcapi-environmental-data-retrieval-2/1.0/req/pubsub`
A	A landing page SHALL provide a link reference to the description of its Publish-Subscribe capabilities using a link relation of `service-desc`.

REQUIREMENT 1	
B	An API SHALL provide the description of its Publish-Subscribe capabilities using AsyncAPI to describe supported protocols, channels, and message payload descriptions.

8.2.2. Providing notification metadata as an OGC API endpoint

For Brokers providing notification metadata (as opposed to actual data payloads), an implementation of OGC API Building Blocks can, in parallel, readily provide GeoJSON-based notification messages via an OGC API — Features endpoint. Providing message payloads via an implementation of OGC API Standard(s) provides the additional benefit of querying for past messages over time in case of a lost connection.

8.2.3. Providing Pub/Sub links to collection updates

The links array could also provide references to the Pub/Sub capabilities available on the service. A **collection** link could reference a collection update notification channel.

NOTE: In the OGC API Suite of Standards, a collection is a geospatial resource (such as a dataset) that may be available as one or more sub-resource distributions that conform to one or more OGC API standards. See OGC API-Common: Part 2

```
{
    "rel": "collection",
    "type": "application/json",
    "title": "Data notifications",
    "href": "mqtt://example.org:8883",
    "channel": "collections"
}
```

Figure 3 — OGC API Pub/Sub link example to new collection notifications

8.2.4. Providing Pub/Sub links to collection item notifications

An **items** link could reference a data payload channel:

An OGC API — Features example

```
{
    "rel": "items",
    "type": "application/json",
    "title": "Data notifications",
    "href": "mqtt://example.org:8883",
    "channel": "collections/{collectionId}"
}
```

Figure 4 — OGC API - Features example linking to a data payload channel

An OGC API — EDR example

```
{
```

```
    "rel": "items",
    "type": "application/json",
    "title": "Data notifications",
    "href": "mqtt://example.org:8883",
    "channel": "collections/{collectionId}/items"
}
```

Figure 5 — OGC API - EDR example linking to a data payload channel

PERMISSION 2	
IDENTIFIER	`/per/pubsub/links`
A	A `collection` resource MAY provide a link reference to a Publish-Subscribe server from an OGC API implementation endpoint when Publish-Subscribe capabilities exist related to the `collection` service endpoint.
B	A Publish-Subscribe collection link reference MAY provide the `channel` property to allow for granular subscription.

9

REQUIREMENTS CLASS PUBLISH-SUBSCRIBE (PUB/SUB) CHANNELS

9 REQUIREMENTS CLASS PUBLISH-SUBSCRIBE (PUB/SUB) CHANNELS

9.1. Overview

REQUIREMENTS CLASS 2: REQUIREMENTS CLASS 'PUBLISH-SUBSCRIBE (PUB/SUB) CHANNELS'	
IDENTIFIER	`http://www.opengis.net/spec/ogcapi-environmental-data-retrieval-2/1.0/req/pubsub-channels`
TARGET TYPE	Pub/Sub
NORMATIVE STATEMENT	Requirement 2: `/req/pubsub-channels/channels`

9.1.1. Channels

The OGC API service endpoint specified by a URL path of resources and sub-resources can be used in parallel as a channel description when the data publisher wishes to provide Pub/Sub capability for resources normally available via an OGC API implementations instance in the same way. Below are examples of service endpoints or resources normally available via HTTP, and how they can be re-used as topics for Pub/Sub workflow:

- `/collections`: Notifies Subscribers whenever there is a change to the `/collections` resource (for example, addition of a new collection). The message payload would be collection metadata as defined in the OGC API — Common Standard, or a message referencing the collection metadata.
- `/collections/{collectionId}`: Notifies Subscribers whenever there is an update to a single `collection` resource (for example, spatial or temporal extents, new items, etc.). The message payload would be defined by the resource model of the given collection (items, etc.), or a message referencing the resource model of the collection.

For example, users could use a subscription to metadata records, which are usually small compared to the source data, and are therefore more transportable. This informs and notifies the user of changes prior to requesting the possibly large source data, especially when bandwidth is at a premium.

Using the OGC API service endpoints of the URL path of resource and sub-resources provides the key benefit that developers implementing OGC API Standards do not need to learn a different, additional approach or resource path for Pub/Sub (same content, additional interface).

