Open Geospatial Consortium

Submission Date: 2015-12-19

Approval Date: 2016-02-19

Publication Date: 2016-08-22

External identifier of this OGC® document: http://www.opengis.net/doc/is/pubsub-core/1.0

Internal reference number of this OGC® document: 13-131r1

Version: 1.0

Category: OGC® Implementation Standard

Editors: Aaron Braeckel, Lorenzo Bigagli, Johannes Echterhoff

OGC® Publish/Subscribe Interface Standard 1.0 - Core

Warning

Although OGC 13-131r1 is an OGC Standard. This formatted version of 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. The normative version is available at:
http://docs.opengeospatial.org/is/13-131r1/13-131r1.html
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.

Document type: OGC® Standard
Document subtype: Implementation
Document stage: Approved
Document language: English

Contents

Figures

Tables

i. Abstract

Publish/Subscribe 1.0 is an interface specification that supports the core components and concepts of the Publish/Subscribe message exchange pattern with OGC Web Services. The Publish/Subscribe pattern complements the Request/Reply pattern specified by many existing OGC Web Services. This specification may be used either in concert with, or independently of, existing OGC Web Services to publish data of interest to interested Subscribers.

Publish/Subscribe 1.0 primarily addresses subscription management capabilities such as creating a subscription, renewing a subscription, and unsubscribing. However, this standard also allows Publish/Subscribe services to advertise and describe the supported message delivery protocols such as SOAP messaging, ATOM, and AMQP. Message delivery protocols should be considered to be independent of the Publish/Subscribe 1.0 standard. Therefore, OGC Publish/Subscribe only includes metadata relating to message delivery protocols in sufficient detail to allow for different implementations of Publish/Subscribe 1.0 to interoperate.

This specification defines Publish/Subscribe functionality independently of the binding technology (e.g., KVP, SOAP, REST). Extensions to this specification may realize these core concepts with specific binding technologies.

ii. Keywords

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

ogcdoc pubsub core specification

iii. Preface

The OGC® Abstract Specification does not require any changes to accommodate the technical contents of this document.

No future work is currently anticipated.

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. Submitting organizations

The following organizations submitted this document to the Open Geospatial Consortium Inc.

- National Center for Atmospheric Research (NCAR)
- National Research Council of Italy (CNR)
- International Geospatial Services Institute (iGSI) GmbH
- CubeWerx, Inc.
- Cooperative Institute for Research in the Atmosphere (CIRA)

v. Submitters

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

Name	Company
Aaron Braeckel	NCAR
Lorenzo Bigagli	CNR
Johannes Echterhoff	interactive instruments
Panagiotis (Peter) Vretanos	CubeWerx, Inc.
Chris MacDermaid	CIRA

1. Scope

This OGC standard defines core concepts and mechanisms for enabling the Publish/Subscribe messaging pattern with OGC Web Services. Publish/Subscribe may be used independently of or in conjunction with the Request/Reply messaging pattern.

This standard defines a common Publish/Subscribe conceptual framework and functionality, independently of specific binding technologies (e.g., KVP, SOAP, REST).

Reliable delivery of messages (i.e. assurance that messages that are sent are actually delivered) is out of scope for this specification, as reliable delivery techniques are dependent on the delivery method. Extensions to this specification may specify requirements and conformance for reliable delivery.

Authorization, authentication, and access control are not addressed in this specification. Extensions to this specification may specify requirements and conformance for security-related functionality.

2. Conformance

Conformance with this standard shall be checked using 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[1].

This standard distinguishes several conceptual roles for entities participating in Publish/Subscribe interactions: Sender, Receiver, Subscriber, and Publisher (defined in Clause 4). However, this standard only defines conformance requirements for the following Standardization Target Types.

- **Publisher** – entity that offers publications to Subscribers.
- **Receiver** – entity that receives messages from Senders (e.g. a Publisher).

This standard defines the Conformance Classes summarized in Table 1 and shown in Figure 1.

Requirements and conformance test URIs defined in this document are relative to http://www.opengis.net/spec/pubsub/1.0/.

Table 1: Conformance Classes

Conformance Class Name	Conformance Target	Operation or behavior	Conformance Class URI
Basic Receiver	Receiver	The **Receiver** shall implement the following operation: - *Notify*	**/conf/core/basic-receiver**
Basic Publisher	Publisher	The **Publisher** shall implement the following operations: - *Subscribe* - *Renew* - *Unsubscribe*	**/conf/core/basic-publisher**
Standalone Publisher	Publisher	The **Publisher** shall implement the Basic Publisher conformance class. Additionally the **Publisher** shall implement the following operations: - *GetCapabilities* - *GetSubscription*	**/conf/core/standalone-publisher**
Pausable Publisher	Publisher	The **Publisher** shall implement the Basic Publisher conformance class.	**/conf/core/pausable-publisher**

1 www.opengeospatial.org/cite

		Additionally the **Publisher** shall implement operations for subscription pausing and resuming: - *Pause* - *Resume*	
Message Batching Publisher	Publisher	The **Publisher** shall implement the Basic Publisher conformance class. Additionally the **Publisher** shall enable Subscribers to specify message-batching capabilities on the *Subscribe* operation. The **Publisher** shall follow message batching directives specified by the Subscriber when delivering messages.	**/conf/core/message-batching-publisher**
Heartbeat Publisher	Publisher	The **Publisher** shall implement the Basic Publisher conformance class. Additionally the **Publisher** shall allow Subscribers to specify heartbeat capabilities on the *Subscribe* operation. The **Publisher** shall follow heartbeat directives specified by the Subscriber and send regular heartbeat messages to allow Receivers to detect a failure or communications problem.	**/conf/core/heartbeat-publisher**
Brokering Publisher	Publisher	The **Publisher** shall implement the Standalone Publisher conformance class. Additionally the **Publisher** shall support the management of brokered Publishers: - *RegisterPublisher* - *RemovePublisher* - *GetPublisher* The **Publisher** shall receive messages from the brokered Publishers and republish them.	**/conf/core/brokering-publisher**
Publication Manager	Publisher	The **Publisher** shall implement the Basic Publisher conformance class.	**/conf/core/publication-manager**

		Additionally the **Publisher** shall support the creation, removal, and subscriptions to user-defined publications: • *CreatePublication* • *RemovePublication*	
Capabilities Filtering	Publisher	The **Publisher** shall implement the Standalone Publisher conformance class. Additionally the **Publisher** shall support filtering of the Publications section (i.e., contents section) of *GetCapabilities* responses.	**/conf/core/capabilities-filtering**

The relationships between conformance classes are shown below in Figure 1.

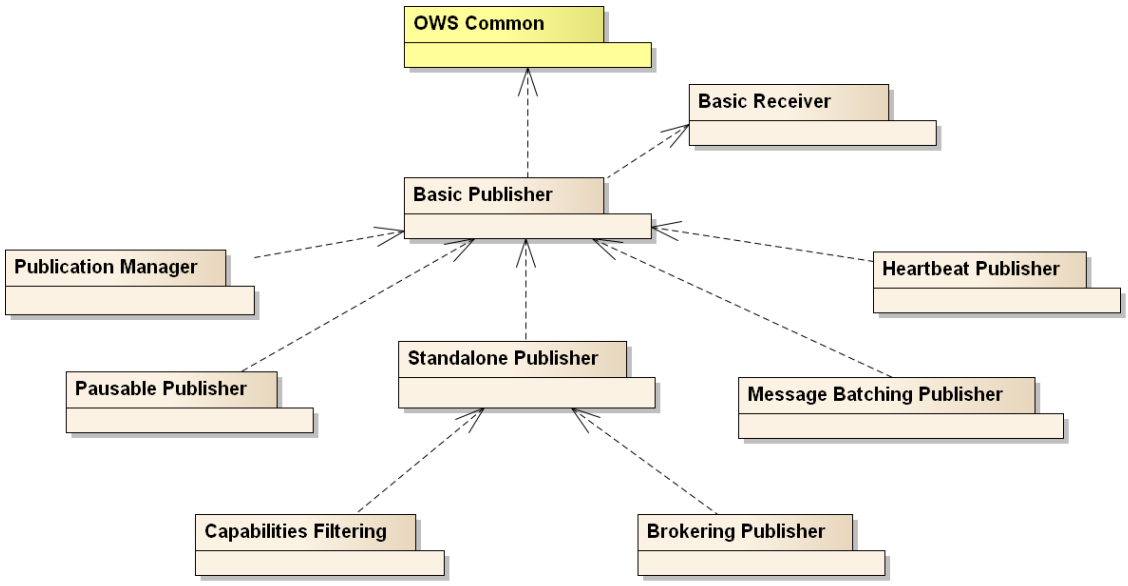


Figure 1: Relationships between Publish/Subscribe Core Conformance Classes

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

3. References

This *OGC Publish/Subscribe 1.0 Core* standard consists of the present document. An associated XML Schema is provided for consistency among extensions to this standard. For this standard, the provided XML Schema may be considered informative.

The complete *OGC Publish/Subscribe 1.0* specification is identified by OGC URI http://www.opengis.net/spec/pubsub/1.0. It is available for download from http://www.opengeospatial.org/standards/pubsub. The informative XML Schema is posted on-line at http://schemas.opengis.net/pubsub/1.0 as part of the OGC schema repository.

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

ISO/TS 19103:2005, *Geographic information* — Conceptual schema language

OGC 06-121r3, *OGC Web Services Common Specification*, OGC® Implementation Standard 1.1.0 (9 February 2007)

W3C XML Schema Part 1, *XML Schema Part 1: Structures*, W3C Recommendation (2 May 2001)

W3C XML Schema Part 2, *XML Schema Part 2: Datatypes*, W3C Recommendation (2 May 2001).

4. Terms and Definitions

This document uses the terms defined in Sub-clause 5.3 of [OGC 06-121r3], 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
Message
A container within which data (such as XML, binary data, or other content) is transported. Messages may include additional information beyond data, including headers or other information used for routing or security purposes.

4.2
Publication
A uniquely identified aggregation of messages published by a Publisher over time. A Publisher may offer any number of publications that Subscribers may subscribe to.

4.3
Publisher
An entity that offers publications to Subscribers; supports subscription management (subscribe, unsubscribe) and is responsible for filtering and matching messages of interest to active subscriptions.

4.4
Receiver
An entity that receives messages from Senders; may (but need not) be the original Subscriber.

4.5
Sender
Entity that sends messages to Receivers; may (but need not) be the initial creator/producer of the data in the message payload.

4.6
Subscriber
Entity that creates a subscription at a Publisher; may (but need not) be the Receiver of delivered messages.

4.7
Subscription
Expression of interest in all or part of a publication offered by a Publisher. When a subscription has been created, the Publisher delivers messages that match the subscription criteria to the Receiver defined in the subscription.

5. Conventions

5.1 Abbreviations

In this document the following abbreviations and acronyms are used or introduced:

HTTP	Hypertext Transfer Protocol
MEP	Message Exchange Pattern
OGC	Open Geospatial Consortium
OMG	Object Management Group
UML	Unified Modeling Language (an object modeling language)

XML eXtensible Markup Language

5.2 UML Notation

All symbols used in this document are UML 2 (Unified Modeling Language) as defined by OMG and accepted as a publicly available standard by ISO in its earlier 1.3 version.

All classes in this standard are *extensible* and may be extended with application- or domain-specific content via `Extension` blocks.

NOTE The UML shown in this standard is considered conceptual and abstract, and should not be interpreted as an implementation strategy for bindings that extend and implement this standard. For example, TM_Instant from ISO 19108 is used to represent time instants for conceptual clarity, but bindings and implementations of this standard may realize TM_Instant as a GML TimeInstant, an ISO 8601 date string, or any other representation that is consistent with TM_Instant.

5.3 Referencing Conventions

This standard references UML classes from other specifications. When referencing UML classes not defined in this standard, the class name will be qualified with the document of origin. For example, a reference to the ISO 19108 TM_Instant is referenced as:

TM_Instant [see ISO/TS 19103:2005]

Many referenced UML classes are instantiated as XML schema, such as the GML realization of ISO TC211 standards. This standard only normatively references UML representations.

6. Publish/Subscribe Overview

Two primary parties characterize the publish/subscribe model: a Publisher that is publishing information and a Subscriber that is interested in all or part of the published information. The publish/subscribe messaging model is distinguished from the request/reply model by the use of an ongoing, persistent, expression of interest (a *subscription*) and the asynchronous delivery of messages that match a subscription.

The entity subscribing for published information (the Subscriber) and the entity to which data is delivered (the Receiver) are often one and the same. However, they are distinguished in this standard to allow for these roles to be segregated in cases such as a system component mass-subscribing on behalf of the ultimate Receivers of messages.

Similarly, while the Publisher and Sender roles may be segregated they are often implemented as the same entity. Senders may be unaware of the ultimate recipients of their messages and of the architecture of the system into which they deliver messages, such as with multi-cast delivery or ATOM feeds.

While multiple entities (Publisher, Subscriber, Sender, and Receiver) are distinguished in this Clause, requirements are only allocated against Publishers and Receivers in this standard.

6.1 Publish/Subscribe workflow

The publish/subscribe workflow is depicted in Figure 2.

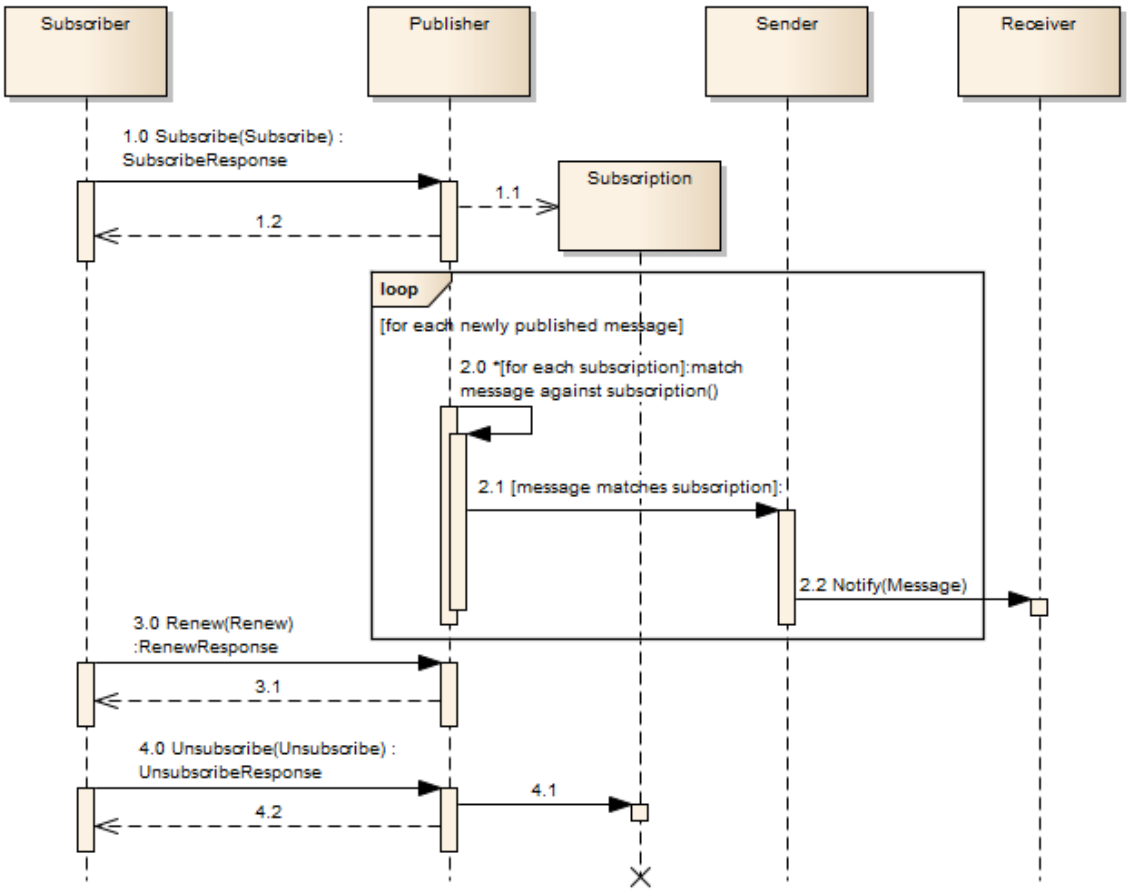


Figure 2: Publish/Subscribe workflow

The first step to initiate a publish/subscribe message exchange is the creation of a subscription. A subscription defines which messages available at the Publisher are of interest to the Subscriber. The Subscriber is an entity that creates a subscription on behalf of a Receiver using the *Subscribe* operation on a Publisher (1.0). If the Publisher accepts the subscribe request, it creates a subscription (1.1) and returns a response informing the requester of the outcome of its request – either success or an exception (1.2).

When a subscription is submitted, a Subscriber may supply filter criteria. Filter expressions evaluate to a boolean value for each individual message. Those messages that evaluate to true for all filter expressions on a subscription are considered to have matched. Filter criteria can filter by message content (such as XPath or OGC Filter Specification), by message metadata (such as header content), or by other criteria.

Whenever a new message is available to the Publisher, it attempts to match it against each subscription (2.0). If the message matches the filter criteria of a subscription the Publisher initiates Sender delivery to the location and/or Receiver specified for the subscription (2.1). Messages are delivered asynchronously as they become available on the Publisher.

Every subscription has a defined time at which it expires. When that time is reached the Publisher terminates the subscription. The *Renew* operation may be utilized (3.0) to set a new termination time for a subscription. If the Publisher accepts the request, the new termination time is set on the subscription and the Publisher returns a response (3.1) informing the Subscriber of the outcome of the request.

Termination of a subscription may be requested any time after the subscription was created using the *Unsubscribe* operation (4.0). If the Publisher accepts the request, it terminates the subscription (4.1) and returns a response (4.2) informing the Subscriber of the outcome of the request.

7. Requirements Class: Basic Receiver

Requirements Class	
http://www.opengis.net/spec/pubsub/1.0/req/core/basic-receiver	
Target type	Receiver
Requirement	/req/core/basic-receiver/notify

This Requirements Class specifies the basic operation of a Receiver:

> ***Notify*** – delivery of a message to the Receiver. In the context of Publish/Subscribe this is often the delivery of a message that matches the filter criteria of a given subscription.

7.1 Notify operation

The *Notify* operation is offered by a Receiver to allow the delivery of a message.

In the context of Publish/Subscribe a Sender may use the *Notify* operation to deliver a message that matches the filter criteria of a subscription to the Receiver associated with that subscription. Some pull-based message delivery methods, such as ATOM, do not require that the Receiver to implement this requirements class for message delivery.

Requirement
/req/core/basic-receiver/notify
Req 1 A **Receiver** shall offer the *Notify* operation

7.1.1 Request

The *Notify* operation is based on the fundamental datagram pattern, where a single message is sent from one system entity to another (the Receiver), without expecting a response. Therefore, it need not be the same as the Request/Response message exchange pattern.

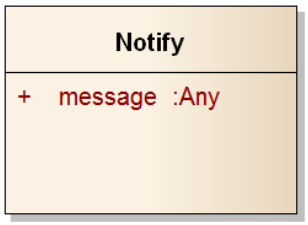


Figure 3: Notify operation message

Table 2: Notify operation message properties

Name	Definition	Data type and values	Multiplicity and use
message	The content of the message.	Any	One (Mandatory)

7.1.2 Response

No response is expected/defined for the Notify operation.

7.1.3 Exceptions

No exception is defined for the Notify operation.

