Open Geospatial Consortium Inc.

Date: 2007-05-14

Reference number of this document: **OGC 06-028r5**

Version: 0.9.0

Category: Candidate OpenGIS® Interface Standard

Editor: Ingo Simonis
Co-Editor: Johannes Echterhoff

OGC® Sensor Alert Service Implementation Specification

Warning

This document is not an OGC Standard. It is distributed for review and comment. It is subject to change without notice and may not be referred to as an OGC Standard.

Recipients of this document are invited to submit, with their comments, notification of any relevant patent rights of which they are aware and to provide supporting documentation.

Document type:	Candidate OpenGIS® Interface Standard
Document subtype:	Candidate Standard
Document stage:	Request for Comment
Document language:	English

Contents

Page

Figures

Page

Tables

Page

i. Preface

Suggested additions, changes, and comments on this draft report are welcome and encouraged. Such suggestions may be submitted by email message or by making suggested changes in an edited copy of this document.

The changes made in this document version, relative to the previous version, are tracked by Microsoft Word, and can be viewed if desired. If you choose to submit suggested changes by editing this document, please first accept all the current changes, and then make your suggested changes with change tracking on.

ii. Document terms and definitions

This document uses the specification terms defined in Subclause 5.3 of [OGC 05-008], 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 specification

iii. Submitting organizations

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

University of Muenster

Oak Ridge National Laboratory

3eti

iv. Document contributor contact points

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

Name	Organization
Mark Priest	3eti
Ingo Simonis (Editor)	Geospatial Research & Consulting
Johnny Tolliver	Oak Ridge National Laboratory
Alexander Walkowski	University of Muenster

Johannes Echterhoff (Co-Editor)	University of Muenster

v. Revision history

Date	Release	Editor	Primary clauses modified	Description
2005-12-28	0.1.0	Ingo Simonis		Initial version
2006-01-03	0.1.1	Alexander Walkowski	All	Examples added and general revision
2006-02-12	0.2.0	Ingo Simonis	All	Final changes to submit this version in time of the three week rule for the OGC TC meeting March 2006.
2006-05-13	0.3.0	Ingo Simonis	All	SAS redesigned to fulfil the requirements set in SAS IE.
2007-01-26	1.0.0	Johannes Echterhoff	All	Revised examples and full text.
2007-01-30	1.0.0	Ingo Simonis	All	Edits prior to release into RFC process
2007-05-10	1.0.0	Johannes Echterhoff	All	Revised schema, examples and full text; added Acknowledgement mechanism; added differentiation between user- and sensor-defined alerts
2007-06-15	1.0.0	Ingo Simonis	All	Final edits prior to release into RFC process
2007-10-24	1.0.0	Ingo Simonis	Annex A, Section 22	added

vi. Changes to the OGC Abstract Specification

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

vii. Future work

This work depicts the initial version of the Sensor Alert Service. Right now, the specification provides only limited capabilities for defining filter criteria during subscription. Emerging technologies like event stream processing (ESP) and complex event processing (CEP) may be suited to provide a much more powerful way to define filter queries on the incoming sensor data streams. Future work should investigate how ESP / CEP can be incorporated by SAS.

There is no mechanism defined for SAS to automatically store the incoming data streams so that the data can later be retrieved by clients or in filter queries. The connection between SAS and SOS should therefore be investigated in the future, especially taking into account that automatic registration of new sensors at SOS is possible and that scalability of the service is unharmed.

XMPP credentials are provided during advertisement of sensors, so that plug-and-play of sensors can be achieved. However, at the moment this feature is not available for clients. Future use cases might find the need for the provision of XMPP credentials for the client side as well.

WS-Notification is an OASIS product that seems to have overlapping functionalities with SAS. Future versions of SAS shall be harmonized therefore (further information on WS-Notification can be found at [6]).

The Sensor Alert Service defines an optional acknowledgement mechanism. Alert consumers may register for reliable communication with the producer, which allows consumers to acknowledge each alert that has been received. However, the handling of unacknowledged alerts is not covered by this specification, because that may be highly implementation dependent. For example, the producer might resend unacknowledged alerts or a client might want to receive a batch of the data he missed so far upon request. The handling of unacknowledged alerts has to be specified in future versions of this specification. It is anticipated that best practices show in which way this can be handled.

This specification defines a rather complex service. It includes functionalities that are not required by a number of scenarios, e.g. the option to acknowledge incoming or outgoing messages. Future versions of the SAS specification will be separated into a core specification that lists mandatory functionalities only. All additional components will be covered by extensions.

Foreword

The Sensor Alert Service is the newest part of the OGC Sensor Web Enablement document suite.

This document includes four annexes; Annexes A and B are normative, and Annexes C and D are informative.

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

Introduction

OGC's Sensor Web Enablement (SWE) activity, which is being executed through the OGC Web Services (OWS) initiatives (under the Interoperability Program) and the SWE Working Group (under the Specification Program), is establishing the interfaces and protocols that will enable a "Sensor Web" through which applications and services will be able to access sensors of all types over the Web. These initiatives have defined, prototyped and tested several foundational components needed for a Sensor Web, namely:

a) Sensor Model Language (SensorML)

b) Transducer Markup Language (TML)

c) Observations & Measurements (O&M)

d) Sensor Observation Service (SOS)

e) Sensor Planning Service (SPS)

f) Sensor Alert Service (SAS)

g) Web Notification Service (WNS)

This document specifies Sensor Alert Service (SAS). The other components are specified under separate cover. The SAS can be compared with an event notification system. The sensor node is the object of interest. Each node has to advertise its publications at a SAS (*advertise*).

Note: Actually, it is the "alerts" that a sensor can send that shall be registered rather than the node/sensor itself.

It is important to notice that both sensors that just send the current observation value as an alert as well as those sensors that are able to send alerts like "above a HIGH threshold" or "below a LOW threshold" can register at the SAS. This allows more "clever" sensors (i.e. sensors that can decide on their own if a predefined condition is matched) to be attached to SAS as well as those that only send current observations. There is no difference made between those types of sensors! If the sensor just sends its current observation value, it is up to the SAS to check its subscription table if a condition set by a user is matched, e.g. the current value is above or below a user defined threshold. The same applies to those alerts that are sent from sensors that can decide on their own if a threshold crossing or a specific condition (e.g. „battery is low") has occurred.

Traditional OGC web services are not suitable for implementing this alert service. The Sensor Alert Service uses the Extensible Messaging and Presence Protocol (XMPP) to provide the push-based notification functionality.

The Extensible Messaging and Presence Protocol (XMPP) is the IETF's formalization of the base XML streaming protocols for instant messaging and presence developed within the Jabber community starting in 1999. As specified in RFC 3920 [7], the core "transport" layer for XMPP is an XML streaming protocol that makes it possible to exchange fragments of XML between any two network endpoints. Authentication and channel encryption happen at the XML streaming layer using the IETF-standard protocols for Simple Authentication and Security Layer [5] and Transport Layer Security [2]. The normal architecture of XMPP is a pure client-server model, wherein clients connect to servers and (optionally) servers connect to each other for interdomain communications. XMPP addresses are fully internationalized, and are of the form <node@domain> for clients (similar to email) [13].

In an early draft of this specification, it was mentioned that although the XMPP protocol was widely used within the specification, it should be emphasized that the SAS specification should be agnostic regarding the used alert protocol. We abandoned the generic approach in favour to provide a more specific specification; thus achieving a higher level of interoperability between SAS implementations.

OpenGIS® Sensor Alert Service Implementation Specification

1 Scope

This OpenGIS© document specifies interfaces for requesting information describing the capabilities of a Sensor Alert Service, for determining the nature of offered alerts, the protocols used, and the options to subscribe to specific alert types.

2 Conformance

Conformance with this specification shall be checked using all the relevant tests specified in Annex A (normative).

3 Normative references

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

ISO 19105:2000, *Geographic information — Conformance and Testing*

OGC 05-008, *OpenGIS® Web Services Common Specification* [12]

OGC 06-095, *OpenGIS® Web Notification Service Specification* [9]

OGC 07-000, *OpenGIS® Sensor Model Language (SensorML) Specification* [1]

In addition to this document, this specification includes several normative XML Schema Document files as specified in Annex B.

4 Terms and definitions

For the purposes of this specification, the definitions specified in Clause 4 of the OWS Common Implementation Specification [OGC 05-008] shall apply. In addition, the following terms and definitions apply.

Alert

An alert is a special kind of notification indicating that an event has occurred at an object of interest, which results in a condition of heightened watchfulness or preparation for action. Alerts always contain a time (from the underlying event) but not necessarily a location value.

Event

An event is an action that occurs at an instant or over an interval of time [4], [3]. Based upon this definition, we define the action and outcome of measuring data at a certain point in time as an event. So what a sensor is producing are events.

Note: whenever a sensor itself performs some intelligent filtering to find out which events are of interest or importance, this is what we call an alert.

Notification

A message sent to a receiver indicating the occurrence of an event.

Message

A message in its most general meaning is an object of communication. In the context of SAS, the term applies to both the information contents and its actual presentation (form). A message contains an alert in its body.

5 Conventions

5.1 Abbreviated terms

Most of the abbreviated terms listed in Subclause 5.1 of the OWS Common Implementation Specification [OGC 05-008] apply to this document, plus the following abbreviated terms.

MUC	Multi User Chat
SAS	Sensor Alert Service
SensorML	Sensor Model Language
SOS	Sensor Observation Service
SWE	Sensor Web Enablement
WNS	Web Notification Service
WSDL	Web Services Description Language

5.2 UML notation

Most diagrams that appear in this specification are presented using the Unified Modeling Language (UML) static structure diagram, as described in Subclause 5.2 of [OGC 05-008].

5.3 XMLSpy notation

Most diagrams that appear in this specification are presented using an XML schema notation defined by the XMLSpy[1] product and described in this subclause. XML schema

[1] XML Spy: http://www.altova.com

diagrams are for informative use only though they shall reflect the accompanied UML and schema perfectly.

5.3.1 Element

A named rectangle represents the most basic part of the XML Schema notation. Each represents an XML "Element" token. Each Element symbol can be elaborated with extra information as shown in the examples below.

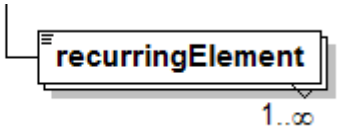
This is a mandatory simple element. Note the upper left corner of the rectangle indicates that data is contained in this element.

5.3.2 Optional Element

Optional (non mandatory) elements are specified with dashed lines used to frame the rectangle.

5.3.3 Recurring Element

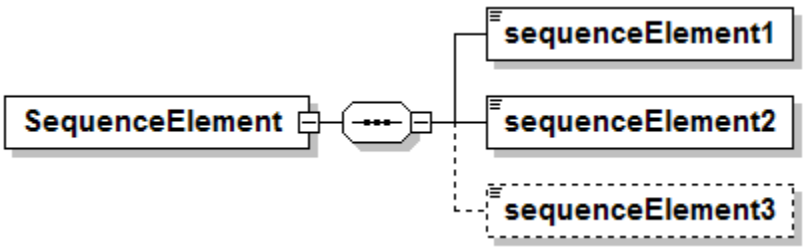
This element (and its child elements if it has any) can occur multiple times.



This example shows a recurring element that shall occur at least once but can occur an unlimited amount of times. The upper bound here is shown with the infinity symbol.

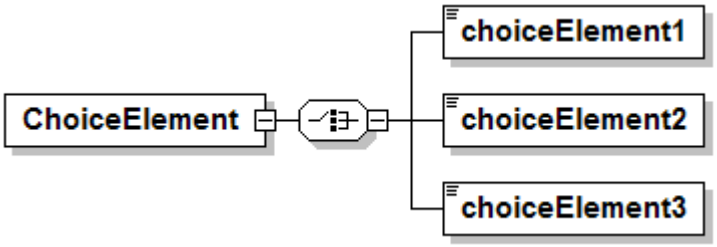
5.3.4 Sequence Connector

The connection box, called a sequence indicator, indicates that the "SequenceElement" data is made up of three elements. In this example, the first two elements are mandatory and the third element is optional



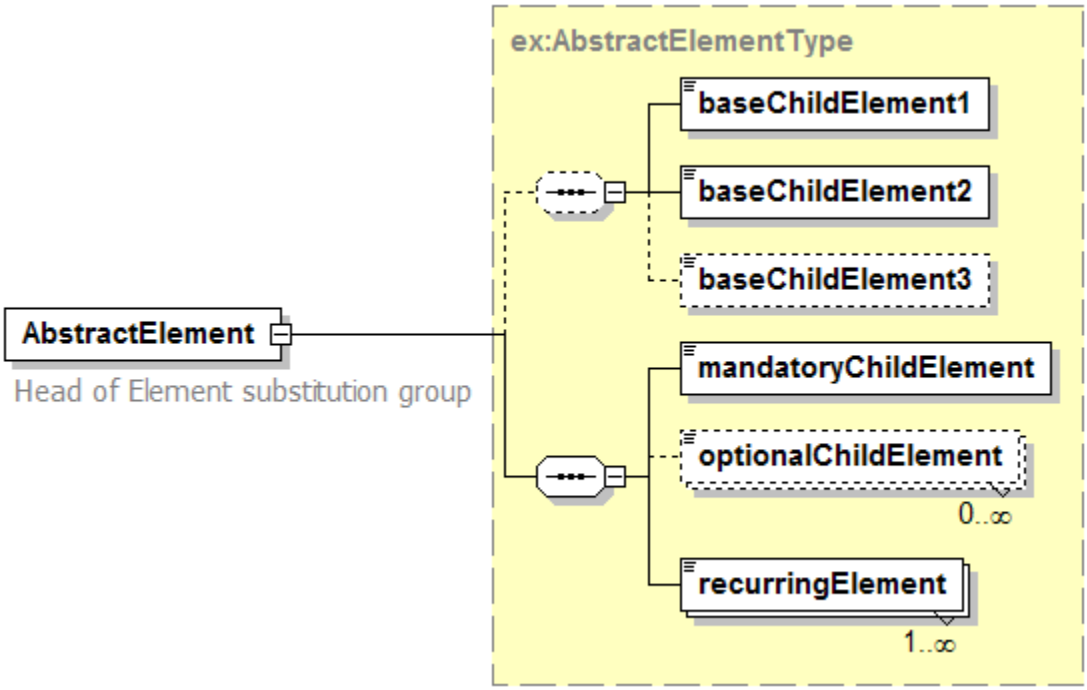
5.3.5 Choice Connector

The connection box here is a "choice" indicator, indicating that there is always going to be exactly one of the child elements listed on the right.



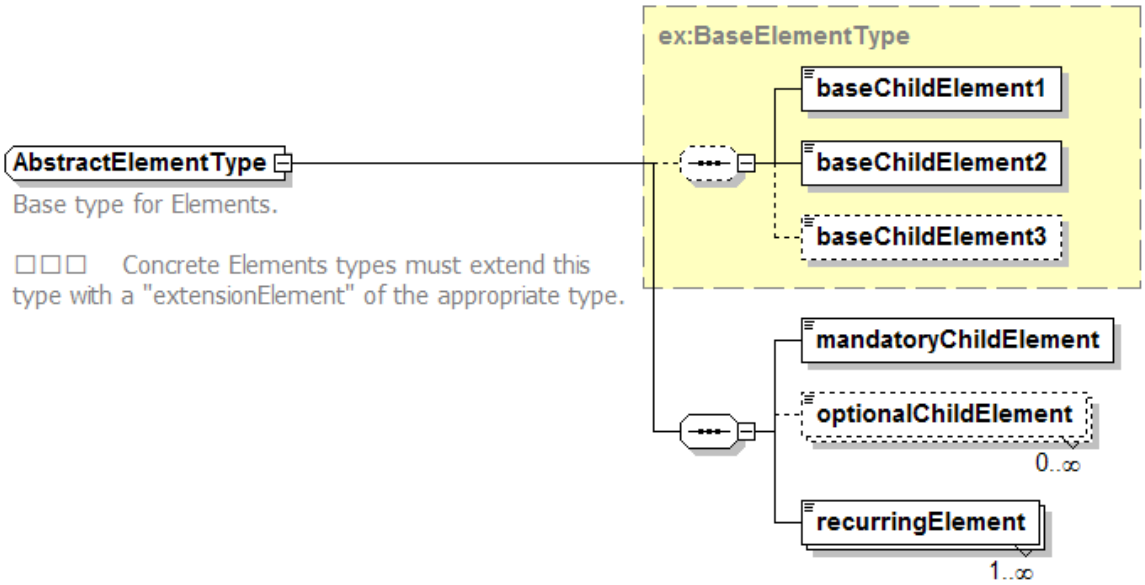
5.3.6 Definition with Complex Type

This diagram illustrates the use of a complex type (i.e., "ex:AbstractElementType") for defining an XML element (e.g., "AbstractElement").



5.3.7 Complex Type

This diagram illustrates the definition of a complex type (i.e., "AbstractElementType"), extending another complex type (i.e.,"ex:BaseElementType") with three additional elements. Complex types can be reused to specify that different elements are of the same type.



5.4 Used parts of other documents

This document uses significant parts of document [OGC 05-008]. To reduce the need to refer to that document, this document copies some of those parts with small modifications. To indicate those parts to readers of this document, the largely copied parts are accompanied by a NOTE.

5.5 Platform-neutral and platform-specific specifications

As specified in Clause 10 of OGC Abstract Specification Topic 12 “OpenGIS Service Architecture” (which contains ISO 19119), this document includes both Distributed Computing Platform-neutral and platform-specific specifications. This document first specifies each operation request and response in platform-neutral fashion. This is done using a table for each data structure, which lists and defines the parameters and other data structures contained. These tables serve as data dictionaries for the UML model, and thus specify the UML model data type and multiplicity of each listed item.

The specified platform-neutral data could be encoded in many alternative ways, each appropriate to one or more specific distributed computing platforms (DCPs). This document now specifies encoding appropriate for use of HTTP GET transfer of operations requests (using key-value-pair (KVP) encoding), and for use of HTTP POST transfer of operation requests (using XML encoding).

6 SAS overview

The specified SAS defines an interface that allows nodes to advertise and publish observational data or alerts and corresponding metadata respectively. It allows clients to

subscribe for this data – or any other data that is produced by the SAS based on incoming messages from sensors – within specific thresholds. Observational data sent from a sensor is referred in this specification as *sensorData* or simply *data*. This data might be a single observation result, a complex observation result or even an *alert* in its nature.

Examples of the three cases are:

- Single observation: <5>
- Complex observation: <5, 23, null, true>
- Alert data: <Battery_Is_Low>

In the following we will not differentiate between *alert* and *observational data* as sent by sensors. It is usually application specific to assess the content of a message as being an alert or purely informative. However, data sent from the SAS to the client will be referred herein as alerts, even though this data might be considered as "data that matches my criteria", rather than an alert in its colloquial meaning.

For performance reasons, SAS instances make use of two different protocols: HTTP and XMPP. Thus the advertise/publish process on the sensor side as well as the subscribe process on the client side are in fact two steps processes. In fine, a sensor advertises its future sensorData on HTTP, receives the response on HTTP and uses XMPP subsequently to publish its data. The client sends its subscription request on HTTP, receives the response with necessary XMPP specific content (i.e., the address of the XMPP channel he has to join) on HTTP and makes use of XMPP in order to receive alert messages from the SAS.

The following figure illustrates a high level view on the SAS and the protocols used at the different steps. Please note that all diagrams presented in this section have been created to support a better understanding on the general functionality of the SAS and therefore do not show all operation details. For a detailed description of every operation please consult the appropriate sections in this document.

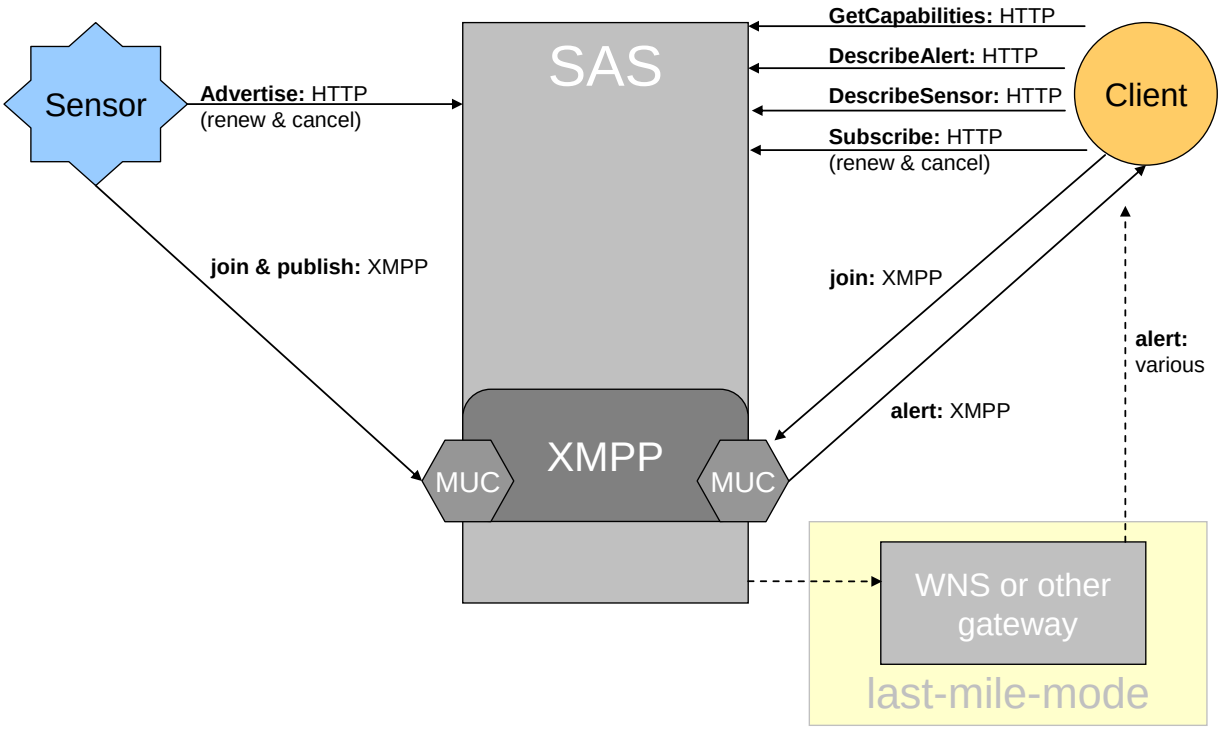


Figure 1: SAS overview

Sensors or other data producers (like preceding SAS services in a service chain) do advertise their offers to the Sensor Alert Service using HTTP POST requests. The SAS instructs its associated XMPP Messaging Server to create a new Multi User Chat (MUC) [8] in case that this or a similar sensor has not advertised the same advertisement before. Otherwise the SAS will simply provide an existing MUC without opening a new one via the Messaging Server. Internally a SAS contains some kind of look up table to store the information about existing MUCs. It is implementation dependent if the SAS decides to open a new MUC or use an existing one; thus it is transparent to the sensor. Theoretically, a SAS instance may use a single MUC for all types of sensors wherever located or whatever measuring. In this case, the amount of necessary filtering simply increases steeply, as all incoming messages have to be scanned before any forwarding to clients can appear.[2]

Note: Computationally, using a single MUC might be very expensive. This applies in particular to MUC for alert messages that might not require further processing by SAS, e.g. a MUC for "battery is low" messages sent by a sensor. Here, sensor and client better share a single MUC.

When the SAS receives a valid advertisement request from a sensor, he will determine a MUC-address as described above and return it to the sensor. This communication is

[2] Please note that the differentiation between Messaging Server and SAS is purely logical! This means that Messaging Server and SAS may work on the same machine, both of them may even work in the very same memory space.

based on HTTP. The sensor will then join this MUC to be able to publish data. This registration is in fact a subscription to the MUC and uses XMPP.[3]

On the client side (the client might be a human user or a machine, even another SAS), the client learns about the capabilities of a SAS by sending a `GetCapabilities` request on HTTP. The HTTP-based `GetCapabilities` response basically contains all information that is of interest for subscribers and was provided by the sensor in its advertisement, plus SAS controlled *SubscriptionOfferingIDs.* A SAS may integrate any number of advertisements into a single *SubscriptionOfferingID*. This means that on the one hand, multiple sensors that share the same data format (e.g. if they are of the same type) can be aggregated under such an ID. On the other hand, the service might process advertised data (either by itself or at another service) and offer the higher value data as a new sensor under a SubscriptionOffering. For example, sensor data like temperature, precipitation and pressure can be processed to provide "severe weather situation" alerts. For the client it does not matter whether the sensors listed in the Capabilities document of the service have actually been advertised or are instances of virtual sensors, provided and administered by SAS.