REQUIREMENT 2	
IDENTIFIER	`/req/pubsub-channels/channels`
INCLUDED IN	Requirements class 2: `http://www.opengis.net/spec/ogcapi-environmental-data-retrieval-2/1.0/req/pubsub-channels`
A	Channels (topic/destination/node depending on protocol) that communicate changes to resources in an OGC Web API SHALL be expressed within an AsyncAPI channel with an `x-ogc-api-link` object.

RECOMMENDATION 1	
IDENTIFIER	`/rec/pubsub-channels/message-payloads`
A	A Publish-Subscribe server's message payload SHOULD be machine readable to enable automated workflow.

10

REQUIREMENTS CLASS PUBLISH-SUBSCRIBE (PUB/SUB) MESSAGE PAYLOADS

10 REQUIREMENTS CLASS PUBLISH-SUBSCRIBE (PUB/SUB) MESSAGE PAYLOADS

10.1. Overview

REQUIREMENTS CLASS 3: REQUIREMENTS CLASS 'PUBLISH-SUBSCRIBE (PUB/SUB) MESSAGE PAYLOADS'	
IDENTIFIER	`http://www.opengis.net/spec/ogcapi-environmental-data-retrieval-2/1.0/req/pubsub-message-payload`
TARGET TYPE	Pub/Sub
CONFORMANCE CLASS	Conformance class A.3: `http://www.opengis.net/spec/ogcapi-environmental-data-retrieval-2/1.0/conf/pubsub-message-payload`
NORMATIVE STATEMENTS	Requirement 3: `/req/pubsub-message-payload/geojson` Requirement 4: `/req/pubsub-message-payload/id` Requirement 6: `/req/pubsub-message-payload/operation` Requirement 5: `/req/pubsub-message-payload/pubtime`

A key component of Pub/Sub workflows is the message payload. Once a client subscribes to one or more channels from a given Pub/Sub server, notifications messages are sent using a given representation or encoding. Notification messages can be issued using any encoding that is deemed suitable by a given publisher.

While the Publish-Subscribe (Pub/Sub) Requirements Class recommends a machine-readable message payload, the Message Payload Requirements Class provides further requirements for interoperability of message payloads as part of an OGC API implementation ecosystem.

10.1.1. JSON, GeoJSON and FG-JSON

JSON is a widely used, flexible, relatively compact, format that is both machine readable and amenable to human inspection. A restricted profile of *JSON*, *GeoJSON*, can be used for geospatial data to improve interoperability, but it specifies that geospatial coordinate data must only use the *WGS84* coordinate reference system. *GeoJSON* is defined in the standard IETF RFC7946.

If *GeoJSON* is used as a payload for message notification, then

REQUIREMENT 3	
IDENTIFIER	`/req/pubsub-message-payload/geojson`
INCLUDED IN	Requirements class 3: `http://www.opengis.net/spec/ogcapi-environmental-data-retrieval-2/1.0/req/pubsub-message-payload`
A	A Pub/Sub notification message encoding SHALL be compliant to IETF RFC7946 GeoJSON.

If a Coordinate Reference System other than *WGS84* is used, then the *OGC Features and Geometry JSON* extension to *GeoJSON* should be used.

RECOMMENDATION 2	
IDENTIFIER	`/rec/pubsub-message-payload/crs`
A	A Pub/Sub notification message encoding describing data in a CRS other than WGS84 SHOULD use OGC Features and Geometries JSON.

10.1.2. Identifier

A universally unique identifier of the message using the UUID standard (RFC4122). The identifier is generated by the originator of the message. It remains the same throughout the lifetime of the message.

```
"id": "31e9d66a-cd83-4174-9429-b932f1abe1be"
```

Figure 6 — Example id property

REQUIREMENT 4	
IDENTIFIER	`/req/pubsub-message-payload/id`
INCLUDED IN	Requirements class 3: `http://www.opengis.net/spec/ogcapi-environmental-data-retrieval-2/1.0/req/pubsub-message-payload`
A	A Pub/Sub notification message SHALL provide an `id` property as a UUID.