8. Requirements Class: Basic Publisher

Requirements Class	
http://www.opengis.net/spec/pubsub/1.0/req/core/basic-publisher	
Target type	Publisher
Dependency	http://www.opengis.net/doc/IS/OWS/1.1/clause/8
Dependency	http://www.opengis.net/doc/IS/OWS/1.1/clause/10
Requirement	/req/core/basic-publisher/getcapabilities-conf-class-listing
Requirement	/req/core/basic-publisher/getcapabilities-filtercapabilities
Requirement	/req/core/basic-publisher/getcapabilities-unique-filter-languages
Requirement	/req/core/basic-publisher/getcapabilities-deliverycapabilities
Requirement	/req/core/basic-publisher/getcapabilities-unique-delivery-method
Requirement	/req/core/basic-publisher/getcapabilities-publications

Requirement	/req/core/basic-publisher/publication-valid-filter-language
Requirement	/req/core/basic-publisher/publication-valid-delivery-method
Requirement	/req/core/basic-publisher/publication-unique-publication-id
Requirement	/req/core/basic-publisher/valid-exceptions
Requirement	/req/core/basic-publisher/exception-version
Requirement	/req/core/basic-publisher/subscribe
Requirement	/req/core/basic-publisher/subscribe-assign-unique-id
Requirement	/req/core/basic-publisher/subscribe-default-termination-time
Requirement	/req/core/basic-publisher/match-active-subscriptions
Requirement	/req/core/basic-publisher/match-inactive-subscriptions
Requirement	/req/core/basic-publisher/interrupt-matching
Requirement	/req/core/basic-publisher/termination
Requirement	/req/core/basic-publisher/filter-id
Requirement	/req/core/basic-publisher/valid-filter-id
Requirement	/req/core/basic-publisher/content-type
Requirement	/req/core/basic-publisher/subscribe-exceptions
Requirement	/req/core/basic-publisher/unsubscribe
Requirement	/req/core/basic-publisher/unsubscribe-halt-matching
Requirement	/req/core/basic-publisher/unsubscribe-exception-state
Requirement	/req/core/basic-publisher/unsubscribe-exceptions
Requirement	/req/core/basic-publisher/renew
Requirement	/req/core/basic-publisher/renew-update-termination-time
Requirement	/req/core/basic-publisher/renew-exception-state
Requirement	/req/core/basic-publisher/renew-exceptions

This Requirements Class specifies the basic Publish/Subscribe operations of a Publisher:

> ***Subscribe*** - allows for the creation of subscriptions against publications offered by a Publisher;
>
> ***Renew*** - allows for the renewal of a subscription on a Publisher; and
>
> ***Unsubscribe*** - allows for removal of a subscription on a Publisher.

Additionally this Requirements Class specifies Publish/Subscribe capabilities metadata that is offered in response to a *GetCapabilities* operation, whether offered as a Publish/Subscribe *GetCapabilities* as defined in Clause 9 or through a *GetCapabilities* operation defined by another OGC Web Service - such as the OGC Web Feature Service (WFS). This Requirements Class does not define a *GetCapabilities* operation, only the capabilities metadata that is offered by a Publish/Subscribe service.

All classes defined in this standard are *extensible* and may therefore contain additional parameters that can be used and/or defined by an extension.

8.1 Capabilities metadata

Capabilities metadata for a Publisher is defined in three parts: filtering capabilities (Clause 8.1.1), delivery capabilities (Clause 8.1.2), and published contents (Clause 8.1.3).

These components are each offered as the result of a *GetCapabilities* operation, either defined by the Standalone Publisher Requirements Class (Clause 9) or another OGC web service. In the latter case an existing *GetCapabilities* operation is extended with Publisher metadata.

NOTE This Standard does not specify mechanisms for incorporating Publisher capabilities metadata into other OGC web services

Publish/Subscribe conformance classes are advertised with the `Profile` section of the `ServiceIdentification` portion of Capabilities documents.

Requirement
/req/core/basic-publisher/getcapabilities-conf-class-listing
Req 2 A **Publisher** shall advertise conformance classes which are supported by the server. Each supported conformance class shall be identified by a unique value of the `Profile` property of the `ServiceIdentification` section of the capabilities document, and the Publisher shall pass all tests defined for each listed conformance class

8.1.1 FilterCapabilities

The `FilterCapabilities` data type describes the filtering-related capabilities of a Publisher. A Publisher may support specific filter languages, such as the OGC Filter Encoding Spec or XPath, that is used by a Subscriber to define a subset of messages of interest on a subscription. In order to support the creation of filtered subscription requests, the Publisher provides metadata about the filter languages it supports, if any.

The `FilterLanguage` type contains information about the filter languages that the Publisher supports for matching messages against subscriptions.

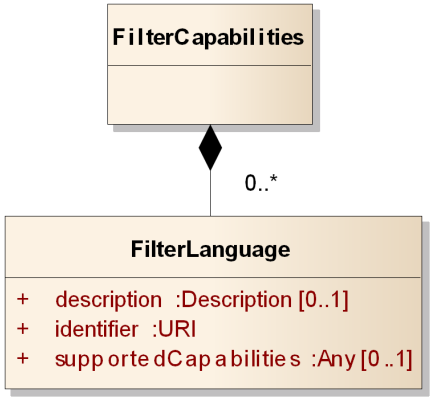


Figure 4: FilterCapabilities

Table 3: FilterLanguage properties

Name	Definition	Data type and values	Multiplicity and use
description	The abstract, title, and other human-readable descriptive information	Description [see OGC 06-121r3]	Zero to one (Optional)
identifier[A]	A unique identifier for the FilterLanguage on this Publisher	URI	One (Mandatory)
supportedCapabilities	Formal definition of the capabilities supported by the service regarding this FilterLanguage. For example, this can include the FES FilterCapabilities, supported operators/operands, filter parameter ranges, etc.	Any	Zero to one (Optional)

A. Example identifiers include "http://www.opengis.net/fes/2.0" and "http://www.opengis.net/wcs/1.1", the latter indicating support for WCS 1.1 filtering mechanisms

`FilterLanguage` identifiers are provided to the *Subscribe* operation along with the actual filter specified in that language. For example, the *Subscribe* operation may be executed with the XPath filter language identifier (e.g., "http://www.w3.org/TR/xpath") along with the specific XPath (e.g., "/messageType1") that defines the messages of interest. The OGC Filter Encoding Specification (see ISO 19143 / OGC 09-026) is an example of a filter language that may be relevant for a Publisher associated with a Web Feature Service (WFS).

`FilterLanguage` identifiers are advertised for specific publications as part of the `Publications` data type. Publishers may choose to support a different set of filter

languages for each publication. `FilterLanguage` identifiers advertised in `FilterCapabilities` need not be associated with any publication offered by the Publisher, such as cases where no publications are offered or the set of offered publications varies over time.

Requirement
/req/core/basic-publisher/getcapabilities-filtercapabilities
Req 3 A **Publisher** shall return a `FilterCapabilities` structure within its *GetCapabilities* response

Requirement
/req/core/basic-publisher/getcapabilities-unique-filter-languages
Req 4 A **Publisher** shall uniquely identify each offered `FilterLanguage` included in `FilterCapabilities`

8.1.2 DeliveryCapabilities

A Publisher must support a set of delivery methods that a Subscriber can use to define a method for delivering messages of interest on a subscription. The `DeliveryCapabilities` type describes the set of delivery methods supported by a Publisher, such as ATOM, AMQP, or SOAP over HTTP.

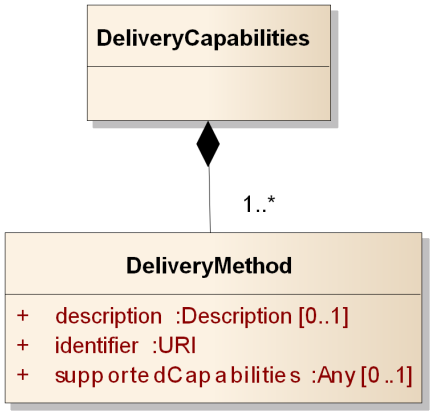


Figure 5: DeliveryCapabilities

The `DeliveryMethod` type contains information on a single method by which a Publisher can deliver messages.

Table 4: DeliveryMethod properties

Name	Definition	Data type and values	Multiplicity and use

Name	Definition	Data type and values	Multiplicity and use
description	The abstract, title, and other human-readable descriptive information	Description [see OGC 06-121r3]	Zero to one (Optional)
identifier[A]	A unique identifier for the DeliveryMethod on this Publisher	URI	One (Mandatory)
supportedCapabilities	The capabilities supported by the service regarding this `DeliveryMethod`. For may include information such as which portions of AMQP are supported and which SOAP version is supported	Any	Zero to one (Optional)

A. Examples identifiers include "http://schemas.xmlsoap.org/soap/http" and "http://www.w3.org/2005/Atom"

Requirement
/req/core/basic-publisher/getcapabilities-deliverycapabilities
Req 5 A **Publisher** shall return a `DeliveryCapabilities` structure within its *GetCapabilities* response

Requirement
/req/core/basic-publisher/getcapabilities-unique-delivery-method
Req 6 A **Publisher** shall uniquely identify each offered `DeliveryMethod` included in the `PublisherCapabilities`

8.1.3 Publications

The contents offered by a Publisher are described in the `Publications` type. The `Publications` type includes all of the offered publications that Subscribers can subscribe to. The `Publication` type contains information on an individual publication.

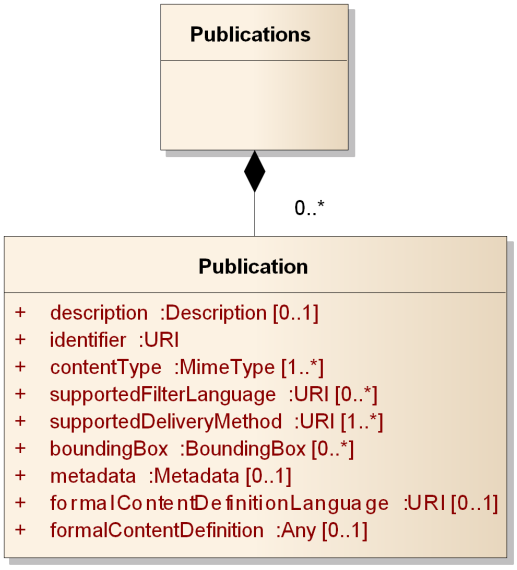


Figure 6: Publications

Table 5: Publication properties

Name	Definition	Data type and values	Multiplicity and use
description	A human-readable description	DescriptionType [see OGC 06-121r3]	Zero to one (Optional)
identifier	A unique identifier	URI	One (Mandatory)
contentType	The content type of the published data contents (e.g. “application/weather +xml”, “text/plain”)	MimeType [see OGC 06-121r3]	One to many (Mandatory)
supportedFilterLanguage	The filter languages that are supported for filtering this publication	URI	Zero to many (Optional)
supportedDeliveryMethod	The supported delivery methods for this publication	URI	One to many (Mandatory)
boundingBox	The area of interest of the published data contents. If multiple bounding boxes are included, this shall be interpreted as the union of the areas of these bounding	BoundingBox [see OGC 06-121r3]	Zero to many (Optional)

Name	Definition	Data type and values	Multiplicity and use
	boxes		
metadata	Additional metadata on this publication	Metadata [see OGC 06-121r3]	Zero to one (Optional)
formalContentDefinition Language	The identifier of the language used to describe the formal publication content definition (e.g., "http://www.w3.org/XML/Schema/1.0")	URI	Zero to many (Optional)
formalContentDefinition	A formal definition of the published data contents. This may take the form of an XML schema or other machine-readable definition for the publication	Any	Zero to many (Optional)

Publication content types specify the media/MIME type of the published content. A publication may be offered in multiple formats, such as ‘application/xml’ and ‘application/json’.

Publication bounding boxes carry the same meaning as that used for [OGC 06-121r3]. Specifically, publications may have any number of bounding boxes whose union describes the extent of the published contents.

Requirement
/req/core/basic-publisher/getcapabilities-publications
Req 7 A **Publisher** shall return a `Publications` structure within its *GetCapabilities* response

Requirement
/req/core/basic-publisher/publication-valid-filter-language
Req 8 Each `supportedFilterLanguage` of a `Publication` shall correspond to one of the `FilterLanguage` identifiers advertised in the `FilterCapabilities`

Requirement
/req/core/basic-publisher/publication-valid-delivery-method
Req 9 Each `supportedDeliveryMethod` of a `Publication` shall correspond to one of the `DeliveryMethod` identifiers advertised in the `DeliveryCapabilities`

Requirement
/req/core/basic-publisher/publication-unique-publication-id
Req 10 The `identifier` on each `Publication` shall be unique among all other `Publication` identifiers on the **Publisher**

8.2 Exception usage

In the event that a Publisher encounters an error while processing a request or receives an invalid request, it shall generate an OWS Exception indicating that an error has occurred. The form of the error response is specified by the ExceptionReport defined in Clause 8 of the OWS Common Specification [OGC 06-121r3].

Requirement
/req/core/basic-publisher/valid-exceptions
Req 11 A **Publisher** shall issue Exceptions that incorporate an ExceptionReport valid according to Clause 8 of the OWS Common Specification [OGC 06-121r3]

The mandatory `version` parameter is used to indicate the version of the service exception report, which shall be "1.0.0". The optional `language` may be used to indicate the language used. The code list for the `language` parameter is defined in [IETF RFC 4646].

Requirement
/req/core/basic-publisher/exception-version
Req 12 A **Publisher** shall raise Exceptions with the ExceptionReport `version` set to the value “1.0.0”

Individual exception messages are contained within the OWS `ExceptionText`. The mandatory `code` is used to associate an exception code with the accompanying message. The optional `locator` may be used to indicate where an exception was encountered in the request that generated the error.

Multiple exceptions may be reported in a single exception report so implementations should endeavor to report as many exceptions as necessary to clearly describe a problem.

8.3 Subscribe operation

The *Subscribe* operation is offered by the Publisher to allow Subscribers to subscribe for messages. To invoke the *Subscribe* operation, a Subscriber sends a Subscribe request message to the Publisher. The Publisher then processes the request and determines if the proposed subscription is acceptable. If so, the Publisher creates a subscription and returns a SubscribeResponse. If it is not acceptable or problems occur while processing the request, the Publisher returns an exception.

Requirement
/req/core/basic-publisher/subscribe
Req 13 The **Publisher** shall offer the *Subscribe* operation

8.3.1 Subscription

Subscribers express their interest in a specific set of messages that are available to a Publisher with a subscription. When a subscription has been submitted to a Publisher, the Publisher delivers messages that match the subscription criteria to the location defined by the subscription.

A Publisher creates a subscription when it accepts a Subscribe request. The subscription has a well-defined termination time. That time is an absolute point in time in the future.

The termination time defines the point in time at which the Publisher terminates the subscription. A subscription can be terminated at any time by explicitly requesting its termination (see *Unsubscribe* in Clause 8.4). In addition, the termination time of a subscription can be updated to a different time (see *Renew* in Clause 8.5) at a later point in time.

The subscription filter is used to express the interest in a certain set of messages. The filter itself is an expression evaluating to a boolean value. Filter languages may support logical combinations of filter expressions, such as the OGC Filter Encoding Specification (see ISO 19143 / OGC 09-026).

A subscription has the properties shown in the following figure.

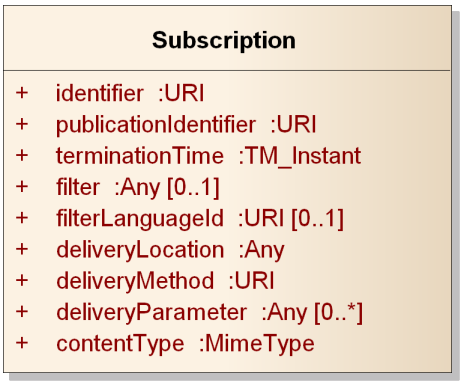


Figure 7: Subscription

Table 6: Subscription properties

Name	Definition	Data type and values	Multiplicity and use
identifier	A unique identifier for the subscription on the Publisher. Assigned by the Publisher when the subscription is created	URI	One (Mandatory)
publicationIdentifier	The identifier of the publication to which the subscription applies	URI	One (Mandatory)
terminationTime	The time at which the subscription is set to terminate	TM_Instant [see ISO/TS 19103:2005]	One (Mandatory)
filter	An expression of interest that evaluates to a Boolean value (true/false) when applied to messages published in a publication. If missing, no messages from the publication are excluded (all messages are delivered for the subscription)	Any	Zero to one (Optional)
filterLanguageId	The identifier of the filter language used to encode the filter. Must be one of the advertised supportedFilterLanguages for the publication	URI	Zero to one (Optional) Required if filter is present
deliveryLocation	The location to which messages are delivered	Any	One (Mandatory)
deliveryMethod	The method used to deliver messages. Must be one of the advertised delivery methods for the	URI	One (Mandatory)

Name	Definition	Data type and values	Multiplicity and use
	publication		
deliveryParameter	Delivery-related parameter that allows for messages to be delivered to the specified delivery location using the delivery method	Any	Zero to many (Optional)
contentType	The media type of the data contents for the subscription	MimeType [see OGC 06-121r3]	One (Mandatory)

Requirement
/req/core/basic-publisher/subscribe-assign-unique-id
Req 14 A **Publisher** shall assign a unique identifier to each created subscription

Requirement
/req/core/basic-publisher/subscribe-default-termination-time
Req 15 A **Publisher** shall assign a default `terminationTime` to created subscriptions if not provided by the Subscriber

The lifecycle of a subscription is shown in Figure 8. The matching process takes place against all active subscriptions whenever a new message is available to the Publisher.

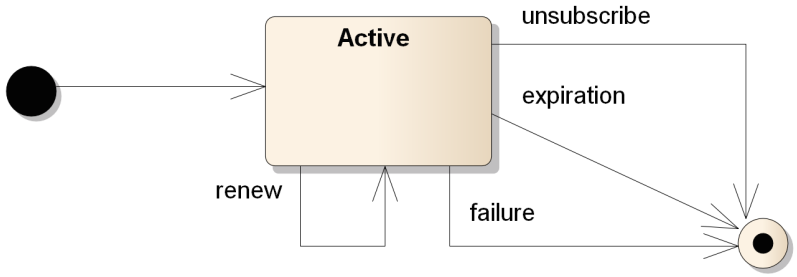


Figure 8: Subscription lifecycle

Requirement
/req/core/basic-publisher/match-active-subscriptions
Req 16 A **Publisher** shall match messages against all active subscriptions

Requirement
/req/core/basic-publisher/match-inactive-subscriptions
Req 17 A **Publisher** shall cease matching and delivery of messages when subscriptions move to an inactive or terminated state

The Publisher performs matching by evaluating the filter against the new message. If the boolean value of the filter evaluates to "true" for a message, then the message matches the subscription. If no filter is defined, all messages match for the publication defined in the subscription. When a message matches, the Publisher is responsible for delivering it to the Receiver specified in the subscription.

NOTE The Basic Publisher conformance class requires that the Publisher attempt to deliver matching messages once. This does not prevent repeated attempts to deliver the message or the use of additional mechanisms to guarantee the message delivery. The delivery method and/or transport mechanism may provide additional delivery guarantees for messages.

The Publisher starts matching new messages against a subscription once that subscription has been created. This can happen at any time after it received the request to create that subscription, and must happen before a SubscribeResponse is returned. Therefore, the Receiver specified for a new subscription should be ready to receive incoming messages before the Subscriber has received the SubscribeResponse.

Likewise, the Publisher stops matching new messages against a subscription once it has been terminated. Message matching and message delivery are independent; message matching will cease after termination, but messages that have previously matched may still be delivered.

Requirement
/req/core/basic-publisher/interrupt-matching
Req 18 When a **Publisher** terminates a subscription it shall interrupt all unfinished matching processes for this subscription

Requirement
/req/core/basic-publisher/termination
Req 19 A **Publisher** shall terminate a subscription when its termination time is reached

8.3.2 Request

A Subscriber sends a Subscribe request to the Publisher in order to create a new subscription.

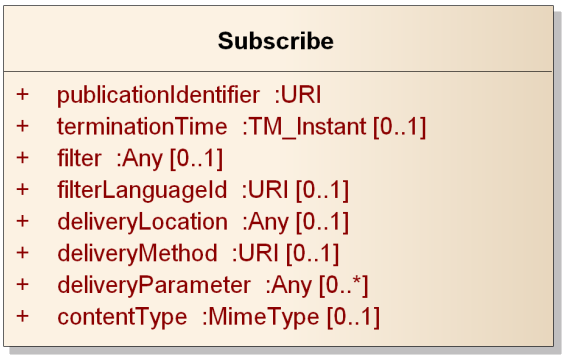


Figure 9: Subscribe request

Table 7: Subscribe request properties

Name	Definition	Data type and values	Multiplicity and use
publicationIdentifier	The publication to which the subscription applies	URI	One (Mandatory)
terminationTime	The requested termination time for the subscription. Must be in the future	TimeInstant [see ISO/TS 19103:2005]	Zero to one (Optional)
filter	The filter to be applied to the publication for the subscription.	URI	Zero to one (Optional)
filterLanguageId	The identifier of the filter language used to encode the filter. Must be one of the advertised supportedFilterLanguages for the publication	URI	Zero to one (Optional) Required if filter is present
deliveryLocation	The location where information will be delivered	Any	Zero to one (Optional)
deliveryMethod	The method used to deliver messages for this subscription. Must be from the list of advertised delivery methods for the publication	URI	Zero to one (Optional)
deliveryParameter	Delivery-related parameter that allows for messages to be delivered to the specified delivery location using the specified delivery method	Any	Zero to many (Optional)

Name	Definition	Data type and values	Multiplicity and use
contentType	The content type of the data contents for the subscription. Must be from the list of advertised content types for the publication	MimeType [see OGC 06-121r3]	Zero to one (Optional) Required if the publication is available in more than one content types

The `deliveryLocation` parameter defines the system endpoint where the Publisher should send messages that match the filter criteria of the requested subscription. The `deliveryLocation` parameter is optional, as in some cases the Publisher may assign a `deliveryLocation` to the subscription rather than accept a `deliveryLocation` from a Subscriber. Extensions to the Basic Publisher conformance class (e.g. bindings) may specialize the use of this parameter.