To subscribe at the service, the client first has to send an HTTP based subscribe request. The response will contain the address of an XMPP MUC. After that, he has to join that MUC, which is XMPP specific. There is one exception to this sequence of operations: in case the client wants to be informed via another communication channel (i.e. an alert will not be sent using XMPP, but on any other supported protocol, such as e.g. E-mail, SMS or phone calls). In this case exclusively, the subscription process terminates after the first HTTP based request/response. Instead of a MUC, the SAS will reply with a status message, indicating that the subscription has been registered successfully. We call this modus operandi "last-mile-mode".

Note: In "last mile mode", a SAS will not forward messages to an outgoing MUC, but will send the message via another protocol. Since a WNS can be used under the hood for providing this kind of functionality, all examples in this chapter mention WNS for last-mile-mode. In fact a SAS does not have to make use of a WNS service at all, but uses any other communication gateway if necessary.

Now that the subscription process has been completed, the client will receive alerts that match the subscription criteria. The data contained in each alert is given in a generic sensor format. For example, sensor *A* may produce data records with a temperature value followed by relative humidity in binary encoding while sensor *B* provides the same data just the other way round (i.e., first relative humidity, then temperature) and in plain XML. To solve this problem of heterogeneous data formats, each sensor has to provide a description of the format in which its data is encoded. This mechanism allows to bind previously unknown sensors at runtime. A client learns all necessary information about the structure of the alert data via the `DescribeAlert` operation and metainformation about the sensor itself via the `DescribeSensor` operation.

3 In fact, the registration process does not have to be initiated by the sensor itself. In particular simple sensors will only be able to publish data, but not to advertise themselves. In this case, the registration process has to be run externally and the specific MUC address has to be transferred to the sensor manually.

The following figures will illustrate the entire SAS usage cycle exemplarily. Imagine a river segment that is observed by a sensor. The sensor has a certain area of operation. It advertises its capabilities to the SAS and a client will make use of it.

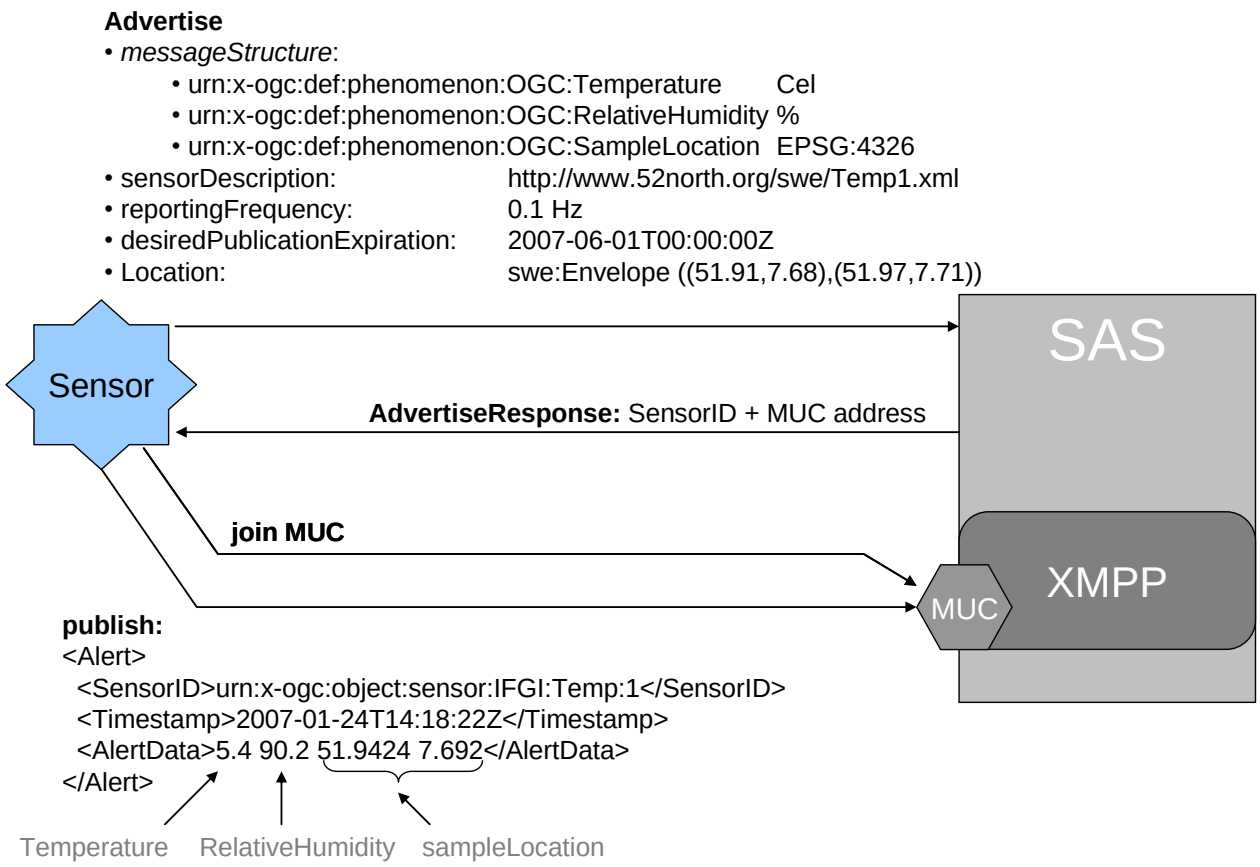


Figure 2: SAS example part 1, advertise

We see that the advertisement contains the following information:

- Message structure: the message consists of any number of fields. We can see that the data generated by the sensor contains a temperature and relative humidity value as well as the sampling location. See section 7 “Alerts” for further information.
- Sensor description: provides an overview of the sensors capabilities encoded in SensorML.
- Reporting frequency: indicates how often the sensor is creating alerts. If that frequency is unpredictable, a value of 0 (default) shall be provided.
- Desired publication expiration: point in time the sensor will not provide alerts anymore. See section 10.2 for further information.
- Location: position of the sensor or its operational area (given as swe:Position or swe:Envelope)

Whenever the sensor sends new data, the SensorID and time of the measurement is included, along with the measurement itself.

The client has to explore the capabilities of the SAS. As mentioned before, it explores those by issuing a GetCapabilities request, which is HTTP based. The following figure illustrates the first communication steps between a client and the SAS.

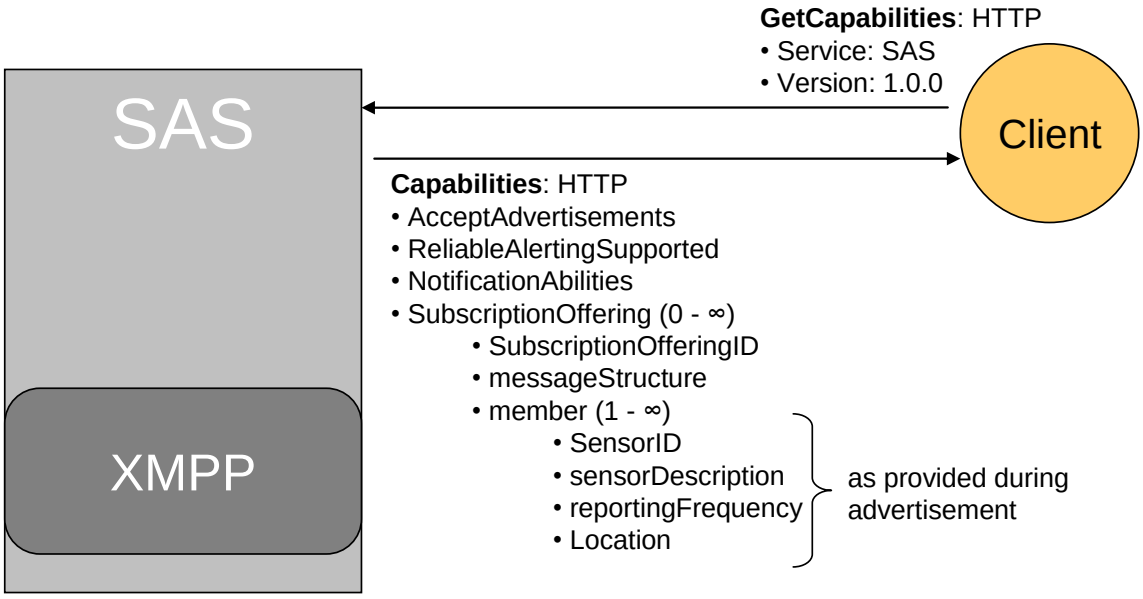


Figure 3: SAS example part 2, GetCapabilities

The only information the client has to send to the Web service is the service identifier and the version tag. The SAS will respond by sending a HTTP based XML encoded Capabilities document. This document contains the following information:

- AcceptAdvertisements: indicates if the SAS supports advertisement of new sensors.
- ReliableAlertingSupported: indicates if alert receivers can send acknowledgements. See section 8 for further information.
- NotificationAbilities: shows which communication channels other than direct XMPP are supported by the SAS for delivering alerts to subscribed clients.
- SubscriptionOfferings: once a sensor has been advertised, the service will assign it to a subscription offering, publishing the data which has been provided during advertisement and which is useful for clients.

As we can see, most of the information is simply forwarded from the sensor to the client. If sensors share the same message structure the SAS may group them under the same SubscriptionOffering, see section 11.3.3 for further information.

Last we will have a close look at the subscription and alerting mechanism as illustrated in the next figure.

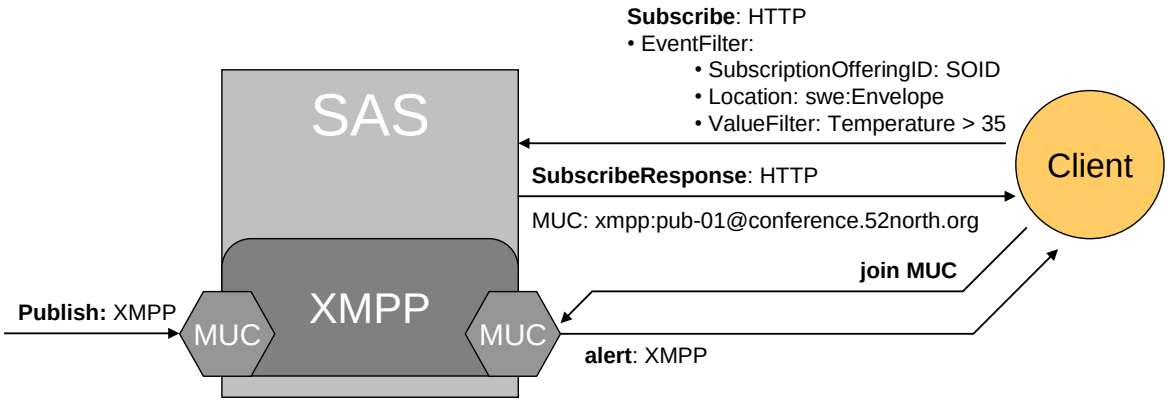


Figure 4: SAS example part 3, subscription and alerting – user defined alerts

As mentioned before, the client sends a HTTP based and XML encoded subscribe request to the SAS. Though dependent on the SAS implementation, in general the SAS will check some kind of internal MUC registry to find out if a corresponding MUC is already existent. That check would require that the service determines whether some kind of previously configured MUC already exists or somebody else has subscribed with the same parameters before. Either way, the SAS will return a MUC at which the alert messages will be sent. If the MUC is not existent at this stage, it will be created dynamically.

The client will then register at this MUC and is now "on air"; so all alert messages sent to this MUC will be received by the client. Again, it is implementation specific how the combination of SAS and Messaging Server maintains its MUCs. The XMPP specification defines *isAlive* messages to detect abandoned MUCs.

If a sensor is publishing data, the SAS will receive this data at one MUC, check the registered conditions and forward the alert message to those MUCs where the condition is fulfilled. Note that the SAS delivers sensor data "as is" if it matches the filter criteria. The service does not shorten the alert data, but provides its full content. For example, if the client is only interested in temperature but an alert also contains relative humidity, the values of both phenomena will be sent to the client. The receiving application should be smart enough to find out which parts of an alert are of interest, especially because it can always determine the actual data structure via the `DescribeAlert` operation.

Performing subscription in this way is what we call subscribing for user-defined alerts. The user defines the conditions that shall hold true for all alerts sent to him by the service. Recognizing the fact that some users are interested in every event generated by a sensor and that some sensors are smart enough to determine for themselves whether an interesting condition or dangerous situation occurred or not, we also allow subscribing for sensor defined alerts.

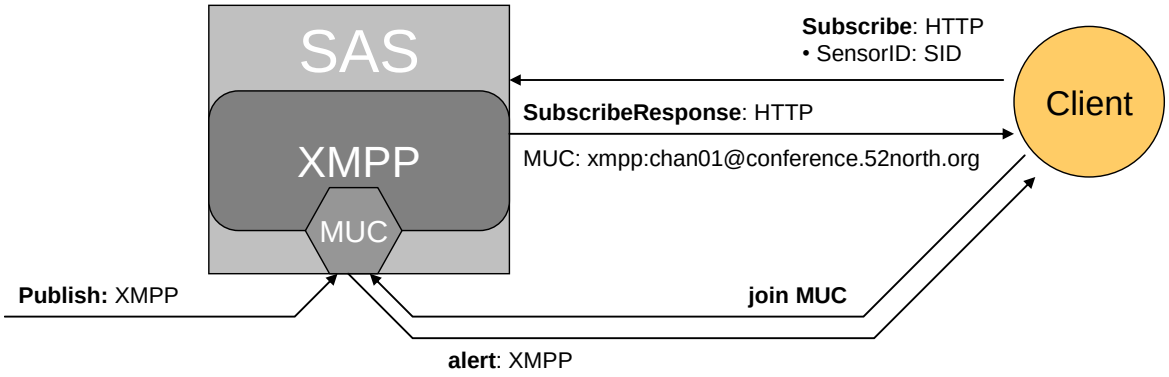


Figure 5: SAS example part 4, subscription and alerting – sensor defined alerts

Here, the user subscribes by only providing the ID of the sensor from which he wants to get all generated alerts. The service will then provide the address of the MUC where the sensor publishes its data. In this way, a lot of computational burden is taken away from the service, as it no longer needs to cope with those subscriptions that have not defined any filter criteria. The service only has to make sure that subscriptions are handled correctly when they expire. This mechanism improves scalability and also provides faster access to sensor data for clients.

The last part of our example highlights the case that the client does not want to receive alerts via XMPP, but by email or any other protocol that is supported by a WNS. Again, the separation between SAS and WNS is purely logical, both systems may run on the same machine or even the same memory space. The following figure illustrates the situation.

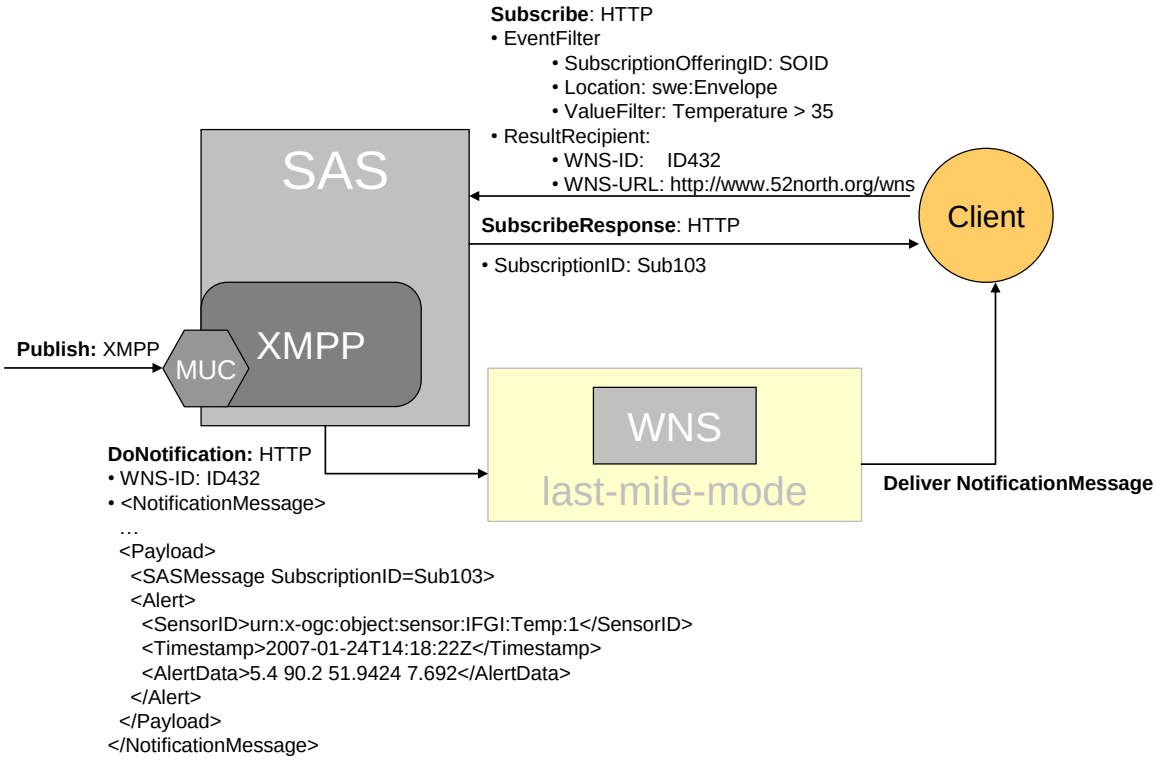


Figure 6: SAS example, part 5, subscription and alerting in last-mile-mode

The subscription request sent by the client looks slightly different compared to the previous example. Again, the client defines a location (the area the observation has to be located in) and an observation result filter. Additionally, it provides the WNS data: The URL of the WNS where the client has registered itself before, and the user ID that was provided by this WNS. If a sensor publishes an alert message that fits the client's conditions, then the message will be forwarded to the WNS. The Web Notification Service will act as a simple protocol transformer at this stage.

Note: this mechanism applies to subscriptions with user- as well as sensor-defined alerts.

Any receiving party – clients or services, can acknowledge alerts. This allows SAS and sensor providers to check if alert messages are delivered properly. For further information on alert message acknowledgement, see section 8.

7 Alerts

Messages are sent from a sensor to the SAS (and maybe to clients as well) and from the SAS to the client; independently of whether the message contains data of type "my battery is low" (i.e. represents an alert) or simple or complex values of an observation (i.e. raw data). Nevertheless, all data messages sent from sensors to SAS use the element identifier "Alert". This is necessary, as all messages sent from sensors to SAS might be alerts. All messages from SAS to clients are called alerts. As some of the incoming messages will be forwarded without modification by the SAS, even incoming messages use the element identifier "alert". All "alerts" follow the schema given below.

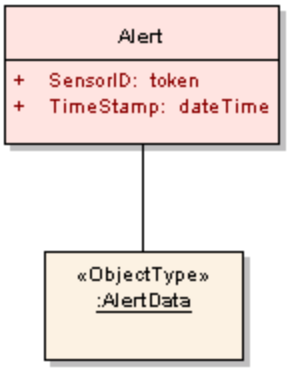


Figure 7: Structure of an alert in UML notation

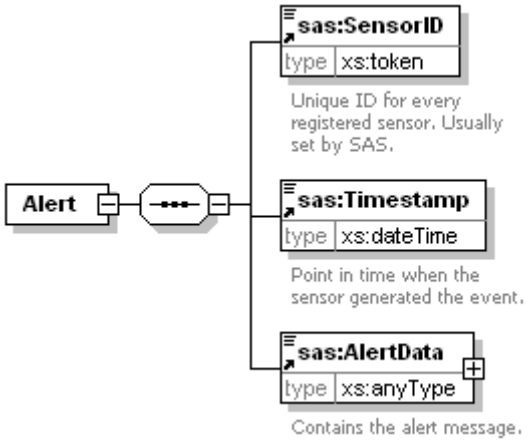


Figure 8: structure of an alert sent by SAS in XMLSpy notation

An Alert contains the data that has been sent by the sensor and – in case of a user defined alert subscription – matches the filter criteria provided during the subscribe process. The value of the SensorID element can be used in a DescribeSensor (see section 19) request to receive information about the sensor, which generated the alert data. The SensorID can also be used in a DescribeAlert (see section 20) request to determine the actual structure of the alert data. Thus, if the client receives an Alert with an unknown SensorID he can always determine how to parse the incoming data. The timestamp provides the point in time when the data contained in the alert was measured. The precision of this timestamp shall be high enough to accomplish that the timestamp provided in one alert is always after the timestamp of the previous alert. Thus, together with the SensorID, each alert is unique within one SAS instance.

Regardless of whether the alert is sent by sensors or SAS instances, the data contained in an alert follows a specific pattern. This pattern gives a lot of freedom to the alert creators. The data contained in an alert is described via the *swe:DataBlockDefinition* element defined in SweCommon.

The following listing provides an example of an alert structure definition.

Listing 1: example of swe:DataBlockDefinition

```
<swe:DataBlockDefinition>
  <swe:components name="sensor data structure">
    <swe:DataRecord>
      <swe:field name="component1">
        <swe:Quantity
definition="urn:x-ogc:def:phenomenon:OGC:Temperature">
          <swe:uom code="Cel"/>
        </swe:Quantity>
      </swe:field>
      <swe:field name="component2">
        <swe:Quantity
definition="urn:x-ogc:def:phenomenon:OGC:RelativeHumidity">
          <swe:uom code="%"/>
        </swe:Quantity>
      </swe:field>
      <swe:field name="component3">
        <swe:Position
definition="urn:x-ogc:def:phenomenon:OGC:sampleLocation"
referenceFrame="urn:x-ogc:def:crs:EPSG:6.11:4326">
        <swe:location>
          <swe:Vector>
            <swe:coordinate name="latitude">
              <swe:Quantity>
                <swe:uom code="deg"/>
              </swe:Quantity>
            </swe:coordinate>
            <swe:coordinate name="longitude">
              <swe:Quantity>
                <swe:uom code="deg"/>
              </swe:Quantity>
            </swe:coordinate>
          </swe:Vector>
          </swe:location>
        </swe:Position>
      </swe:field>
    </swe:DataRecord>
  </swe:components>
```

```
  <swe:encoding>
    <swe:TextBlock decimalSeparator="." blockSeparator="@@"
tokenSeparator=" "/>
   </swe:encoding>
</swe:DataBlockDefinition>
```

This alert definition contains values for three phenomena: temperature, relative humidity and sampling location. Given this description the SAS knows how to interpret an event sent by the sensor. A corresponding alert given in the following listing thus indicates that the sensor measured a temperature of 5.4 °C and relative humidity of 90.2 % at time 2007-01-24T14:18:22Z and at location 51.9424 / 7.692 (lat / lon in WGS84).

Listing 2: example of an alert sent by a sensor

```
<?xml version="1.0" encoding="UTF-8"?>
<Alert xmlns="http://www.opengis.net/sas/0.0"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
 <SensorID>urn:x-ogc:object:sensor:IFGI:Temp:1</SensorID>
 <Timestamp>2007-01-24T14:18:22Z</Timestamp>
      <AlertData>5.4 90.2 51.9424 7.692</AlertData>
</Alert>
```

8 Alert Message Acknowledgement

For use cases that require reliable filtering and transmission of alerts, the SAS offers an acknowledgement mechanism. Whether a SAS instance supports this mechanism or not is indicated via its Capabilities document. The following UML sequence diagram provides a brief overview of how the mechanism works for a client.

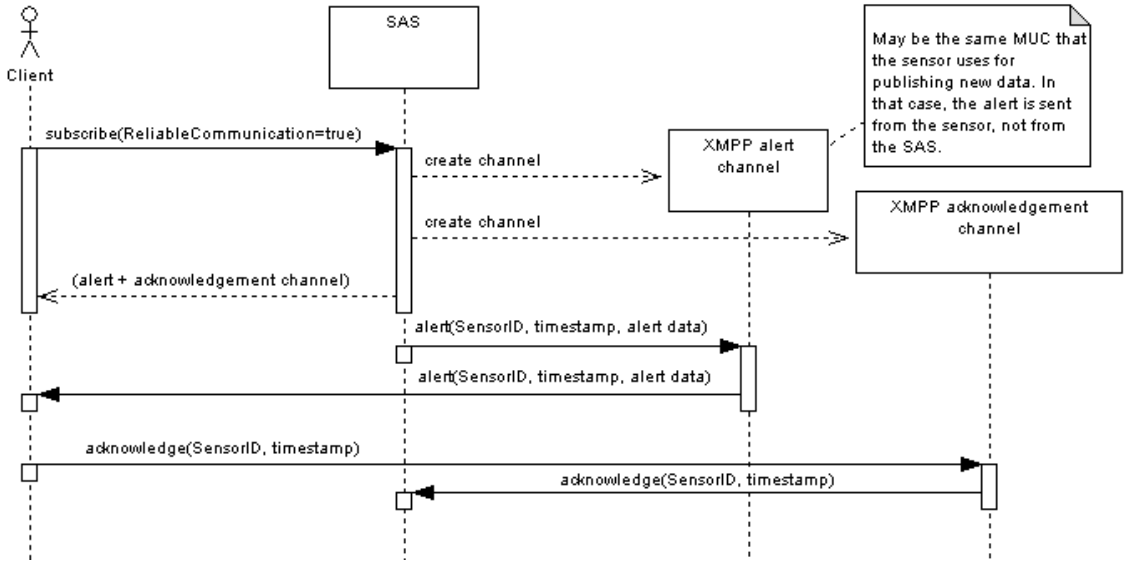


Figure 9: alert acknowledgement – client side

Upon subscribing at the SAS, a client may indicate that he requires reliable communication. The SAS creates two channels - one for sending alerts and one for receiving acknowledgements (this has to be a new channel for each subscription, because an acknowledgement only contains SensorID and timestamp of the corresponding alert) - and provides the channel addresses in the response. Both the SAS and the client would register with these channels (not shown in the diagram). Now, regardless on whether an alert was filtered and provided by the SAS or generated by a sensor and directly provided to the client, the client acknowledges the incoming alert by sending an Acknowledgement (see Figure 11) back to the SAS via the given XMPP channel.