10.1.2.1. pubtime

The `pubtime` property identifies the date/time of when the message was posted/published. `datetime` is published as specified in RFC3339 Clause 5.6 in the UTC timezone (`Z`). The

publication date/time is critical for subscribers to prevent message loss in providing awareness of how far behind the publisher they may be.

```
"properties": {
  ...
  "pubtime": "2022-03-20T04:50:18.314854383Z"
  ...
}
```

Figure 7 — Example pubtime property

REQUIREMENT 5	
IDENTIFIER	`/req/pubsub-message-payload/pubtime`
INCLUDED IN	Requirements class 3: `http://www.opengis.net/spec/ogcapi-environmental-data-retrieval-2/1.0/req/pubsub-message-payload`
A	A Pub/Sub notification message SHALL provide a `properties.pubtime` property in RFC3339 format.

10.1.2.2. operation

The `operation` property indicates the stage of the lifecycle for the resource described in the notification, and can be used to notify users that a resource has been updated or deleted. If not specified, the default value is `create`.

```
"properties": {
  ...
  "operation": "delete",
  ...
}
```

Figure 8 — Example operation property

REQUIREMENT 6	
IDENTIFIER	`/req/pubsub-message-payload/operation`
INCLUDED IN	Requirements class 3: `http://www.opengis.net/spec/ogcapi-environmental-data-retrieval-2/1.0/req/pubsub-message-payload`
A	A Pub/Sub Notification Message SHALL provide the `properties.operation` property to indicate if a resource has been updated or deleted.

PERMISSION 3	
IDENTIFIER	`/per/pubsub-message-payload/operation`

PERMISSION 3	
A	**A:** A Pub/Sub Notification Message MAY provide the `properties.operation` property to indicate if a resource has been inserted.

A

ANNEX A (NORMATIVE) CONFORMANCE CLASS ABSTRACT TEST SUITE

A

ANNEX A (NORMATIVE) CONFORMANCE CLASS ABSTRACT TEST SUITE

A.1. Conformance Class Publish-Subscribe (Pub/Sub)

CONFORMANCE CLASS A.1	
IDENTIFIER	`http://www.opengis.net/spec/ogcapi-environmental-data-retrieval-2/1.0/conf/pubsub`
REQUIREMENTS CLASS	Requirements class 1: `http://www.opengis.net/spec/ogcapi-environmental-data-retrieval-2/1.0/req/pubsub`
TARGET TYPE	Pub/Sub
CONFORMANCE TEST	Abstract test A.1: `/conf/pubsub/api`

ABSTRACT TEST A.1	
IDENTIFIER	`/conf/pubsub/api`
REQUIREMENT	Requirement 1: `/req/pubsub/api`
TEST PURPOSE	Validate that an EDR service provides AsyncAPI capabilities.
TEST METHOD	1. Construct a path for the API landing page 2. Issue a HTTP GET request on that path 3. Inspect all `link` objects in the response 4. Ensure that at least one exists with `rel=service-desc` that corresponds to an AsyncAPI 3.0 description

A.2. Conformance Class Publish-Subscribe (Pub/Sub) Channels

CONFORMANCE CLASS A.2	
IDENTIFIER	`http://www.opengis.net/spec/ogcapi-environmental-data-retrieval-2/1.0/conf/pubsub-message-channels`
REQUIREMENTS CLASS	http://www.opengis.net/spec/ogcapi-environmental-data-retrieval-2/1.0/req/pubsub-message-channels
CONFORMANCE TEST	Abstract test A.2: `/conf/pubsub-channels/channels`

ABSTRACT TEST A.2	
IDENTIFIER	`/conf/pubsub-channels/channels`
REQUIREMENT	Requirement 2: `/req/pubsub-channels/channels`
TEST PURPOSE	Validate that an EDR service provides AsyncAPI capabilities.
TEST METHOD	1. Construct a path for the API landing page 2. Issue a HTTP GET request on that path 3. Inspect all `link` objects in the response 4. Ensure that at least one exists with `rel=service-desc` that corresponds to an AsyncAPI 3.0 description 5. Issue a HTTP GET request on that path 6. Ensure that at least one exists with `rel=service-desc` that corresponds to an OpenAPI 3 description 7. Issue a HTTP GET request on that path 8. In the AsyncAPI response, for each channel object, look for an `x-ogc-api-link, noting the `href` property 9. Issue a HTTP GET request on that path 10. In the matching channel object from the previous two steps, test that the channel name has an equivalent path in the OpenAPI document