For example, in WS-BaseNotification[2] it is mandatory to specify an endpoint in a Subscribe request. In a RESTful binding with ATOM-based delivery, the Publisher might create an ATOM feed to which all messages matching a given subscription are sent. In the latter case, the Publisher determines the delivery location and will raise an Exception if one is provided in the Subscribe request.

When a Subscribe request includes a `deliveryMethod` it must be among those listed in the `DeliveryCapabilities` section of the `PublisherCapabilities` document.

The `terminationTime` parameter defines the requested time when a subscription terminates. That time must be an absolute time in the future. The Publisher may choose to reject the requested termination time with an Exception.

The `filter` parameter in a Subscribe request defines which messages match the requested subscription, i.e., it defines the subset of messages available in a publication that are of interest to the Subscriber.

The `filterLanguageId` parameter defines the language using for encoding the Filter in the Subscribe request. The supported filter languages are advertised in the supportedFilterLanguage of each Publication, and in the FilterCapabilities of the Publisher.

Requirement
/req/core/basic-publisher/filter-id

[2] OASIS WS-BaseNotification, *Web Services Base Notification*, OASIS Standard 1.3 (1 October 2006).

Req 20 A **Publisher** shall raise an Exception if the `Subscribe` request includes a `filter`, but does not include a `filterLanguageId`

Requirement
/req/core/basic-publisher/valid-filter-id
Req 21 A **Publisher** shall raise an Exception if the `Subscribe` request includes a `filterLanguageId` that does not correspond to a `supportedFilterLanguage` for the publication

The `contentType` parameter defines the format of the data contents for the subscription. Must be from the list of content types for the publication, advertised in the Publications of the service instance. It can be omitted if there is only one content type advertised for the Publication.

Requirement
/req/core/basic-publisher/content-type
Req 22 A **Publisher** shall raise an Exception if the `Subscribe` request does not include a `contentType` and the offered Publication advertises multiple content types

8.3.3 Response

If the request is accepted and no Exception is raised, the Publisher creates a new subscription with information from the Subscribe request, determines any other information not provided by the Subscriber (such as delivery location, termination, etc.) and returns a SubscribeResponse. The SubscribeResponse includes the complete and valid subscription that was created.

Figure 10: Subscribe response

Table 8: Subscribe response properties

Name	Definition	Data type and values	Multiplicity and use
subscription	The newly created subscription	Subscription	One (Mandatory)

8.3.4 Exceptions

Exceptions raised as a result of the *Subscribe* operation are described below.

Requirement
/req/core/basic-publisher/subscribe-exceptions
Req 23 A **Publisher** shall raise Exceptions in accordance with Table 9 when executing the *Subscribe* operation

Table 9: Subscribe Exceptions

Exception Code	Description	Locator Values
InvalidPublicationIdentifier	The referenced publication is unknown to the Publisher	Comma-separated list of invalid publication identifiers
TerminationUnacceptable	The requested termination time is not acceptable for the Publisher	Comma-separated list of unacceptable termination times
PastTermination	The requested termination time is in the past	Comma-separated list of unacceptable termination times
InvalidDeliveryMethod	The DeliveryMethod identifier is unknown to this Publisher	Comma-separated list of unacceptable DeliveryMethod identifiers
InvalidFilter	The requested filter is not valid for the subscription or Publisher	XPath to invalid request filter section, or other relevant request location information
MissingParameterValue	Operation request does not include a parameter value, and this server did not declare a default value for that parameter	Name of missing parameter
InvalidParameterValue	Operation request contains an invalid parameter value	Name of parameter with invalid value
NoApplicableCode	No other exceptionCode specified by this service	None, omit “locator”

	and server applies to this exception	parameter

8.4 Unsubscribe operation

The *Unsubscribe* operation allows Subscribers to terminate a subscription. To invoke the *Unsubscribe* operation, a client sends an Unsubscribe request message to the Publisher. The Publisher then processes the request and determines if it is acceptable. If so, the Publisher terminates the subscription identified in the request and returns an *Unsubscribe* operation response. If it is not acceptable or problems occur while processing the request, the Publisher returns an exception.

Requirement
/req/core/basic-publisher/unsubscribe
Req 24 The **Publisher** shall offer the *Unsubscribe* operation

8.4.1 Request

The Unsubscribe request identifies the subscription that the client wants to terminate, as shown in Figure 11.

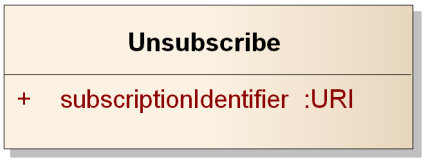


Figure 11: Unsubscribe request

Table 10: Unsubscribe request properties

Name	Definition	Data type and values	Multiplicity and use
subscriptionIdentifier	The identifier of the subscription to be terminated	URI	One (Mandatory)

8.4.2 Response

If the request is accepted and no Exception is raised, the Publisher terminates the subscription and ceases message matching. Undelivered messages that matched before termination may be delivered after termination.

Requirement
/req/core/basic-publisher/unsubscribe-halt-matching

Req 25 A **Publisher** shall cease subscription matching for the subscription identified in the `Unsubscribe` request

Figure 12: Unsubscribe response

8.4.3 Exceptions

Exceptions raised as a result of the *Unsubscribe* operation are described below. Unsuccessful *Unsubscribe* requests do not change any subscription state.

Requirement
/req/core/basic-publisher/unsubscribe-exception-state
Req 26 A **Publisher** shall leave subscription state unchanged when an Exception occurs during the *Unsubscribe* operation

Requirement
/req/core/basic-publisher/unsubscribe-exceptions
Req 27 A **Publisher** shall raise Exceptions in accordance with Table 11 when executing the *Unsubscribe* operation

Table 11: Unsubscribe Exceptions

Exception Code	Description	Locator Values
InvalidSubscriptionIdentifier	The requested subscription is unknown to the Publisher.	Comma-separated list of invalid subscription identifiers
NoApplicableCode	No other exceptionCode specified by this service and server applies to this exception	None, omit “locator” parameter

8.5 Renew operation

The *Renew* operation allows subscribers to set the termination time on a subscription to a new time. This new time may be before or after the current termination time.

NOTE A subscription that has already been terminated (either automatically expired or explicitly via the *Unsubscribe* operation) cannot be renewed.

To invoke the *Renew* operation, a client sends a Renew request message to the Publisher. The Publisher then processes the request and determines if the proposed termination time is acceptable.

If so, the Publisher updates the subscription and returns a RenewResponse. If it is not acceptable or problems occur while processing the request, the Publisher returns an exception.

Requirement
/req/core/basic-publisher/renew
Req 28 The **Publisher** shall offer the *Renew* operation

8.5.1 Request

A client sends a Renew request to the Publisher in order to update the termination time of an existing subscription.

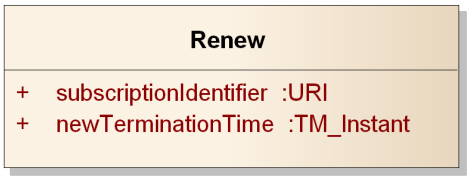


Figure 13: Renew request

Table 12: Renew request properties

Name	Definition	Data type and values	Multiplicity and use
subscriptionIdentifier	Unique identifier for the subscription	URI	One (Mandatory)
newTerminationTime	The new date and time when the identified subscription is requested to terminate. The new termination time cannot be in the past	TM_Instant [see ISO/TS 19103:2005]	One (Mandatory)

Requirement
/req/core/basic-publisher/renew-update-termination-time
Req 29 A **Publisher** shall update the `terminationTime` on the identified subscription to be the value of `newTerminationTime` provided as part of a successful *Renew* operation

8.5.2 Response

If the request is accepted and no Exception is raised, the Publisher accepts the request, updates the termination time of the subscription, and returns a RenewResponse.

Figure 14: Renew response

NOTE This Requirements Class does not define any content to be returned in a RenewResponse. Extensions may include more information, such as further information about the updated subscription.

8.5.3 Exceptions

Exceptions raised as a result of the *Renew* operation are described below. Unsuccessful *Renew* requests do not change any subscription state, in particular termination time.

Requirement
/req/core/basic-publisher/renew-exception-state
Req 30 A **Publisher** shall leave subscription state unchanged when an Exception occurs during the *Renew* operation

Requirement
/req/core/basic-publisher/renew-exceptions
Req 31 A **Publisher** shall raise Exceptions in accordance with Table 13 when executing the *Renew* operation

Table 13: Renew Exceptions

Exception Code	Description	Locator Values
InvalidSubscriptionIdentifier	The requested subscription is	Comma-separated list of invalid

	unknown to the Publisher.	subscription identifiers
TerminationUnacceptable	The requested termination time is not acceptable for the Publisher.	Comma-separated list of unacceptable termination times
PastTermination	The requested termination time is in the past.	Comma-separated list of unacceptable termination times
MissingParameterValue	Operation request does not include a parameter value, and this server did not declare a default value for that parameter	Name of missing parameter
InvalidParameterValue	Operation request contains an invalid parameter value	Name of parameter with invalid value
NoApplicableCode	No other exceptionCode specified by this service and server applies to this exception	None, omit "locator" parameter

9. Requirements Class – Standalone Publisher extends Basic Publisher

Requirements Class	
http://www.opengis.net/spec/pubsub/1.0/req/core/standalone-publisher	
Target type	Publisher
Dependency	**http://www.opengis.net/spec/pubsub/1.0/req/core/basic-publisher**
Dependency	**http://www.opengis.net/doc/IS/OWS/1.1/clause/7**
Requirement	/req/core/standalone-publisher/getcapabilities
Requirement	/req/core/standalone-publisher/getsubscription
Requirement	/req/core/standalone-publisher/getsubscription-all-subscriptions
Requirement	/req/core/standalone-publisher/getsubscription-exceptions

This Requirements Class enables standalone publishing, wherein Publishers offer metadata concerning Publisher capabilities. This Requirements Class requires that a Publisher implement two operations:

GetCapabilities - allows for the discovery of Publisher metadata, including offered publications, service capabilities, and service provider information; and

GetSubscription - allows for the retrieval of subscription information.

The Standalone Publisher includes a Publish/Subscribe *GetCapabilities* operation extended from OWS Common [OGC 06-121r3] that integrates `FilterCapabilities`, `DeliveryCapabilities`, and `Publications` metadata as specified in Clause 8.1.

9.1 GetCapabilities operation

The *GetCapabilities* operation allows clients to retrieve the capabilities metadata (also called the "capabilities document") of a Publisher. This includes supported functionality (e.g. filter functionality, or functionality defined in other Publish/Subscribe Requirements Classes) requirements for use (e.g. that Subscribers authenticate themselves to the service) and content information (e.g., formal description of published contents).

The Publish/Subscribe `GetCapabilities` type derives from the OWS Common `GetCapabilities` type described in Table 3 of [OGC 06-121r3].

Requirement
/req/core/standalone-publisher/getcapabilities
Req 32 The **Publisher** shall offer the *GetCapabilities* operation

9.1.1 Request

The Publish/Subscribe GetCapabilities request extends the OWS Common `GetCapabilitiesType` with limited information.

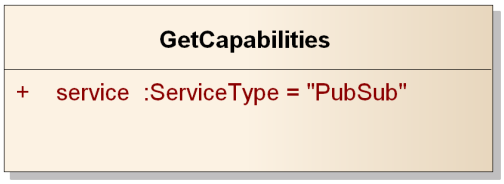


Figure 15: GetCapabilities request

Table 14: GetCapabilities properties

Name	Definition	Data type and values	Multiplicity and use
service	The service type	ServiceType [see OGC 06-121r3]	One (Mandatory) Always the fixed value "PubSub"

9.1.2 Response

If the request is accepted and no Exception is raised, the Publisher returns a PublisherCapabilities. PublisherCapabilities is an extension of the OWS Common

Capabilities document that adds filter capabilities, delivery capabilities, and publications/contents metadata. These additional portions of the Capabilities document are specified in the FilterCapabilities, DeliveryCapabilities, and Publication clauses in Clause 8.1.

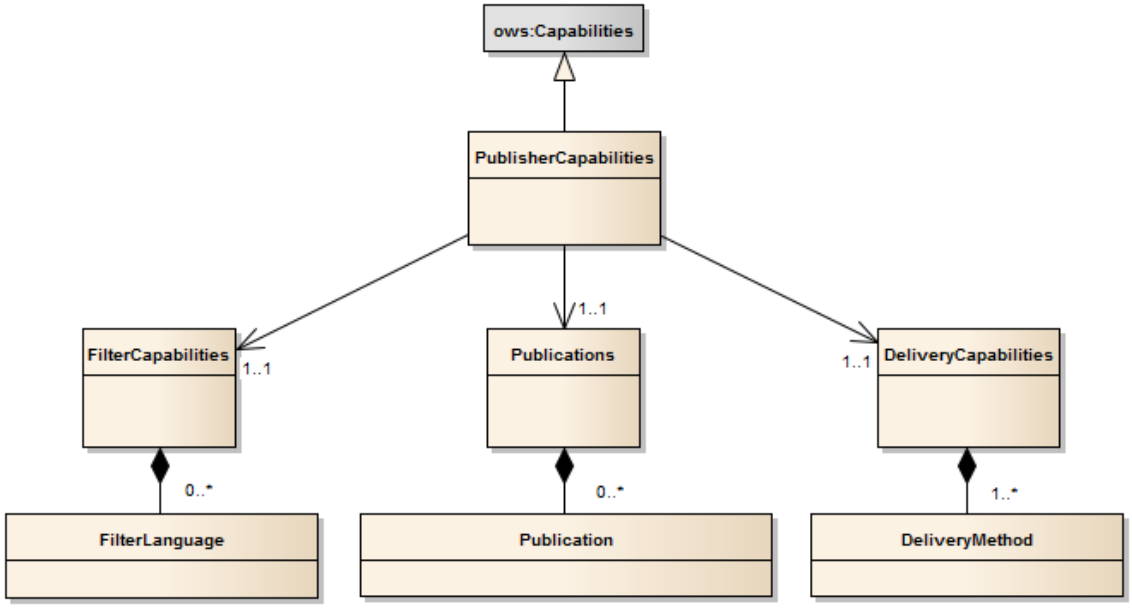


Figure 16: PublisherCapabilities

9.1.3 Exceptions

Exception behavior for the *GetCapabilities* operation is defined in Table 8 and Clause 8 of the OWS Common Specification [OGC 06-121r3].

9.2 GetSubscription operation

A Subscriber invokes the *GetSubscription* operation in order to retrieve information on one or more subscriptions.

NOTE Terminated subscriptions are not returned. Publishers may return an empty list if all the requested subscriptions have expired or were explicitly terminated via the *Unsubscribe* operation.

To invoke the *GetSubscription* operation, a client sends a *GetSubscription* request message to the Publisher. The Publisher then processes the request and determines if it is acceptable. If so, the Publisher returns a *GetSubscription* operation response. If it is not acceptable or problems occur while processing the request, the Publisher returns an exception.

Requirement
/req/core/standalone-publisher/getsubscription
Req 33 The **Publisher** shall offer the *GetSubscription* operation

9.2.1 Request

A client sends a GetSubscription request to the Publisher in order to retrieve the active subscriptions. The Publisher needs to determine if the request is acceptable. In order to do so, the Publisher performs syntactic as well as semantic checks regarding the request.

Figure 17: GetSubscription request

Table 15: GetSubscription request properties

Name	Definition	Data type and values	Multiplicity and use
subscriptionIdentifier	The identifier of the subscription(s) to be described. If missing, all subscriptions are requested	URI	Zero to many (Optional)

9.2.2 Response

If the request is accepted and no Exception is raised, the Publisher returns the requested active subscriptions in a GetSubscriptionResponse. If no subscription identifiers are specified in the request, the Publisher returns all active subscriptions (see the state diagram in Figure 8).

Requirement
/req/core/standalone-publisher/getsubscription-all-subscriptions
Req 34 A **Publisher** shall return a `GetSubscriptionResponse` with all the active subscriptions when no subscription identifiers are provided as part of the `GetSubscription` request

Figure 18: GetSubscription response

Table 16: GetSubscription response properties

Name	Definition	Data type and values	Multiplicity and use
subscription	The requested subscription description	Subscription	One (Mandatory)

9.2.3 Exceptions

Exceptions raised as a result of the *GetSubscription* operation are described below.

Requirement
/req/core/standalone-publisher/getsubscription-exceptions
Req 35 A **Publisher** shall raise Exceptions in accordance with Table 17 when executing the *GetSubscription* operation

Table 17: GetSubscription Exceptions

Exception Code	Description	Locator Values
InvalidSubscriptionIdentifier	The requested subscription is unknown to the Publisher.	Comma-separated list of invalid subscription identifiers
NoApplicableCode	No other exceptionCode specified by this service and server applies to this exception	None, omit "locator" parameter

10. Requirements Class – Pausable Publisher extends Basic Publisher

Requirements Class	
http://www.opengis.net/spec/pubsub/1.0/req/core/pausable-publisher	
Target type	Publisher
Dependency	**http://www.opengis.net/spec/pubsub/1.0/req/core/basic-publisher**
Requirement	/req/core/pausable-publisher/pause
Requirement	/req/core/pausable-publisher/pause-halt-delivery
Requirement	/req/core/pausable-publisher/pause-unchanged-paused-subscription
Requirement	/req/core/pausable-publisher/pause-exceptions
Requirement	/req/core/pausable-publisher/resume
Requirement	/req/core/pausable-publisher/resume-resume-delivery
Requirement	/req/core/pausable-publisher/resume-unchanged-active-subscription

Requirement	/req/core/pausable-publisher/resume-exceptions

The Pausable Publisher Requirements Class enables subscription pausing, wherein Publishers may be directed to pause and resume message delivery for a subscription. Message matching for a paused subscription continues unchanged, but matching messages are not delivered until the subscription is resumed. This Requirements Class requires that a Publisher implement two operations:

> ***Pause*** - allows for the pausing of an unpaused subscription, which pauses message delivery; and
>
> ***Resume*** - allows for the resumption of a paused subscription, which resumes message delivery.

NOTE Pausing and resuming of subscriptions is independent of subscription termination. Paused subscriptions are subject to subscription termination (through expiry or other means) in an identical manner to active subscriptions.

When a paused subscription is resumed, the Publisher delivers all matched but undelivered messages for the subscription. Message delivery (as well as message matching) may also be halted with the *Unsubscribe* and *Subscribe* operations, except that matching messages that arrive between the *Unsubscribe* and the new *Subscribe* call will be lost.

NOTE Pausable Publishers deliver messages for resumed subscriptions on a best-effort basis. Not all messages are guaranteed to be delivered upon resumption of the subscription. Storage limitations and other practical considerations may prevent the delivery of some messages that arrived while a subscription is paused. A future revision of this Standard may specify delivery guarantees, but none are currently made.

In cases of asynchronous message delivery, some messages may be in transit when the *Pause* operation is executed. When this occurs, message delivery may continue after the *Pause* operation is successfully completed and the Publisher has ceased initiating the delivery of messages.

The valid subscription states and transitions between states are shown in Figure 19. Execution of the *Pause* operation is equivalent to executing a *pause* state transition. Similarly, execution of the *Resume* operation is equivalent to executing a *resume* state transition.