Figure 10: Acknowledgement in UML notation

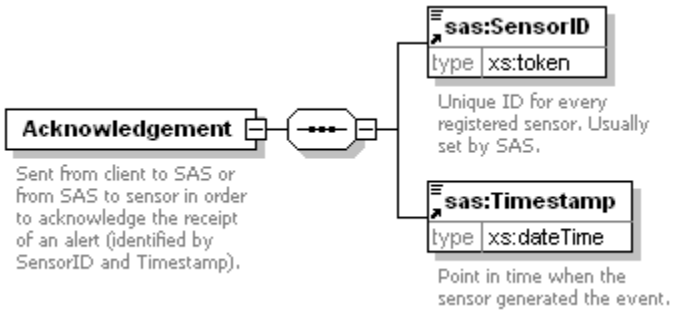


Figure 11: Acknowledgement in XMLSpy notation

Alerts can not only be acknowledged by clients but by the service as well – to indicate that the service received an alert from a sensor. So the use case is slightly different, but the acknowledgement mechanism for sensors works in quite the same way as for clients. The major difference is that now the service has to send an acknowledgement for every alert sent by the sensor.

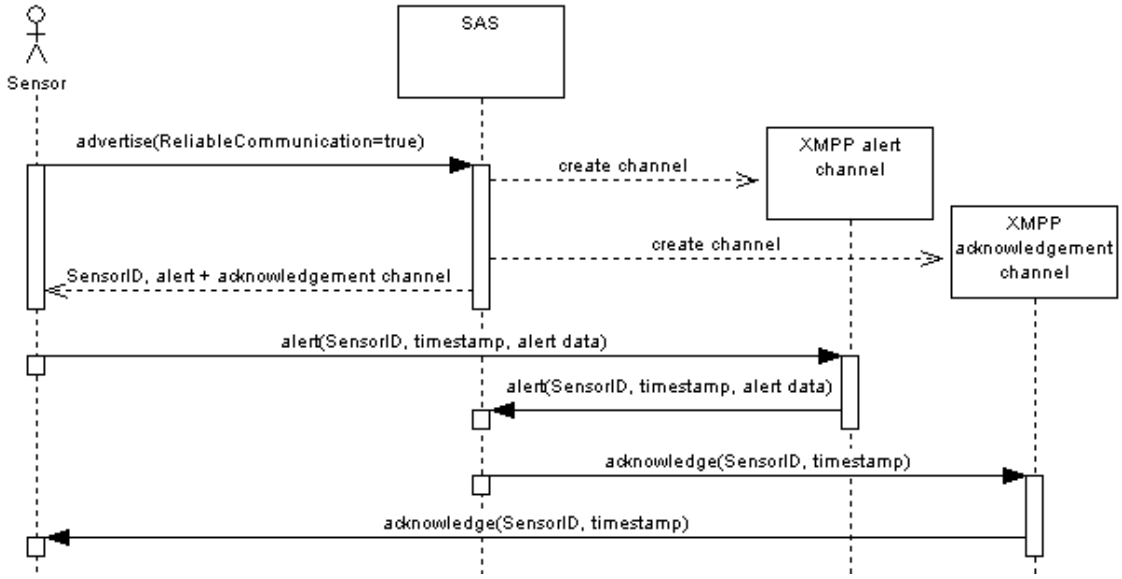


Figure 12: alert acknowledgement – sensor side

It is up to the receiver how acknowledgements are handled. They might be persisted or discarded after a certain amount of time has passed. The receiver does not take care in any means of non-acknowledged alerts. So it does not resend data – neither SAS nor a sensor. Clients may be able to receive historic data from a SOS associated with the SAS.

9 SAS Operations

The SAS interface (currently) specifies ten operations that can be requested by a client and performed by a SAS server. Those operations are:

GetCapabilities (required implementation by servers) – This operation allows a client to request and receive back service metadata (or Capabilities) documents that describe the abilities of the specific server implementation. This operation also supports negotiation of the specification version being used for client-server interactions.

GetWSDL (optional implementation by servers) – This operation allows a client to request and receive back the WSDL definition of the server interface.

Advertise (optional implementation by servers) – This operation allows producers to advertise the type of information published. In case that this data is the product of some pre-processing, e.g. sensor announces a *battery low* status, this data might be considered as an alert. In fact, it is only a published observation!

CancelAdvertisement (optional implementation by servers) – This operation allows producers to cancel an advertisement.

RenewAdvertisement (optional implementation by servers) – This operation allows producers to renew an advertisement

Subscribe (required implementation by servers) – This operation allows consumers to subscribe to alerts (keep in mind that this depicts a virtual subscription; the real subscription takes place at the messaging service interface).

CancelSubscription (required implementation by servers) – This operation allows consumers to cancel a subscription.

RenewSubscription (required implementation by servers) – This operation allows consumers to renew a subscription.

DescribeAlert (required implementation by servers) – This operation allows consumers to receive a template of the alert message structure.

DescribeSensor (required implementation by servers) – This operation allows consumers to receive information about the sensor.

Figure 9 is a simple UML diagram summarizing the SAS interface. This class diagram shows that the SAS interface class inherits the GetCapabilities operation from the OGCWebService interface class, and adds the SAS operations. (This capitalization of names uses the OGC/ISO profile of UML.)

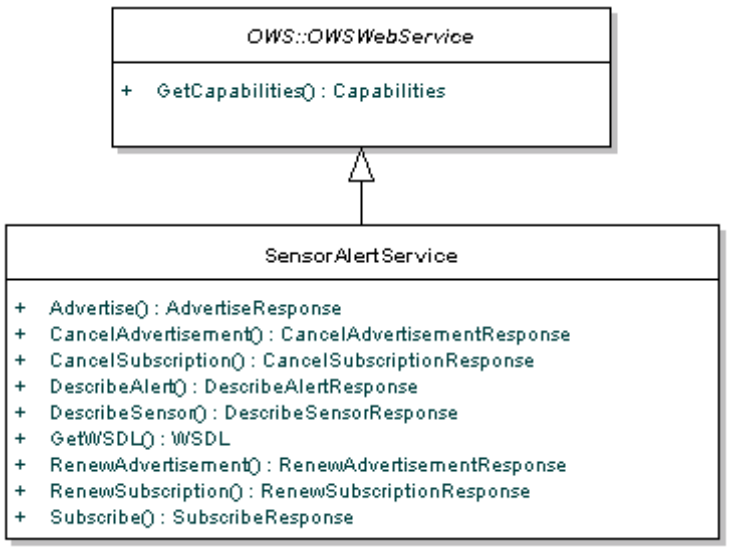


Figure 13: SAS interface UML diagram

NOTE In this UML diagram, the request and response for each operation is shown as a single parameter that is a data structure containing multiple lower-level parameters, which are discussed in subsequent clauses.

Each of the SAS operations is described in more detail in subsequent clauses.

10 Shared aspects

10.1 Introduction

This clause specifies aspects of the SAS behavior that are shared by several operations.

10.2 Shared operation parameters

This clause specifies some of the parameters used by multiple operations specified in the following clauses. The parameter names, meanings, data types, and multiplicity shall be as specified in Table 1.

Table 1 — Definitions of some operation request and response parameters

Name	Definition	Data type and value
reportingFrequency	Indicates how often the sensor is generating and sending observations. A value of 0 (default) indicates that the frequency cannot is unpredictable. Is defined by a swe:Quantity.	complex
AcknowledgementChannel	Contains the address of the MUC used for sending and receiving acknowledgements	complex
AlertChannel	Contains the address of the MUC used for sending and receiving alerts	complex
messageStructure	Defines the structure of the message. Uses swe:DataBlockDefinition	complex
desiredPublicationExpiration	Point in time when the sensor will not publish observations anymore. Uses swe:Time.	complex
Location	Area of operation of the sensor, uses swe:Envelope or swe:Position	complex
ReliableCommunication	True if the client wishes to perform reliable communication with the SAS, else false (default).	xs:boolean
SensorID	Unique identifier for a sensor. Provided by SAS server.	xs:token
sensorDescription	Describes the sensor using SensorML (description is either inline or referenced).	complex
SubscriptionID	Unique identifier for a subscription. Provided by SAS server.	xs:token
SubscriptionOfferingID	Identifies a SubscriptionOffering. It is part of the Capabilities document and can be used by clients in Subscribe operation requests.	xs:stoken
XMPPCredentials	The JID [7] of a sensor. Will be created by SAS or is given by a sensor. SAS can use the JID for identifying clients that join a MUC. JID is of the form node@domain e.g. jondoe@foo.bar.com.	xs:token

10.2.1 Location

The `Location` element (see Figure 14) provides a simple point or bounding box which is used by SAS in two ways: on the one hand, it is used to represent all possible locations a sensor can be placed at, and on the other hand it is used to define spatial filters for subscribing clients.

Let us have a closer look at how the `Location` is used in the former case. In case of a stationary in-situ sensor, usually a point location will be provided. In case that an envelope is provided this envelope defines the region where all of a sensors measurements will be situated in. E.g., a mobile sensor which operates on a river would provide the envelope which encloses the whole river while an in-situ camera would provide the area which is visible for it.

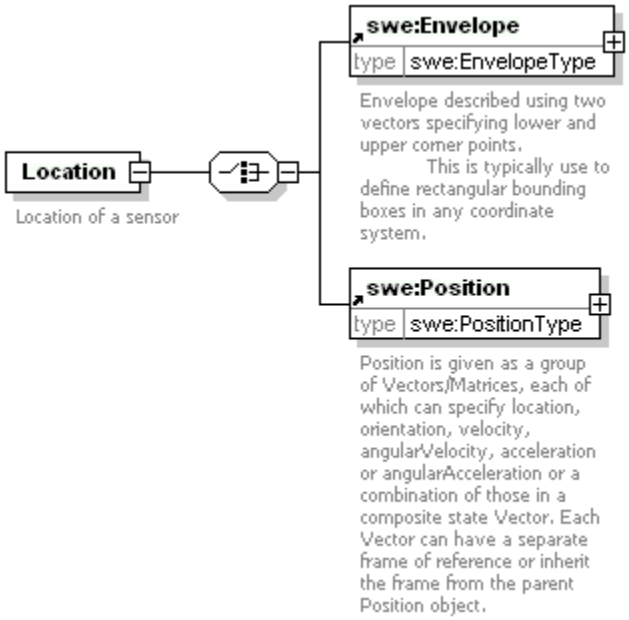


Figure 14: Location element in XMLSpy notation

The `swe:Position` and `swe:Envelope` elements are defined in SweCommon. Please review the SensorML document for further information.

10.2.2 XMPPCredentials

Clients that subscribe for alerts at the SAS shall always have their own XMPP account for logging in to a MUC if XMPP is used for delivering alerts to them.

But: for enabling plug-and-play of sensors and also for having more secure communication between sensors and SAS, XMPP credentials can be provided during advertisement of sensors. If the entity which sends the Advertise request does not provide its XMPP credentials, which in fact is just their JID used for identifying the entity throughout the XMPP network, then the SAS shall create an account for them at its own messaging server and provide the JID via the XMPPCredentials element contained in the

response. Otherwise, the SAS may log the JID of the advertised sensor. In either case, the SAS knows which JID belongs to the advertised sensor and can thus make use of member lists for the MUCs where sensors send data to. In the future this mechanism might be specified for subscribers as well.

10.3 Operation request encoding

The encoding of operation requests shall use HTTP GET with KVP encoding and HTTP POST with XML encoding as specified in Clause 11 of [OGC 05-008]. Table 2 summarizes the SAS operations and their encoding methods defined in this specification.

Table 2 — Operation request encoding

Operation name	Request encoding
GetCapabilities (required)	KVP and optional XML
GetWSDL	KVP
All others	XML

11 GetCapabilities operation (mandatory)

11.1 Introduction

The mandatory GetCapabilities operation allows clients to retrieve service metadata from a server. The response to a GetCapabilities request shall be an XML document containing service metadata about the server, including specific information about SAS. This clause specifies the XML document that a SAS server shall return to describe its capabilities.

11.2 GetCapabilities operation request

The GetCapabilities operation request shall be as specified in Subclauses 7.2 and 7.3 of [OGC 05-008]. The value of the "service" parameter shall be "SAS". The allowed set of service metadata (or Capabilities) XML document section names and meanings shall be as specified in Tables 3 and 7 of [OGC 05-008], with the additions listed in Table 3 below.

Table 3 — Additional Section name values and meanings

Section name	Meaning
sas:Contents	Return the contents section in service metadata document that contains information about the subscription offerings and advertisement offerings.
wns:NotificationAbilities	Return the NotificationAbilities section in service metadata document that indicates on the one hand which communication protocols and formats the service supports.

The "Multiplicity and use" column in Table 1 of [OGC 05-008] specifies the optionality of each listed parameter in the GetCapabilities operation request. Table 4 specifies the implementation of those parameters by SAS clients and servers.

Table 4 — Implementation of parameters in GetCapabilities operation request

Name	Multiplicity	Client implementation	Server implementation
service	One (mandatory)	Each parameter shall be implemented by all clients, using specified values	Each parameter shall be implemented by all servers, checking that each parameter is received with specified values
request	One (mandatory)		
Accept Versions	Zero or one (optional)	Should be implemented by all software clients, using specified values	Shall be implemented by all servers, checking if parameter is received with specified value(s)
Sections	Zero or one (optional)	Each parameter may be implemented by each client If parameter not provided, shall expect default response If parameter provided, shall allow default or specified response	Each parameter may be implemented by each server If parameter not implemented or not received, shall provide default response If parameter implemented and received, shall provide specified response
update Sequence	Zero or one (optional)		
Accept Formats	Zero or one (optional)		

All SAS servers shall implement HTTP GET transfer of the GetCapabilities operation request, using KVP encoding. Servers may also implement HTTP POST transfer of the GetCapabilities operation request, using XML encoding only.

11.2.1 GetCapabilities operation request example

EXAMPLE 1 To request a SAS capabilities document, a client could issue the following KVP encoded GetCapabilities operation request with near-minimum contents:

```
http://mars.uni-muenster.de/SAS/SAS?Request=GetCapabilities&Service=SAS
```

EXAMPLE 2 The corresponding GetCapabilities operation request XML encoded for HTTP POST is:

```
<?xml version="1.0" encoding="UTF-8"?>
<GetCapabilities xmlns="http://www.opengis.net/sas/0.0"
xmlns:ows="http://www.opengis.net/ows"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" service="SAS"/>
```

11.3 GetCapabilities operation response

11.3.1 Normal response

The normal response to a valid GetCapabilities operation request shall be a `Capabilities` document. More precisely, a response from the GetCapabilities operation shall include the parts shown in the following figure.

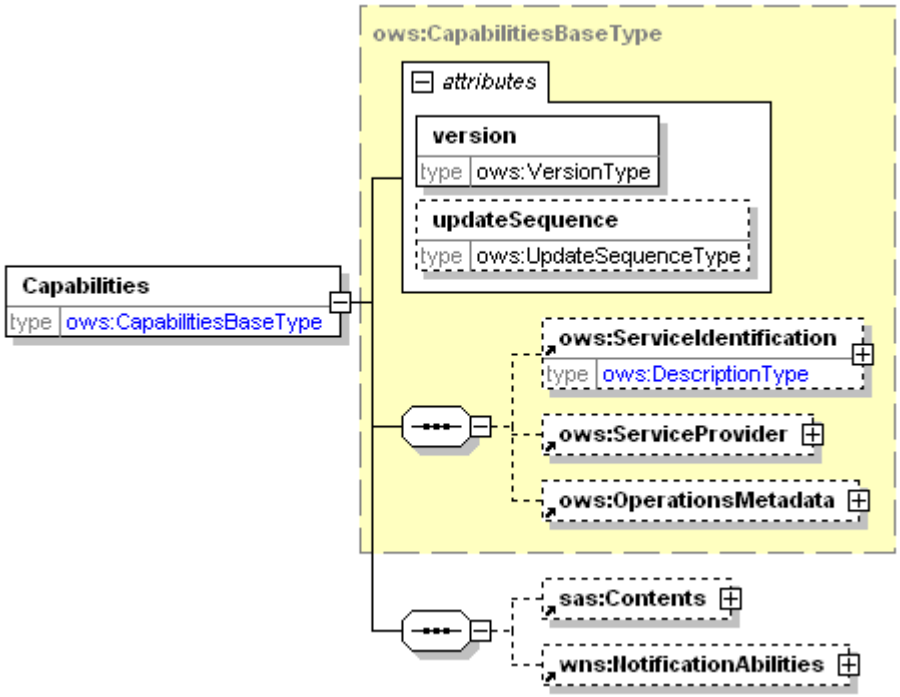


Figure 15: Capabilities element in XMLSpy notation

The service metadata document shall contain the SAS sections specified in Table 5. Depending on the values in the Sections parameter of the GetCapabilities operation request, any combination of these sections can be requested and shall be returned when requested.

Table 5 — Section name values and contents

Section name	Contents
ServiceIdentification	Metadata about this specific server. The schema of this section shall be the same as for all OWSs, as specified in Subclause 7.4.3 and owsServiceIdentification.xsd of [OGC 05-008].
ServiceProvider	Metadata about the organization operating this server. The schema of this section shall be the same for all OWSs, as specified in Subclause 7.4.4 and owsServiceProvider.xsd of [OGC 05-008].
OperationsMetadata	Metadata about the operations specified by this service and implemented by this server, including the URLs for operation requests. The basic contents and organization of this section shall be the same as for all OWSs, as specified in Subclause 7.4.5 and owsOperationsMetadata.xsd of [OGC 05-008].
Contents	Metadata about the data served by this server. For the SAS, this section shall contain data as specified in following Subclauses.
NotificationAbilities	Metadata about the communication protocols and formats supported by this service. (See WNS specification for further details)

In addition to these sections, each service metadata document shall include the mandatory "version" and optional "updateSequence" parameters specified in Table 6 in Subclause 7.4.1 of [OGC 05-008].

11.3.2 OperationsMetadata section standard contents

For the SAS, the OperationsMetadata section shall be the same as for all OGC Web Services, as specified in Subclause 7.4.5 and owsOperationsMetadata.xsd of [OGC 05-008]. The mandatory values of various (XML) attributes shall be as specified in Table 6. Similarly, the optional attribute values listed shall be included or not depending on whether that operation is implemented by that server. In Table 6, the "Attribute name" column uses dot-separator notation to identify parts of a parent item. The "Attribute value**"** column references an operation parameter, in this case an operation name, and the meaning of including that value is listed in the righthand column.

Table 6 — Required values of OperationsMetadata section attributes

Attribute name	Attribute value	Meaning of attribute value
Operation.name	GetCapabilities	The operation is implemented by this server.
	GetWSDL	The operation is implemented by this server.
	Advertise	The operation is implemented by this server.
	CancelAdvertisement	The operation is implemented by this server.
	RenewAdvertisement	The operation is implemented by this server.
	Subscribe	The operation is implemented by this server.
	CancelSubscription	The operation is implemented by this server.
	RenewSubscription	The operation is implemented by this server.
	DescribeAlert	The operation is implemented by this server
	DescribeSensor	The operation is implemented by this server

11.3.3 Contents section

The Contents section of a service metadata document contains metadata about the data served by this server. For the SAS, this Contents section shall contain data about subscription options: The SAS interface allows clients to subscribe to alerts or to publish alerts.

In case a client wants to advertise new publications, all it has to check is if the SAS instance accepts advertisements. This is indicated by the `AcceptAdvertisements` element. The only possible values are true and false. In case the SAS accepts advertisements, the next step would be to send an Advertise request.

`ReliableCommunicationSupported` is a flag that indicates if alert receivers can send acknowledgements. Acknowledgements will be sent via XMPP (see section 8).

In case that a client wants to subscribe to alerts, the `SubscriptionOffering` Element contains all information for subscribing at SAS. Multiple `SubcriptionOffering` elements are pooled in a `SubscriptionOfferingList`. The following figure shows the structure of the Contents element.

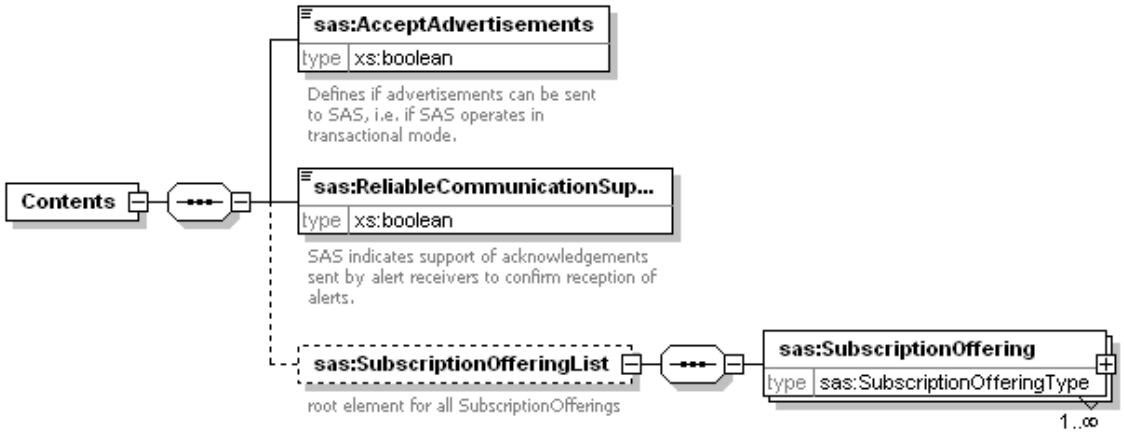


Figure 16: Contents element in XMLSpy notation

The `SubscriptionOffering` is shown in the following figures in UML and XMLSpy notation.