A.3. Conformance Class Publish-Subscribe (Pub/Sub) Message Payloads

CONFORMANCE CLASS A.3	
IDENTIFIER	`http://www.opengis.net/spec/ogcapi-environmental-data-retrieval-2/1.0/conf/pubsub-message-payload`
REQUIREMENTS CLASS	Requirements class 3: `http://www.opengis.net/spec/ogcapi-environmental-data-retrieval-2/1.0/req/pubsub-message-payload`
CONFORMANCE TESTS	Abstract test A.3: `/conf/pubsub-message-payload/geojson` Abstract test A.4: `/conf/pubsub-message-payload/id` Abstract test A.5: `/conf/pubsub-message-payload/operation` Abstract test A.6: `/conf/pubsub-message-payload/pubtime`

ABSTRACT TEST A.3	
IDENTIFIER	`/conf/pubsub-message-payload/geojson`
REQUIREMENT	Requirement 3: `/req/pubsub-message-payload/geojson`
TEST PURPOSE	Validate that a notification message is a valid GeoJSON document.
TEST METHOD	1. Construct a path for the API landing page 2. Issue a HTTP GET request on that path 3. Inspect all `link` objects in the response 4. Find a link that contains `rel=service-desc` that corresponds to an AsyncAPI 3.0 description 5. Using the server endpoint, subscribe to a given topic. 6. Upon receiving a message against the selected topic, validate that the message is GeoJSON compliant.

ABSTRACT TEST A.4	
IDENTIFIER	`/conf/pubsub-message-payload/id`
REQUIREMENT	Requirement 4: `/req/pubsub-message-payload/id`

ABSTRACT TEST A.4

TEST PURPOSE	Validate that a notification message provides an `id` property that is a UUID.
TEST METHOD	1. Construct a path for the API landing page 2. Issue a HTTP GET request on that path 3. Inspect all `link` objects in the response 4. Find a link that contains `rel=service-desc` that corresponds to an AsyncAPI 3.0 description 5. Using the server endpoint, subscribe to a given topic. 6. Upon receiving a message against the selected topic, ensure that the message has an `id` property. 7. Ensure that the `id` property is a valid UUID.

ABSTRACT TEST A.5

IDENTIFIER	`/conf/pubsub-message-payload/operation`
REQUIREMENT	Requirement 6: `/req/pubsub-message-payload/operation`
TEST PURPOSE	Validate that a notification message provides a `properties.operation` property.
TEST METHOD	1. Construct a path for the API landing page 2. Issue a HTTP GET request on that path 3. Inspect all `link` objects in the response 4. Find a link that contains `rel=service-desc` that corresponds to an AsyncAPI 3.0 description 5. Using the server endpoint, subscribe to a given topic. 6. Upon receiving a message against the selected topic, ensure that the message has a `properties.operation` property whose value is one of `insert`, `update`, `delete`..

ABSTRACT TEST A.6

IDENTIFIER	`/conf/pubsub-message-payload/pubtime`
REQUIREMENT	Requirement 5: `/req/pubsub-message-payload/pubtime`
TEST PURPOSE	Validate that a notification message provides a `properties.pubtime` property.
TEST METHOD	1. Construct a path for the API landing page

ABSTRACT TEST A.6

2. Issue a HTTP GET request on that path
3. Inspect all `link` objects in the response
4. Find a link that contains `rel=service-desc` that corresponds to an AsyncAPI 3.0 description
5. Using the server endpoint, subscribe to a given topic.
6. Upon receiving a message against the selected topic, ensure that the message has a `properties.pubtime` property whose value a valid RFC3339 datetime.

B

ANNEX B (INFORMATIVE) EXAMPLES

ANNEX B (INFORMATIVE) EXAMPLES

B.1. Pub/Sub API Description Example

The API is described using the AysncAPI 3.0.0 specification and an example response can be found below:

Example 1

```
asyncapi: '3.0.0'
info:
    title: AsyncAPI demo
    version: '0.0.1'
    description: |
        AsyncAPI description of the proposed Pub/Sub functionality
    contact:
        name: Contact Name
        email: you@example.org

servers:
    mqtt_prod:
        host: example.org
        protocol: mqtt
        description: MQTT endpoint
        security:
            - user-password: []

defaultContentType: application/json

channels:
    notify-collections:
        address: collections
        messages:
            $ref: '#/components/messages/collection_msg'
    notify-collections-wthr-stn:
        address: collections/wthr_st:
        messages:
            $ref: '#/components/messages/collection_msg'
    notify-collections-stream-gage:
        address: collections/stream_gage:
        messages:
            $ref: '#/components/messages/collection_msg'
    notify-collections-wthr_stn-items:
        address: collections/wthr_stn/items:
        messages:
            $ref: '#/components/messages/wthr_stn_msg'
```

```
    notify-collections-stream_gage-items:
        address: collections/stream_gage/items:
        messages:
            $ref: '#/components/messages/stream_gage_msg'

operations:
    notify-collections:
        action: receive
        channel:
            $ref: '#/channels/notify-collections'
    notify-collections-wthr-stn:
        action: receive
        channel:
            $ref: '#/channels/notify-collections-wthr-stn'
    notify-collections-stream-gage:
        action: receive
        channel:
            $ref: '#/channels/notify-collections-stream_gage'
    notify-collections-wthr_stn-items:
        action: receive
        channel:
            $ref: '#/channels/notify-collections-wthr_stn-items'
    notify-collections-stream_gage-items:
        action: receive
        channel:
            $ref: '#/channels/notify-collections-stream_gage-items'

components:
    messages:
        collection_msg:
            description: collection updated notification
            payload:
                type: object
                required:
                    - id
                    - href
                properties:
                    id:
                        type: string
                        description: collection name
                    time:
                        type: string
                        format: date-time
                        description: time collection changed
                    href:
                        type: string
                        format: uri
                        description: URL of the changed collection
        wthr_stn_msg:
            description: An observation formatted as GeoJSON
            payload:
                type: object
                additionalProperties: false
                properties:
                    id:
                        type: string
                    type:
                        type: string
                    geometry:
                        type: object
                        properties:
                            type:
                                type: string
```

```
                        coordinates:
                            type: array
                            items:
                            type: number
                            format: float
                    properties:
                        type: object
                        properties:
                            time:
                                type: string
                                format: date-time
                            id:
                                type: string
                            wind_direction:
                                type: number
                                format: float
                            wind_speed:
                                type: number
                                format: float
                            wind_gust:
                                type: number
                                format: float
                            visibility:
                                type: number
                                format: float
                            air_temperature:
                                type: number
                                format: float
                            dew_point:
                                type: number
                                format: float
                            mean_sea_level_pressure:
                                type: number
                                format: float
stream_gage_msg:
    description: Monitoring station data formatted as GeoJSON
    payload:
        type: object
        additionalProperties: false
        properties:
            id:
                type: string
            type:
                type: string
            geometry:
                type: object
                properties:
                    type:
                        type: string
                    coordinates:
                        type: array
                        items:
                        type: number
                        format: float
        links:
            type: array
                items:
                    type: object
                    properties:
                        rel:
                            type: string
                        type:
                           type: string
```

```
                                title:
                                   type: string
                                href:
                                    type: string
                                format: uri
                properties:
                    type: object
                    properties:
                        datetime:
                            type: string
                            format: date-time
                        label:
                            type: string
                        parametername:
                            type: array
                            items:
                                type: string
                        edrqueryendpoint:
                            type: string
                            format: uri
    securitySchemes:
        user-password:
            type: userPassword
```

Breaking down into the components:

Example 2

```
asyncapi: '3.0.0'
info:
    title: AsyncAPI demo
    version: '0.0.1'
    description: |
        AsyncAPI description of the proposed Pub/Sub functionality
    contact:
        name: Contact Name
        email: you@example.org
```

- The `asyncapi` field indicates you use the AsyncAPI version 3.0.0.
- The `info` field holds information about the API, such as its name, version, description, and license.

Example 3

```
servers:
    mqtt_prod:
        host: example.org
        protocol: mqtt
        protocolVersion: 5.0
        description: MQTT endpoint
        security:
            - user-password: []
```

- Each `server` object provides the following fields:
  - `host`: the server hostname and port
  - `protocol`: Pub/Sub protocol supported by the server

- `protocolVersion`: version of the Pub/Sub protocol supported by the server
- `description`: string describing the host
- `security`: reference to supported authentication types

Example 4

```
servers:
    mqtt:
        host: example.org
        protocol: mqtt
        protocolVersion: 5.0
        description: MQTT endpoint
        variables:
            port:
                enum:
                    - 1883
                default: 1883
    amqp:
        host: example.org
        protocol: amqp
        protocolVersion: 1.0.0
        description: AMQP endpoint
        variables:
            port:
                enum:
                    - 5672
                default: 5672
```

Different Pub/Sub protocols are supported as additional server objects, and can be defined accordingly.