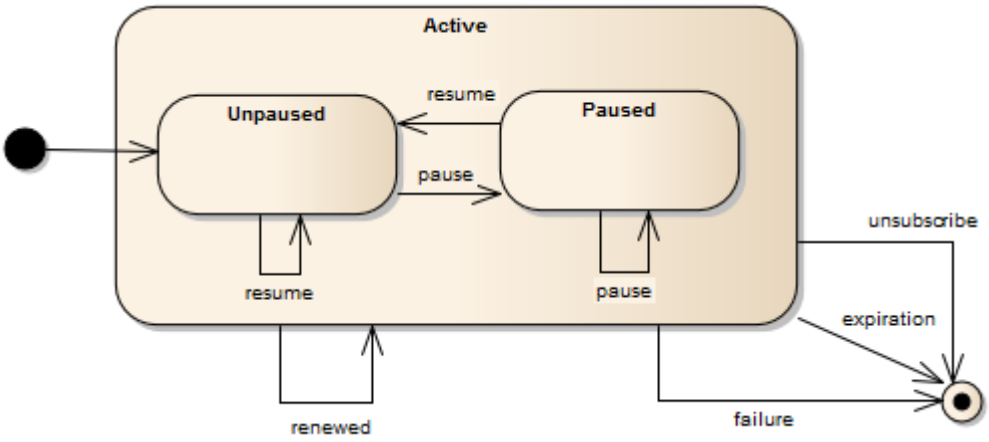


Figure 19: Subscription Pausing state

Paused subscriptions only differ from active subscriptions in terms of message delivery. Therefore, they are valid targets and valid responses from all operations that include active subscriptions, such as *GetSubscription* responses.

10.1 Pause operation

10.1.1 Request

The Pause request includes a single property that identifies the subscription to be paused.

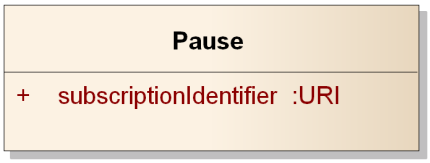


Figure 20: Pause request

Table 18: Pause properties

Name	Definition	Data type and values	Multiplicity and use
subscriptionIdentifier	The identifier of the subscription to be paused	URI	One (Mandatory)

Requirement
/req/core/pausable-publisher/pause
Req 36 A **Publisher** shall offer the *Pause* operation

Requirement
/req/core/pausable-publisher/pause-halt-delivery
Req 37 A **Publisher** shall cease the initiation of message delivery processes for the subscription when the *Pause* operation is successfully completed. Message delivery processes already underway continue unchanged

Requirement
/req/core/pausable-publisher/pause-unchanged-paused-subscription
Req 38 When a **Publisher** executes the *Pause* operation on a subscription that is already paused, no change in subscription matching or subscription state will be made

10.1.2 Response

If the request is accepted and no Exception is raised, the Publisher pauses the subscription and returns a PauseResponse. The PauseResponse is returned when the relevant subscription has been successfully paused.

Figure 21: PauseResponse

10.1.3 Exceptions

Exceptions raised as a result of the *Pause* operation are described below. Unsuccessful *Pause* requests do not change any subscription state.

Requirement
/req/core/pausable-publisher/pause-exceptions
Req 39 A **Publisher** shall raise Exceptions in accordance with Table 19 when executing the *Pause* operation

Table 19: Pause Exceptions

Exception Code	Description	Locator Values
InvalidSubscriptionIdentifier	The requested subscription is unknown to the Publisher.	Comma-separated list of invalid subscription identifiers

NoApplicableCode	No other exceptionCode specified by this service and server applies to this exception	None, omit "locator" parameter

10.2 Resume operation

10.2.1 Request

The Resume request includes a single property that identifies the subscription to be resumed. All messages that have matched for a subscription but have not yet been delivered will be delivered when the *Resume* operation is completed.

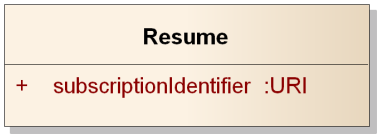


Figure 22: Resume request

Table 20: Resume properties

Name	Definition	Data type and values	Multiplicity and use
subscriptionIdentifier	The identifier of the subscription to be resumed	URI	One (Mandatory)

Requirement
/req/core/pausable-publisher/resume
Req 40 A **Publisher** shall offer the *Resume* operation

Requirement
/req/core/pausable-publisher/resume-resume-delivery
Req 41 A **Publisher** shall re-start all message delivery processes for the appropriate subscription when the *Resume* operation is successfully completed

Requirement
/req/core/pausable-publisher/resume-unchanged-active-subscription
Req 42 When a **Publisher** executes the *Resume* operation on a subscription that is already active, no change in subscription matching or subscription state will be made

10.2.2 Response

If the request is accepted and no Exception is raised, the Publisher resumes the subscription and returns a ResumeResponse. The ResumeResponse is returned when the relevant subscription has been successfully resumed.

Figure 23: ResumeResponse

10.2.3 Exceptions

Exceptions raised as a result of the *Resume* operation are described below. Unsuccessful *Resume* requests do not change subscription state.

Requirement
/req/core/pausable-publisher/resume-exceptions
Req 43 A **Publisher** shall raise Exceptions in accordance with Table 21 when executing the *Resume* operation

Table 21: Resume Exceptions

Exception Code	Description	Locator Values
InvalidSubscriptionIdentifier	The requested subscription is unknown to the Publisher.	Comma-separated list of invalid subscription identifiers
NoApplicableCode	No other exceptionCode specified by this service and server applies to this exception	None, omit “locator” parameter

11. Requirements Class – Message Batching Publisher extends Basic Publisher

Requirements Class	
http://www.opengis.net/spec/pubsub/1.0/req/core/message-batching-publisher	
Target type	Publisher
Dependency	**http://www.opengis.net/spec/pubsub/1.0/req/core/basic-publisher**

Requirement	/req/core/message-batching-publisher/subscribe-message-batching
Requirement	/req/core/message-batching-publisher/withheld-delivery
Requirement	/req/core/message-batching-publisher/reset-batching
Requirement	/req/core/message-batching-publisher/subscription-termination
Requirement	/req/core/message-batching-publisher/pausing
Requirement	/req/core/message-batching-publisher/subscribe-exceptions

The Message Batching Publisher Requirements Class specifies capabilities for Subscribers to communicate message-batching directives. Message batching allows Subscribers to specify desired message delivery at a different rate than the messages are natively generated. This includes cases where frequent, small messages are published that can be consumed more efficiently in batches by the Receiver.

11.1 Batching criteria

Message-batching criteria are optionally set by providing a BatchingCriteria object to the *Subscribe* operation. The batching criteria supported include:

- Time period (e.g. every 5 minutes, every hour); and
- Batch size (e.g. every 20 messages, every 150 messages).

More than one criterion may be supplied at once. When multiple criteria are supplied, the first criterion that applies triggers the delivery of the batch.

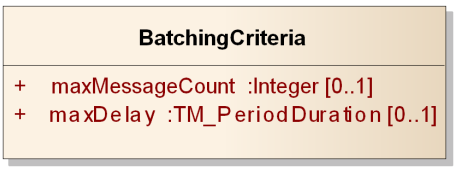


Figure 24: BatchingCriteria

Table 22: BatchingCriteria properties

Name	Definition	Data type and values	Multiplicity and use
maxDelay	The maximum amount of time that may pass between the delivery of message batches	TM_PeriodDuration [see ISO/TS 19103:2005]	Zero to one (Optional)
maxMessageCount	The maximum number of messages accumulated before a batch is delivered	Integer - greater than 0	Zero to one (Optional)

Messages matching a BatchingCriteria are accumulated and withheld by the Publisher. When either the number of messages equals `maxMessageCount` or the time passed since the last delivery exceeds `maxDelay`, all the withheld messages are delivered.

Subscription termination will trigger the batch delivery of any withheld (undelivered) messages for that subscription.

If the `maxDelay` period is reached without any withheld messages to deliver, no message delivery will take place. No message batch will ever be delivered with more messages than `maxMessageCount`.

For example, in the case where a Subscriber submitted a subscription via the *Subscribe* operation with batching criteria indicating a `maxDelay` of 10 minutes and a `maxMessageCount` of 30 the Publisher would withhold the messages for this publication until 30 matching messages become available or 10 minutes pass, whichever occurs first.

Requirement
/req/core/message-batching-publisher/subscribe-message-batching
Req 44 A **Publisher** shall accept `MessageBatchingCriteria` with other subscription criteria on the *Subscribe* operation

Requirement
/req/core/message-batching-publisher/withheld-delivery
Req 45 A **Publisher** shall withhold delivery of messages until any of the subscription message batching criteria are met, at which time all withheld messages will be delivered together as a batch

Requirement
/req/core/message-batching-publisher/reset-batching
Req 46 A **Publisher** shall reset tracking information (e.g., last batch delivery time and number of withheld messages) for subscription message batching criteria whenever a message batch is delivered

Requirement
/req/core/message-batching-publisher/subscription-termination
Req 47 A **Publisher** shall deliver withheld messages in a batch when a subscription is terminated

Requirement
/req/core/message-batching-publisher/pausing
Req 48 A **Publisher** shall deliver withheld messages in a batch when a subscription is paused as described in the Pausable Publisher Requirements Class (see Clause 10)

NOTE The use of this conformance class in conjunction with the Heartbeat Publisher conformance class can result in batched heartbeats. Subscribers are recommended to exercise caution when both message batching and heartbeats are used in conjunction.

11.2 Exceptions

Exceptions raised as a result of the *Subscribe* operation are described below.

Requirement
/req/core/message-batching-publisher/subscribe-exceptions
Req 49 A **Publisher** shall raise Exceptions in accordance with Table 23 when executing the *Subscribe* operation, in addition to those specified in Section 8.3.4

Table 23: Message Batching Subscribe Exceptions

Exception Code	Description	Locator Values
MissingParameterValue	Operation request does not include a parameter value, and this server did not declare a default value for that parameter	Name of missing parameter
InvalidParameterValue	Operation request contains an invalid parameter value	Name of parameter with invalid value
NoApplicableCode	No other exceptionCode specified by this service and server applies to this exception	None, omit "locator" parameter

12. Requirements Class – Heartbeat Publisher extends Basic Publisher

Requirements Class	
http://www.opengis.net/spec/pubsub/1.0/req/core/heartbeat-publisher	
Target type	Publisher
Dependency	**http://www.opengis.net/spec/pubsub/1.0/req/core/basic-publisher**
Requirement	/req/core/heartbeat-publisher/subscribe-heartbeat
Requirement	/req/core/heartbeat-publisher/publish-heartbeat
Requirement	/req/core/heartbeat-publisher/pausing
Requirement	/req/core/heartbeat-publisher/subscribe-exceptions

The Heartbeat Publisher Requirements Class specifies capabilities to ensure that the Receiver is sent notifications on a regular basis. This Requirements Class enables Receivers to detect outages due to network failures, Publisher failures, or other issues preventing communication of messages for an active subscription. This Requirements Class addresses end-to-end subscription delivery life, and as such is a capability that is most useful when the original Publisher or Sender is capable of issuing heartbeats.

12.1 Heartbeat criteria

Subscribers may optionally specify a rate for heartbeat messages to be issued by the Publisher.

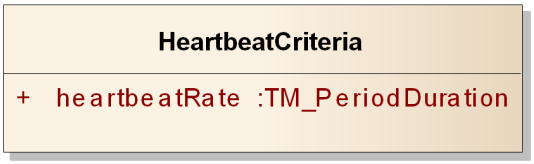


Figure 25: HeartbeatCriteria

Table 24: HeartbeatCriteria properties

Name	Definition	Data type and values	Multiplicity and use
heartbeatRate	The rate at which heartbeat messages should be sent for this subscription	TM_PeriodDuration [see ISO/TS 19103:2005]	One (Mandatory)

Requirement
/req/core/heartbeat-publisher/subscribe-heartbeat
Req 50 A **Publisher** shall accept `HeartbeatCriteria` with other subscription criteria on the *Subscribe* operation

HeartbeatMessages are messages sent on a regular period, each of which includes the heartbeat issuance time from the Publisher. The arrival of these messages indicates that the Publisher was able to deliver messages as of that time, as observed by the Publisher's clock when it initiated the delivery of the HeartbeatMessage.

NOTE HeartbeatMessages may be represented as a header entry, unique message, or other representation depending on the delivery method.

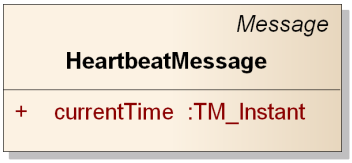


Figure 26: Heartbeat Message

Table 25: Heartbeat Message properties

Name	Definition	Data type and values	Multiplicity and use
currentTime	The time of issuance of the heartbeat message	TM_Instant [see ISO/TS 19103:2005]	One (Mandatory)

Requirement
/req/core/heartbeat-publisher/publish-heartbeat
Req 51 A **Publisher** shall send regular HeartbeatMessages for each subscription as specified by each subscription's HeartbeatCriteria

Requirement
/req/core/heartbeat-publisher/pausing
Req 52 A **Publisher** shall cease sending HeartbeatMessages for a subscription when it is paused as described in the Pausable Publisher Requirements Class (see Clause 10)

12.2 Exceptions

Exceptions raised as a result of the *Subscribe* operation are described below.

Requirement
/req/core/heartbeat-publisher/subscribe-exceptions
Req 53 A **Publisher** shall raise Exceptions in accordance with Table 26 when executing the *Subscribe* operation, in addition to those specified in Section 8.3.4

Table 26: Heartbeat Subscribe Exceptions

Exception Code	Description	Locator Values
MissingParameterValue	Operation request does not include a parameter value, and this server did not declare a default value for that parameter	Name of missing parameter
InvalidParameterValue	Operation request contains an invalid parameter value	Name of parameter with invalid value
NoApplicableCode	No other exceptionCode specified by this service and server applies to this exception	None, omit "locator" parameter

13. Requirements Class – Brokering Publisher extends Standalone Publisher

Requirements Class	
http://www.opengis.net/spec/pubsub/1.0/req/core/brokering-publisher	
Target type	Publisher
Dependency	http://www.opengis.net/spec/pubsub/1.0/req/core/standalone-publisher
Requirement	/req/core/brokering-publisher/registerpublisher
Requirement	/req/core/brokering-publisher/registerpublisher-connect
Requirement	/req/core/brokering-publisher/registerpublisher-exceptions
Requirement	/req/core/brokering-publisher/removepublisher
Requirement	/req/core/brokering-publisher/removepublisher-exceptions
Requirement	/req/core/brokering-publisher/getcapabilities-registered-publishers

A Brokering Publisher, or broker, is an intermediary between Subscribers and other Publishers which have been previously registered with the broker. The broker is not the original producer of messages, but only acts as a message middleman, re-publishing messages received from other Publishers and decoupling them from their Subscribers. This Requirements Class requires that a Publisher implement the operations:

RegisterPublisher - allows the connection of an external Publisher to the broker; and

RemovePublisher - allows the disconnection of a Publisher from the broker.

A broker is a distinct third party that acts as a communication intermediary between the source and the target of a communication, mediating their interfaces and in some cases

adding new behavior. A broker may aggregate the messages into different publications, may provide the same publications with a with different delivery methods, or otherwise process the messages (e.g. converting their format). A broker may also provide advanced messaging features such as load balancing. However, a broker should not advertise capabilities on behalf of another Publisher, unless the latter provides identical guarantees (e.g. heartbeat capabilities).

Examples of Brokering Publisher applications include the following.

> **Publisher Aggregation –** a broker subscribes to several Publishers and relays their publications (without modification) to interested Subscribers, acting like a Proxy to multiple Publishers. Optionally, the broker may adapt the service interface (binding) of the aggregated Publishers.
>
> **Publication Aggregation –** a broker receives messages generated by several Publishers (e.g. dumb sensors) and publishes them to the interested Subscribers as a single publication.

This Requirement Class does not mandate any specific behavior to be implemented by a Brokering Publisher, in particular as regards the support to Delivery Capabilities, Filtering Capabilities, and Publications of connected Publishers. Implementations of this Requirement Class are free to interact with the connected Publishers as appropriate for their specific application. Interactions may include subscribing, loading and/or proxying capabilities documents, or other behavior. Future extensions to this Requirement Class may standardize the behavior of Brokering Publishers in specific application scenarios.

NOTE WS-Notification has a similar abstraction, the NotificationBroker, as defined in WS-BrokeredNotification[3].

Figure 27 illustrates the typical broker interaction. The broker behaves like a Basic Publisher in the core Publish/Subscribe. However, the broker relays messages received from an external Publisher, which is assumed to have been previously registered.

[3] OASIS WS-BrokeredNotification, *Web Services Brokered Notification*, OASIS Standard 1.3 (1 October 2006).

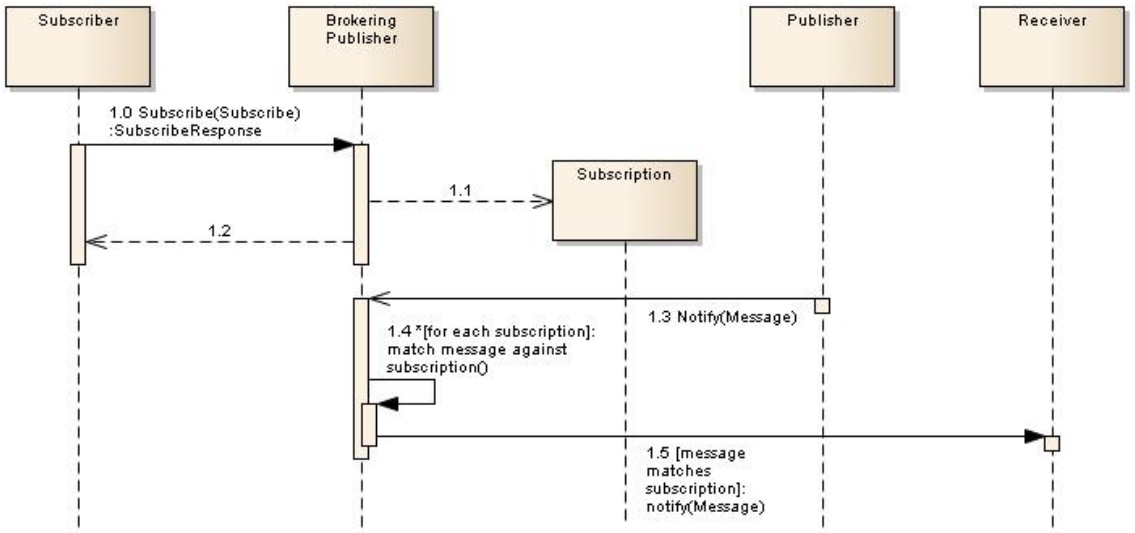


Figure 27: Broker workflow

The broker provides additional functionalities that support the management of brokered Publishers. The operations described herein allow external Publishers to be registered to and removed from the broker.

13.1 RegisterPublisher operation

The *RegisterPublisher* operation is used to connect the broker to a given Publisher. As a result of this operation, the broker capabilities may change (e.g. exposing part or all of the `FilterCapabilities`, `DeliveryCapabilities`, and `Publications` of the brokered Publisher); the specification of such changes is out of the scope of this Requirements Class.

Requirement
/req/core/brokering-publisher/registerpublisher
Req 54 A **Publisher** shall offer the *RegisterPublisher* operation

13.1.1 Request

The following diagram and table list the request parameters for the *RegisterPublisher* operation:

Figure 28: RegisterPublisher request

Table 27: RegisterPublisher properties

Name	Definition	Data type and values	Multiplicity and use
capabilitiesReference	Reference to the capabilities document of the Publisher to be registered	URL	One (Mandatory)

13.1.2 Response

If the request is accepted and no Exception is raised, the broker retrieves the capabilities document, verifies that the document is a valid Publish/Subscribe capabilities document, and returns a RegisterPublisherResponse. If there is a failure retrieving or verifying the capabilities document, an Exception is raised.

Figure 29: RegisterPublisher response

Requirement
/req/core/brokering-publisher/registerpublisher-connect
Req 55 When the *RegisterPublisher* operation is executed a **Publisher** shall retrieve the capabilities document of the registered Publisher and verify that it contains integrates `FilterCapabilities`, `DeliveryCapabilities`, and `Publications` sections before returning the RegisterPublisherResponse

13.1.3 Exceptions

Exceptions raised as a result of the *RegisterPublisher* operation are described below.

Requirement
/req/core/brokering-publisher/registerpublisher-exceptions
Req 56 A **Publisher** shall raise Exceptions in accordance with Table 28 when executing the *RegisterPublisher* operation

Table 28: RegisterPublisher Exceptions

Exception Code	Description	Locator Values
PublisherRegistrationRejected	Registration of the Publisher was rejected by	None, omit "locator" parameter

	the broker	
PublisherRegistrationFailed	Registration of the Publisher on the broker failed	None, omit “locator” parameter
NoApplicableCode	No other exceptionCode specified by this service and server applies to this exception	None, omit “locator” parameter

13.2 RemovePublisher operation

The *RemovePublisher* operation removes a Publisher from the broker. As a result of this operation, the broker capabilities may change (e.g. removing the Publications, FilterCapabilities, DeliveryCapabilities of the removed Publisher); the specification of such changes is out of the scope of this Requirements Class.