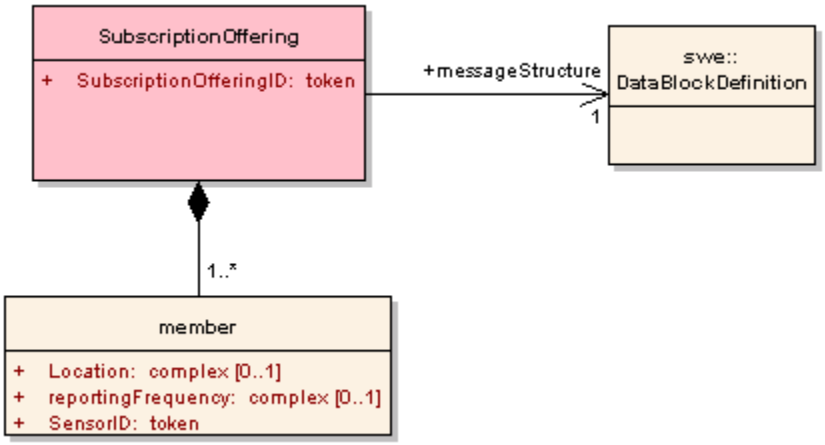


Figure 17: SubscriptionOffering element in UML notation

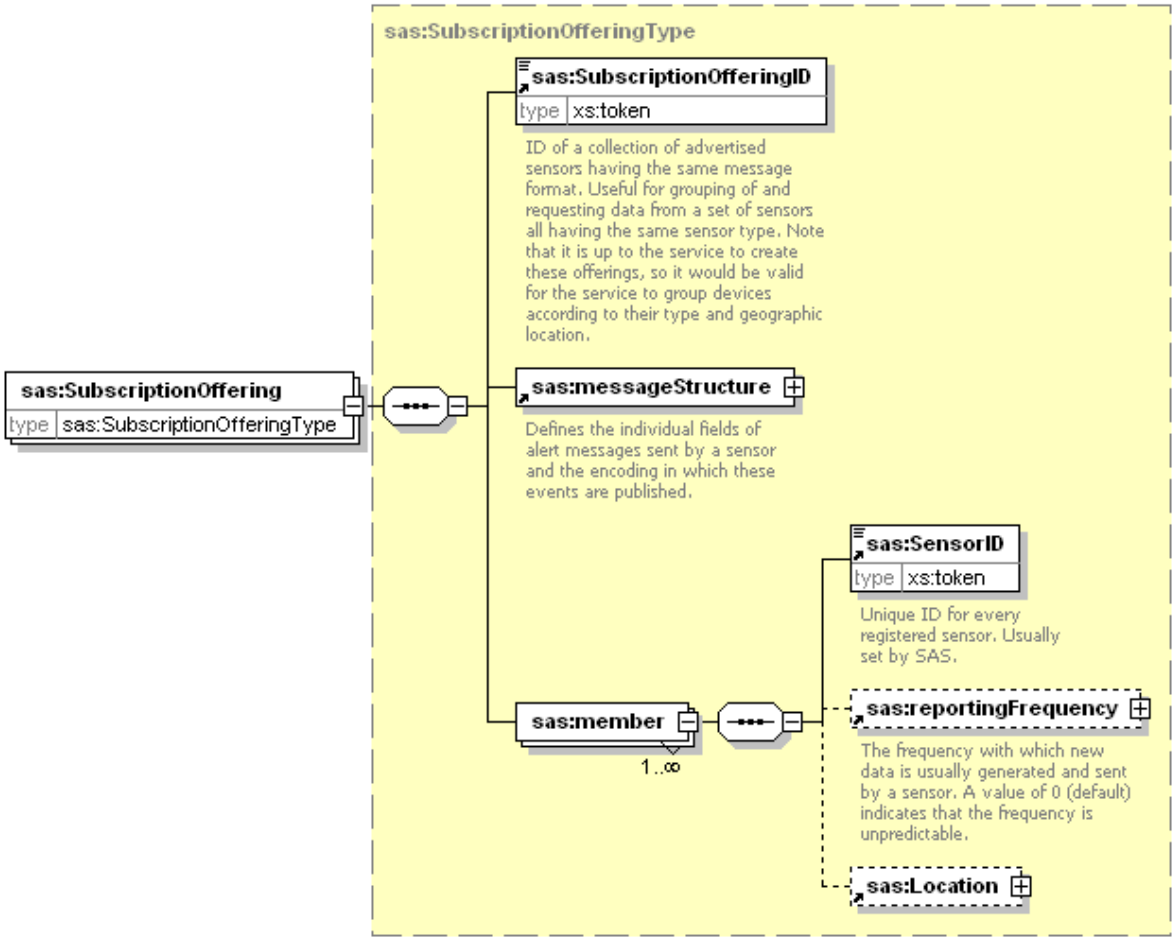


Figure 18: SubscriptionOffering in XMLSpy notation

11.3.4 NotificationAbilities section

The `NotificationAbilities` defines if the SAS can operate in Last-Mile-Mode. SAS usually uses external or internal Web Notification Services to deliver alert messages on protocols other than XMPP. `NotificationAbilities` defines the supported communication protocols and message formats. For further information, see Web Notification Service Specification [9].

11.3.5 Capabilities document XML encoding

An XML schema fragment for a SAS metadata document extends ows:CapabilitiesBaseType in owsCommon.xsd of [OGC 05-008], and can be found in Annex B (sasCapabilities.xsd).

11.3.6 Capabilities document example

In response to GetCapabilities operation request, a SAS server might generate a document that looks like (See Annex C for an entire capabilities document):

Listing 3: Example Capabilities

```
<?xml version="1.0" encoding="UTF-8"?>
<Capabilities xmlns="http://www.opengis.net/sas/0.0"
xmlns:xlink="http://www.w3.org/1999/xlink"
xmlns:ows="http://www.opengis.net/ows"
xmlns:wns="http://www.opengis.net/wns/0.0"
xmlns:swe="http://www.opengis.net/swe/1.0"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" version="1.0.0">
 <Contents>
   <AcceptAdvertisements>true</AcceptAdvertisements>
<ReliableCommunicationSupported>true</ReliableCommunicationSupported>
   <SubscriptionOfferingList>
     <SubscriptionOffering>
       <SubscriptionOfferingID>sub1</SubscriptionOfferingID>
       <messageStructure>
         <swe:DataBlockDefinition>
           <swe:components name="sensor data structure">
             <swe:DataRecord>
               <swe:field name="component1">
                 <swe:Quantity definition="urn:x-
ogc:def:phenomenon:OGC:Temperature">
                   <swe:uom code="Cel"/>
                 </swe:Quantity>
               </swe:field>
               <swe:field name="component2">
                 <swe:Quantity definition="urn:x-
ogc:def:phenomenon:OGC:RelativeHumidity">
                   <swe:uom code="%"/>
                 </swe:Quantity>
               </swe:field>
               <swe:field name="component3">
                 <swe:Position definition="urn:x-
ogc:def:phenomenon:OGC:sampleLocation" referenceFrame="urn:x-
ogc:def:crs:EPSG:6.11:4326">
                   <swe:location>
                     <swe:Vector>
                       <swe:coordinate name="latitude">
                         <swe:Quantity>
                           <swe:uom code="deg"/>
                         </swe:Quantity>
```

```
                        </swe:coordinate>
                        <swe:coordinate name="longitude">
                          <swe:Quantity>
                            <swe:uom code="deg"/>
                          </swe:Quantity>
                        </swe:coordinate>
                      </swe:Vector>
                    </swe:location>
                  </swe:Position>
                </swe:field>
              </swe:DataRecord>
            </swe:components>
            <swe:encoding>
              <swe:TextBlock decimalSeparator="." blockSeparator="@@"
tokenSeparator=" "/>
            </swe:encoding>
          </swe:DataBlockDefinition>
        </messageStructure>
        <member>
          <SensorID>urn:x-ogc:object:sensor:IFGI:Temp:1</SensorID>
          <reportingFrequency>
            <swe:Quantity>
              <swe:uom code="Hz"/>
              <swe:value>0.1</swe:value>
            </swe:Quantity>
          </reportingFrequency>
          <Location>
            <swe:Envelope referenceFrame="urn:x-
ogc:def:crs:EPSG:6.11:4326">
              <swe:lowerCorner>
                <swe:Vector>
                  <swe:coordinate name="latitude">
                    <swe:Quantity>
                      <swe:uom code="deg"/>
                      <swe:value>51.916945</swe:value>
                    </swe:Quantity>
                  </swe:coordinate>
                  <swe:coordinate name="longitude">
```

```
                    <swe:Quantity>
                      <swe:uom code="deg"/>
                      <swe:value>7.680686</swe:value>
                    </swe:Quantity>
                  </swe:coordinate>
                </swe:Vector>
              </swe:lowerCorner>
              <swe:upperCorner>
                <swe:Vector>
                  <swe:coordinate name="latitude">
                    <swe:Quantity>
                      <swe:uom code="deg"/>
                      <swe:value>51.974551</swe:value>
                    </swe:Quantity>
                  </swe:coordinate>
                  <swe:coordinate name="longitude">
                    <swe:Quantity>
                      <swe:uom code="deg"/>
                      <swe:value>7.716518</swe:value>
                    </swe:Quantity>
                  </swe:coordinate>
                </swe:Vector>
              </swe:upperCorner>
            </swe:Envelope>
          </Location>
        </member>
      </SubscriptionOffering>
    </SubscriptionOfferingList>
  </Contents>
  <wns:NotificationAbilities>
    <wns:SupportedCommunicationProtocols>
      <wns:XMPP>true</wns:XMPP>
      <wns:SMS>false</wns:SMS>
      <wns:Phone>false</wns:Phone>
      <wns:Fax>false</wns:Fax>
      <wns:Email>true</wns:Email>
      <wns:WSAddressing>false</wns:WSAddressing>
```

```
      <wns:WNS>true</wns:WNS>
    </wns:SupportedCommunicationProtocols>
    <wns:SupportedCommunicationFormats>
      <wns:NotificationFormat>basic</wns:NotificationFormat>
    </wns:SupportedCommunicationFormats>
  </wns:NotificationAbilities>
</Capabilities>
```

11.3.7 Exceptions

When a SAS server encounters an error while performing a GetCapabilities operation, it shall return an exception report message as specified in Clause 8 of [OGC 05-008]. The allowed exception codes shall include those listed in Table 5 of Subclause 7.4.1 of [OGC 05-008], if the updateSequence parameter is implemented by the server.

12 GetWSDL operation (optional)

12.1 Introduction

The optional GetWSDL operation allows clients to retrieve the WSDL description of the service interface.

12.2 GetWSDL operation request

Only KVP encoding of the GetWSDL operation request is allowed. Table 7 specifies the parameters of a GetWSDL request.

12.2.1 GetWSDL request parameters

This table also specifies the UML model data type, source of values, and multiplicity of each listed parameter, plus the meaning to servers when each optional parameter is not included in the operation request. Although some values listed in the "Name" column appear to contain spaces, they shall not contain spaces.

Table 7 – Parameters in the GetWSDL operation request

Name [a]	Definition	Data type and values	Multiplicity and use
service	Service type identifier	Character String type, not empty ("SAS")	One (mandatory)
request	Operation name	Character String type, not empty ("GetWSDL")	One (mandatory)
version	Specification version for operation	Version of this specification	One (mandatory)

a The name capitalization rules being used here are specified in Subclause 11.6.2 of [OGC 05-008].

12.2.2 GetWSDL request KVP encoding (mandatory)

Servers may implement HTTP GET transfer of the GetWSDL operation request, using KVP encoding. The KVP encoding of the GetWSDL operation request shall use the parameters specified in Table 8. The parameters listed in Table 8 shall be as specified in Table 7 above.

Table 8 — GetWSDL operation request URL parameters

Name and example [a]	Optionality and use	Definition and format
service=SAS	Mandatory	Service type identifier
request= GetWSDL	Mandatory	Operation name
version=1.0.0	Mandatory	Specification and schema version for this operation

a All parameter names are here listed using mostly lower case letters. However, any parameter name capitalization shall be allowed in KVP encoding, see Subclause 11.5.2 of [OGC 05-008].

12.2.3 GetWSDL request XML encoding

XML encoding is not supported.

12.2.4 GetWSDL operation request example

To request the WSDL description of the service interface, a KVP encoded request like the following could be sent to the SAS.

```
http://www.52north.org:8080/52nSAS/SAS?request=GetWSDL&service=SAS&vers
ion=1.0.0
```

12.2.5 GetWSDL Response

The response to a GetWSDL operation request is a WSDL document [11]. This document describes the interface of the service, with the operations supported by the specific service instance.

12.2.6 GetWSDL operation response example

In response to GetWSDL operation request, a SAS server might generate a document that looks like the document provided in Annex D.

12.2.7 GetWSDL Exceptions

When a SAS server encounters an error while performing a `GetWSDL` operation, it shall return an exception report message as specified in clause 8 of [OGC 05-008]. The allowed standard exception codes shall include those listed in Table 12. For each listed exceptionCode, the contents of the "locator" parameter value shall be as specified in the righthand column of Table 12.

NOTE To reduce the need for readers to refer to other documents, the first four values listed below are copied from Table 20 in Subclause 8.3 of [OGC 05-008].

Table 9 — Exception codes for GetWSDL operation

exceptionCode value	Meaning of code	"locator" value
OperationNotSupported	Request is for an operation that is not supported by this server	Name of operation not supported
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
NonApplicableCode	No other exceptionCode specified by this service and server applies to this exception	None, omit "locator" parameter

13 Advertise operation (optional)

13.1 Introduction

The advertise operation allows SAS clients to advertise what kind of data will be published. The operation might be used by sensors able to launch an HTTP call or by sensor administrators that want to register sensors. **Note**: The entity sending the advertise request may but doesn't have to be identical with the entity that registers with the Messaging Server (and eventually sends the alerts). A sensor provides information about the `messageStructure` and the `sensorDescription` (either inline or linked SensorML document). `reportingFrequency`, `desiredPublicationExpiration`, `Location`, `XMPPCredentials` (see section 10.2.2) and `ReliableCommunication` (see section 8) might be defined.

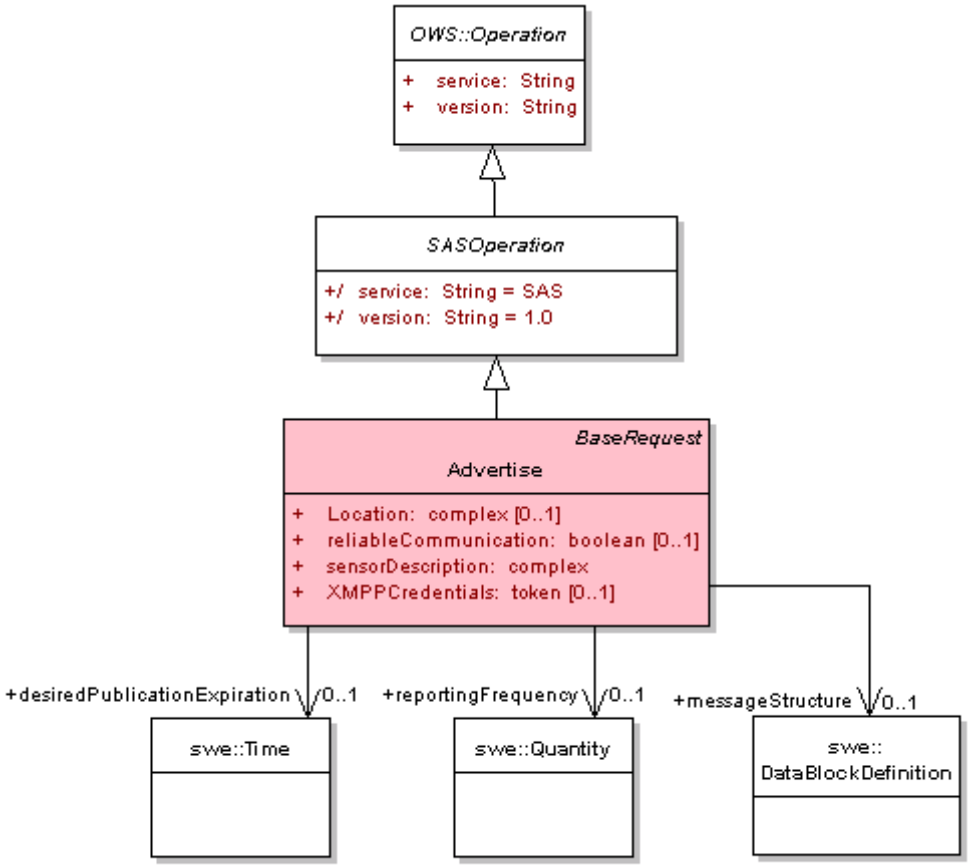


Figure 19: Advertise element in UML notation

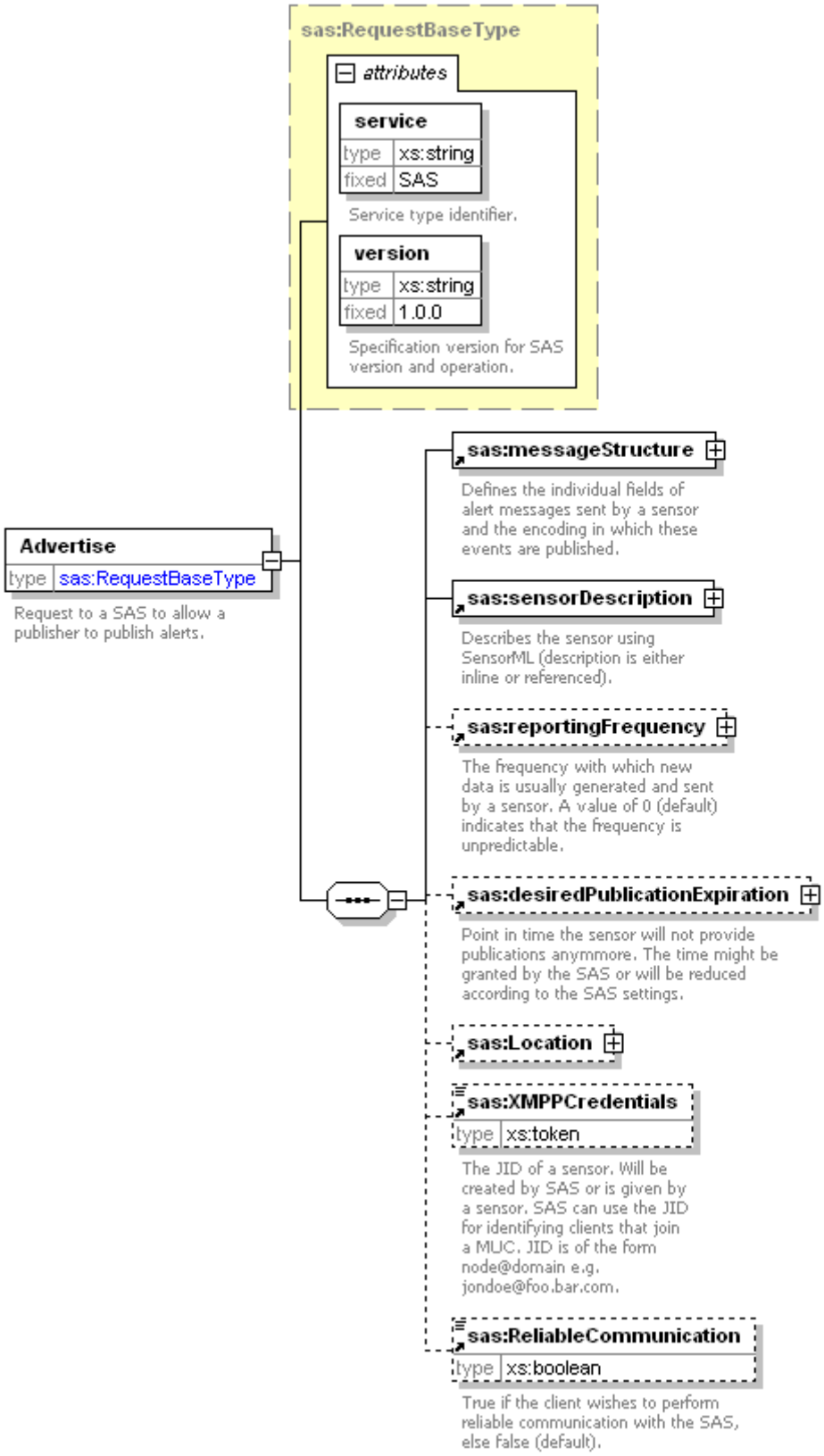


Figure 20: Advertise element in XMLSpy notation

13.2 Advertise operation request

13.2.1 Advertise request parameters

A request to perform the `Advertise` operation shall include the parameters listed and defined in Table 10. This table also specifies the UML model data type, source of values, and multiplicity of each listed parameter, plus the meaning to servers when each optional parameter is not included in the operation request. Although some values listed in the "Name" column appear to contain spaces, they shall not contain spaces.

Table 10 — Parameters in Advertise operation request

Name [a]	Definition	Data type and values	Multiplicity and use
service	Service type identifier	Character String type, not empty Value is OWS type abbreviation ("SAS")	One (mandatory)
version	Specification version for operation	Character String type, not empty Value is specified by this Implementation Specification and Schemas version	One (mandatory)
sensorDescription	describes the sensor	either inline or referenced SensorML document	One (mandatory)
Location	defines the area of operation of the sensor	Complex, uses swe:Envelope or swe:Position	One (optional)
reportingFrequency	see Subclause 10	complex, uses swe:Quantity	One (optional)
desiredPublicationExpiration	see Subclause 10	complex, uses swe:Time	One (optional)
messageStructure	see Subclause 10	Complex (uses swe:DataBlockDefinition)	One (mandatory)
XMPPCredentials	see Subclause 10; shall be provided if the SAS shall not create an XMPP account at its messaging server for the advertised entity	Character String type, not empty	One (optional)
ReliableCommunication	See Subclauses 8 and 10	xs:boolean	One (optional)
a The name capitalization rules being used here are specified in Subclause 11.6.2 of [OGC 05-008].			

The "Multiplicity and use" columns in all tables specify the optionality of each listed parameter and data structure in the Advertise operation request. All the "mandatory" parameters and data structures shall be implemented by all SAS clients, using a specified value(s). Similarly, all the "mandatory" parameters and data structures shall be

implemented by all SAS servers, checking that each request parameter or data structure is received with any specified value(s).

13.2.2 Advertise request KVP encoding

KVP encoding is not supported.

13.2.3 Advertise request XML encoding (mandatory)

All SAS servers shall implement HTTP POST transfer of the Advertise operation request, using XML encoding only. The following schema fragment specifies the contents and structure of an Advertise operation request encoded in XML: See Annex B (sasAdvertise.xsd).

13.2.4 Advertise operation request example

To advertise a new sensor, a request like the following could be sent to the SAS.

Listing 4: Example Advertise operation request

```
<?xml version="1.0" encoding="UTF-8"?>
<Advertise xmlns="http://www.opengis.net/sas/0.0"
xmlns:swe="http://www.opengis.net/swe/1.0"
xmlns:xlink="http://www.w3.org/1999/xlink"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://www.opengis.net/sas/0.0 ../sasAdvertise.xsd"
service="SAS" version="1.0.0">
 <messageStructure>
   <swe:DataBlockDefinition>
     <swe:components name="sensor data structure">
       <swe:DataRecord>
         <swe:field name="component1">
           <swe:Quantity
definition="urn:x-ogc:def:phenomenon:OGC:Temperature">
             <swe:uom code="Cel"/>
           </swe:Quantity>
         </swe:field>
         <swe:field name="component2">
           <swe:Quantity
definition="urn:x-ogc:def:phenomenon:OGC:RelativeHumidity">
             <swe:uom code="%"/>
           </swe:Quantity>
         </swe:field>
         <swe:field name="component3">
```

```
          <swe:Position
definition="urn:x-ogc:def:phenomenon:OGC:sampleLocation"
referenceFrame="urn:x-ogc:def:crs:EPSG:6.11:4326">
            <swe:location>
              <swe:Vector>
                <swe:coordinate name="latitude">
                  <swe:Quantity>
                    <swe:uom code="deg"/>
                  </swe:Quantity>
                </swe:coordinate>
                <swe:coordinate name="longitude">
                  <swe:Quantity>
                    <swe:uom code="deg"/>
                  </swe:Quantity>
                </swe:coordinate>
              </swe:Vector>
            </swe:location>
          </swe:Position>
        </swe:field>
      </swe:DataRecord>
    </swe:components>
    <swe:encoding>
      <swe:TextBlock decimalSeparator="." blockSeparator="@@"
tokenSeparator=" "/>
    </swe:encoding>
  </swe:DataBlockDefinition>
 </messageStructure>
 <sensorDescription xlink:href="http://www.52north.org/swe/Temp1.xml"/>
 <reportingFrequency>
   <swe:Quantity>
     <swe:uom code="Hz"/>
     <swe:value>0.1</swe:value>
   </swe:Quantity>
 </reportingFrequency>
 <desiredPublicationExpiration>
   <swe:Time>
     <swe:uom xlink:href="urn:x-ogc:def:uom:OGC:iso8601"/>
     <swe:value>2007-01-24T13:06:12Z</swe:value>
```

```
  </swe:Time>
</desiredPublicationExpiration>
<Location>
  <swe:Envelope referenceFrame="urn:x-ogc:def:crs:EPSG:6.11:4326">
    <swe:lowerCorner>
      <swe:Vector>
        <swe:coordinate name="latitude">
          <swe:Quantity>
            <swe:uom code="deg"/>
            <swe:value>51.916945</swe:value>
          </swe:Quantity>
        </swe:coordinate>
        <swe:coordinate name="longitude">
          <swe:Quantity>
            <swe:uom code="deg"/>
            <swe:value>7.680686</swe:value>
          </swe:Quantity>
        </swe:coordinate>
      </swe:Vector>
    </swe:lowerCorner>
    <swe:upperCorner>
      <swe:Vector>
        <swe:coordinate name="latitude">
          <swe:Quantity>
            <swe:uom code="deg"/>
            <swe:value>51.974551</swe:value>
          </swe:Quantity>
        </swe:coordinate>
        <swe:coordinate name="longitude">
          <swe:Quantity>
            <swe:uom code="deg"/>
            <swe:value>7.716518</swe:value>
          </swe:Quantity>
        </swe:coordinate>
      </swe:Vector>
    </swe:upperCorner>
  </swe:Envelope>
```

```
 </Location>
</Advertise>
```

13.3 Advertise operation response

13.3.1 Normal response parameters

The normal response to a valid Advertise operation request shall be an `AdvertiseResponse`. More precisely, a response from the Advertise operation shall include the parts shown in the following figures.