Example 5

```
    channels:
        notify-collections:
            address: collections
                x-ogc-api-link:
                    rel: data
                    type: application/json
                    href: https://example.org/collections
                message:
                    $ref: '#/components/messages/collection_msg'
        notify-collections/wthr_stn:
            address: collections/wthr_stn
                x-ogc-api-link:
                    rel: collection
                    type: application/json
                    href: https://example.org/collections/wthr_stn
                messages:
                    $ref: '#/components/messages/collection_msg'
        notify-collections-stream_gage:
            address: collections/stream_gage
                x-ogc-api-link:
                    rel: collection
                    type: application/json
                    href: https://example.org/collections/stream_gage
                messages:
                    $ref: '#/components/messages/collection_msg'
```

```
    notify-collections-wthr_stn-items:
        address: collections/wthr_stn/items
            x-ogc-api-link:
                rel: items
                type: application/json
                href: https://example.org/collections/stream_gage/items
            messages:
                $ref: '#/components/messages/wthr_stn_msg'
    collections-stream_gage-items:
        address: collections/stream_gage/items
            x-ogc-api-link:
                rel: items
                type: application/json
                href: https://example.org/collections/stream_gage/items
            messages:
                $ref: '#/components/messages/stream_gage_msg'
```

- The `channels` section lists the events a user can subscribe to and can provide a schema for the associated message payloads.
- In the example the following events can be subscribed to:
  - `collections`
  - `collections/wthr_stn`
  - `collections/stream_gage`
  - `collections/wthr_stn/items`
  - `collections/stream_gage/items`

Example 6

```
operations:
    notify-collections:
        action: receive
        channel:
            $ref: '#/channels/notify-collections'
    notify-collections-wthr-stn:
        action: receive
        channel:
            $ref: '#/channels/notify-collections-wthr-stn'
    notify-collections-stream-gage:
        action: receive
        channel:
            $ref: '#/channels/notify-collections-stream_gage'
    notify-collections-wthr_stn-items:
        action: receive
        channel:
            $ref: '#/channels/notify-collections-wthr_stn-items'
    notify-collections-stream_gage-items:
        action: receive
        channel:
            $ref: '#/channels/notify-collections-stream_gage-items'
```

- The `operations` section lists the required operations and their send and receive capabilities.

Example 7

```
components:
```

- As in the OpenAPI specification, the `components` section is used to define reusable objects for different aspects of the AsyncAPI specification.

C

ANNEX C (INFORMATIVE) PUB/SUB MESSAGE PAYLOAD EXAMPLES

C

ANNEX C
(INFORMATIVE)
PUB/SUB MESSAGE PAYLOAD EXAMPLES

C.1. Pub/Sub Message Payload Example

The WMO WIS2 standard notification message format ensures that the WIS2 ecosystem (data publisher, data user, and global services) is a robust, effective, and unified exchange platform for weather, climate, and water data. The message provides notification metadata about the availability of a new data granule. The message is encoded using a GeoJSON object, and provides detailed information on the data notification (associated datetime of the granule, publishing datetime, integrity), as well as access to the data via a link object or inline content (useful for encoding small messages). Geometry is required (given GeoJSON requirements), however geometry can be expressed with a null value when generating the geometry in the message is not possible, practical or timely for data publishers. To support extensibility, additional properties are also valid (given the default definition in JSON Schema).

Using a GeoJSON object as the message payload supports broad interoperability given the large ecosystem of tooling (decoders, encoders) supporting the same approach. An example web application demonstrating the ease of integration can be found at https://kralidis.ca/tmp/wis2-data-notifications.html.

An example WIS2 Notification Message can be found below, extending the OGC API — Pub/Sub Notification Message Requirements with domain specific properties as required:

Example


```
{
    "id": "31e9d66a-cd83-4174-9429-b932f1abe1be",
    "version": "v04",
    "type": "Feature",
    "geometry": {
        "type": "Point",
        "coordinates": [
            6.146255135536194,
            46.223296618227444
        ]
    },
    "properties": {
        "pubtime": "2022-03-20T04:50:18.314854383Z",
        "operation": "create",
        "datetime": "2022-03-20T04:45:00Z",
        "integrity": {
            "method": "sha512",
```

```
            "value": "A2KNxvks...S8qfSCw=="
        },
        "data_id": "dataset/123/data-granule/UANT01_CWAO_200445___15103.bufr4",
        "metadata_id": "urn:x-wmo:md:can:eccc-msc:observations.swob",
        "content": {
            "encoding": "utf-8",
            "value": "encoded bytes from the file",
            "size": 457
        }
    },
    "links": [
        {
            "href": "https://example.org/data/4Pubsub/92c557ef-d28e-4713-91af-
2e2e7be6f8ab.bufr4",
            "rel": "canonical",
            "type": "application/x-bufr"
        },
        {
            "href": "https://example.org/oapi/collections/my-dataset/items/my-
data-granule",
            "rel": "item",
            "type": "application/json"
        }
    ]
}
```