Requirement
/req/core/brokering-publisher/removepublisher
Req 57 A **Publisher** shall offer the *RemovePublisher* operation

13.2.1 Request

The following figure and table list the parameters for the *RemovePublisher* operation:

Figure 30: RemovePublisher request

Table 29: RemovePublisher properties

Name	Definition	Data type and values	Multiplicity and use
capabilitiesReference	The Capabilities reference of the Publisher(s) to be removed and disconnected	URL	One to many (Mandatory)

13.2.2 Response

If the request is accepted and no Exception is raised, the broker accepts the request, removes the specified Publishers and returns a RemovePublisherResponse.

Figure 31: RemovePublisher response

13.2.3 Exceptions

Exceptions raised as a result of the *RemovePublisher* operation are described below.

Requirement
/req/core/brokering-publisher/removepublisher-exceptions
Req 58 A **Publisher** shall raise Exceptions in accordance with Table 30 when executing the *RemovePublisher* operation

Table 30: RemovePublisher Exceptions

Exception Code	Description	Locator Values
UnknownPublisher	The Publisher identified by the `capabilitiesReference` parameter is unknown to the broker	Comma-separated list of invalid `capabilitiesReference` parameters
NoApplicableCode	No other exceptionCode specified by this service and server applies to this exception	None, omit "locator" parameter

13.3 GetCapabilities operation

In addition to the three parts offered by Standalone Publishers: filtering capabilities (Clause 8.1.1), delivery capabilities (Clause 8.1.2), and published contents (Clause 8.1.3) Brokering Publishers add RegisteredPublishers: the set of registered Publishers.

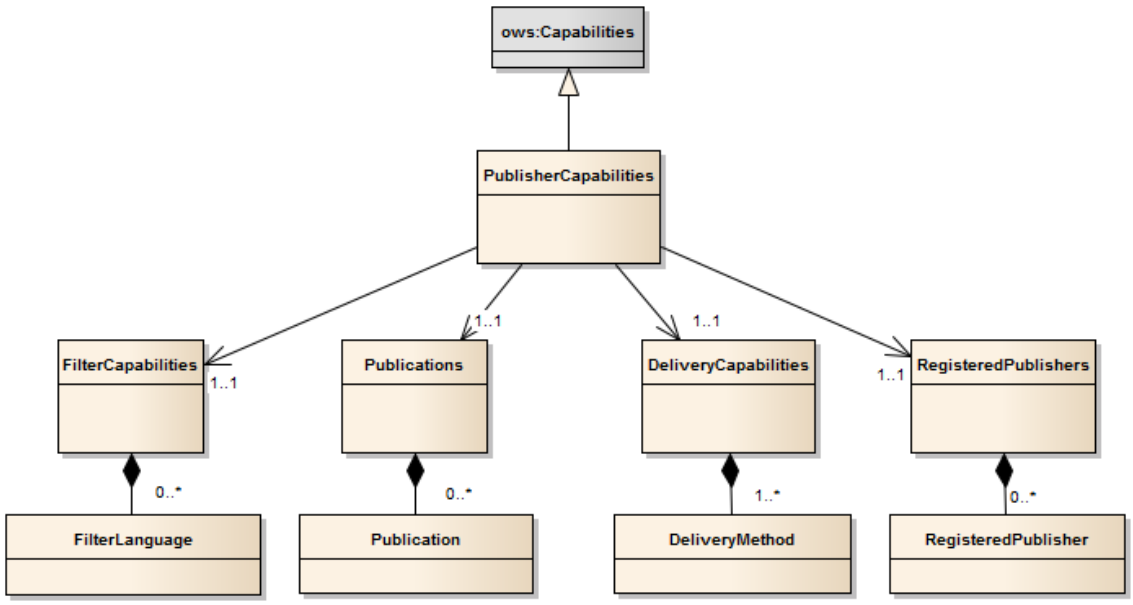


Figure 32: Brokering Capabilities

13.3.1 RegisteredPublishers

The set of registered Publishers on a broker is described with the RegisteredPublishers type. RegisteredPublishers is returned as part of the PublisherCapabilities type as a result of the *GetCapabilities* operation.

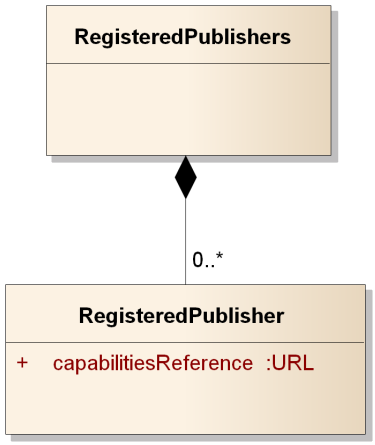


Figure 33: RegisteredPublishers metadata

Requirement
/req/core/brokering-publisher/getcapabilities-registered-publishers
Req 59 A **Publisher** shall return a `RegisteredPublishers` as part of the PublisherCapabilities type as a result of the *GetCapabilities* operation

14. Requirements Class – Publication Manager extends Basic Publisher

Requirements Class	
http://www.opengis.net/spec/pubsub/1.0/req/core/publication-manager	
Target type	Publisher
Dependency	http://www.opengis.net/spec/pubsub/1.0/req/core/basic-publisher
Requirement	/req/core/publication-manager/getcapabilities-processingcapabilities
Requirement	/req/core/publication-manager/getcapabilities-unique-processinglanguage
Requirement	/req/core/publication-manager/publication-valid-processinglanguage
Requirement	/req/core/publication-manager/createpublication
Requirement	/req/core/publication-manager/createpublication-publication-id
Requirement	/req/core/publication-manager/createpublication-assign-properties
Requirement	/req/core/publication-manager/createpublication-assign-identifier
Requirement	/req/core/publication-manager/createpublication-filter-id
Requirement	/req/core/publication-manager/createpublication-valid-filter-id
Requirement	/req/core/publication-manager/createpublication-processinglanguage-id
Requirement	/req/core/publication-manager/createpublication-valid-processinglanguage-id
Requirement	/req/core/publication-manager/createpublication-exceptions
Requirement	/req/core/publication-manager/removepublication
Requirement	/req/core/publication-manager/removepublication-nesting
Requirement	/req/core/publication-manager/removepublication-base-publication-removal
Requirement	/req/core/publication-manager/subscribe-derived-publications
Requirement	/req/core/publication-manager/derived-publication-identifiers
Requirement	/req/core/publication-manager/removepublication-exceptions

The Publication Manager Requirements Class supports the creation, removal, and subscriptions to user-defined publications that are derived from an existing publication. This Requirements Class requires that a Publisher implement two operations:

> ***CreatePublication*** - allows for the creation of a new derived publication based upon an existing publication with an optional filter; and
>
> ***RemovePublication*** - allows for the removal of a derived publication.

A derived publication is a publication that is created by applying an optional additional filter and/or processing expression to messages aggregated within an existing publication. As with any publication, a Subscriber may subscribe to a derived publication. Derived publications allow subscription filters to be shared among a large number of Subscribers rather than having each Subscriber create a subscription with the same filter and/or processing transformation. This capability can be useful in cases such as large enterprises

where different filtering and processing criteria on publications is required for different sets of Subscribers in order to satisfy policy and/or legal requirements.

This clause describes operations that support the management of derived publications. The operations described herein allow derived publications to be created and removed from the system.

14.1 Publication Types

The `DerivedPublication` type is a specialized type of publication that is a distinct type of offered publication. Specifically, an offered publication is published content in its original form, whereas `DerivedPublications` are derived from a processed and/or filtered form of another publication. Therefore, there may be any number of `DerivedPublications` that apply processing or filtering to a single base publication. `DerivedPublication` identifiers are accepted as publication identifiers to all Publish/Subscribe operations and are included among publications results in *GetCapabilities* and other relevant operation responses. `DerivedPublications` themselves may be used as a base publication when creating a new `DerivedPublication`, and it is therefore possible to create a hierarchy of publications with a single base publication at its root.

This Requirements Class also adds an extension to the `Publication` type to allow Publishers to advertise the supported processing languages for each base Publication. The mechanism for advertising the supported processing languages is similar to that of supported filter languages. Therefore, as with filter languages, a set of supported processing languages is advertised in the *GetCapabilities* response and each Publication lists the set of processing languages allowed for each specific Publication.

Table 31: Publication extension properties

Name	Definition	Data type and values	Multiplicity and use
supportedProcessingLanguage	The processing languages that are supported for processing this publication	URI	Zero to many (Optional)

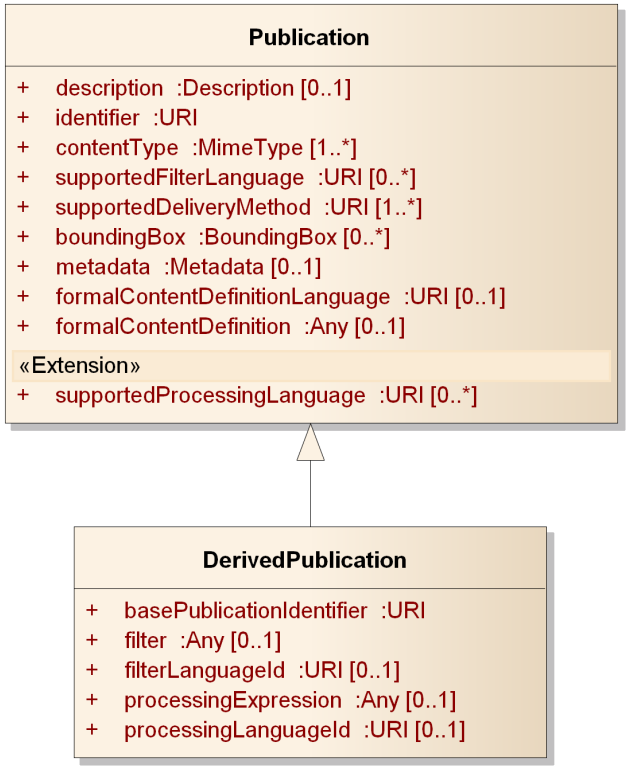


Figure 34: DerivedPublication

Table 32: DerivedPublication properties

Name	Definition	Data type and values	Multiplicity and use
basePublicationIdentifier	Identifier of the base publication	URI	One (Mandatory)
filter	The filter applied to messages produced by the base publication that are available to Subscribers of this publication	Any	Zero to one (Optional)
filterLanguageId	The identifier (unique in the scope of a Publisher) for the language used to encode the filter	URI	Zero to one (Optional) Required if filter is present
processingExpression	A formal expression that specifies processing instructions for transforming the base publication contents to a new form. If a processing expression is not provided, the DerivedPublication message contents are those of the base	Any	Zero to one (Optional)

	publication		
processingLanguageId	The identifier (unique in the scope of a Publisher) for the language used to process the messages from the base publication	URI	Zero to one (Optional) Required if processingExpression is present

14.1.1 ProcessingCapabilities

The `ProcessingCapabilities` data type describes the processing-related capabilities of a Publisher. A Publisher may support specific processing languages, such as XSLT, or WCPS. In order to support the creation of processed derived publications, the Publisher provides metadata about the processing languages it supports, if any.

The `ProcessingLanguage` type contains information about the processing languages that the Publisher supports for processing matching messages before the delivery.

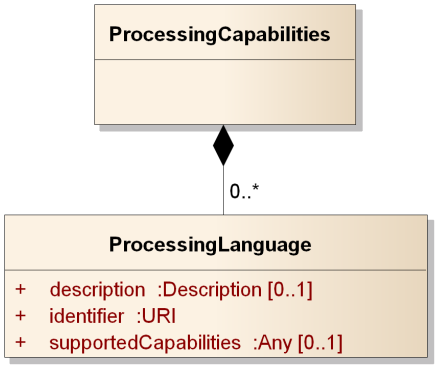


Figure 35: ProcessingCapabilities

Table 33: ProcessingLanguage properties

Name	Definition	Data type and values	Multiplicity and use
description	The abstract, title, and other human-readable descriptive information	Description [see OGC 06-121r3]	Zero to one (Optional)
identifier[A]	A unique identifier for the ProcessingLanguage on this Publisher	URI	One (Mandatory)

Name	Definition	Data type and values	Multiplicity and use
supportedCapabilities	Formal definition of the capabilities supported by the service regarding this ProcessingLanguage. For example, this can include supported operators/operands, process parameter ranges, etc.	Any	Zero to one (Optional)
A. Example identifiers include "http://www.opengis.net/wcps/1.0", indicating support for WCPS 1.0 processing mechanisms			

`ProcessingLanguage` identifiers are provided to the *CreatePublication* operation along with the actual processing expression specified in that language. For example, the *CreatePublication* operation can be executed with the WCPS processing language identifier (e.g., "http://www.opengis.net/wcps/1.0") along with the specific expression that defines the process of interest.

`ProcessingLanguage` identifiers are advertised for specific publications as part of the `DerivedPublications` data type. Publishers may choose to support a different set of processing languages for each publication. `ProcessingLanguage` identifiers advertised in `ProcessingCapabilities` need not be associated with any publication offered by the Publisher, such as cases where no publications are offered or the set of offered publications varies over time.

Requirement
/req/core/publication-manager/getcapabilities-processingcapabilities
Req 60 A **Publisher** shall return a `ProcessingCapabilities` structure within its *GetCapabilities* response

Requirement
/req/core/publication-manager/getcapabilities-unique-processinglanguage
Req 61 A **Publisher** shall uniquely identify each offered `ProcessingLanguage` included in the `PublisherCapabilities`

Requirement
/req/core/publication-manager/publication-valid-processinglanguage
Req 62 Each `supportedProcessingLanguage` of a `Publication` shall correspond to one of the `ProcessingLanguage` identifiers advertised in the `ProcessingCapabilities`

14.2 CreatePublication operation

The *CreatePublication* operation is used to create a filtered view of a publication offered by a publisher. The salient parameters for the operation are a base publication identifier, an optional filter that is used to identify the active set of messages, and an optional processing expression. A derived publication with only filtering applied may be considered conceptually equivalent to a stored query.

While filtering allows for a subset of messages from the base publication to be offered, processing expressions change the nature of the delivered messages (relative to the base publication) in some fashion. One example of processing capabilities would be a Web Coverage Processing Service (WCPS) augmented with Publish/Subscribe that allows for the creation of derived publications with WCPS processing expressions. Subscribers may then subscribe to a derived publication and receive the processed messages. Processing is applied to messages by the Publisher prior to being made available as part of the derived publication.

14.2.1 Request

The following diagram and table list the request parameters for the *CreatePublication* operation:

Figure 36: CreatePublication request

Table 34: CreatePublication properties

Name	Definition	Data type and values	Multiplicity and use
identifier	The identifier of the DerivedPublication to be created. If specified, it must be unique within the Publisher. If unspecified, the Publisher will generate a unique identifier	URI	Zero to one (Optional)
basePublicationIdentifier	Identifier of the base publication upon which the DerivedPublication is derived	URI	One (Mandatory)

Name	Definition	Data type and values	Multiplicity and use
description	A human-readable description of the DerivedPublication	String	One (Mandatory)
filter	An expression that evaluates to a Boolean value (true/false) when applied to messages published in the base publication. It determines whether a message from the base publication appears as a message in this DerivedPublication. If a filter is not provided, no filtering is applied	Any	Zero to one (Optional)
filterLanguageId	The identifier of the filter language used to encode the filter. Must be one of the `supportedFilterLanguages` advertised for the base publication	URI	Zero to one (Optional) Required if filter is present
processingExpression	A formal expression that specifies processing instructions for transforming the base publication contents to a new form. If a processing expression is not provided, the DerivedPublication message contents are those of the base publication	Any	Zero to one (Optional)
processingLanguageId	The identifier of the processing language used to encode the processing expression. Must be one of the `supportedFilterLanguages` advertised for the base publication	URI	Zero to one (Optional) Required if processingExpression is present

Requirement
/req/core/publication-manager/createpublication
Req 63 The **Publisher** shall offer the *CreatePublication* operation

Requirement
/req/core/publication-manager/createpublication-publication-id
Req 64 The **Publisher** shall raise an Exception if the

`basePublicationIdentifier` specified in the *CreatePublication* request is not a member of the list of offered publications at the time the derived publication is created

Requirement
/req/core/publication-manager/createpublication-assign-properties
Req 65 The **Publisher** shall assign publication properties (`contentType`, `supportedFilterLanguage`, `supportedDeliveryMethod`, `boundingBox`, `formalContentDefinitionLanguage`, and `formalContentDefinition`) from the base publication to the created DerivedPublication when a derived publication is created, excepting the `identifier` and `description` properties

Requirement
/req/core/ publication-manager/createpublication-assign-identifier
Req 66 The **Publisher** shall assign a unique `identifier` to the created DerivedPublication, when not specified in the *CreatePublication* request

Requirement
/req/core/publication-manager/createpublication-filter-id
Req 67 A **Publisher** shall raise an Exception if the *CreatePublication* request includes a `filter`, but does not include a `filterLanguageId`

Requirement
/req/core/ publication-manager/createpublication-valid-filter-id
Req 68 A **Publisher** shall raise an Exception if the *CreatePublication* request includes a `filterLanguageId` that does not correspond to a `supportedFilterLanguage` for the base publication

Requirement
/req/core/publication-manager/createpublication-processinglanguage-id
Req 69 A **Publisher** shall raise an Exception if the *CreatePublication* request includes a `processingExpression`, but does not include a `processingLanguageId`

Requirement
/req/core/publication-manager/createpublication-valid-processinglanguage-id
Req 70 A **Publisher** shall raise an Exception if the *CreatePublication* request includes a

`processingLanguageId` that does not correspond to a `supportedProcessingLanguage` for the publication

14.2.2 Response

If the request is accepted and no Exception is raised, the Publisher creates a new `DerivedPublication` and returns a `CreatePublicationResponse`.

Figure 37: CreatePublication response

Table 35: CreatePublication response properties

Name	Definition	Data type and values	Multiplicity and use
publication	The newly created DerivedPublication	DerivedPublication	One (Mandatory)

14.2.3 Exceptions

Exceptions raised as a result of the *CreatePublication* operation are described below.

Requirement
/req/core/publication-manager/createpublication-exceptions
Req 71 A **Publisher** shall raise Exceptions in accordance with Table 36 when executing the *CreatePublication* operation

Table 36: CreatePublication Exceptions

Exception Code	Description	Locator Values
InvalidPublicationIdentifier	The requested base publication is unknown to the Publisher and/or the identifier specified for the publication to be created is not unique	Comma-separated list of invalid publication identifiers
InvalidFilter	The requested filter is not valid for the subscription or not known to the	XPath to invalid request filter section, or other relevant request location

	Publisher	information
InvalidParameterValue	Operation request contains an invalid parameter value	Name of parameter with invalid value
NoApplicableCode	No other exceptionCode specified by this service and server applies to this exception	None, omit "locator" parameter

14.3 RemovePublication operation

The *RemovePublication* operation deletes one or more derived publications from the system.

14.3.1 Request

The following figure and table list the parameters for the *RemovePublication* operation:

Figure 38: RemovePublication request

Table 37: RemovePublication properties

Name	Definition	Data type and values	Multiplicity and use
publicationIdentifier	The identifiers of the derived publication(s) to be removed	URI	One to many (Mandatory)

DerivedPublications may be created using other DerivedPublications as the base publication. However, any publications with active DerivedPublications cannot be removed until their child DerivedPublications have first been removed.

Requirement
/req/core/publication-manager/removepublication
Req 72 The **Publisher** shall offer the *RemovePublication* operation

Requirement
/req/core/publication-manager/removepublication-nesting

Req 73 The **Publisher** shall raise an Exception if the *RemovePublication* operation specifies a publication that is an active base publication for one or more derived publications

Requirement
/req/core/publication-manager/removepublication-base-publication-removal
Req 74 The **Publisher** shall raise an Exception if the publicationIdentifier parameter to the *RemovePublication* operation specifies a publication that is not a derived publication

DerivedPublications are publications, and as such the Publisher shall follow normal subscription termination procedures as described in Clause 8.3.1 when a DerivedPublication is removed to which active subscriptions are associated.

14.3.2 Response

If the request is accepted and no Exception is raised, the Publisher removes the specified DerivedPublication and returns a RemovePublicationResponse.