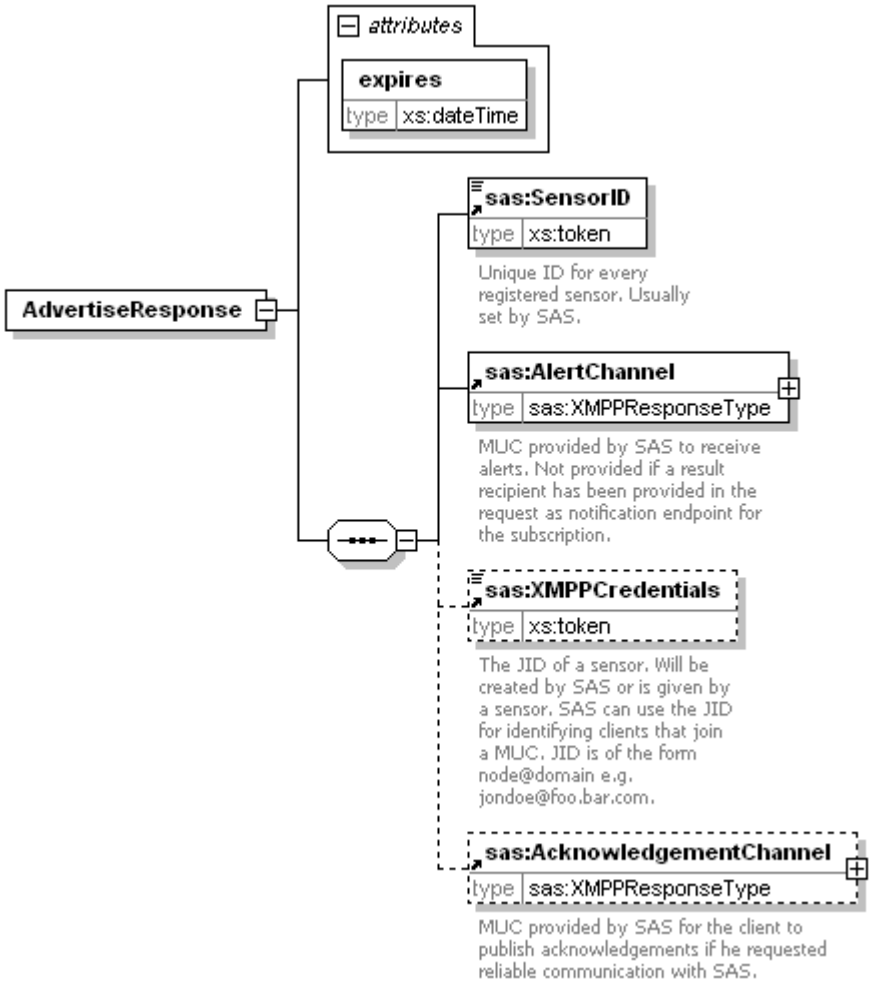


Figure 21: AdvertiseResponse element in XMLSpy notation

Figure 22: AdvertiseResponse element in UML notation

Table 11 — Parameters in AdvertiseResponse

Name	Definition	Data type and values	Multiplicity and use
expires	Defines the time until the option to advertise will expire. This enables SAS to close abandoned MUCs.	xs:dateTime; the value follows ISO 8601:2000, i.e. YYYY-MM-DDThh:mm:ss ±hh:mm	One (mandatory)
SensorID	Unique identifier for the sensor, provided by SAS	token	One (mandatory)
AlertChannel	Subscription endpoint where the sensor has to register using XMPP to publish alerts (see Subclause 10)	complex	One (mandatory)
XMPPCredentials	will be provided by SAS if no credentials were given in the Advertise request (see Subclause 10.2.2)	Character String type, not empty	One (optional)
Acknowledgement Channel	Contains the address of the MUC used for sending and receiving acknowledgements (see Subclauses 8 and 10)	complex	One (optional)

13.3.2 Normal response XML encoding

The following schema fragment specifies the contents and structure of an Advertise operation response, always encoded in XML.

See Annex B (sasAdvertise.xsd).

13.3.3 Advertise operation response example

In response to an Advertise operation request, a SAS server might generate a document that looks like the following.

Listing 5: Example advertise response

```
<?xml version="1.0" encoding="UTF-8"?>
<AdvertiseResponse xmlns="http://www.opengis.net/sas/0.0"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
expires="2007-01-01T00:00:00Z">
 <SensorID>urn:x-ogc:object:sensor:IFGI:Temp:1</SensorID>
 <AlertChannel>
   <XMPPURI>xmpp:pub301@conference.52North.org</XMPPURI>
 </AlertChannel>
</AdvertiseResponse>
```

13.3.4 Advertise exceptions

When a SAS server encounters an error while performing an `Advertise` operation, it shall return an exception report message as specified in clause 8 of [OGC 05-008]. The allowed standard exception codes shall include those listed in Table 12. For each listed exceptionCode, the contents of the "locator" parameter value shall be as specified in the righthand column of Table 12.

NOTE To reduce the need for readers to refer to other documents, the first four values listed below are copied from Table 20 in Subclause 8.3 of [OGC 05-008].

Table 12 — Exception codes for Advertise operation

exceptionCode value	Meaning of code	"locator" value
OperationNotSupported	Request is for an operation that is not supported by this server	Name of operation not supported
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
NonApplicableCode	No other exceptionCode specified by this service and server applies to this exception	None, omit "locator" parameter

14 RenewAdvertisement operation (optional)

14.1 Introduction

The `RenewAdvertisement` operation allows SAS clients to extend a previously advertised advertisement. The operation is mainly used by sensor management units that

had registered a sensor at the SAS but the advertisement time that was set by the SAS has expired.

14.2 RenewAdvertisement operation request

14.2.1 RenewAdvertisement request parameters

A request to perform the `RenewAdvertisement` operation shall include the parameters listed and defined in Table 13. This table also specifies the UML model data type, source of values, and multiplicity of each listed parameter, plus the meaning to servers when each optional parameter is not included in the operation request. Although some values listed in the "Name" column appear to contain spaces, they shall not contain spaces.

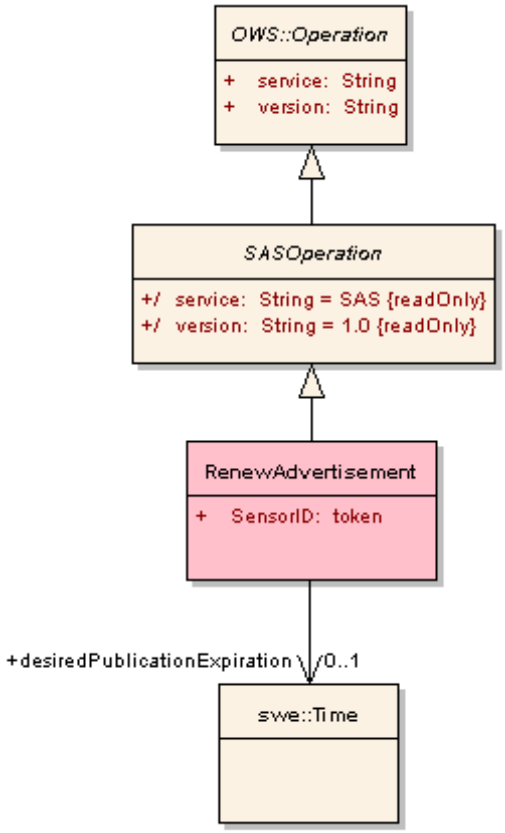


Figure 23: RenewAdvertisement operation in UML notation

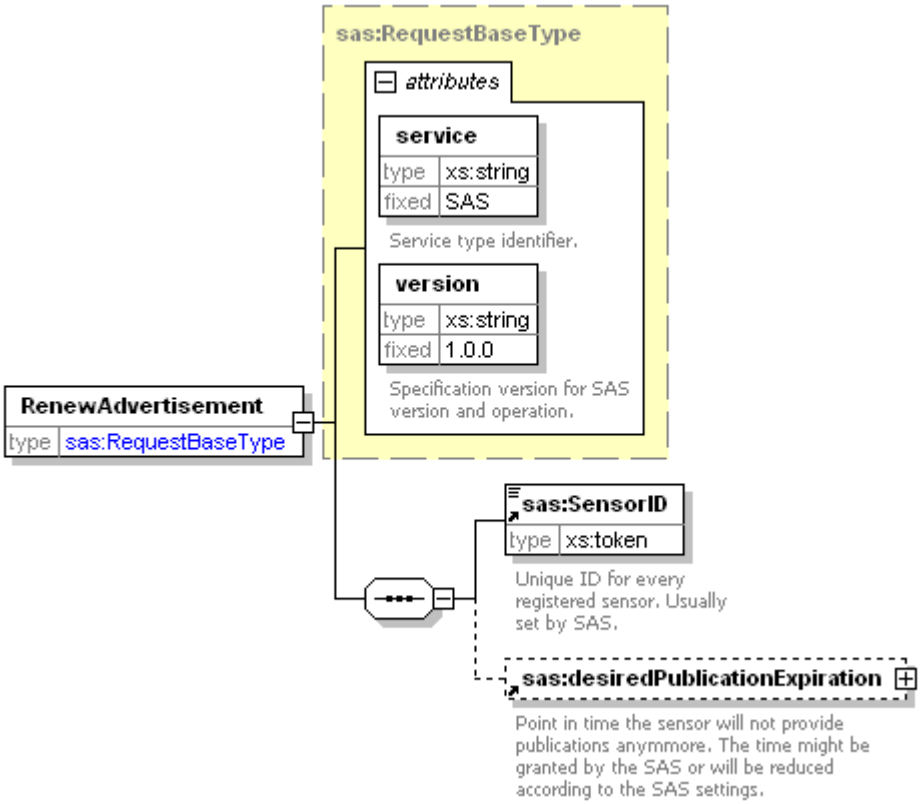


Figure 24: RenewAdvertisement operation in XMLSpy notation

Table 13 — Parameters in RenewAdvertisement operation request

Name [a]	Definition	Data type and values	Multiplicity and use
service	Service type identifier	Character String type, not empty Value is OWS type abbreviation (”SAS”)	One (mandatory)
version	Specification version for operation	Character String type, not empty Value is specified by this Implementation Specification and Schemas version	One (mandatory)
SensorID	Unique identifier for the sensor, provided by SAS	xs:token	One (mandatory)
desiredPublicationExpiration	see Subclause 10	Complex, uses swe:Time	One (optional)
a The name capitalization rules being used here are specified in Subclause 11.6.2 of [OGC 05-008].			

NOTE 2 The data type of many parameters is specified as “Character String type, not empty”. In the XML Schema Documents specified herein, these parameters are encoded with the xsd:string type, which does NOT require that these strings not be empty.

The “Multiplicity and use” column in Table 13 specifies the optionality of each listed parameter and data structure in the `RenewAdvertisement` operation request. Since all parameters and data structures are mandatory in the operation request, all parameters and data structures shall be implemented by all SAS clients, using a specified value(s). Similarly, all parameters and data structures shall be implemented by all SAS servers, checking that each request parameter is received with any specified value(s).

14.2.2 RenewAdvertisement request KVP encoding

KVP encoding not supported.

14.2.3 RenewAdvertisement request XML encoding (mandatory)

All SAS servers shall implement HTTP POST transfer of the RenewAdvertisement operation request, using XML encoding only. The following schema fragment specifies the contents and structure of a RenewAdvertisement operation request encoded in XML:

See Annex B (sasAdvertise.xsd).

14.2.4 RenewAdvertisement operation request example

To extend the duration of an advertisement for a specific sensor, a request like the following could be sent to the SAS.

Listing 6: Example RenewAdvertisement request

```
<?xml version="1.0" encoding="UTF-8"?>
<RenewAdvertisement xmlns="http://www.opengis.net/sas/0.0"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" service="SAS"
version="1.0.0">
  <SensorID>urn:x-ogc:object:sensor:IFGI:Temp:1</SensorID>
</RenewAdvertisement>
```

14.3 RenewAdvertisement operation response

14.3.1 Normal response parameters

The normal response to a valid RenewAdvertisement operation request shall be RenewAdvertisementResponse. More precisely, a response from the RenewAdvertisement operation shall include the parts listed in Table 14. This table also specifies the UML model data type plus the multiplicity and use of each listed part.

Figure 25: RenewAdvertisementResponse in UML notation

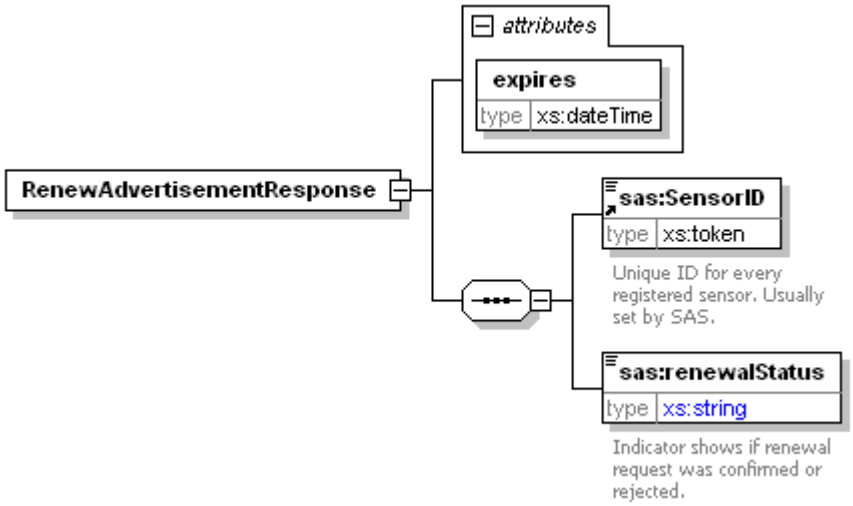


Figure 26: RenewAdvertisementResponse in XMLSpy notation

Table 14 — Parts of RenewAdvertisement operation response

Name	Definition	Data type and values	Multiplicity and use
SensorID	Unique identifier for the sensor, provided by SAS	xs:token	one (mandatory)
expires	Defines the time this advertisement will expire	xs:dateTime	one (mandatory)
renewalStatus	Identifier if renewal was successful	String “confirmed” or “rejected”	one (mandatory)

14.3.2 Normal response XML encoding

The following schema fragment specifies the contents and structure of a RenewAdvertisement operation response, always encoded in XML:

See Annex B (sasAdvertise.xsd).

14.3.3 RenewAdvertisement operation response example

In response to RenewAdvertisement operation request, a SAS server might generate a document that looks like the following.

Listing 7: Example RenewAdvertisementResponse

```
<?xml version="1.0" encoding="UTF-8"?>
<RenewAdvertisementResponse xmlns="http://www.opengis.net/sas/0.0"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
expires="2007-06-01T00:00:00Z">
  <SensorID>urn:x-ogc:object:sensor:IFGI:Temp:1</SensorID>
  <renewalStatus>confirmed</renewalStatus>
</RenewAdvertisementResponse>
```

14.3.4 RenewAdvertisement exceptions

When a SAS server encounters an error while performing a RenewAdvertisement operation, it shall return an exception report message as specified in Subclause 7 of [OGC 05-008]. The allowed standard exception codes shall include those listed in Table 15. For each listed exceptionCode, the contents of the "locator" parameter value shall be as specified in the righthand column of Table 15.

NOTE To reduce the need for readers to refer to other documents, the first four values listed below are copied from Table 20 in Subclause 8.3 of [OGC 05-008].

Table 15 — Exception codes for RenewAdvertisement operation

exceptionCode value	Meaning of code	"locator" value
OperationNotSupported	Request is for an operation that is not supported by this server	Name of operation not supported
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
NonApplicableCode	No other exceptionCode specified by this service and server applies to this exception	None, omit "locator" parameter

15 CancelAdvertisement operation (optional)

15.1 Introduction

The `CancelAdvertisement` operation allows SAS clients to cancel a previously registered advertisement offering. The operation is mainly used by sensor management units that want to unregister the sensor at the SAS. The only request payload is the SensorID that was provided by the SAS server.

15.2 CancelAdvertisement operation request

15.2.1 CancelAdvertisement request parameters

A request to perform the `CancelAdvertisement` operation shall include the parameters listed and defined in Table 10. This table also specifies the UML model data type, source of values, and multiplicity of each listed parameter, plus the meaning to servers when each optional parameter is not included in the operation request. Although some values listed in the "Name" column appear to contain spaces, they shall not contain spaces.

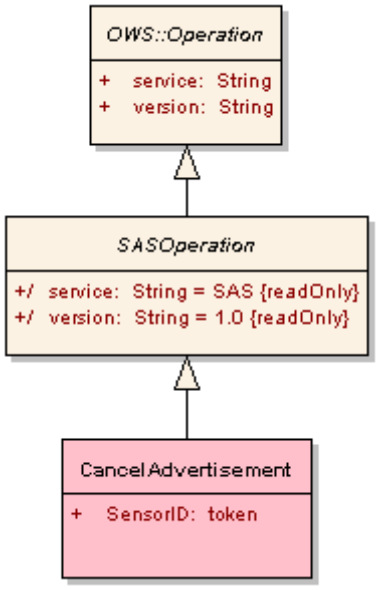


Figure 27: CancelAdvertisement in UML notation

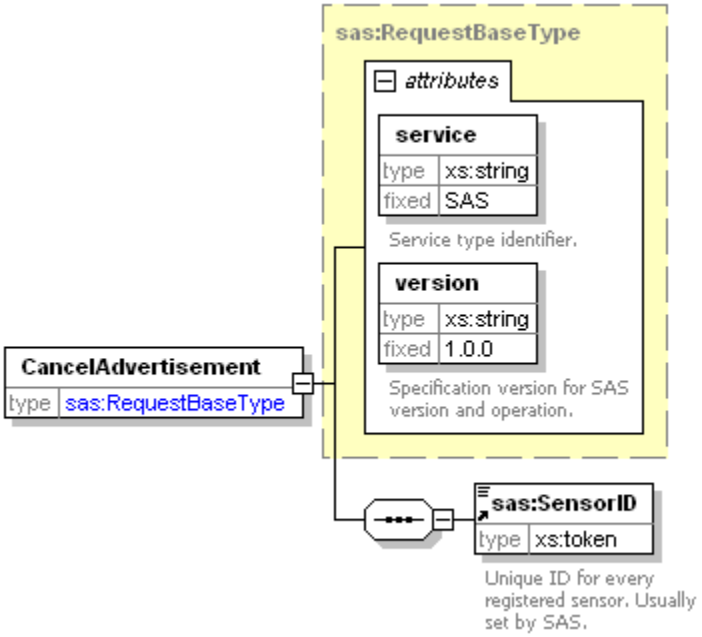


Figure 28: CancelAdvertisement in XMLSpy notation

Table 16 — Parameters in CancelAdvertisement operation request

Name	Definition	Data type and values	Multiplicity and use
service	Service type identifier	Character String type, not empty Value is OWS type abbreviation ("SAS")	One (mandatory)
version	Specification version for operation	Character String type, not empty Value is specified by this Implementation Specification and Schemas version	One (mandatory)
SensorID	Unique identifier for the sensor, provided by SAS	xs:token	One (mandatory)

NOTE 2 The data type of many parameters is specified as "Character String type, not empty". In the XML Schema Documents specified herein, these parameters are encoded with the xs:string or xs:token type, which does NOT require that these strings not be empty.

The "Multiplicity and use" column in Table 16 specifies the optionality of each listed parameter and data structure in the `CancelAdvertisement` operation request. Since all parameters and data structures are mandatory in the operation request, all parameters and data structures shall be implemented by all SAS clients, using a specified value(s). Similarly, all parameters and data structures shall be implemented by all SAS servers, checking that each request parameter is received with any specified value(s).

15.2.2 CancelAdvertisement request KVP encoding

KVP encoding not supported.

15.2.3 CancelAdvertisement request XML encoding

All SAS servers shall implement HTTP POST transfer of the CancelAdvertisement operation request, using XML encoding only. The following schema fragment specifies the contents and structure of a CancelAdvertisement operation request encoded in XML:

See Annex B (sasAdvertise.xsd).

15.2.4 CancelAdvertisement operation request example

To cancel the advertisement for a specific sensor, a request like the following could be sent to SAS.

Listing 8: example CancelAdvertisement operation request

```
<?xml version="1.0" encoding="UTF-8"?>
<CancelAdvertisement xmlns="http://www.opengis.net/sas/0.0"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" service="SAS"
version="1.0.0">
  <SensorID>urn:x-ogc:object:sensor:IFGI:Temp:1</SensorID>
</CancelAdvertisement>
```

15.3 CancelAdvertisement operation response

15.3.1 Normal response parameters

The normal response to a valid `CancelAdvertisement` operation request shall be `CancelAdvertisementResponse`. More precisely, a response from the `CancelAdvertisement` operation shall include the parts listed in Table 17. This table also specifies the UML model data type plus the multiplicity and use of each listed part.

Figure 29: CancelAdvertisementResponse in UML notation

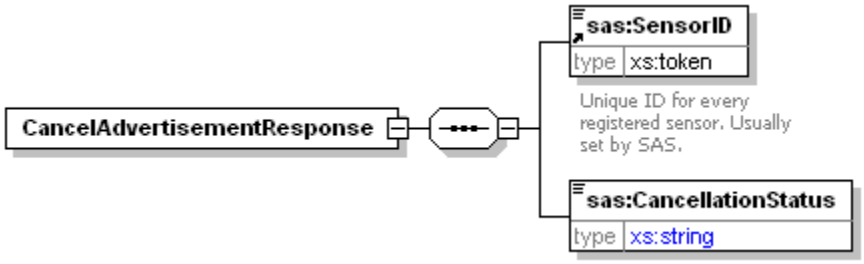


Figure 30: CancelAdvertisementResponse in XMLSpy notation

Table 17 — Parts of CancelAdvertisement operation response

Name	Definition	Data type and values	Multiplicity and use
SensorID	Unique identifier for the sensor, provided by SAS	xs:token	one (mandatory)
CancellationStatus	Identifies the status of the cancellation request	String “confirmed” “expired” or “invalidSensorID”	one (mandatory)

15.3.2 Normal response XML encoding

The following schema fragment specifies the contents and structure of a CancelAdvertisement operation response, always encoded in XML:

See Annex B (sasAdvertise.xsd).

15.3.3 CancelAdvertisement operation response example

In response to CancelAdvertisement operation request, a SAS server might generate a document that looks like the following.

Listing 9: Example CancelAdvertisement response

```
<?xml version="1.0" encoding="UTF-8"?>
<CancelAdvertisementResponse xmlns="http://www.opengis.net/sas/0.0"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
  <SensorID>urn:x-ogc:object:sensor:IFGI:Temp:1</SensorID>
  <CancellationStatus>confirmed</CancellationStatus>
</CancelAdvertisementResponse>
```

15.3.4 CancelAdvertisement exceptions

When a SAS server encounters an error while performing a CancelAdvertisement operation, it shall return an exception report message as specified in Subclause 8 of [OGC 05-008]. The allowed standard exception codes shall include those listed in Table 18. For each listed exceptionCode, the contents of the "locator" parameter value shall be as specified in the righthand column of Table 18.

NOTE To reduce the need for readers to refer to other documents, the first four values listed below are copied from Table 20 in Subclause 8.3 of [OGC 05-008].

Table 18 — Exception codes for CancelAdvertisement operation

exceptionCode value	Meaning of code	"locator" value
OperationNotSupported	Request is for an operation that is not supported by this server	Name of operation not supported
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
NonApplicableCode	No other exceptionCode specified by this service and server applies to this exception	None, omit "locator" parameter

16 Subscribe operation (mandatory)

16.1 Introduction

The `Subscribe` operation allows SAS clients to subscribe for alerts. The client can define in which data he is interested in. This is done by using constraining elements like value filters, location areas etc.

16.2 Subscribe operation request

As was mentioned in section 6, there are two levels in which a client can subscribe at the SAS – on the one hand the client can subscribe for all events of a specific sensor (which

is especially valuable for use cases in which the sensor itself recognizes that a situation of interest occurred), on the other hand a client may provide filter criteria for the SAS to find only those events that are of interest to the user (e.g. all events in a certain area or above a certain threshold). If specific filter criteria are provided (via the EventFilter element), all of them are concatenated by an implicit AND. So if one would provide a SubscriptionOfferingID, location and list of value filters, only those events that are generated by the sensors grouped under the referenced offering, occur at the given location AND match the value filters are sent to the client. If the value filters contain definitions of different observed properties, the service shall discard all events that do not contain values for all of these properties.

16.2.1 Subscribe request parameters

A request to perform the `Subscribe` operation shall include the parameters listed and defined in Table 19. This table also specifies the UML model data type, source of values, and multiplicity of each listed parameter, plus the meaning to servers when each optional parameter is not included in the operation request. Although some values listed in the "Name" column appear to contain spaces, they shall not contain spaces.