C.2. Pub/Sub Message Payload Schema

Example


```
$schema: 'https://json-schema.org/draft/2020-12/schema'
$id: 'https://raw.githubusercontent.com/opengeospatial/ogcapi-environmental-
data-retrieval/master/extensions/pubsub/openapi/schemas/pubsub-message-payload-
schema.yaml'
title: OGC API - Pub/Sub message payload definition
description: OGC API - Pub/Sub message payload definition

required:
  - id
  - type
  - geometry
  - properties

properties:
  id:
    $ref: 'https://raw.githubusercontent.com/opengeospatial/ogcapi-features/
master/core/openapi/schemas/featureGeoJSON.yaml#/properties/id'
  type:
    $ref: 'https://raw.githubusercontent.com/opengeospatial/ogcapi-features/
master/core/openapi/schemas/featureGeoJSON.yaml#/properties/type'
  geometry:
    $ref: 'https://raw.githubusercontent.com/opengeospatial/ogcapi-features/
master/core/openapi/schemas/featureGeoJSON.yaml#/properties/geometry'
  properties:
    type: object
    required:
      - pubtime
    properties:
```

```
      pubtime:
        type: string
        format: date-time
        description: |
          Identifies the date/time of when the file was posted/published, in
RFC3339 format.
          The publication date/time is critical for subscribers to prevent
message loss by knowing
          their lag (how far behind the publisher they are).
      operation:
        type: string
        description: the event associated with the lifecycle of a resource.
        enum:
          - create
          - update
          - delete
        default: create
  links:
    $ref: 'https://raw.githubusercontent.com/opengeospatial/ogcapi-features/
master/core/openapi/schemas/link.yaml'
```

D

ANNEX D (INFORMATIVE) USE CASES

D

ANNEX D (INFORMATIVE) USE CASES

D.1. Earth System Prediction model run and data granules notification

A given numerical weather prediction system produces a weather forecast as part of a model run. A model run typically has associated forecast hours. Each forecast hour makes available one or many weather elements at different pressure levels of the atmosphere.

For example, Canada's Global Deterministic Prediction System (GDPS) produces two model runs per day, providing forecast of numerous weather elements for 33 pressure levels, at a resolution of 15 kilometres.[1].

A Pub/Sub workflow can be applied to an NWP system where:

- notifications are sent as individual weather elements (data granules) become available
- notifications are sent once the model run generation is complete, or "fully qualified"

As a result, Pub/Sub would eliminate the need for continuous polling while a model run is in progress.

[1] https://eccc-msc.github.io/open-data/msc-data/nwp_gdps/readme_gdps_en

E

ANNEX E (INFORMATIVE) REVISION HISTORY

ANNEX E (INFORMATIVE) REVISION HISTORY

Table E.1

DATE	RELEASE	EDITOR	PRIMARY CLAUSES MODIFIED	DESCRIPTION
2023-08-28	0.1	T. Kralidis	all	bootstrap
2024-01-10	0.2	C. Little	all	editorial consistency
2024-02-16	0.3	C. Little	all	workflow consistency

BIBLIOGRAPHY

BIBLIOGRAPHY

[1] Tom Kralidis, Mark Burgoyne, Steve Olson, Shane Mill: OGC 23-013, *Discussion paper for Publish-Subscribe workflow in OGC APIs*. Open Geospatial Consortium (2023). http://www.opengis.net/doc/discussion-paper/ogcapi-pubsub/1.0.

[2] Aaron Braeckel, Lorenzo Bigagli, Johannes Echterhoff: OGC 13-131r1, *OGC® Publish/Subscribe Interface Standard 1.0 – Core*. Open Geospatial Consortium (2016). http://www.opengis.net/doc/IS/pubsub-core/1.0.0.

[3] *OGC Publish-Subscribe White Paper* (2020)