Figure 39: RemovePublication response

Requirement
/req/core/publication-manager/subscribe-derived-publications
Req 75 The **Publisher** shall perform DerivedPublication message matching and message delivery on messages that match on the base publication, but filtered by any filters on the DerivedPublication

Requirement
/req/core/publication-manager/derived-publication-identifiers
Req 76 The **Publisher** shall accept DerivedPublication identifiers as valid publication identifiers to all Publish/Subscribe operations (e.g., the *Subscribe* operation) and include DerivedPublications among publication results (e.g., the *GetCapabilities* operation)

14.3.3 Exceptions

Exceptions raised as a result of the *RemovePublication* operation are described below.

Requirement
/req/core/publication-manager/removepublication-exceptions
Req 77 A **Publisher** shall raise Exceptions in accordance with Table 38 when executing the *RemovePublication* operation

Table 38: RemovePublication Exceptions

Exception Code	Description	Locator Values
InvalidPublicationIdentifier	The publication identifier is unknown to the Publisher, or the `publicationIdentifier` parameter is the identifier of a non-derived publication	Comma-separated list of invalid publication identifiers
InvalidParameterValue	Operation request contains an invalid parameter value	Name of parameter with invalid value
NoApplicableCode	No other exceptionCode specified by this service and server applies to this exception	None, omit "locator" parameter

15. Requirements Class – Capabilities Filtering extends Basic Publisher

Requirements Class	
http://www.opengis.net/spec/pubsub/1.0/req/core/capabilities-filtering	
Target type	Publisher
Dependency	**http://www.opengis.net/spec/pubsub/1.0/req/core/standalone-publisher**
Requirement	/req/core/capabilities-filtering/getcapabilities-content-sort
Requirement	/req/core/capabilities-filtering/getcapabilities-content-filter
Requirement	/req/core/capabilities-filtering/getcapabilities-search
Requirement	/req/core/capabilities-filtering/getcapabilities-exceptions

15.1 Introduction

Clause 8.1 of this standard and Clause 7.4 of OGC 06-121r3 define the response to a GetCapabilities request. The response is composed of a number of sections including a `Publications` section, which lists the publications offered for subscription by a Publisher, otherwise known as a content section. This `Publications` section can become quite large, hindering the efficient transmission of a capabilities document over

the Internet. For example, a Publisher may offer many thousands of publications resulting in a large and cumbersome capabilities document. The content section of OGC web services may include multiple items, in this case individual publications.

This clause defines syntactic and semantic extensions to the *GetCapabilities* operation in order to support very large `Publications` sections. Specifically, this clause defines additional parameters for the GetCapabilities request that allow a client to:

- control the number of items that appear in the `Publications` section;
- page through a `Publications` section that includes a large number of items; and
- specify query predicates, including spatial and temporal predicates, which allow a client to control which items are listed in the `Publications` section.

15.2 Request

Table 3 in Clause 7 of OGC 06-121r3 defines the standard set of request parameters for the *GetCapabilities* operation. Table 39 below defines additional parameters for the *GetCapabilities* operation that enables support for large `Publications` sections.

The default maximum number of content items returned is 15 unless specified otherwise by the `count` parameter described in Table 39. As this conformance class addresses the requirements of web services with large numbers of content items, this means that clients executing the *GetCapabilities* operation will not be overwhelmed by large Capabilities responses before being able to discover the functional capabilities.

Table 39: Additional request parameters for GetCapabilities operation

Name	Definition	Data type and values	Multiplicity and use
searchTerms	A list of terms, one or more of which, matching content items appearing in the capabilities document shall contain.	String	Zero to one (Optional)
count	The maximum number of content items that shall appear in the `Publications` section of a capabilities document at one time.	Integer (default=15)[A]	Zero to one (Optional)
startIndex[B]	The item offset, starting from zero, from which the service shall begin presenting content items in the `Publications` section of a capabilities document.	Integer (default=0)	Zero to one (Optional)

Name	Definition	Data type and values	Multiplicity and use
bbox	A spatial search box as defined in clause 10.2 of OGC 06-121r3.	BoundingBox [see OGC 06-121r3]	Zero to one (Optional)
start	The starting point of a temporal search range. When omitted, no start time filtering is applied	TM_Instant [see ISO/TS 19103:2005]	Zero to one (Optional)
end	The ending point of a temporal search range. When omitted, no end time filtering is applied	TM_Instant [see ISO/TS 19103:2005]	Zero to one (Optional)

A. When no content filtering parameters are provided, the default values apply. Unless the count parameter is provided with the request, at most 15 items are returned in the `Publications` section by services that implement this conformance class

B. See requirement /req/core/basic-publisher/getcapabilities/content/sort

Requirement
/req/core/capabilities-filtering/getcapabilities-content-sort
Req 78 A **Publisher** shall impose a consistent sort order on the items listed in the `Publications` section. The sorting methodology is not specified by this Standard, but GetCapabilities responses shall present a consistent order between GetCapabilities requests, regardless of filtering criteria

15.3 Response

Without the parameters defined in Table 39, the *GetCapabilities* operation behaves as described in OGC 06-121r3 and generates a complete `Publications` section as defined in Clause 9.1.4 of this standard and Clause 7.4 of OGC 06-121r3. The response to a GetCapabilities filtering query will always be a valid Capabilities document. The parameters in Table 39, if specified, only affect what items appear in the `Publications` section of the response. Contents filtering will not take effect if the GetCapabilities request excludes the `Publications` section from appearing in the response via the `sections` parameter described in OGC 06-121r3.

Requirement
/req/core/capabilities-filtering/getcapabilities-content-filter
Req 79 A **Publisher** shall filter the items in the `Publications` section of the Capabilities response in accordance with Clause 15.2 when the parameters from Table 39 are provided in the request

Requirement
/req/core/capabilities-filtering/getcapabilities-search
Req 80 When a **Publisher** receives a GetCapabilities request that causes the

`Publications` section to be excluded from the response, the Publisher shall ignore any of the parameters defined in Table 39

15.4 Examples

The following request fragments exemplify (in a KVP encoding) how the parameters in Table 39 affect the behavior of the *GetCapabilities* operation.

Example 1: Excluded Publications section

```
…&sections=ServiceIdentification,ServiceProvider&searchTerms=blue,ox&…
```

In this example, the `searchTerms` parameter is ignored since the request specifically excludes the `Publications` section (as specified by `sections=ServiceIdentification,ServiceProvider`).

Example 2: Publications section paging

```
…&sections=Publications &count=10&startIndex=11&…
```

In this example, only the Publications section is presented in the response and the Publications section contains 10 items (i.e., items 11 through 20).

Example 3: Publications section filtering

```
…&sections=Publications &searchTerms=restaurants&bbox=43.57,-
79.64,43.89,-79.12&…
```

In this example, only the Publications section is presented, and the records that contain the search term “restaurants” and lie within the rough boundary of Toronto, Ontario, Canada.

Example 4: Publications section filtering

```
…&sections=Contents&searchTerms=javascript&start=01-01-2013&end=06-30-
2013&…
```

In this example, only the Publications section is presented, and the records that contain the search term “javascript” and have a salient date in the first 6 months of 2013.

15.5 Exceptions

Exceptions raised as a result of the *GetCapabilities* operation are described below.

Requirement
/req/core/capabilities-filtering/getcapabilities-exceptions
Req 81 A **Publisher** shall raise Exceptions in accordance with Table 40 when executing the *GetCapabilities* operation, in addition to those specified in Clause 9.1.3

Table 40: GetCapabilities Filtering Exceptions

Exception Code	Description	Locator Values
MissingParameterValue	Operation request does not include a parameter value, and this server did not declare a default value for that parameter	Name of missing parameter
InvalidParameterValue	Operation request contains an invalid parameter value	Name of parameter with invalid value
NoApplicableCode	No other exceptionCode specified by this service and server applies to this exception	None, omit “locator” parameter

Annex A. Abstract Test Suite (Normative)

A Publish/Subscribe implementation must satisfy the following system characteristics to be conformant with this specification.

Test, requirement, requirements class, and conformance class identifiers below are relative to http://www.opengis.net/spec/pubsub/1.0/.

The minimum set of conformance classes an implementation needs to pass are either Basic Receiver (section A.1) or Basic Publisher (section A.2)

A.1 Conformance class: Basic Receiver

/conf/core/basic-receiver	
Requirements Class	/req/core/basic-receiver

Test: /conf/core/basic-receiver/notify

Requirement	**/req/core/basic-receiver/notify**
Test Purpose	A **Receiver** shall offer the *Notify* operation
Test Method	Execute a *Notify* operation with test data

A.2 Conformance class: Basic Publisher

/conf/core/basic-publisher	
Dependency	http://www.opengis.net/doc/IS/OWS/1.1/clause/8
Dependency	http://www.opengis.net/doc/IS/OWS/1.1/clause/10
Requirements Class	/req/core/basic-publisher

Test: /conf/core/basic-publisher/getcapabilities-conf-class-listing

Requirement	**/req/core/basic-publisher/getcapabilities-conf-class-listing**

Test Purpose	A **Publisher** shall advertise conformance classes which are supported by the server. Each supported conformance class shall be identified by a unique value of the `Profile` property of the `ServiceIdentification` section of the capabilities document
Test Method	Execute a *GetCapabilities* operation against the service that includes the `ServiceIdentification` section and verify that the service returns a Capabilities document with a `ServiceIdentification` section with a `Profile` section with a value starting with "http://www.opengis.net/spec/pubsub/1.0/"

Test: /conf/core/basic-publisher/getcapabilities-filtercapabilities

Requirement	**/req/core/basic-publisher/getcapabilities-filtercapabilities**
Test Purpose	A **Publisher** shall return a `FilterCapabilities` structure within its *GetCapabilities* response
Test Method	Execute a *GetCapabilities* operation against the service and verify that the service returns a Capabilities document with a `FilterCapabilities` section

Test: /conf/core/basic-publisher/unique-filter-languages

Requirement	**/req/core/basic-publisher/unique-filter-languages**
Test Purpose	A **Publisher** shall uniquely identify each offered `FilterLanguage` included in `FilterCapabilities`
Test Method	Execute a *GetCapabilities* operation on the service, ensure that every `FilterLanguage identifier` property in the `PublisherCapabilities` section is unique among all `FilterLanguage` identifiers

Test: /conf/core/basic-publisher/deliverycapabilities

Requirement	**/req/core/basic-publisher/deliverycapabilities**
Test Purpose	A **Publisher** shall return a `DeliveryCapabilities` structure within its *GetCapabilities* response

Test Method	Execute a *GetCapabilities* operation against the service and verify that the service returns a Capabilities document with a `DeliveryCapabilities` section

Test: /conf/core/basic-publisher/unique-delivery-method

Requirement	**/req/core/basic-publisher/unique-delivery-method**
Test Purpose	A **Publisher** shall uniquely identify each offered `DeliveryMethod` included in the `PublisherCapabilities`
Test Method	Execute a *GetCapabilities* operation on the service, ensure that every `DeliveryMethod` identifier property in the `DeliveryCapabilities` section is unique among all other `DeliveryMethod` identifiers

Test: /conf/core/basic-publisher/publications

Requirement	**/req/core/basic-publisher/publications**
Test Purpose	A **Publisher** shall return a `Publications` structure within its *GetCapabilities* response
Test Method	Execute a *GetCapabilities* operation against the service and verify that the service returns a Capabilities document with a `Publications` section

Test: /conf/core/basic-publisher/publication-valid-filter-language

Requirement	**/req/core/basic-publisher/publication-valid-filter-language**
Test Purpose	Each `supportedFilterLanguage` of a `Publication` shall correspond to one of the `FilterLanguage` identifiers advertised in the `FilterCapabilities`
Test Method	Execute a *GetCapabilities* operation against the service and verify that each `supportedFilterLanguage` identifier in each `Publication` section exactly matches a `FilterLanguage identifier`

Test: /conf/core/basic-publisher/publication-valid-delivery-method

Requirement	**/req/core/basic-publisher/publication-valid-delivery-method**
Test Purpose	Each `supportedDeliveryMethod` of a `Publication` shall correspond to one of the `DeliveryMethod` identifiers advertised in the `DeliveryCapabilities`
Test Method	Execute a *GetCapabilities* operation against the service and verify that each `supportedDeliveryMethod` identifier in each `Publication` section exactly matches a `DeliveryMethod identifier`

Test: /conf/core/basic-publisher/publication-unique-publication-id

Requirement	**/req/core/basic-publisher/publication-unique-publication-id**
Test Purpose	The `identifier` on each `Publication` shall be unique among all other `Publication` identifiers on the **Publisher**
Test Method	Execute a *GetCapabilities* operation on the service, ensure that every `Publication` identifier property in the `Publications` section is unique among all other `Publication` identifiers

Test: /conf/core/basic-publisher/valid-exceptions

Requirement	**/req/core/basic-publisher/valid-exceptions**
Test Purpose	A **Publisher** shall issue Exceptions that incorporate an ExceptionReport valid according to Clause 8 of the OWS Common Specification [OGC 06-121r3]
Test Method	Execute a request that raises an exception on the service and ensure that the response message contains a valid ExceptionReport from [OGC 06-121r3]

Test: /conf/core/basic-publisher/exception-version

Requirement	**/req/core/basic-publisher/exception-version**

Test Purpose	A **Publisher** shall raise Exceptions with the ExceptionReport `version` set to the value "1.0.0"
Test Method	Execute a request that raises an exception on the service and ensure that the response Exception message `version` parameter is "1.0.0"

Test: /conf/core/basic-publisher/subscribe

Requirement	**/req/core/basic-publisher/subscribe**
Test Purpose	The **Publisher** shall offer the *Subscribe* operation
Test Method	Execute a *Subscribe* operations against a test publication and ensure that the SubscribeResponse includes a valid `Subscription`

Test: /conf/core/basic-publisher/subscribe-assign-unique-id

Requirement	**/req/core/basic-publisher/subscribe-assign-unique-id**
Test Purpose	A **Publisher** shall assign a unique identifier to each created subscription
Test Method	Execute three *Subscribe* operations against a test publication and ensure that the `Subscription identifier` is unique among all returned `Subscriptions`

Test: /conf/core/basic-publisher/subscribe-default-termination-time

Requirement	**/req/core/basic-publisher/subscribe-default-termination-time**
Test Purpose	A **Publisher** shall assign a default `terminationTime` to created subscriptions if not provided by the Subscriber
Test Method	Execute a *Subscribe* operations against a test publication without an `terminationTime` parameter and ensure that the returned `Subscription terminationTime` is set

Test: /conf/core/basic-publisher/match-active-subscriptions

Requirement	**/req/core/basic-publisher/match-active-subscriptions**
Test Purpose	A **Publisher** shall match messages against all active subscriptions
Test Method	Execute two *Subscribe* operations against two different test publications and ensure that matching messages are delivered for each subscription

Test: /conf/core/basic-publisher/match-inactive-subscriptions

Requirement	**/req/core/basic-publisher/match-inactive-subscriptions**
Test Purpose	A **Publisher** shall cease matching and delivery of messages when subscriptions move to an inactive or terminated state
Test Method	Execute a *Subscribe* operation against a test publication with an `terminationTime` parameter that specifies 1 minute in the future. Ensure that messages are delivered on the subscription for the 1-minute period, and ensure that message delivery ceases shortly after the 1-minute period (i.e., on subscription expiry)

Test: /conf/core/basic-publisher/interrupt-matching

Requirement	**/req/core/basic-publisher/interrupt-matching**
Test Purpose	When a **Publisher** terminates a subscription it shall interrupt all unfinished matching processes for this subscription
Test Method	Execute a *Subscribe* operation against a test publication with an `terminationTime` parameter that specifies 1 hour in the future. Wait 1 minute and ensure that messages are delivered on the subscription. Execute an *Unsubscribe* operation against the test subscription, and ensure that message delivery ceases within a brief period

Test: /conf/core/basic-publisher/termination

Requirement	**/req/core/basic-publisher/termination**
Test Purpose	A **Publisher** shall terminate a subscription when its termination time is reached

Test Method	Execute a *Subscribe* operation against a test publication with an `terminationTime` parameter that specifies 1 minute in the future. Ensure that messages are delivered on the subscription for the 1-minute period, and ensure that message delivery ceases shortly after the 1-minute period (i.e., on subscription expiry)

Test: /conf/core/basic-publisher/filter-id

Requirement	**/req/core/basic-publisher/filter-id**
Test Purpose	A **Publisher** shall raise an Exception if the `Subscribe` request includes a `filter`, but does not include a `filterLanguageId`
Test Method	Execute a *Subscribe* operation including in the request a `filter`, but not a `filterLanguageId`. Ensure that the response is a MissingParameterValue Exception with a locator value set to "filterLanguageId"

Test: /conf/core/basic-publisher/valid-filter-id

Requirement	**/req/core/basic-publisher/valid-filter-id**
Test Purpose	A **Publisher** shall raise an Exception if the `Subscribe` request includes a `filterLanguageId` that does not correspond to a `supportedFilterLanguage` for the publication
Test Method	Execute a *Subscribe* operation including in the request a `filterLanguageId` that does not correspond to any `supportedFilterLanguage` for the publication. Ensure that the response is an InvalidParameterValue Exception with a locator value set to "filterLanguageId"

Test: /conf/core/basic-publisher/content-type

Requirement	**/req/core/basic-publisher/content-type**
Test Purpose	A **Publisher** shall raise an Exception if the `Subscribe` request does not include a `contentType` and the offered Publication advertises multiple content types

Test Method	Execute a *Subscribe* operation without a `contentType` parameter in the request against a test publication with multiple (offered) `contentType` parameters. Ensure that the response is a MissingParameterValue Exception with a locator value set to "contentType"

Test: /conf/core/basic-publisher/subscribe-exceptions

Requirement	**/req/core/basic-publisher/subscribe-exceptions**
Test Purpose	A **Publisher** shall raise Exceptions in accordance with Table 9 when executing the *Subscribe* operation
Test Method	Execute the *Subscribe* operation with the following scenarios: 1. A `publicationIdentifier` parameter set to "urn:pubsub:ats:InvalidPublication", and ensure that an InvalidPublicationIdentifier Exception is returned with a `locator` value of "urn:pubsub:ats:InvalidPublication" 2. An `terminationTime` parameter specifying a point in time a year ago, and ensure that the response is a PastTermination Exception with a `locator` value set to the requested termination time 3. A `deliveryMethod` parameter of "urn:pubsub:ats:InvalidDeliveryMethod", and ensure that the response is an InvalidDeliveryMethod Exception with a `locator` value set to the requested delivery method identifier 4. A `filter` parameter containing the text "Invalid filter", and ensure that the response is an InvalidFilter Exception 5. A missing `publicationIdentifier` parameter, and ensure that the response is a MissingParameterValue Exception with a `locator` value set to "publicationIdentifier" 6. A `deliveryMethod` parameter of "not a URN", and ensure that the response is a InvalidParameterValue Exception with a `locator` value set to "deliveryMethod" 7. An empty request (request sent to the Subscribe endpoint with no content), and ensure that the response is a NoApplicableCode Exception with an empty `locator` value

Test: /conf/core/basic-publisher/unsubscribe

Requirement	/req/core/basic-publisher/unsubscribe
Test Purpose	The **Publisher** shall offer the *Unsubscribe* operation
Test Method	Execute a *Subscribe* operation against a test publication, record the returned subscription identifier, and execute an *Unsubscribe* operation with the subscription identifier, and ensure that the response is a valid UnsubscribeResponse

Test: /conf/core/basic-publisher/halt-matching

Requirement	/req/core/basic-publisher/halt-matching
Test Purpose	A **Publisher** shall cease subscription matching for the subscription identified in the `Unsubscribe` request
Test Method	Execute a *Subscribe* operation against a test publication and record the returned subscription identifier, wait for test messages to be received for that subscription, then execute an *Unsubscribe* operation with the subscription identifier and after a reasonable delay ensure that no further messages are received

Test: /conf/core/basic-publisher/unsubscribe-exception-state

Requirement	/req/core/basic-publisher/unsubscribe-exception-state
Test Purpose	A **Publisher** shall leave subscription state unchanged when an Exception occurs during the *Unsubscribe* operation
Test Method	Execute a *Subscribe* operation against a test publication and record the returned test subscription identifier, wait for test messages to be received for that subscription, then execute an *Unsubscribe* operation with the subscription identifier "urn:pubsub:ats:invalidSubscriptionId" and ensure that messages continue to be received for the test subscription