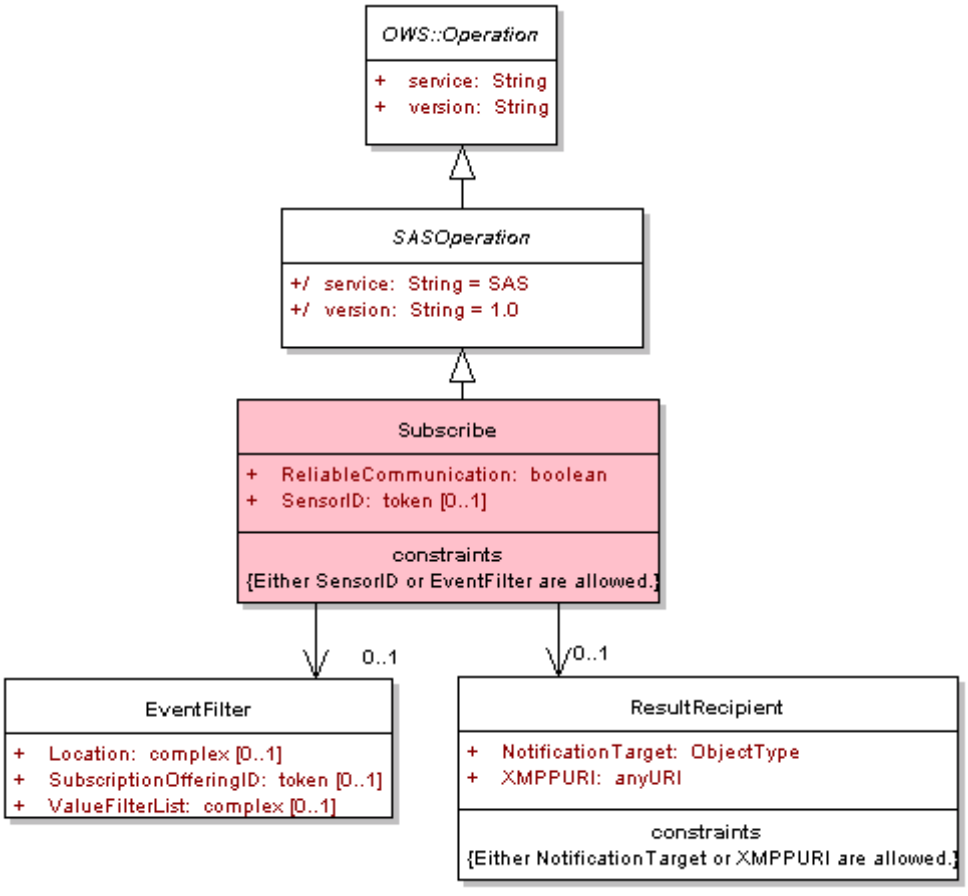


Figure 31: Subscribe in UML notation

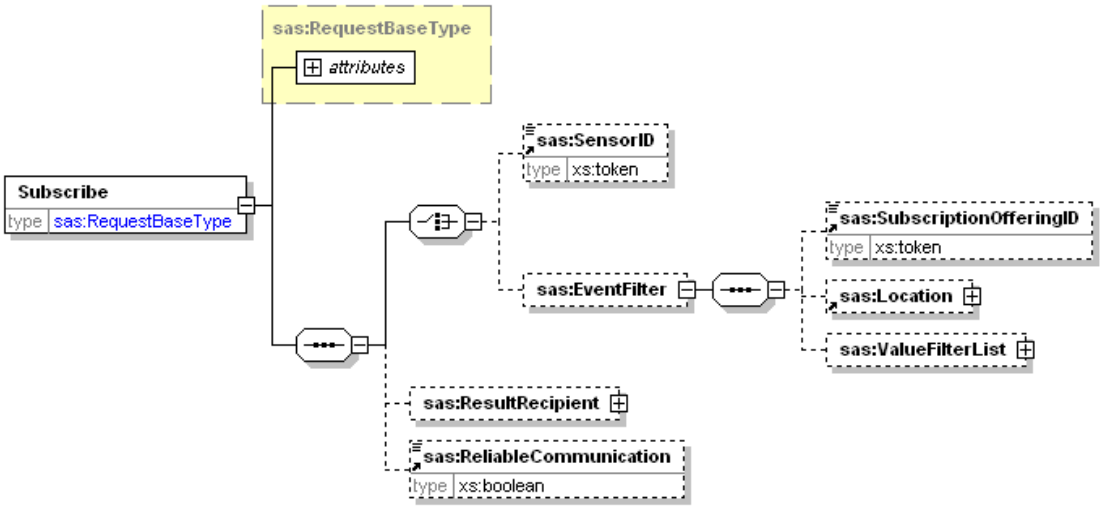


Figure 32: Subscribe in XMLSpy notation

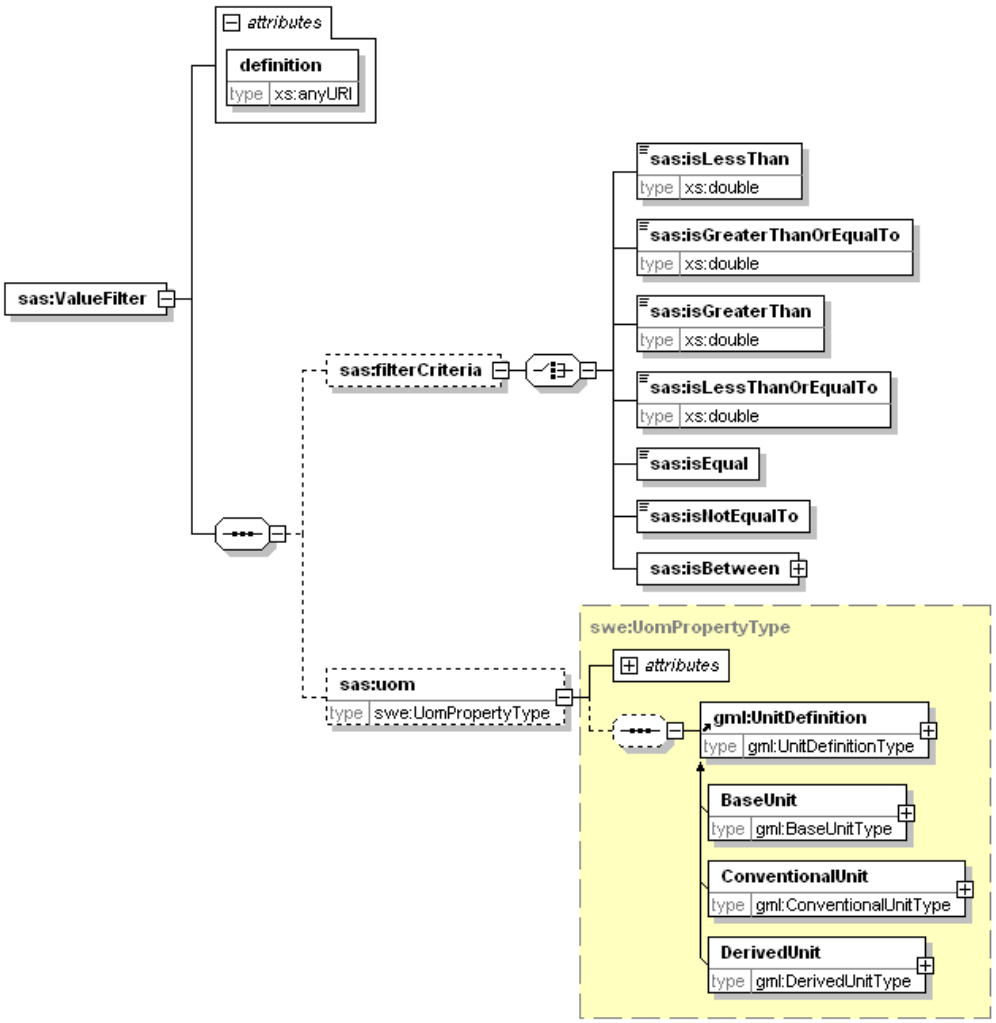


Figure 33: ValueFilter (element of the Subscribe request) in XMLSpy notation

Table 19 — Parameters in Subscribe operation request

Name [a]	Definition	Data type and values	Multiplicity and use
service	Service type identifier	Character String type, not empty Value is OWS type abbreviation (”SAS”)	One (mandatory)
version	Specification version for operation	Character String type, not empty Value is specified by this Implementation Specification and Schemas version	One (mandatory)
SensorID	Sensor identifier, administered by SAS; if used in the request, the client subscribes for all alerts generated by the referenced sensor (see section 6 for further details)	xs:token	One (optional)
Subscription OfferingID	see Subclause 10	xs:token	One (optional)
Location	Defines the position or bounding box from where the alert has to be triggered (see section 10.2.1)	swe:Position or swe:Envelope	One (optional)
ValueFilter	Allows the definition of a value filter condition. Each filter has a definition of the observed property for which the filter applies. This allows filtering for events that contain values for the given properties. In addition, a comparison filter can be defined which – in case that the observed property represents a quantity – shall be accompanied by a unit of measure for the value which is used in the filter. For example, one could define a filter “temperature > 30 °C”. The information contained in the filter allows for the service to perform conversion between units of measure if applicable. The filter defined in this specification[4] enables the definition of basic filter criteria for properties that are represented by boolean, textual or numeric values.	Complex	One to many (optional)

[4] We are aware of the fact that an OGC filter encoding implementation specification already exists. [10] However, for this version of the service we chose not to make use of that specification. On the one hand, we did not want to allow too complex queries at SAS which could be expressed with the elements of the filter encoding specification. SAS is designated for filtering huge amount of data on the fly, we therefore chose to provide only support for simple (value) comparison based filtering. On the other hand, the filter encoding specification cannot express the unit of measure of the comparison values used in filters. We therefore had to extend our filter. Being able to recognize the unit of measure of observed properties and performing unit conversions (if necessary) is a crucial part of SAS.

ResultRecipient	Used to instruct the SAS to send alerts to a specific XMPP-MUC or to use a WNS for alert message forwarding. If ResultRecipient is not set, the SAS will provide a MUC where the client has to subscribe in order to receive alerts.	anyURI if XMPP-MUC should be used, complex type WNS in case that a WNS is used	One to many (optional)
ReliableAlertingRequired	Defines if alerts have to be acknowledged (see chapter 8)	Boolean	One (optional)
a The name capitalization rules being used here are specified in Subclause 11.6.2 of [OGC 05-008].			

The "Multiplicity and use" column in Table 19 specifies the optionality of each listed parameter and data structure in the Subscribe operation request. All the "mandatory" parameters and data structures shall be implemented by all SAS clients, using a specified value(s). Similarly, all the "mandatory" parameters and data structures shall be implemented by all SAS servers, checking that each request parameter or data structure is received with any specified value(s).

All the "optional" parameters and data structures in the Subscribe operation request should be implemented by all SAS clients using specified values for each implemented process to which that parameter or data structure applies. Similarly, all the "optional" parameters and data structures shall be implemented by all SAS servers for each implemented process to which that parameter or data structure applies.

16.2.2 Subscribe request KVP encoding

KVP encoding not supported.

16.2.3 Subscribe request XML encoding (mandatory)

All SAS servers shall implement HTTP POST transfer of the Subscribe operation request, using XML encoding only. The following schema fragment specifies the contents and structure of a Subscribe operation request encoded in XML:

See Annex B (sasSubscribe.xsd).

16.2.4 Subscribe operation request example

The following example request would tell the service to deliver all events that fulfill the condition "relative humidity >= 80 %" and "air temperature > 30 °C" regardless of location. No specific result recipient was provided, thus the service will provide the address of an XMPP channel where all matching events are posted in the operation response. Furthermore, the client did not request reliable communication, so there is no need for the service to open an acknowledgement channel. This also means that the service cannot be held responsible if important events have not been received by the client.

Listing 10: Example Subscribe request

```
<?xml version="1.0" encoding="UTF-8"?>
<Subscribe xmlns="http://www.opengis.net/sas/0.0"
xmlns:swe="http://www.opengis.net/swe/1.0"
xmlns:wns="http://www.opengis.net/wns/0.0"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" service="SAS"
version="1.0.0">
 <EventFilter>
   <ValueFilterList>
     <member>
       <ValueFilter
definition="urn:x-ogc:def:phenomenon:OGC:RelativeHumidity">
         <filterCriteria>
           <isGreaterThanOrEqualTo>80</isGreaterThanOrEqualTo>
         </filterCriteria>
         <uom code="%"/>
       </ValueFilter>
     </member>
     <member>
       <ValueFilter
definition="urn:x-ogc:def:phenomenon:OGC:AirTemperature">
         <filterCriteria>
           <isGreaterThan>30</isGreaterThan>
         </filterCriteria>
         <uom code="CEL"/>
       </ValueFilter>
     </member>
   </ValueFilterList>
 </EventFilter>
</Subscribe>
```

16.3 Subscribe operation response

A valid subscription request results in a new subscription at the service. This subscription has a specific ID and expires at a point in time in the future, determined by the SAS. By constraining the duration of subscriptions, the SAS can make sure that idle subscriptions can be removed. Clients may extend this duration by issuing a RenewSubscription request (see section 17). If the client did not provide another endpoint for delivering alerts to (by including a result recipient in the Subscribe request), the service will add the address of an XMPP channel where all the alerts designated for the subscriber will be

posted. If reliable communication is required, the service will also provide another XMPP channel address where alert acknowledgements can be posted by clients.

16.3.1 Normal response parameters

The normal response to a valid Subscribe operation request shall be a SubscribeResponse. More precisely, a response from the Subscribe operation shall include the parts listed in Table 20. This table also specifies the UML model data type plus the multiplicity and use of each listed part.

Figure 34: SubscribeResponse in UML notation

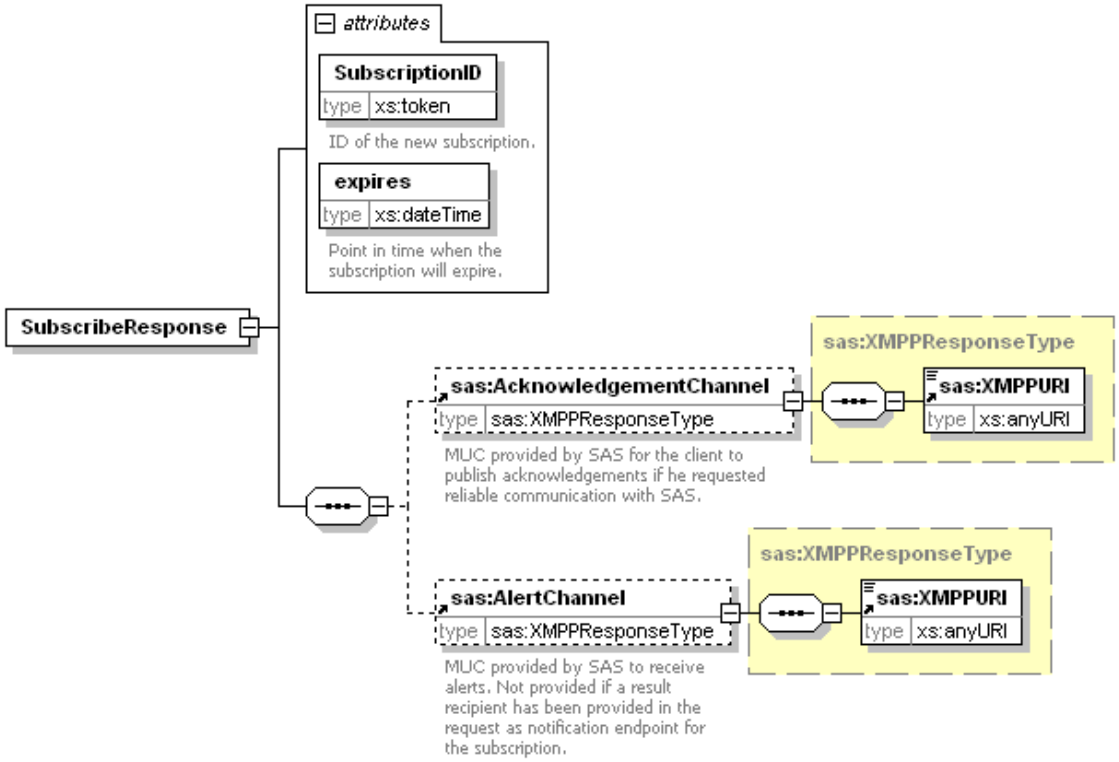


Figure 35: SubscribeResponse in XMLSpy notation

Table 20 — Parts of Subscribe operation response

Name	Definition	Data type and values	Multiplicity and use
SubscriptionID	Each subscription is identified by a unique identifier which is provided by the SAS.	ID	one (mandatory)
expires	The time that this subscription automatically terminates. Each SAS is free to choose the time a subscription remains valid. During this period, the client has to send a RenewSubscription request in order to extend it.	xs:dateTime, follows ISO 8601:2000 and is restricted to YYYY-MM-DDThh:mm:ss±hh:mm	one (mandatory)
AcknowledgementChannel	XMPP channel used to send acknowledgements	XMPPResponseType	one (optional)
AlertChannel	XMPP channel where alerts will be sent to. Client has to subscribe to it using XMPP operations in order to receive alerts. Only existent in the response if the client did not provide a ResultRecipient in the request.	XMPPResponseType	one (optional)

16.3.2 Normal response XML encoding

The following schema fragment specifies the contents and structure of a Subscribe operation response, always encoded in XML:

See Annex B (sasSubscribe.xsd).

16.3.3 Subscribe response example

In response to Subscribe operation request, a SAS server might generate a document that looks like the following.

Listing 11: Example Subscribe response

```
<?xml version="1.0" encoding="UTF-8"?>
<SubscribeResponse SubscriptionID="sub01" expires="2007-01-
01T00:00:00Z" xsi:schemaLocation="http://www.opengis.net/sas/0.0
../sasSubscribe.xsd" xmlns="http://www.opengis.net/sas/0.0"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
      <AcknowledgementChannel>
            <XMPPURI>xmpp:sub01-ack@conference.52North.org</XMPPURI>
      </AcknowledgementChannel>
      <AlertChannel>
            <XMPPURI>xmpp:sub01@conference.52North.org</XMPPURI>
      </AlertChannel>
</SubscribeResponse>
```

16.3.4 Subscribe exceptions

When a SAS server encounters an error while performing a Subscribe operation, it shall return an exception report message as specified in Subclause 8 of [OGC 05-008]. The allowed standard exception codes shall include those listed in Table 21. For each listed exceptionCode, the contents of the "locator" parameter value shall be as specified in the righthand column of Table 21.

NOTE To reduce the need for readers to refer to other documents, the first four values listed below are copied from Table 20 in Subclause 8.3 of [OGC 05-008].

Table 21 — Exception codes for Subscribe operation

exceptionCode value	Meaning of code	"locator" value
OperationNotSupported	Request is for an operation that is not supported by this server	Name of operation not supported
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
NotApplicableCode	No other exceptionCode specified by this service and server applies to this exception	None, omit "locator" parameter

17 RenewSubscription operation (mandatory)

17.1 Introduction

The RenewSubscription operation allows SAS clients to renew a subscription before it expires. This operation becomes necessary as SAS servers determine the maximal time before a subscription expires automatically. Of course the service does not have to extend the subscriptions duration – in this case the previously determined expiration time will still be in effect.

17.2 RenewSubscription operation request

17.2.1 RenewSubscription request parameters

A request to perform the `RenewSubscription` operation shall include the parameters listed and defined in Table 22. This table also specifies the UML model data type, source of values, and multiplicity of each listed parameter, plus the meaning to servers when each optional parameter is not included in the operation request. Although some values listed in the "Name" column appear to contain spaces, they shall not contain spaces.

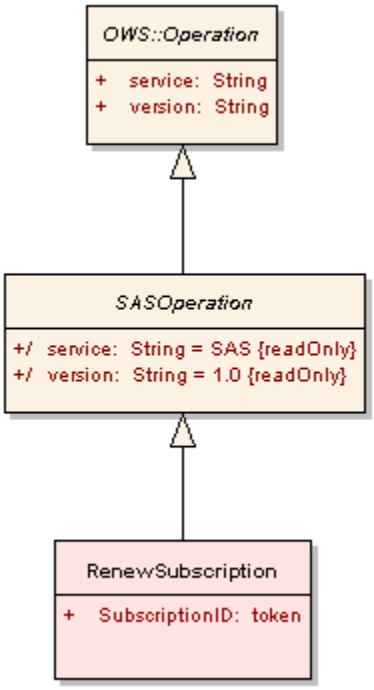


Figure 36: RenewSubscription in UML notation

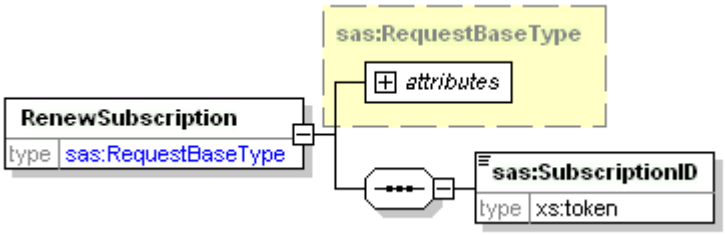


Figure 37: RenewSubscription in XMLSpy notation

Table 22 — Parameters in RenewSubscription operation request

Name	Definition	Data type and values	Multiplicity and use
service	Service type identifier	Character String type "SAS"	One (mandatory)
version	Specification version for operation	Character String type, not empty Value is specified by this Implementation Specification and Schemas version	One (mandatory)
SubscriptionID	Identifier provided by SAS as part of a SubscribeResponse (see Subclause 16)	token	one (mandatory)

The "Multiplicity and use" column in Table 22 specifies the optionality of each listed parameter and data structure in the RenewSubscription operation request. Since all parameters and data structures are mandatory in the operation request, all parameters and data structures shall be implemented by all SAS clients, using a specified value(s). Similarly, all parameters and data structures shall be implemented by all SAS servers, checking that each request parameter is received with any specified value(s).

17.2.2 RenewSubscription request KVP encoding

KVP encoding not supported.

17.2.3 RenewSubscription request XML encoding (mandatory)

All SAS servers shall implement HTTP POST transfer of the RenewSubscription operation request, using XML encoding only. The following schema fragment specifies the contents and structure of a SAS operation request encoded in XML:

See Annex B (sasSubscribe.xsd).

17.2.4 RenewSubscription operation request example

To renew the subscription with ID "sub01", a request like the following could be sent to SAS.

Listing 12: Example RenewSubscription request

```
<?xml version="1.0" encoding="UTF-8"?>
<RenewSubscription xmlns="http://www.opengis.net/sas/0.0"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://www.opengis.net/sas/0.0 ../sasAll.xsd"
service="SAS" version="1.0.0">
  <SubscriptionID>sub01</SubscriptionID>
</RenewSubscription>
```

17.3 RenewSubscription operation response

17.3.1 Normal response parameters

The normal response to a valid RenewSubscription operation request shall be RenewSubscriptionResponse. More precisely, a response from the RenewSubscription operation shall include the parts listed in Table 23. This table also specifies the UML model data type plus the multiplicity and use of each listed part.

Figure 38: RenewSubscriptionResponse in UML notation

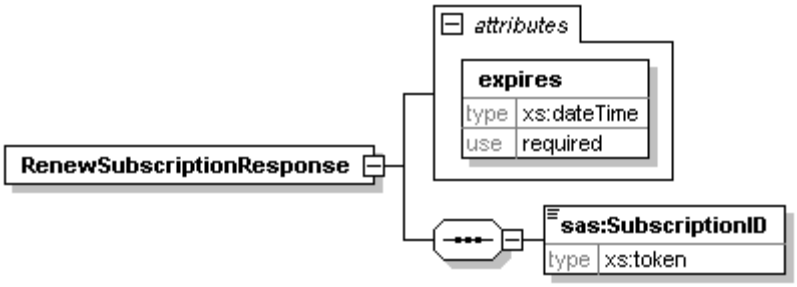


Figure 39: RenewSubscriptionResponse in XMLSpy notation

Table 23 — Parts of RenewSubscription operation response

Name	Definition	Data type and values	Multiplicity and use
SubscriptionID	Identifier provided by SAS as part of a SubscribeResponse (see Subclause 16)	token	one (mandatory)
expires	The time that this subscription automatically terminates. Each SAS is free to choose the maximal time a subscription remains valid. During this period, the client has to send a RenewSubscription request in order to extend it.	String, follows ISO 8601:2000 and is restricted to YYYY-MM-DDThh:mm:ss±hh:mm	one (mandatory)

17.3.2 Normal response XML encoding

The following schema fragment specifies the contents and structure of a RenewSubscription operation response, always encoded in XML:

See Annex B (sasSubscribe.xsd).

17.3.3 RenewSubscription operation response example

In response to RenewSubscription operation request, a SAS server might generate a document that looks like the following.

Listing 13: Example RenewSubscription response

```
<?xml version="1.0" encoding="UTF-8"?>
<RenewSubscriptionResponse xmlns="http://www.opengis.net/sas/0.0"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://www.opengis.net/sas/0.0 ../sasAll.xsd"
expires="2007-06-01T00:00:00Z">
  <SubscriptionID>sub013</SubscriptionID>
</RenewSubscriptionResponse>
```

17.3.4 RenewSubscription exceptions

When a SAS server encounters an error while performing a RenewSubscription operation, it shall return an exception report message as specified in Subclause 8 of [OGC 05-008]. The allowed standard exception codes shall include those listed in Table 24. For each listed exceptionCode, the contents of the “locator” parameter value shall be as specified in the righthand column of Table 24.

NOTE To reduce the need for readers to refer to other documents, the first four values listed below are copied from Table 20 in Subclause 8.3 of [OGC 05-008].

Table 24 — Exception codes for RenewSubscription operation

exceptionCode value	Meaning of code	"locator" value
OperationNotSupported	Request is for an operation that is not supported by this server	Name of operation not supported
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
NotApplicableCode	No other exceptionCode specified by this service and server applies to this exception	None, omit "locator" parameter

18 CancelSubscription operation (mandatory)

18.1 Introduction

The CancelSubscription operation allows SAS clients to cancel a subscription.

18.2 CancelSubscription operation request

18.2.1 CancelSubscription request parameters

A request to perform the `CancelSubscription` operation shall include the parameters listed and defined in Table 25. This table also specifies the UML model data type, source of values, and multiplicity of each listed parameter, plus the meaning to servers when each optional parameter is not included in the operation request. Although some values listed in the "Name" column appear to contain spaces, they shall not contain spaces.

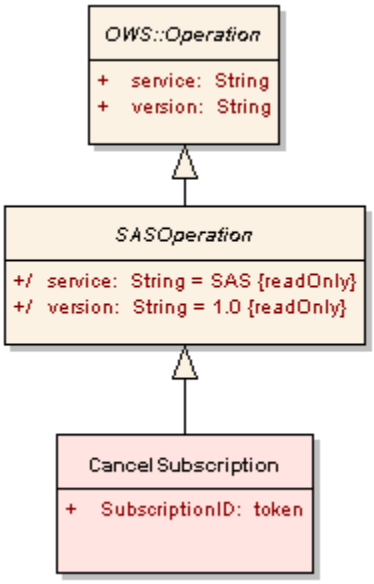


Figure 40: CancelSubscription in UML notation

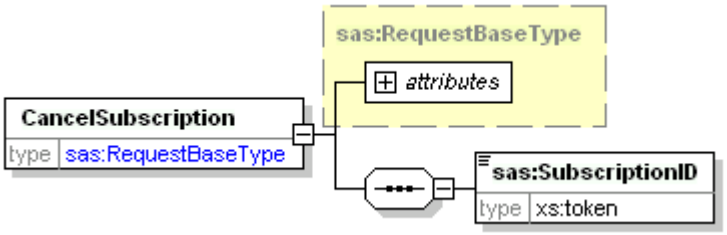


Figure 41: CancelSubscription in XMLSpy notation

Table 25 — Parameters in CancelSubscription operation request

Name	Definition	Data type and values	Multiplicity and use
service	Service type identifier	Character String type, not empty Value is "SAS"	One (mandatory)
version	Specification version for operation	Character String type, not empty Value is specified by this Implementation Specification and Schemas version	One (mandatory)
SubscriptionID	Unique Identifier for the subscription. Provided by SAS server.	ID	One (mandatory)

The "Multiplicity and use" column in Table 25 specifies the optionality of each listed parameter and data structure in the CancelSubscription operation request. Since all parameters and data structures are mandatory in the operation request, all parameters and data structures shall be implemented by all SAS clients, using a specified value(s).

Similarly, all parameters and data structures shall be implemented by all SAS servers, checking that each request parameter is received with any specified value(s).

18.2.2 CancelSubscription request KVP encoding

KVP encoding not supported.

18.2.3 CancelSubscription request XML encoding (mandatory)

All SAS servers shall implement HTTP POST transfer of the CancelSubscription operation request, using XML encoding only. The following schema fragment specifies the contents and structure of a CancelSubscription operation request encoded in XML:

See Annex B (sasSubscribe.xsd).

18.2.4 CancelSubscription operation request example

To cancel the subscription with ID “sub01”, a request like the following would have to be sent to SAS.

Listing 14: Example CancelSubscription request

```
<?xml version="1.0" encoding="UTF-8"?>
<CancelSubscription xmlns="http://www.opengis.net/sas/0.0"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://www.opengis.net/sas/0.0 ../sasAll.xsd"
service="SAS" version="1.0.0">
  <SubscriptionID>sub01</SubscriptionID>
</CancelSubscription>
```

18.3 CancelSubscription operation response

18.3.1 Normal response parameters

The normal response to a valid CancelSubscription operation request shall be CancelSubscriptionResponse. More precisely, a response from the CancelSubscription operation shall include the parts listed in Table 26. This table also specifies the UML model data type plus the multiplicity and use of each listed part.

Figure 42: CancelSubscriptionResponse in UML notation

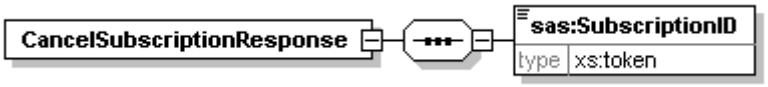


Figure 43: CancelSubscriptionResponse in XMLSpy notation

Table 26 — Parts of CancelSubscription operation response

Name	Definition	Data type and values	Multiplicity and use
SubscriptionID	Unique identifier. Just returned as given in the CancelSubscription request. Helps the calling client to differentiate multiple calls of the same operation, though the synchronous schema should make this superficial.	ID	One (mandatory)

18.3.2 Normal response XML encoding

The following schema fragment specifies the contents and structure of a CancelSubscription operation response, always encoded in XML:

See Annex B (sasSubscribe.xsd).

18.3.3 CancelSubscription operation response example

In response to CancelSubscription operation request, a SAS server might generate a document that looks like the following.

Listing 15: Example CancelSubscription response

```
<?xml version="1.0" encoding="UTF-8"?>
<CancelSubscriptionResponse xmlns="http://www.opengis.net/sas/0.0"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://www.opengis.net/sas/0.0 ../sasAll.xsd">
  <SubscriptionID>sub01</SubscriptionID>
</CancelSubscriptionResponse>
```

18.3.4 CancelSubscription exceptions

When a SAS server encounters an error while performing a CancelSubscription operation, it shall return an exception report message as specified in Subclause 8 of [OGC 05-008]. The allowed standard exception codes shall include those listed in Table 27. For each listed exceptionCode, the contents of the “locator” parameter value shall be as specified in the righthand column of Table 27.

NOTE To reduce the need for readers to refer to other documents, the first four values listed below are copied from Table 20 in Subclause 8.3 of [OGC 05-008].

Table 27 — Exception codes for CancelSubscription operation

exceptionCode value	Meaning of code	"locator" value
OperationNotSupported	Request is for an operation that is not supported by this server	Name of operation not supported
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
NotApplicableCode	No other exceptionCode specified by this service and server applies to this exception	None, omit "locator" parameter

19 DescribeSensor operation (mandatory)

19.1 Introduction

The DescribeSensor operation allows SAS clients to receive information about a sensor, encoded in SensorML.

19.2 DescribeSensor operation request

19.2.1 DescribeSensor request parameters

A request to perform the `DescribeSensor` operation shall include the parameters listed and defined in Table 28. This table also specifies the UML model data type, source of values, and multiplicity of each listed parameter, plus the meaning to servers when each optional parameter is not included in the operation request. Although some values listed in the "Name" column appear to contain spaces, they shall not contain spaces.

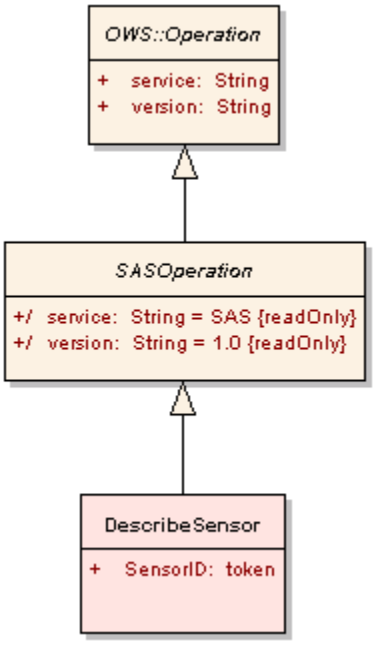


Figure 44: DescribeSensor in UML notation

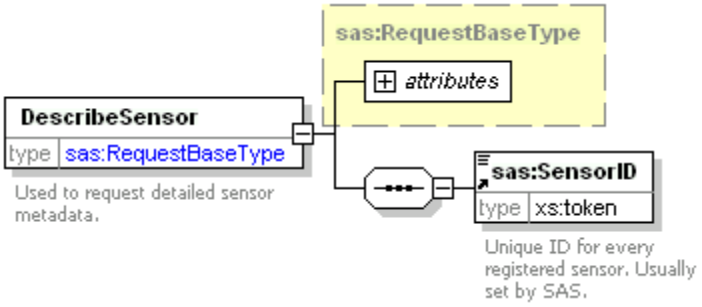


Figure 45: DescribeSensor in XMLSpy notation

Table 28 — Parameters in DescribeSensor operation request

Name	Definition	Data type and values	Multiplicity and use
service	Service type identifier	Character String type, not empty Value is "SAS"	One (mandatory)
version	Specification version for operation	Character String type, not empty Value is specified by this Implementation Specification and Schemas version	One (mandatory)
SensorID	Unique Identifier for the sensor. Provided by SAS server.	token	One (mandatory)

The "Multiplicity and use" column in Table 28 specifies the optionality of each listed parameter and data structure in the `DescribeSensor` operation request. Since all parameters and data structures are mandatory in the operation request, all parameters and data structures shall be implemented by all SAS clients, using a specified value(s). Similarly, all parameters and data structures shall be implemented by all SAS servers, checking that each request parameter is received with any specified value(s).

19.2.2 DescribeSensor request KVP encoding

KVP encoding not supported.

19.2.3 DescribeSensor request XML encoding (mandatory)

All SAS servers shall implement HTTP POST transfer of the `DescribeSensor` operation request, using XML encoding only. The following schema fragment specifies the contents and structure of a `DescribeSensor` operation request encoded in XML:

See Annex B (sasDescribeSensor.xsd).

19.2.4 DescribeSensor operation request example

To receive the SensorML description of sensor '`urn:x-ogc:object:sensor:IFGI:Temp:1`', a request like the following would have to be sent to SAS.

Listing 16: DescribeSensor request example

```
<?xml version="1.0" encoding="UTF-8"?>
<DescribeSensor xmlns="http://www.opengis.net/sas/0.0"
xmlns:sml="http://www.opengis.net/sensorML/1.0"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
service="SAS" version="1.0.0">
 <SensorID>urn:x-ogc:object:sensor:IFGI:Temp:1</SensorID>
</DescribeSensor>
```

19.3 DescribeSensor operation response

The DescribeSensor response contains the SensorML document for the sensor – either inline or by reference.

19.3.1 Normal response parameters

The normal response to a valid `DescribeSensor` operation request shall be `DescribeSensorResponse`. More precisely, a response from the `DescribeSensor` operation shall include the parts listed in Table 29. This table also specifies the UML model data type plus the multiplicity and use of each listed part.

Figure 46: DescribeSensorResponse in UML notation

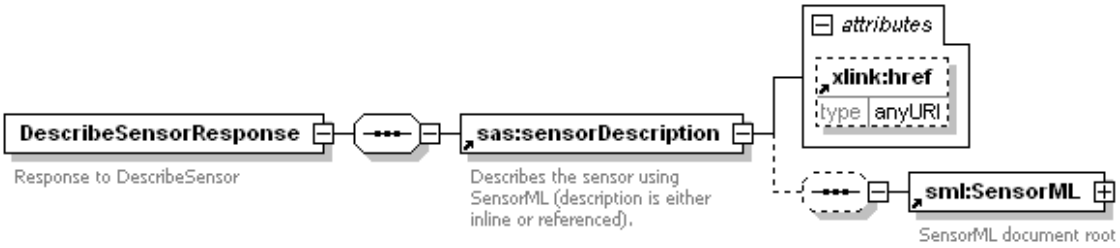


Figure 47: DescribeSensorResponse in XMLSpy notation

Table 29 — Parts of DescribeSensor operation response

Name	Definition	Data type and values	Multiplicity and use
sensorDescription	provides information about the sensor either inline or by reference	href or complex XML	One (mandatory)

19.3.2 Normal response XML encoding

The following schema fragment specifies the contents and structure of a `DescribeSensor` operation response, always encoded in XML:

See Annex B (sasDescribeSensor.xsd).

19.3.3 DescribeSensor operation response example

A `DescribeSensor` operation response for SAS can look like the following.

Listing 17: Example DescribeSensor response

```
<?xml version="1.0" encoding="UTF-8"?>
<DescribeSensorResponse xmlns="http://www.opengis.net/sas/0.0"
xmlns:xlink="http://www.w3.org/1999/xlink"
xmlns:sml="http://www.opengis.net/sensorML/1.0"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
 <sensorDescription xlink:href="http://www.52north.org/swe/Temp1.xml"/>
</DescribeSensorResponse>
```

19.3.4 DescribeSensor exceptions

When a SAS server encounters an error while performing a `DescribeSensor` operation, it shall return an exception report message as specified in Subclause 8 of [OGC 05-008]. The allowed standard exception codes shall include those listed in Table 30. For each listed exceptionCode, the contents of the "locator" parameter value shall be as specified in the righthand column of Table 30.

NOTE To reduce the need for readers to refer to other documents, the first four values listed below are copied from Table 20 in Subclause 8.3 of [OGC 05-008].

Table 30 — Exception codes for DescribeSensor operation

exceptionCode value	Meaning of code	"locator" value
OperationNotSupported	Request is for an operation that is not supported by this server	Name of operation not supported
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
NotApplicableCode	No other exceptionCode specified by this service and server applies to this exception	None, omit "locator" parameter

20 DescribeAlert operation (mandatory)

20.1 Introduction

The `DescribeAlert` operation allows SAS clients to get the description of the alert message structure of a particular sensor referenced by its ID. This operation is useful in scenarios where clients do not subscribe for all alerts generated by a specific sensor but create a custom filter condition instead – which also applies for events from previously unknown sensors. As the structure of data contained in an alert may differ from sensor to sensor, clients should make use of the DescribeAlert operation to receive the description of data coming from a previously unknown sensor. Once this description has been parsed by the client, it is able to consume all alerts from that sensor.

20.2 DescribeAlert operation request

20.2.1 DescribeAlert request parameters

A request to perform the `DescribeAlert` operation shall include the parameters listed and defined in Table 31. This table also specifies the UML model data type, source of values, and multiplicity of each listed parameter, plus the meaning to servers when each optional parameter is not included in the operation request. Although some values listed in the "Name" column appear to contain spaces, they shall not contain spaces.

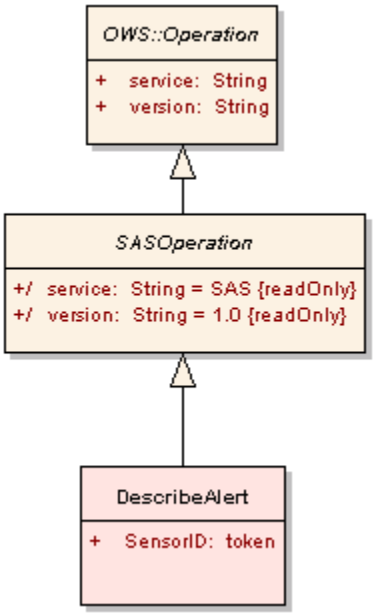


Figure 48: DescribeAlert in UML notation

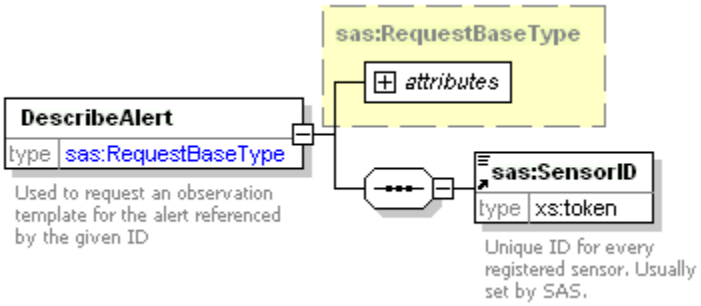


Figure 49: DescribeAlert in XMLSpy notation

Table 31 — Parameters in DescribeAlert operation request

Name	Definition	Data type and values	Multiplicity and use
service	Service type identifier	Character String type, not empty Value is “SAS”	One (mandatory)
version	Specification version for operation	Character String type, not empty Value is specified by this Implementation Specification and Schemas version	One (mandatory)
SensorID	Unique Identifier for the sensor. Provided by SAS server.	xs:token	One (mandatory)

The “Multiplicity and use” column in Table 31 specifies the optionality of each listed parameter and data structure in the DescribeAlert operation request. Since all parameters and data structures are mandatory in the operation request, all parameters and data

structures shall be implemented by all SAS clients, using a specified value(s). Similarly, all parameters and data structures shall be implemented by all SAS servers, checking that each request parameter is received with any specified value(s).

20.2.2 DescribeAlert request KVP encoding

KVP encoding not supported.

20.2.3 DescribeAlert request XML encoding (mandatory)

All SAS servers shall implement HTTP POST transfer of the `DescribeAlert` operation request, using XML encoding only. The following schema fragment specifies the contents and structure of a `DescribeAlert` operation request encoded in XML:

See Annex B (sasDescribeAlert.xsd).

20.2.4 DescribeAlert operation request example

To receive the description of the message structure for data sent by sensor '`urn:x-ogc:object:sensor:IFGI:Temp:1`', a request like the following would have to be sent to SAS.

Listing 18: DescribeAlert request example

```
<?xml version="1.0" encoding="UTF-8"?>
<DescribeAlert xmlns="http://www.opengis.net/sas/0.0"
xmlns:swe="http://www.opengis.net/swe/1.0"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
service="SAS" version="1.0.0">
 <SensorID>urn:x-ogc:object:sensor:IFGI:Temp:1</SensorID>
</DescribeAlert>
```

20.3 DescribeAlert operation response

20.3.1 Normal response parameters

The normal response to a valid `DescribeAlert` operation request shall be `DescribeAlertResponse`. More precisely, a response from the `DescribeAlert` operation shall include the parts listed in Table 32. This table also specifies the UML model data type plus the multiplicity and use of each listed part.

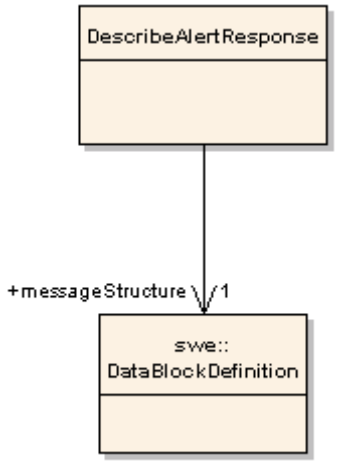


Figure 50: DescribeAlertResponse in UML notation

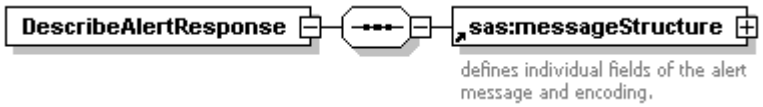


Figure 51: DescribeAlertResponse in XMLSpy notation

Table 32 — Parts of DescribeAlertResponse operation response

Name	Definition	Data type and values	Multiplicity and use
messageStructure	Defines the structure of an alert message	complex, swe:DataBlockDefinition	One (mandatory)

20.3.2 Normal response XML encoding

The following schema fragment specifies the contents and structure of a `DescribeAlert` operation response, always encoded in XML:

See Annex B (sasDescribeAlert.xsd).

20.3.3 DescribeAlert operation response example

A `DescribeAlert` operation response for SAS can look like the following.

Listing 19: DescribeAlertResponse example

```
<?xml version="1.0" encoding="UTF-8"?>
<DescribeAlertResponse xmlns="http://www.opengis.net/sas/0.0"
xmlns:swe="http://www.opengis.net/swe/1.0"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
 <messageStructure>
```

```
   <swe:DataBlockDefinition>
     <swe:components name="sensor data structure">
       <swe:DataRecord>
         <swe:field name="component1">
           <swe:Quantity definition="urn:x-
ogc:def:phenomenon:OGC:Temperature">
             <swe:uom code="Cel"/>
           </swe:Quantity>
         </swe:field>
         <swe:field name="component2">
           <swe:Quantity definition="urn:x-
ogc:def:phenomenon:OGC:RelativeHumidity">
             <swe:uom code="%"/>
           </swe:Quantity>
         </swe:field>
         <swe:field name="component3">
           <swe:Position definition="urn:x-
ogc:def:phenomenon:OGC:sampleLocation" referenceFrame="urn:x-
ogc:def:crs:EPSG:6.11:4326">
             <swe:location>
               <swe:Vector>
                 <swe:coordinate name="latitude">
                   <swe:Quantity>
                     <swe:uom code="deg"/>
                   </swe:Quantity>
                 </swe:coordinate>
                 <swe:coordinate name="longitude">
                   <swe:Quantity>
                     <swe:uom code="deg"/>
                   </swe:Quantity>
                 </swe:coordinate>
               </swe:Vector>
             </swe:location>
           </swe:Position>
         </swe:field>
       </swe:DataRecord>
     </swe:components>
     <swe:encoding>
```

```
       <swe:TextBlock decimalSeparator="." blockSeparator="@@"
tokenSeparator=" "/>
     </swe:encoding>
   </swe:DataBlockDefinition>
 </messageStructure>
</DescribeAlertResponse>
```

20.3.4 DescribeAlertResponse exceptions

When a SAS server encounters an error while performing a `DescribeAlert` operation, it shall return an exception report message as specified in Subclause 8 of [OGC 05-008]. The allowed standard exception codes shall include those listed in Table 33. For each listed exceptionCode, the contents of the "locator" parameter value shall be as specified in the righthand column of Table 33.

NOTE To reduce the need for readers to refer to other documents, the first four values listed below are copied from Table 20 in Subclause 8.3 of [OGC 05-008].

Table 33 — Exception codes for DescribeAlert operation

exceptionCode value	Meaning of code	"locator" value
OperationNotSupported	Request is for an operation that is not supported by this server	Name of operation not supported
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
NotApplicableCode	No other exceptionCode specified by this service and server applies to this exception	None, omit "locator" parameter

21 Alerting via other communication channels

This chapter describes sending alerts to clients via WNS or any other supported channel. The information provided by the client when subscribing for an alert (in the NotificationTarget) is sufficient for a SAS to alert the client. The format in which these alerts have to be sent is specified by SAS and WNS as well.

The basic message format defined by SAS for delivering alerts to clients that do not directly communicate via XMPP (i.e., if they provided a NotificationTarget in the Subscribe request) is presented in the following figure.

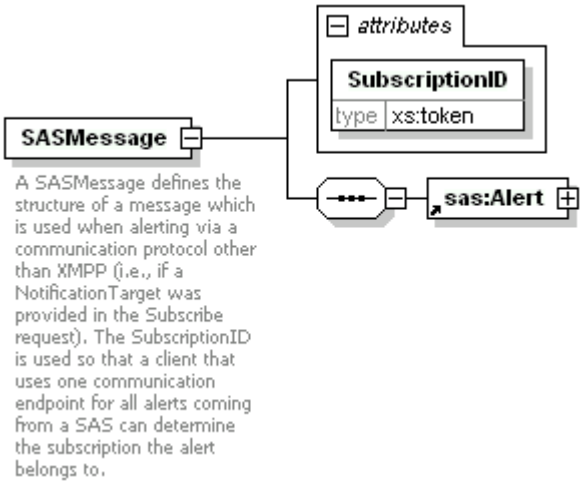


Figure 52: SASMessage in XMLSpy notation

In addition, a SASMessage is encapsulated in a NotificationMessage (see WNS specification [9] for more detailed information). A NotificationMessage provides information to automatically parse incoming messages and to uniquely identify which service sent the message. Thus, one communication endpoint of the client can even be used to receive alerts from multiple subscriptions at multiple SASs. Listing 20 provides an example of a NotificationMessage sent by SAS.

Listing 20: Example NotificationMessage sent by SAS

```
<?xml version="1.0" encoding="UTF-8"?>
<NotificationMessage xmlns="http://www.opengis.net/wns/0.0"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
 <ServiceDescription>
   <ServiceType>SAS</ServiceType>
   <ServiceTypeVersion>1.0.0</ServiceTypeVersion>
   <ServiceURL>http://mars.uni-muenster.de/52nSAS/SAS</ServiceURL>
 </ServiceDescription>
 <Payload>
   <SASMessage xmlns="http://www.opengis.net/sas/0.0"
SubscriptionID="sub01">
     <Alert>
       <SensorID>urn:x-ogc:object:sensor:IFGI:Temp:1</SensorID>
       <Timestamp>2007-01-24T14:18:22Z</Timestamp>
       <AlertData>5.4 90.2 51.9424 7.692</AlertData>
     </Alert>
   </SASMessage>
 </Payload>
```

```
</NotificationMessage>
```

When notifiying via SMS, phone or WNS the SAS should use a short, human readable message which indicates at least the service URL and the ID of the subscription for which an alert was generated. If required by the communication protocol (e.g. SMS) the message should be as short as possible.