Test: /conf/core/basic-publisher/unsubscribe-exceptions

Requirement	/req/core/basic-publisher/unsubscribe-exceptions
Test Purpose	A **Publisher** shall raise Exceptions in accordance with Table 11 when

	executing the *Unsubscribe* operation
Test Method	Execute an *Unsubscribe* operation with the following cases: 1. The `subscriptionIdentifier` parameter is set to "urn:pubsub:ats:invalidSubscriptionId" and ensure that the response is an InvalidSubscriptionIdentifier Exception with a `locator` value of "subscriptionIdentifier" 2. The body of the Unsubscribe request is empty (missing), and ensure that the response is a NoApplicableCode Exception with a missing `locator` value

Test: /conf/core/basic-publisher/renew

Requirement	**/req/core/basic-publisher/renew**
Test Purpose	The **Publisher** shall offer the *Renew* operation
Test Method	Execute a *Subscribe* operation against a test publication with an `terminationTime` parameter set to one minute after now, record the returned subscription identifier, execute a *Renew* operation with the subscription identifier with an `terminationTime` parameter set to two minutes after now, and ensure that the response is a valid RenewResponse

Test: /conf/core/basic-publisher/renew-update-termination-time

Requirement	**/req/core/basic-publisher/renew-update-termination-time**
Test Purpose	A **Publisher** shall update the `terminationTime` on the identified subscription to be the value of `newTerminationTime` provided as part of a successful *Renew* operation
Test Method	Execute a *Subscribe* operation against a test publication with an `trminationTime` parameter set to one minute after now, record the returned subscription identifier, execute a *Renew* operation with the subscription identifier and a `newTerminationTime` parameter set to two minutes after now, ensure that the response is a valid RenewResponse, and ensure that messages continue to arrive for approximately two minutes

Test: /conf/core/basic-publisher/renew-exception-state

Requirement	**/req/core/basic-publisher/renew-exception-state**
Test Purpose	A **Publisher** shall leave subscription state unchanged when an Exception occurs during the *Renew* operation
Test Method	Execute a *Subscribe* operation against a test publication with an `terminationTime` parameter set to one minute after now, record the returned subscription identifier, execute a *Renew* operation with the subscription identifier and a `newTerminationTime` parameter set to two days before now, ensure that the response is a PastTermination Exception, and ensure that messages cease being delivered after approximately one minute from the initial *Subscribe* operation call

Test: /conf/core/basic-publisher/renew-exceptions

Requirement	**/req/core/basic-publisher/renew-exceptions**
Test Purpose	A **Publisher** shall raise Exceptions in accordance with Table 13 when executing the *Renew* operation
Test Method	Execute a *Subscribe* operation against a test publication with an `terminationTime` parameter set to one minute after now, record the returned subscription identifier, execute a *Renew* operation with the following scenarios: 1. The `subscriptionIdentifier` parameter is set to "urn:pubsub:ats:InvalidSubscriptionIdentifier", and ensure that the response is an InvalidSubscriptionIdentifier Exception with a `locator` value set to "urn:pubsub:ats:InvalidSubscriptionIdentifier" 2. The `newTerminationTime` parameter set to 100 years after now, and ensure that the response is an TerminationUnacceptable Exception with a `locator` value set to the `newTerminationTime` parameter value passed in the request 3. The `newTerminationTime` parameter set to 1 day before now, and ensure that the response is a PastTermination Exception with a `locator` value set to the `newTerminationTime` parameter value passed in the request 4. A missing `newTerminationTime` parameter (not present

	in the request), and ensure that the response is an MissingParameterValue Exception with a `locator` value set to the value "newTerminationTime" 5. The newTerminationTime parameter set to the literal value "a day or two", and ensure that the response is a MissingParameterValue Exception with a `locator` value set to the value "newTerminationTime" 6. An empty request (request sent to the Renew endpoint with no content), and ensure that the response is a NoApplicableCode Exception with an empty `locator` value

A.3 Conformance class: Standalone Publisher

/conf/core/standalone-publisher	
Dependency	/conf/core/basic-publisher
Dependency	http://www.opengis.net/doc/IS/OWS/1.1/clause/7
Requirements Class	/req/core/standalone-publisher

Test: /conf/core/standalone-publisher/getcapabilities

Requirement	**/req/core/standalone-publisher/getcapabilities**
Test Purpose	The **Publisher** shall offer the *GetCapabilities* operation
Test Method	Execute the *GetCapabilities* operation with an `AcceptVersions` section with a single `Version` parameter set to "1.0.0" and the `service` parameter set to "PubSub", and ensure that the response is a valid PublisherCapabilities document

Test: /conf/core/standalone-publisher/getsubscription

Requirement	**/req/core/standalone-publisher/getsubscription**
Test Purpose	The **Publisher** shall offer the *GetSubscription* operation
Test Method	Execute the *GetSubscription* operation without any `subscriptionIdentifier` parameters, and ensure that the

	response is a valid GetSubscriptionResponse document. For every subscription in the GetSubscriptionResponse, execute the *GetSubscription* operation with the corresponding `subscriptionIdentifier` parameter, and ensure that the response is a valid GetSubscriptionResponse document related to that subscription

Test: /conf/core/standalone-publisher/getsubscription-all-subscriptions

Requirement	**/req/core/standalone-publisher/getsubscription-all-subscriptions**
Test Purpose	A **Publisher** shall return a `GetSubscriptionResponse` with all the active subscriptions when no subscription identifiers are provided as part of the `GetSubscription` request
Test Method	Execute the *Subscribe* operation on a test publication, record the returned subscription identifier, execute the GetSubscription operation with no subscriptionIdentifier parameters, ensure that the response is a valid GetSubscriptionResponse, and ensure that exactly one subscription with the recorded subscription identifier is present in the response

Test: /conf/core/standalone-publisher/getsubscription-exceptions

Requirement	**/req/core/standalone-publisher/getsubscription-exceptions**
Test Purpose	A **Publisher** shall raise Exceptions in accordance with Table 17 when executing the *GetSubscription* operation
Test Method	Execute the GetSubscription operation with the following scenarios: 1. A `subscriptionIdentifier` parameter set to the value "urn:pubsub:ats:InvalidSubscriptionIdentifier", and ensure that the response is an InvalidSubscriptionIdentifier Exception with the `locator` value set to "urn:pubsub:ats:InvalidSubscriptionIdentifier" 2. An empty request (request sent to the GetSubscription endpoint with no content), and ensure that the response is a NoApplicableCode Exception with an empty `locator` value

A.4 Conformance class: Pausable Publisher

/conf/core/basic-publisher	
Dependency	/conf/core/basic-publisher
Requirements Class	/req/core/pausable-publisher

Test: /conf/core/pausable-publisher/pause

Requirement	**/req/core/pausable-publisher/pause**
Test Purpose	A **Publisher** shall offer the *Pause* operation
Test Method	Execute the *Subscribe* operation on a test publication, record the returned subscription identifier, wait for messages to be received for the subscription, then execute the *Pause* operation with the `subscriptionIdentifier` parameter set to the recorded subscription identifier, and ensure that the response is a valid PauseResponse document

Test: /conf/core/pausable-publisher/pause-halt-delivery

Requirement	**/req/core/pausable-publisher/pause-halt-delivery**
Test Purpose	A **Publisher** shall cease the initiation of message delivery processes for the subscription when the *Pause* operation is successfully completed. Message delivery processes already underway continue unchanged
Test Method	Create a test subscription on the service via the *Subscribe* operation, wait for a message to be delivered, execute the *Pause* operation to pause the test subscription, and verify that no message is delivered for that subscription within a reasonable period to account for normal delays with delivery processes

Test: /conf/core/pausable-publisher/pause-unchanged-paused-subscription

Requirement	**/req/core/pausable-publisher/pause-unchanged-paused-subscription**

Test Purpose	When a **Publisher** executes the *Pause* operation on a subscription that is already paused, no change in subscription matching or subscription state will be made
Test Method	Create a test subscription on the service via the *Subscribe* operation, wait for a message to be delivered, execute the *Pause* operation to pause the test subscription, execute the Pause operation again on the test subscription, and ensure that no messages are delivered

Test: /conf/core/pausable-publisher/pause-exceptions

Requirement	**/req/core/pausable-publisher/pause-exceptions**
Test Purpose	A **Publisher** shall raise Exceptions in accordance with Table 19 when executing the *Pause* operation
Test Method	Create a test subscription on the service via the *Subscribe* operation, wait for a message to be delivered, execute the *Pause* operation with the following scenarios: 1. A `subscriptionIdentifier` set to the value "urn:pubsub:ats:InvalidSubscriptionIdentifier", and ensure that the response is a InvalidSubscriptionIdentifier Exception with a `locator` value of "urn:pubsub:ats:InvalidSubscriptionIdentifier" 2. An empty request (request sent to the Pause endpoint with no content), and ensure that the response is a NoApplicableCode Exception with an empty `locator` value

Test: /conf/core/pausable-publisher/resume

Requirement	**/req/core/pausable-publisher/resume**
Test Purpose	A **Publisher** shall offer the *Resume* operation
Test Method	Execute the *Subscribe* operation on a test publication, record the returned subscription identifier, wait for messages to be received for the subscription, execute the *Pause* operation with the `subscriptionIdentifier` parameter set to the recorded subscription identifier, execute the *Resume* operation with the `subscriptionIdentifier` parameter set to the recorded subscription identifier, and ensure that the response is a valid

	ResumeResponse document

Test: /conf/core/pausable-publisher/resume-resume-delivery

Requirement	**/req/core/pausable-publisher/resume-resume-delivery**
Test Purpose	A **Publisher** shall re-start all message delivery processes for the appropriate subscription when the *Resume* operation is successfully completed
Test Method	Execute the *Subscribe* operation on a test publication that publishes messages at a fixed rate (e.g., 1 message per second), record the returned subscription identifier, wait for a message to be received for the subscription, execute the *Pause* operation on the test subscription, wait until 5 messages will have been produced for the test subscription, execute the *Resume* operation on the test subscription, ensure that the response is a valid ResumeResponse document, and ensure that the expected 5 messages are received

Test: /conf/core/pausable-publisher/resume-unchanged-active-subscription

Requirement	**/req/core/pausable-publisher/resume-unchanged-active-subscription**
Test Purpose	When a **Publisher** executes the *Resume* operation on a subscription that is already active, no change in subscription matching or subscription state will be made
Test Method	Execute the *Subscribe* operation on a test publication, wait for messages to be received for the subscription, execute the *Resume* operation on the test subscription, ensure that the response is a valid ResumeResponse document, and ensure that messages continue to be received on the subscription

Test: /conf/core/pausable-publisher/resume-exceptions

Requirement	**/req/core/pausable-publisher/resume-exceptions**
Test Purpose	A **Publisher** shall raise Exceptions in accordance with Table 21 when executing the *Resume* operation

Test Method	Create a test subscription on the service via the *Subscribe* operation, wait for a message to be delivered, execute the *Pause* operation on the test subscription, execute the *Resume* operation with the following scenarios: 1. A `subscriptionIdentifier` set to the value "urn:pubsub:ats:InvalidSubscriptionIdentifier", and ensure that the response is a InvalidSubscriptionIdentifier Exception with a `locator` value of "urn:pubsub:ats:InvalidSubscriptionIdentifier" 2. An empty request (request sent to the Resume endpoint with no content), and ensure that the response is a NoApplicableCode Exception with an empty `locator` value

A.5 Conformance class: Message Batching Publisher

/conf/core/message-batching-publisher	
Dependency	/conf/core/basic-publisher
Requirements Class	/req/core/message-batching-publisher

Test: /conf/core/message-batching-publisher/subscribe-message-batching

Requirement	**/req/core/message-batching-publisher/subscribe-message-batching**
Test Purpose	A **Publisher** shall accept `MessageBatchingCriteria` with other subscription criteria on the *Subscribe* operation
Test Method	Execute the *Subscribe* operation to create a test subscription with message batching criteria with the parameter `maxMessageCount` set to "1", ensure that the response is a valid SubscribeResponse

Test: /conf/core/message-batching-publisher/withheld-delivery

Requirement	**/req/core/message-batching-publisher/withheld-delivery**
Test Purpose	A **Publisher** shall withhold delivery of messages until any of the subscription message batching criteria are met, at which time all withheld messages will be delivered together as a batch

Test Method	Create a test publication that starting the first second of every minute (11:00, 11:01…): produces 10 messages, waits 15 seconds, produces 3 more messages, and produces no further messages for the remainder of each minute. Execute the *Subscribe* operation to create a test subscription against the test publication, with message batching criteria with the parameter `maxMessageCount` set to "5" and the `maxDelay` parameter set to 30 seconds, ensure that the response is a valid SubscribeResponse, wait 1 minute, ensure that messages were delivered in 3 batches in the following order: 1. First batch with the first 5 of 10 messages (messages 1-5) 2. Second batch with the second 5 of 10 messages (messages 6-10) 3. Third batch with the final 3 messages (messages 11-13)

Test: /conf/core/message-batching-publisher/reset-batching

Requirement	**/req/core/message-batching-publisher/reset-batching**
Test Purpose	A **Publisher** shall reset tracking information (e.g., last batch delivery time and number of withheld messages) for subscription message batching criteria whenever a message batch is delivered
Test Method	Create a test publication that starting the first second of every minute (11:00, 11:01…): produces 10 messages, waits 15 seconds, produces 3 more messages, and produces no further messages for the remainder of each minute. Execute the *Subscribe* operation to create a test subscription against the test publication, with message batching criteria with the parameter `maxMessageCount` set to "5" and the `maxDelay` parameter set to 30 seconds, ensure that the response is a valid SubscribeResponse, wait 1 minute, ensure that messages were delivered in 3 batches in the following order: 1. First batch with the first 5 of 10 messages (messages 1-5) 2. Second batch with the second 5 of 10 messages (messages 6-10) Third batch with the final 3 messages (messages 11-13)

Test: /conf/core/message-batching-publisher/subscription-termination

Requirement	/req/core/message-batching-publisher/subscription-termination
Test Purpose	A **Publisher** shall deliver withheld messages in a batch when a subscription is terminated
Test Method	Create a test publication that produces 10 messages starting the first second of every minute (11:00, 11:01…). Execute the *Subscribe* operation to create a test subscription against the test publication with message batching criteria with the parameter `maxDelay` set to 60 seconds, ensure that the response is a valid SubscribeResponse, wait 30 seconds, ensure no messages were received for the test subscription, execute the *Unsubscribe* operation on the test subscription, and ensure that 10 messages are received for the subscription

Test: /conf/core/message-batching-publisher/pausing

Requirement	/req/core/message-batching-publisher/pausing
Test Purpose	A **Publisher** shall deliver withheld messages in a batch when a subscription is paused as described in the Pausable Publisher Requirements Class (see Clause 10)
Precondition	The Pausable Publisher conformance class is supported
Test Method	Create a test publication that produces 10 messages starting the first second of every minute (11:00, 11:01…). Execute the *Subscribe* operation to create a test subscription against the test publication with message batching criteria with the parameter `maxDelay` set to 60 seconds, ensure that the response is a valid SubscribeResponse, wait 30 seconds, ensure no messages were received for the test subscription, execute the *Pause* operation on the test subscription, and ensure that 10 messages are received for the subscription

Test: /conf/core/message-batching-publisher/subscribe-exceptions

Requirement	/req/core/message-batching-publisher/subscribe-exceptions
Test Purpose	A **Publisher** shall raise Exceptions in accordance with Table 23 when executing the *Subscribe* operation, in addition to those specified in

	Section 8.3.4
Test Method	Execute the *Subscribe* operation with the following scenarios: 1. MessageBatchingCriteria present with missing `maxDelay` and `maxMessageCount` parameters, ensure that the response is a MissingParameterValue Exception with a `locator` value of either "maxDelay" or "maxMessageCount" 2. MessageBatchingCriteria present with the `maxDelay` parameter set to the value "some period", ensure that the response is a InvalidParameterValue Exception with a `locator` value of "maxDelay" 3. MessageBatchingCriteria present with the `maxMessageCount` parameter set to the value "-999", ensure that the response is a InvalidParameterValue Exception with a `locator` value of "maxMessageCount"

A.6 Conformance class: Heartbeat Publisher

/conf/core/heartbeat-publisher	
Dependency	/conf/core/basic-publisher
Requirements Class	/req/core/heartbeat-publisher

Test: /conf/core/heartbeat-publisher/subscribe-heartbeat

Requirement	**/req/core/heartbeat-publisher/subscribe-heartbeat**
Test Purpose	A **Publisher** shall accept `HeartbeatCriteria` with other subscription criteria on the *Subscribe* operation
Test Method	Execute the *Subscribe* operation to create a test subscription with heartbeat criteria with the parameter `heartbeatRate` set to "1 minute", ensure that the response is a valid SubscribeResponse

Test: /conf/core/heartbeat-publisher/publish-heartbeat

Requirement	**/req/core/heartbeat-publisher/publish-heartbeat**

Test Purpose	A **Publisher** shall send regular HeartbeatMessages for each subscription as specified by each subscription's HeartbeatCriteria
Test Method	Execute the *Subscribe* operation to create a test subscription with heartbeat criteria with the parameter `heartbeatRate` set to "10 seconds", ensure that the response is a valid SubscribeResponse, wait 35 seconds, ensure that 3 heartbeat messages were received

Test: /conf/core/heartbeat-publisher/pausing

Requirement	**/req/core/heartbeat-publisher/pausing**
Test Purpose	A **Publisher** shall cease sending HeartbeatMessages for a subscription when it is paused as described in the Pausable Publisher Requirements Class (see Clause 10)
Precondition	The Pausable Publisher conformance class is supported
Test Method	Execute the *Subscribe* operation to create a test subscription against the test publication with heartbeat criteria with the parameter `heartbeatDelay` set to 10 seconds, ensure that the response is a valid SubscribeResponse, wait 30 seconds, ensure that 3 heartbeat messages were received for the test subscription, execute the *Pause* operation on the test subscription, wait 30 seconds, ensure that no further messages were received for the subscription

Test: /conf/core/heartbeat-publisher/subscribe-exceptions

Requirement	**/req/core/heartbeat-publisher/subscribe-exceptions**
Test Purpose	A **Publisher** shall raise Exceptions in accordance with Table 26 when executing the *Subscribe* operation, in addition to those specified in Section 8.3.4
Test Method	Execute the *Subscribe* operation with the following scenarios: 1. HeartbeatCriteria present with missing `heartbeatRate`, ensure that the response is a MissingParameterValue Exception with a `locator` value of "heartbeatRate" 2. HeartbeatCriteria present with the `heartbeatRate` parameter set to the value "42", ensure that the response is a InvalidParameterValue Exception with a `locator` value of

	“heartbeatRate”

A.7 Conformance class: Brokering Publisher

/conf/core/brokering-publisher	
Dependency	/conf/core/standalone-publisher
Requirements Class	/req/core/brokering-publisher

Test: /conf/core/brokering-publisher/registerpublisher

Requirement	**/req/core/brokering-publisher/registerpublisher**
Test Purpose	A **Publisher** shall offer the *RegisterPublisher* operation
Test Method	Execute the *RegisterPublisher* operation and ensure that the response is a valid RegisterPublisherResponse

Test: /conf/core/brokering-publisher/registerpublisher-connect

Requirement	**/req/core/brokering-publisher/registerpublisher-connect**
Test Purpose	When the *RegisterPublisher* operation is executed a **Publisher** shall retrieve the capabilities document of the registered Publisher and verify that it contains integrates `FilterCapabilities`, `DeliveryCapabilities`, and `Publications` sections before returning the RegisterPublisherResponse
Test Method	Execute the *RegisterPublisher* operation with a `capabilitiesReference` parameter that is resolvable to a valid capabilities document with `FilterCapabilities`, `DeliveryCapabilities`, and `Publications` sections, and ensure that the response is a valid RegisterPublisherResponse

Test: /conf/core/brokering-publisher/registerpublisher-exceptions

Requirement	**/req/core/brokering-publisher/registerpublisher-exceptions**

Test Purpose	A **Publisher** shall raise Exceptions in accordance with Table 28 when executing the *RegisterPublisher* operation
Test Method	Execute the *RegisterPublisher* operation with the following scenarios: 1. A `capabilitiesReference` parameter containing a URL that is not resolvable, and ensure that the response is an PublisherRegistrationFailed Exception 2. An empty request (request sent to the RegisterPublisher endpoint with no content), and ensure that the response is a NoApplicableCode Exception with an empty `locator` value

Test: /conf/core/brokering-publisher/removepublisher

Requirement	**/req/core/brokering-publisher/removepublisher**
Test Purpose	A **Publisher** shall offer the *RemovePublisher* operation
Test Method	Execute the *RegisterPublisher* operation and ensure that the response is a valid RegisterPublisherResponse, execute the *RemovePublisher* operation against the same `capabilitiesReference` parameter and ensure that the response is a valid RemovePublisherResponse

Test: /conf/core/brokering-publisher/removepublisher-exceptions

Requirement	**/req/core/brokering-publisher/removepublisher-exceptions**
Test Purpose	A **Publisher** shall raise Exceptions in accordance with Table 30 when executing the *RemovePublisher* operation
Test Method	Execute the *RemovePublisher* operation with the following scenarios: 1. A `capabilitiesReference` parameter containing a "http://ats.opengeospatial.org/invalid-capabilities-reference", and ensure that the response is an UnknownPublisher Exception 2. An empty request (request sent to the RemovePublisher endpoint with no content), and ensure that the response is a NoApplicableCode Exception with an empty `locator` value

Test: /conf/core/brokering-publisher/getcapabilities-registered-publishers

Requirement	**/req/core/brokering-publisher/getcapabilities-registered-publishers**
Test Purpose	A **Publisher** shall return a `RegisteredPublishers` as part of the PublisherCapabilities type as a result of the *GetCapabilities* operation
Test Method	Execute the *GetCapabilities* operation and ensure that the response is a valid Capabilities document with a PublisherCapabilities section with a RegisteredPublishers section.

A.8 Conformance class: Publication Manager

/conf/core/publication-manager	
Dependency	/conf/core/basic-publisher
Requirements Class	/req/core/publication-manager

Test: /conf/core/publication-manager/getcapabilities-processingcapabilities

Requirement	**/req/core/publication-manager/getcapabilities-processingcapabilities**
Test Purpose	A **Publisher** shall return a `ProcessingCapabilities` structure within its *GetCapabilities* response
Test Method	Execute a *GetCapabilities* operation against the service and verify that the service returns a Capabilities document with a `ProcessingCapabilities` section

Test: /conf/core/publication-manager/getcapabilities-unique-processinglanguage

Requirement	**/req/core/publication-manager/getcapabilities-unique-processinglanguage**
Test Purpose	A **Publisher** shall uniquely identify each offered `ProcessingLanguage` included in the `PublisherCapabilities`
Test Method	Execute a *GetCapabilities* operation on the service, ensure that every `ProcessingLanguage identifier` property in the

	`PublisherCapabilities` section is unique among all `ProcessingLanguage` identifiers

Test: /conf/core/publication-manager/publication-valid-processinglanguage

Requirement	**/req/core/publication-manager/publication-valid-processing-language**
Test Purpose	Each `supportedProcessingLanguage` of a `Publication` shall correspond to one of the `ProcessingLanguage` identifiers advertised in the `ProcessingCapabilities`
Test Method	Execute a *GetCapabilities* operation against the service and verify that each `supportedProcessingLanguage` identifier in each `Publication` section exactly matches a `ProcessingLanguage identifier`

Test: /conf/core/publication-manager/createpublication

Requirement	**/req/core/publication-manager/createpublication**
Test Purpose	The **Publisher** shall offer the *CreatePublication* operation
Test Method	Create a test publication with a publication identifier of "urn:pubsub:ats:BasePub". Execute the *CreatePublication* operation with the `basePublicationIdentifier` parameter set to "urn:pubsub:ats:BasePub" and the `identifier` parameter set to "urn:pubsub:ats:DerivedPub" and the `description` parameter set to "Test description", ensure that the response is a valid CreatePublicationResponse document

Test: /conf/core/publication-manager/createpublication-publication-id

Requirement	**/req/core/publication-manager/createpublication-publication-id**
Test Purpose	The **Publisher** shall raise an Exception if the `basePublicationIdentifier` specified in the *CreatePublication* request is not a member of the list of offered

	publications at the time the derived publication is created
Test Method	Execute the *CreatePublication* operation with the `basePublicationIdentifier` parameter set to "urn:pubsub:ats:InvalidBasePub" ensure that the response is a InvalidPublicationIdentifier Exception with a `locator` value of "basePublicationIdentifier"

Test: /conf/core/publication-manager/createpublication-assign-properties

Requirement	**/req/core/publication-manager/createpublication-assign-properties**
Test Purpose	The **Publisher** shall assign publication properties (`contentType`, `supportedFilterLanguage`, `supportedDeliveryMethod`, `boundingBox`, `formalContentDefinitionLanguage`, and `formalContentDefinition)` from the base publication to the created DerivedPublication when a derived publication is created, excepting the `identifier` and `description` properties
Test Method	Create a test publication with a publication identifier of "urn:pubsub:ats:BasePub". Execute the *GetCapabilities* operation, ensure a publication with an identifier of "urn:pubsub:ats:BasePub" exists in the Publications section, record the `contentType`, `supportedFilterLanguage`, `supportedDeliveryMethod`, `boundingBox`, `formalContentDefinitionLanguage`, and `formalContentDefinition` sections of the test publication. Execute the *CreatePublication* operation with the `basePublicationIdentifier` parameter set to "urn:pubsub:ats:BasePub" and the `identifier` parameter set to "urn:pubsub:ats:DerivedPub" and the `description` parameter set to "Test description", ensure that the response is a valid CreatePublicationResponse document, ensure that the `contentType`, `supportedFilterLanguage`, `supportedDeliveryMethod`, `boundingBox`, `formalContentDefinitionLanguage`, and `formalContentDefinition` sections exactly match those recorded from the "urn:pubsub:ats:BasePub" publication

Test: /conf/core/publication-manager/createpublication-assign-identifier

Requirement	**/req/core/publication-manager/createpublication-assign-identifier**
Test Purpose	The **Publisher** shall assign a unique `identifier` to the created DerivedPublication, when not specified in the *CreatePublication* request
Test Method	Execute a valid *CreatePublication* operation with a request missing the `identifier` parameter. Verify that the response includes a Publication whose `identifier` is unique among the publications of this Publisher

Test: /conf/core/publication-manager/createpublication-filter-id

Requirement	**/req/core/publication-manager/createpublication-filter-id**
Test Purpose	A **Publisher** shall raise an Exception if the *CreatePublication* request includes a `filter`, but does not include a `filterLanguageId`
Test Method	Execute a *CreatePublication* operation including in the request a `filter`, but not a `filterLanguageId`. Ensure that the response is a MissingParameterValue Exception with a locator value set to "filterLanguageId"

Test: /conf/core/publication-manager/createpublication-valid-filter-id

Requirement	**/req/core/publication-manager/createpublication-valid-filter-id**
Test Purpose	A **Publisher** shall raise an Exception if the *CreatePublication* request includes a `filterLanguageId` that does not correspond to a `supportedFilterLanguage` for the base publication
Test Method	Execute a *CreatePublication* operation including in the request a `filterLanguageId` that does not correspond to any `supportedFilterLanguage` for the publication. Ensure that the response is an InvalidParameterValue Exception with a locator value set to "filterLanguageId"

Test: /conf/core/publication-manager/createpublication-processinglanguage-id

Requirement	/req/core/publication-manager/createpublication-processinglanguage-id
Test Purpose	A **Publisher** shall raise an Exception if the *CreatePublication* request includes a `processingExpression`, but does not include a `processingLanguageId`
Test Method	Execute a *CreatePublication* operation including in the request a `processingExpression`, but not a `processingLanguageId`. Ensure that the response is a MissingParameterValue Exception with a locator value set to "processingLanguageId"

Test: /conf/core/publication-manager/createpublication-valid-processinglanguage-id

Requirement	/req/core/publication-manager/createpublication-valid-processinglanguage-id
Test Purpose	A **Publisher** shall raise an Exception if the *CreatePublication* request includes a `processingLanguageId` that does not correspond to a `supportedProcessingLanguage` for the publication
Test Method	Execute a *CreatePublication* operation including in the request a `processingLanguageId` that does not correspond to any `supportedProcessingLanguage` for the publication. Ensure that the response is an InvalidParameterValue Exception with a locator value set to "processingLanguageId"

Test: /conf/core/publication-manager/createpublication-exceptions

Requirement	/req/core/publication-manager/createpublication-exceptions
Test Purpose	A **Publisher** shall raise Exceptions in accordance with Table 36 when executing the *CreatePublication* operation
Test Method	Execute the *CreatePublication* operation with the following scenarios: 1. A `basePublicationIdentifier` parameter set to "urn:pubsub:ats:InvalidPublication", and ensure that an InvalidPublicationIdentifier Exception is returned with a `locator` value of "urn:pubsub:ats:InvalidPublication" 2. A `filter` parameter containing the text "Invalid filter", and ensure that the response is an InvalidFilter Exception

	3. A `identifier` parameter set to the value "Not A URI", ensure that the response is a InvalidParameterValue Exception with a `locator` value of "publicationIdentifier" 4. An empty request (request sent to the CreatePublication endpoint with no content), and ensure that the response is a NoApplicableCode Exception with an empty `locator` value

Test: /conf/core/publication-manager/removepublication

Requirement	**/req/core/publication-manager/removepublication**
Test Purpose	The **Publisher** shall offer the *RemovePublication* operation
Test Method	Execute the *CreatePublication* operation with the `basePublicationIdentifier` parameter set to a test publication identifier, ensure that the response is a valid CreatePublicationResponse document, execute the *RemovePublication* operation against the newly-created Publication, and ensure that the response is a valid RemovePublicationResponse

Test: /conf/core/publication-manager/removepublication-nesting

Requirement	**/req/core/publication-manager/removepublication-nesting**
Test Purpose	The **Publisher** shall raise an Exception if the *RemovePublication* operation specifies a publication that is an active base publication for one or more derived publications
Test Method	Create a test (base) publication with a publication identifier of "urn:pubsub:ats:BasePublication". Execute the *CreatePublication* operation with the `basePublicationIdentifier` parameter set to "urn:pubsub:ats:BasePublication" and an `identifier` parameter set to "urn:pubsub:ats:DerivedPublication". Execute the *CreatePublication* operation with the `basePublicationIdentifier` parameter set to "urn:pubsub:ats:DerivedPublication" and an `identifier` parameter set to "urn:pubsub:ats:NestedDerivedPublication". Execute the RemovePublication operation with the `publicationIdentifier` parameter set to "urn:pubsub:ats:DerivedPublication", and ensure that an InvalidParameterValue Exception is returned with a `locator` value of "publicationIdentifier"

Test: /conf/core/publication-manager/removepublication-base-publication-removal

Requirement	**/req/core/publication-manager/removepublication-base-publication-removal**
Test Purpose	The **Publisher** shall raise an Exception if the publicationIdentifier parameter to the *RemovePublication* operation specifies a publication that is not a derived publication
Test Method	Create a test (base) publication with a publication identifier of "urn:pubsub:ats:BasePublication". Execute the RemovePublication operation with the `publicationIdentifier` parameter set to "urn:pubsub:ats:BasePublication", and ensure that an InvalidParameterValue Exception is returned with a `locator` value of "publicationIdentifier"

Test: /conf/core/publication-manager/subscribe-derived-publications

Requirement	**/req/core/publication-manager/subscribe-derived-publications**
Test Purpose	The **Publisher** shall perform DerivedPublication message matching and message delivery on messages that match on the base publication, but filtered by any filters on the DerivedPublication
Test Method	Create a test (base) publication with a publication identifier of "urn:pubsub:ats:BasePublication". Execute the *CreatePublication* operation with the `basePublicationIdentifier` parameter set to "urn:pubsub:ats:BasePublication" and an `identifier` parameter set to "urn:pubsub:ats:DerivedPublication". Subscribe to both "urn:pubsub:ats:BasePublication" and "urn:pubsub:ats:DerivedPublication" and ensure that messages delivered on the base publication are also delivered to the derived publication.

Test: /conf/core/publication-manager/derived-publication-identifiers

Requirement	**/req/core/publication-manager/derived-publication-identifiers**
Test Purpose	The **Publisher** shall accept DerivedPublication identifiers as valid publication identifiers to all Publish/Subscribe operations (e.g., the *Subscribe* operation) and include DerivedPublications among publication results (e.g., the *GetCapabilities* operation)

Test Method	Create a test (base) publication. Execute the *CreatePublication* operation with the `basePublicationIdentifier` parameter set to the test publication to create a derived publication. Execute a *Subscribe* operation against the derived publication and ensure messages are delivered. Execute the *GetCapabilities* operation and ensure that the Publications section of the response includes both the base publication and derived publication identifiers.

Test: /conf/core/publication-manager/removepublication-exceptions

Requirement	**/conf/core/publication-manager/removepublication-exceptions**
Test Purpose	A **Publisher** shall raise Exceptions in accordance with Table 38 when executing the *RemovePublication* operation
Test Method	Execute the *RemovePublication* operation with the following scenarios: 1. A single `publicationIdentifier` parameter set to "urn:pubsub:ats:InvalidPublication", and ensure that an InvalidPublicationIdentifier Exception is returned with a `locator` value of "urn:pubsub:ats:InvalidPublication" 2. A single `publicationIdentifier` parameter containing the text "Not a URI", and ensure that the response is an InvalidParameterValue Exception with a `locator` value of "publicationIdentifier" 3. An empty request (request sent to the CreatePublication endpoint with no content), and ensure that the response is a NoApplicableCode Exception with an empty `locator` value

A.9 Conformance class: Capabilities Filtering

/conf/core/capabilities-filtering-publisher	
Dependency	/conf/core/standalone-publisher
Requirements Class	/req/core/capabilities-filtering-publisher

Test: /conf/core/capabilities-filtering-publisher/getcapabilities-content-sort

Requirement	/req/core/capabilities-filtering-publisher/getcapabilities-content-sort
Test Purpose	A **Publisher** shall impose a consistent sort order on the items listed in the `Publications` section. The sorting methodology is not specified by this Standard, but GetCapabilities responses shall present a consistent order between GetCapabilities requests, regardless of filtering criteria
Test Method	Execute the *GetCapabilities* operation without any capabilities filtering parameters. Execute the GetCapabilities operation with a `bbox` parameter that returns at least three results. Ensure that the order is the same between the contents of the Publications section is consistent between requests (ignoring filtered contents)

Test: /conf/core/capabilities-filtering-publisher/getcapabilities-content-filter

Requirement	/req/core/capabilities-filtering-publisher/getcapabilities-content-filter
Test Purpose	A **Publisher** shall filter the items in the `Publications` section of the Capabilities response in accordance with Clause 15.2 when the parameters from Table 39 are provided in the request
Test Method	Execute the *GetCapabilities* operation without any capabilities filtering parameters. Record the contents of the Publications section. Execute the *GetCapabilities* operation with the following scenarios: - A `searchTerms` parameter with a single term that is contained in a single advertised publication, ensure the response Publications section contains a single Publication - A `bbox` parameter that encompasses a single advertised publication, ensure the response Publications section contains a single Publication - A `count` parameter that is set to the value “1”, ensure the response Publications section contains only the first Publication - A `count` parameter that is set to the value “1” and a `startIndex` parameter set to the value “1”, ensure the response Publications section contains only the second advertised Publication

Test: /conf/core/capabilities-filtering-publisher/getcapabilities-search

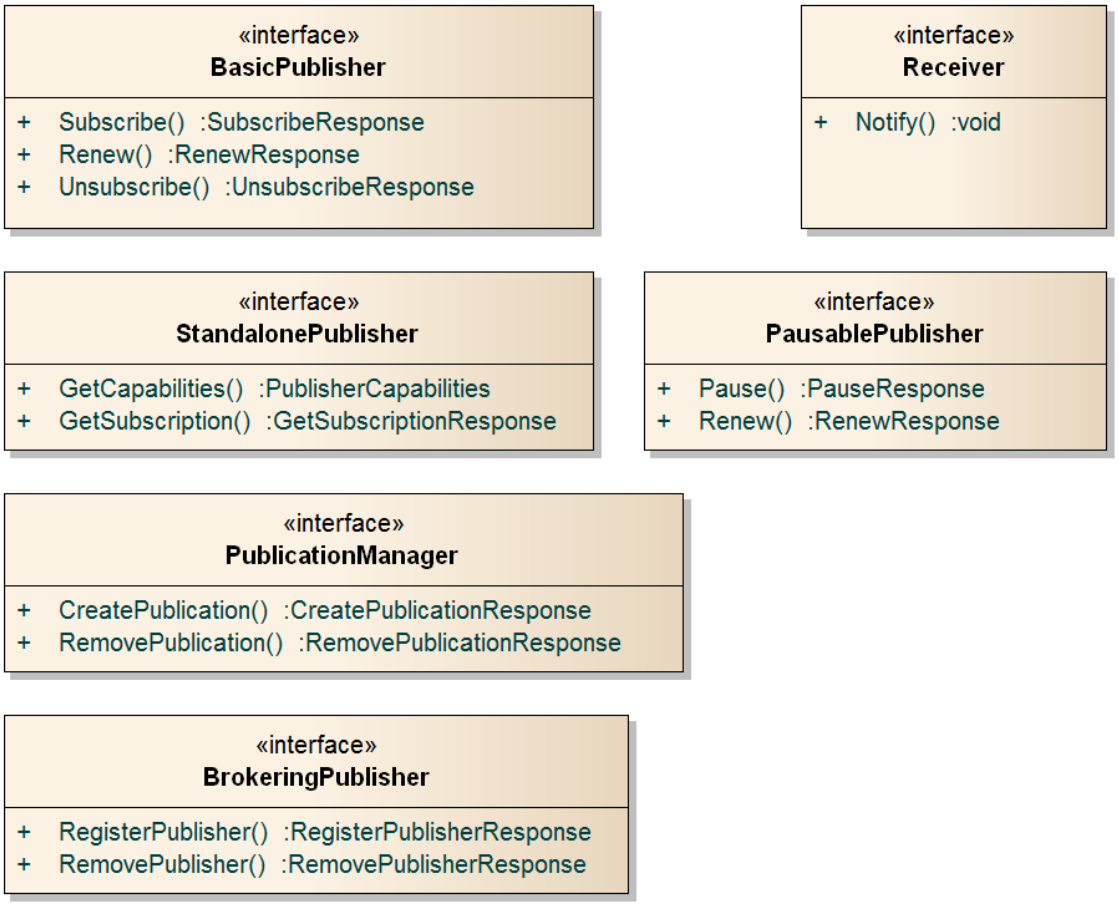
Requirement	**/req/core/capabilities-filtering-publisher/getcapabilities-search**
Test Purpose	When a **Publisher** receives a GetCapabilities request that causes the `Publications` section to be excluded from the response, the Publisher shall ignore any of the parameters defined in Table 39
Test Method	Execute the *GetCapabilities* operation with a sections parameter set to "ServiceIdentification" and the `count` parameter set to "1", ensure that the response is a valid document with a ServiceIdentification section.

Test: /conf/core/capabilities-filtering-publisher/getcapabilities-exceptions

Requirement	**/req/core/capabilities-filtering-publisher/getcapabilities-exceptions**
Test Purpose	A **Publisher** shall raise Exceptions in accordance with Table 40 when executing the *GetCapabilities* operation, in addition to those specified in Clause 9.1.3
Test Method	Execute the *GetCapabilities* operation with the following scenarios: - A `count` parameter with the value "-1", ensure that the response is an InvalidParameterValue Exception with a `locator` value of "count"

Annex B. Publish/Subscribe Interfaces (Informative)

This standard defines operations that can be combined in interfaces as follows:



Annex C. Revision history

Date	Release	Editor	Paragraph(s) modified	Description
2013-07-25	1.0-RC0	Aaron Braeckel, Lorenzo Bigagli, Johannes Echterhoff	All	First draft for internal SWG review
2013-12-17	1.0-RC1	Aaron Braeckel, Lorenzo Bigagli, Johannes Echterhoff	All	Incorporated comments from PubSub SWG review Added Basic Receiver
2015-06-26	1.0-RC2	Aaron Braeckel, Lorenzo Bigagli	All	Incorporated edits resulting from the SOAP Binding draft Second draft for internal SWG review
2015-07-31	1.0-RC3	Aaron Braeckel, Lorenzo Bigagli	All	Revised URIs, revised figures in BasicPublisher
2015-09-08	1.0-RC4	Aaron Braeckel, Lorenzo Bigagli	All	Incorporated comments from OAB review in preparation for public comment
2015-12-07	1.0-RC5	Aaron Braeckel, Lorenzo Bigagli	All	Incorporated changes related to feedback from public comments in preparation for adoption vote
2016-02-12	1.0	Aaron Braeckel, Lorenzo Bigagli	Front page, Abstract	Incorporated comments received during adoption vote Finalisation
2016-03-31	1.0	Scott Simmons	All	Minor edits, preparation for publication