22 Version numbering and negotiation

22.1 Version number form and value

The Sensor Alert Service (SAS) defines a protocol version number. The version number applies to the XML schema and the request encodings defined in this International Standard. The version number contains three non-negative integers, separated by decimal points, in the form "*x*.*y*.*z*". The numbers "*y*" and "*z*" shall not exceed 99. Implementations of this International Standard shall use the value "1.0.0" as the protocol version number.

22.2 Version number changes

The protocol version number shall be changed with each revision of this International Standard. The number shall increase monotonically and shall comprise no more than three integers separated by decimal points, with the first integer being the most significant. There may be gaps in the numerical sequence. Some numbers may denote draft versions. Servers and their clients need not support all defined versions, but shall obey the negotiation rules below.

22.3 Appearance in requests and in service metadata

The version number shall appear in at least two places: in the service metadata and in the parameter list of client requests to a server. The version number used in a client's request of a particular server shall be equal to a version number which that server has declared it supports (except during negotiation, as described below). A server may support several versions, whose values clients may discover according to the negotiation rules.

22.4 Version number negotiation

A WMS client may negotiate with a server to determine a mutually agreeable protocol version. Negotiation is performed using the GetCapabilities operation (described in 11) according to the following rules.

All service metadata shall include a protocol version number and shall comply with the XML Schema defined for that version. In response to a GetCapabilities request that does not specify a version number, the server shall respond with the highest version it

supports. In response to a GetCapabilities request containing a version number that the server implements, the server shall send that version. If the server does not support the requested version, the server shall respond with output that conforms to a version it does support, as determined by the following rules:

- If a version unknown to the server and higher than the lowest supported version is requested, the server shall send the highest version it supports that is less than the requested version.
- If a version lower than any of those known to the server is requested, then the server shall send the lowest version it supports.
- If the client does not support the version sent by the server, it may either cease communicating with the server or send a new request with a different version number that the client does support.

The process may be repeated until a mutually understood version is reached, or until the client determines that it will not or cannot communicate with that particular server.

EXAMPLE 1 Server understands versions 1, 2, 4, 5 and 8. Client understands versions 1, 3, 4, 6, and 7. Client requests version 7. Server responds with version 5. Client requests version 4. Server responds with version 4, which the client understands, and the negotiation ends successfully.

Annex A
(normative)

Abstract test suite

A.1 SAS Client

Basic Service Elements

Information for the basis service elements is as follows:

a) Test Purpose: Verify that a SAS client satisfies the requirements for request parameter rules.

b) Test Method: Generate an adequate sample of requests from the client and verify that each is a valid request.

c) Reference: 10

d) Test Type: Basic

GetCapabilities Request

Information for the GetCapabilities request is as follows:

a) Test Purpose: Verify that a SAS client satisfies all requirements for a GetCapabilities request.

b) Test Method: Generate an adequate sample of GetCapabilities requests from the client and verify that each is a valid request.

c) Reference: 11

d) Test Type: Basic

GetWSDL Request

Information for the GetWSDL request is as follows:

a) Test Purpose: Verify that a SAS client satisfies all requirements for a GetWSDL request.

b) Test Method: Generate an adequate sample of GetWSDL requests from the client and verify that each is a valid request.

c) Reference: 12

d) Test Type: Basic

Advertise Request

Information for the Advertise request is as follows:

a) Test Purpose: Verify that a SAS client satisfies all requirements for a Advertise request.

b) Test Method: Generate an adequate sample of Advertise requests from the client and verify that each is a valid request.

c) Reference: 13

d) Test Type: Basic

RenewAdvertisement Request

Information for the RenewAdvertisement request is as follows:

a) Test Purpose: Verify that a SAS client satisfies all requirements for a RenewAdvertisement request.

b) Test Method: Generate an adequate sample of RenewAdvertisement requests from the client and verify that each is a valid request.

c) Reference: 14

d) Test Type: Basic

CancelAdvertisement Request

Information for the CancelAdvertisement request is as follows:

a) Test Purpose: Verify that a SAS client satisfies all requirements for a CancelAdvertisement request.

b) Test Method: Generate an adequate sample of CancelAdvertisement requests from the client and verify that each is a valid request.

c) Reference: 15

d) Test Type: Basic

Subscribe Request

Information for the Subscribe request is as follows:

a) Test Purpose: Verify that a SAS client satisfies all requirements for a Subscribe request.

b) Test Method: Generate an adequate sample of Subscribe requests from the client and verify that each is a valid request.

c) Reference: 16

d) Test Type: Basic

RenewSubscription Request

Information for the RenewSubscription request is as follows:

a) Test Purpose: Verify that a SAS client satisfies all requirements for a RenewSubscription request.

b) Test Method: Generate an adequate sample of RenewSubscription requests from the client and verify that each is a valid request.

c) Reference: 17

d) Test Type: Basic

CancelSubscription Request

Information for the CancelSubscription request is as follows:

a) Test Purpose: Verify that a SAS client satisfies all requirements for a CancelSubscription request.

b) Test Method: Generate an adequate sample of CancelSubscription requests from the client and verify that each is a valid request.

c) Reference: 18

d) Test Type: Basic

DescribeSensor Request

Information for the DescribeSensor request is as follows:

a) Test Purpose: Verify that a SAS client satisfies all requirements for a DescribeSensor request.

b) Test Method: Generate an adequate sample of DescribeSensor requests from the client and verify that each is a valid request.

c) Reference: 19

d) Test Type: Basic

DescribeAlert Request

Information for the DescribeAlert request is as follows:

a) Test Purpose: Verify that a SAS client satisfies all requirements for a DescribeAlert request.

b) Test Method: Generate an adequate sample of DescribeAlert requests from the client and verify that each is a valid request.

c) Reference: 20

d) Test Type: Basic

A.2 SAS Server

Version negotiation

Information for the version negotiation is as follows:

a) Test Purpose: Verify that a basic WMS server satisfies the requirements for version negotiation.

b) Test Method: Submit requests containing version numbers both lower than and higher than the version supported by the server. Verify that the server responses are in accord with the rules for version negotiation.

c) Reference: 6.2

d) Test Type: Basic

Request parameter rules

Information for the request parameter rules is as follows:

a) Test Purpose: Verify that a basic WMS server satisfies the requirements for request parameter rules.

b) Test Method: Generate a sample of requests from a client. Include both invalid requests and valid request that vary within the limits allowed by the rules. Verify that the server provides an appropriate response in each case.

c) Reference: 5-21

d) Test Type: Basic

GetCapabilities Response

Information for the GetCapabilities response is as follows:

a) Test Purpose: Verify that a SAS server satisfies all requirements of the GetCapabilities operation.

b) Test Method: Make several GetCapabilities requests using a variety of input parameters. Verify that an appropriate response is returned in each case.

c) Reference: 11

d) Test Type: Basic

GetWSDL Response

Information for the GetWSDL response is as follows:

a) Test Purpose: Verify that a SAS server satisfies all requirements of the GetWSDL operation.

b) Test Method: Make several GetWSDL requests using a variety of input parameters. Verify that an appropriate response is returned in each case.

c) Reference: 12

d) Test Type: Basic

Advertise Response

Information for the Advertise response is as follows:

a) Test Purpose: Verify that a SAS server satisfies all requirements of the Advertise operation.

b) Test Method: Make several Advertise requests using a variety of input parameters. Verify that an appropriate response is returned in each case.

c) Reference: 13

d) Test Type: Basic

RenewAdvertisement Response

Information for the RenewAdvertisement response is as follows:

a) Test Purpose: Verify that a SAS server satisfies all requirements of the RenewAdvertisement operation.

b) Test Method: Make several RenewAdvertisement requests using a variety of input parameters. Verify that an appropriate response is returned in each case.

c) Reference: 14

d) Test Type: Basic

CancelAdvertisement Response

Information for the CancelAdvertisement response is as follows:

a) Test Purpose: Verify that a SAS server satisfies all requirements of the CancelAdvertisement operation.

b) Test Method: Make several CancelAdvertisement requests using a variety of input parameters. Verify that an appropriate response is returned in each case.

c) Reference: 15

d) Test Type: Basic

Subscribe Response

Information for the Subscribe response is as follows:

a) Test Purpose: Verify that a SAS server satisfies all requirements of the Subscribe operation.

b) Test Method: Make several Subscribe requests using a variety of input parameters. Verify that an appropriate response is returned in each case.

c) Reference: 16

d) Test Type: Basic

RenewSubscription Response

Information for the RenewSubscription response is as follows:

a) Test Purpose: Verify that a SAS server satisfies all requirements of the RenewSubscription operation.

b) Test Method: Make several RenewSubscription requests using a variety of input parameters. Verify that an appropriate response is returned in each case.

c) Reference: 17

d) Test Type: Basic

CancelSubscription Response

Information for the CancelSubscription response is as follows:

a) Test Purpose: Verify that a SAS server satisfies all requirements of the CancelSubscription operation.

b) Test Method: Make several CancelSubscription requests using a variety of input parameters. Verify that an appropriate response is returned in each case.

c) Reference: 18

d) Test Type: Basic

DescribeSensor Response

Information for the DescribeSensor response is as follows:

a) Test Purpose: Verify that a SAS server satisfies all requirements of the DescribeSensor operation.

b) Test Method: Make several DescribeSensor requests using a variety of input parameters. Verify that an appropriate response is returned in each case.

c) Reference: 19

d) Test Type: Basic

DescribeAlert Response

Information for the DescribeAlert response is as follows:

a) Test Purpose: Verify that a SAS server satisfies all requirements of the DescribeAlert operation.

b) Test Method: Make several DescribeAlert requests using a variety of input parameters. Verify that an appropriate response is returned in each case.

c) Reference: 20

d) Test Type: Basic

Annex B
(normative)

XML Schema Documents

In addition to this document, this specification includes several normative XML Schema Documents. These XML Schema Documents are bundled in a zip file with the present document. After OGC acceptance of a Version 1.0.0 of this specification, these XML Schema Documents will also be posted online at the URL http://schemas.opengeospatial.net/sas/1.0.0. In the event of a discrepancy between the bundled and online versions of the XML Schema Documents, the online files shall be considered authoritative.

These XML Schema Documents use and build on the OWS common XML Schema Documents specified [OGC 05-008], named:

ows19115subset.xsd

owsCommon.xsd

owsDataIdentification.xsd

owsExceptionReport.xsd

owsGetCapabilities.xsd

owsOperationsMetadata.xsd

owsServiceIdentification.xsd

owsServiceProvider.xsd

All these XML Schema Documents contain documentation of the meaning of each element and attribute, and this documentation shall be considered normative as specified in Subclause 11.6.3 of [OGC 05-008].

sasAdvertise.xsd

```
<?xml version="1.0" encoding="UTF-8"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema"
xmlns:sas="http://www.opengis.net/sas/0.0"
xmlns:swe="http://www.opengis.net/swe/1.0"
xmlns:sml="http://www.opengis.net/sensorML/1.0"
targetNamespace="http://www.opengis.net/sas/0.0"
elementFormDefault="qualified" attributeFormDefault="unqualified"
version="0.0.0" xml:lang="en">
 <!-- ============================================================
      and imports
      ============================================================ -->
 <xs:include schemaLocation="./sasCommon.xsd"/>
 <!-- ============================================================


      ============================================================ -->
 <xs:element name="Advertise">
   <xs:annotation>
     <xs:documentation>Request to a SAS to allow a publisher to publish
alerts.</xs:documentation>
   </xs:annotation>
   <xs:complexType>
     <xs:complexContent>
       <xs:extension base="sas:RequestBaseType">
         <xs:sequence>
           <xs:element ref="sas:messageStructure"/>
           <xs:element ref="sas:sensorDescription"/>
           <xs:element ref="sas:reportingFrequency" minOccurs="0"/>
           <xs:element ref="sas:desiredPublicationExpiration"
minOccurs="0"/>
           <xs:element ref="sas:Location" minOccurs="0"/>
           <xs:element ref="sas:XMPPCredentials" minOccurs="0"/>
           <xs:element ref="sas:ReliableCommunication" minOccurs="0"/>
         </xs:sequence>
       </xs:extension>
     </xs:complexContent>
   </xs:complexType>
 </xs:element>
 <xs:element name="AdvertiseResponse">
   <xs:complexType>
```

```
    <xs:sequence>
      <xs:element ref="sas:SensorID">
        <xs:annotation>
          <xs:documentation>unique ID, submitted in response to an
advertise-request. Is used in cancel- or renewAdvertisement
requests.</xs:documentation>
        </xs:annotation>
      </xs:element>
      <xs:element ref="sas:AlertChannel"/>
      <xs:element ref="sas:XMPPCredentials" minOccurs="0"/>
      <xs:element ref="sas:AcknowledgementChannel" minOccurs="0"/>
    </xs:sequence>
    <xs:attribute name="expires" type="xs:dateTime" use="required"/>
  </xs:complexType>
</xs:element>
<xs:element name="CancelAdvertisement">
  <xs:complexType>
    <xs:complexContent>
      <xs:extension base="sas:RequestBaseType">
        <xs:sequence>
          <xs:element ref="sas:SensorID"/>
        </xs:sequence>
      </xs:extension>
    </xs:complexContent>
  </xs:complexType>
</xs:element>
<xs:element name="CancelAdvertisementResponse">
  <xs:complexType>
    <xs:sequence>
      <xs:element ref="sas:SensorID"/>
      <xs:element name="CancellationStatus">
        <xs:simpleType>
          <xs:restriction base="xs:string">
            <xs:enumeration value="confirmed"/>
            <xs:enumeration value="expired"/>
            <xs:enumeration value="invalidSensorID"/>
          </xs:restriction>
        </xs:simpleType>
```

```
      </xs:element>
    </xs:sequence>
  </xs:complexType>
 </xs:element>
 <xs:element name="RenewAdvertisement">
  <xs:complexType>
    <xs:complexContent>
      <xs:extension base="sas:RequestBaseType">
        <xs:sequence>
          <xs:element ref="sas:SensorID"/>
          <xs:element ref="sas:desiredPublicationExpiration"
minOccurs="0"/>
        </xs:sequence>
      </xs:extension>
    </xs:complexContent>
  </xs:complexType>
 </xs:element>
 <xs:element name="RenewAdvertisementResponse">
  <xs:complexType>
    <xs:sequence>
      <xs:element ref="sas:SensorID"/>
      <xs:element name="renewalStatus">
        <xs:annotation>
          <xs:documentation>Indicator shows if renewal request was
confirmed or rejected.</xs:documentation>
        </xs:annotation>
        <xs:simpleType>
          <xs:restriction base="xs:string">
            <xs:enumeration value="confirmed"/>
            <xs:enumeration value="rejected"/>
          </xs:restriction>
        </xs:simpleType>
      </xs:element>
    </xs:sequence>
    <xs:attribute name="expires" type="xs:dateTime" use="required"/>
  </xs:complexType>
 </xs:element>
</xs:schema>
```

sasAll.xsd

```
<?xml version="1.0" encoding="UTF-8"?>

<xs:schema xmlns:sas="http://www.opengis.net/sas/0.0"
xmlns:xs="http://www.w3.org/2001/XMLSchema"
targetNamespace="http://www.opengis.net/sas/0.0"
elementFormDefault="qualified" attributeFormDefault="unqualified"
version="0.0.0" xml:lang="en">

 <xs:annotation>

   <xs:appinfo>sasAll.xsd 2006/05/11</xs:appinfo>

   <xs:documentation>

     <description>This XML Schema includes, directly and indirectly,
all the XML Schemas defined by the OGC Sensor Alert Service
(SAS).</description>

     <copyright>Copyright (c) 2006 Institut for Geoinformatics
University of Muenster</copyright>

   </xs:documentation>

 </xs:annotation>

 <!-- ============================================================

      and imports

      ============================================================ -->

 <xs:include schemaLocation="sasCapabilities.xsd"/>

 <xs:include schemaLocation="sasAdvertise.xsd"/>

 <xs:include schemaLocation="sasSubscribe.xsd"/>

 <xs:include schemaLocation="sasDescribeSensor.xsd"/>

 <xs:include schemaLocation="sasDescribeAlert.xsd"/>

 <xs:include schemaLocation="sasMessageSchema.xsd"/>

</xs:schema>
```

sasCapabilities.xsd

```
<?xml version="1.0" encoding="UTF-8"?>

<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema"
xmlns:ows="http://www.opengis.net/ows"
xmlns:sas="http://www.opengis.net/sas/0.0"
xmlns:wns="http://www.opengis.net/wns/0.0"
targetNamespace="http://www.opengis.net/sas/0.0"
elementFormDefault="qualified" attributeFormDefault="unqualified"
version="0.0.0" xml:lang="en">

 <!-- ============================================================

      and imports

      ============================================================ -->
```

```
 <xs:import namespace="http://www.opengis.net/ows"
schemaLocation="http://schemas.opengis.net/ows/1.0.0/owsGetCapabilities
.xsd"/>

 <xs:import namespace="http://www.opengis.net/wns/0.0"
schemaLocation="../../wns/0.0.0/wnsShared.xsd"/>

 <xs:include schemaLocation="./sasContents.xsd"/>

 <!-- ============================================================

      and types

      ============================================================ -->

 <xs:element name="GetCapabilities">

   <xs:annotation>

     <xs:documentation>Request to a SAS to perform the GetCapabilities
operation. This operation allows a client to retrieve service metadata
(capabilities XML) providing metadata for the specific SAS server. In
this XML encoding, no "request" parameter is included, since the
element name specifies the specific operation. </xs:documentation>

   </xs:annotation>

   <xs:complexType>

     <xs:complexContent>

       <xs:extension base="ows:GetCapabilitiesType">

         <xs:attribute name="service" type="ows:ServiceType"
use="required" fixed="SAS"/>

       </xs:extension>

     </xs:complexContent>

   </xs:complexType>

 </xs:element>

 <!-- ========================================================== -->

 <xs:element name="Capabilities">

   <xs:annotation>

     <xs:documentation>XML encoded SAS GetCapabilities operation
response. This document provides clients with service metadata about a
specific service instance, including metadata about the tightly-coupled
data served. If the server does not implement the updateSequence
parameter, the server shall always return the complete Capabilities
document, without the updateSequence parameter. When the server
implements the updateSequence parameter and the GetCapabilities
operation request included the updateSequence parameter with the
current value, the server shall return this element with only the
"version" and "updateSequence" attributes. Otherwise, all optional
elements shall be included or not depending on the actual value of the
Sections parameter in the GetCapabilities operation request.
</xs:documentation>

   </xs:annotation>

   <xs:complexType>
```

```
    <xs:complexContent>
      <xs:extension base="ows:CapabilitiesBaseType">
        <xs:sequence>
          <xs:element ref="sas:Contents" minOccurs="0"/>
          <xs:element ref="wns:NotificationAbilities" minOccurs="0"/>
        </xs:sequence>
      </xs:extension>
    </xs:complexContent>
  </xs:complexType>
 </xs:element>
</xs:schema>
```

sasCommon.xsd

```
<?xml version="1.0" encoding="UTF-8"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema"
xmlns:sas="http://www.opengis.net/sas/0.0"
xmlns:swe="http://www.opengis.net/swe/1.0"
xmlns:sml="http://www.opengis.net/sensorML/1.0"
xmlns:xlink="http://www.w3.org/1999/xlink"
targetNamespace="http://www.opengis.net/sas/0.0"
elementFormDefault="qualified" attributeFormDefault="unqualified"
version="0.0.0" xml:lang="en">
 <xs:annotation>
   <xs:appinfo>sasCommon.xsd</xs:appinfo>
   <xs:documentation>
        This XML Schema encodes the elements and types that are shared
by muliple SAS operations.
     </xs:documentation>
 </xs:annotation>
 <!-- ============================================================
      and imports
      ============================================================ -->
 <xs:import namespace="http://www.w3.org/1999/xlink"
schemaLocation="http://schemas.opengis.net/xlink/1.0.0/xlinks.xsd"/>
 <xs:import namespace="http://www.opengis.net/swe/1.0"
schemaLocation="../../sweCommon/1.0.0/swe.xsd"/>
 <xs:import namespace="http://www.opengis.net/sensorML/1.0"
schemaLocation="../../sensorML/1.0.0/base/sensorML.xsd"/>
 <!-- ============================================================
      and types
      ============================================================ -->
```

```
 <xs:complexType name="RequestBaseType">
   <xs:annotation>
     <xs:documentation>XML encoded SAS operation request base, for all
operations except Get Capabilities. In this XML encoding, no "request"
parameter is included, since the element name specifies the specific
operation. </xs:documentation>
   </xs:annotation>
   <xs:attribute name="service" type="xs:string" use="required"
fixed="SAS">
     <xs:annotation>
       <xs:documentation>Service type identifier. </xs:documentation>
     </xs:annotation>
   </xs:attribute>
   <xs:attribute name="version" type="xs:string" use="required"
fixed="1.0.0">
     <xs:annotation>
       <xs:documentation>Specification version for SAS version and
operation.</xs:documentation>
     </xs:annotation>
   </xs:attribute>
 </xs:complexType>
 <!--================================================================
          core elements for advertisement and capabilities

================================================================-->
 <xs:element name="Acknowledgement">
   <xs:annotation>
     <xs:documentation>Sent from client to SAS or from SAS to sensor in
order to acknowledge the receipt of an alert (identified by SensorID
and Timestamp).</xs:documentation>
   </xs:annotation>
   <xs:complexType>
     <xs:sequence>
       <xs:element ref="sas:SensorID"/>
       <xs:element ref="sas:Timestamp"/>
     </xs:sequence>
   </xs:complexType>
 </xs:element>
 <xs:element name="AcknowledgementChannel" type="sas:XMPPResponseType">
   <xs:annotation>
```

```
     <xs:documentation>MUC provided by SAS for the client to publish
acknowledgements if he requested reliable communication with
SAS.</xs:documentation>
   </xs:annotation>
 </xs:element>
 <xs:element name="Alert">
   <xs:complexType>
     <xs:sequence>
       <xs:element ref="sas:SensorID"/>
       <xs:element ref="sas:Timestamp"/>
       <xs:element ref="sas:AlertData"/>
     </xs:sequence>
   </xs:complexType>
 </xs:element>
 <xs:element name="AlertChannel" type="sas:XMPPResponseType">
   <xs:annotation>
     <xs:documentation>MUC provided by SAS to receive alerts. Not
provided if a result recipient has been provided in the request as
notification endpoint for the subscription.</xs:documentation>
   </xs:annotation>
 </xs:element>
 <xs:element name="AlertData" type="xs:anyType">
   <xs:annotation>
     <xs:documentation>Contains the alert message.</xs:documentation>
   </xs:annotation>
 </xs:element>
 <xs:element name="reportingFrequency">
    <xs:annotation>
     <xs:documentation>The frequency with which new data is usually
generated and sent by a sensor. A value of 0 (default) indicates that
the frequency is unpredictable.</xs:documentation>
   </xs:annotation>
   <xs:complexType>
     <xs:sequence>
       <xs:element ref="swe:Quantity"/>
     </xs:sequence>
   </xs:complexType>
 </xs:element>
 <xs:element name="desiredPublicationExpiration">
```

```
   <xs:annotation>
     <xs:documentation>Point in time the sensor will not provide
publications anymmore. The time might be granted by the SAS or will be
reduced according to the SAS settings.</xs:documentation>
   </xs:annotation>
   <xs:complexType>
     <xs:sequence>
       <xs:element ref="swe:Time"/>
     </xs:sequence>
   </xs:complexType>
 </xs:element>
 <xs:element name="Location">
   <xs:complexType>
     <xs:choice>
       <xs:element ref="swe:Envelope"/>
       <xs:element ref="swe:Position"/>
     </xs:choice>
   </xs:complexType>
 </xs:element>
 <xs:element name="messageStructure">
   <xs:annotation>
     <xs:documentation>Defines the individual fields of alert messages
sent by a sensor and the encoding in which these events are
published.</xs:documentation>
   </xs:annotation>
   <xs:complexType>
     <xs:sequence>
       <xs:element ref="swe:DataBlockDefinition"/>
     </xs:sequence>
   </xs:complexType>
 </xs:element>
 <xs:element name="ReliableCommunication" type="xs:boolean">
   <xs:annotation>
     <xs:documentation>True if the client wishes to perform reliable
communication with the SAS, else false (default).</xs:documentation>
   </xs:annotation>
 </xs:element>
 <xs:element name="sensorDescription">
   <xs:annotation>
```

```
     <xs:documentation>Describes the sensor using SensorML (description
is either inline or referenced).</xs:documentation>
   </xs:annotation>
   <xs:complexType>
     <xs:sequence minOccurs="0">
       <xs:element ref="sml:SensorML"/>
     </xs:sequence>
     <xs:attribute ref="xlink:href">
       <xs:annotation>
         <xs:documentation>references external SensorML
description</xs:documentation>
       </xs:annotation>
     </xs:attribute>
   </xs:complexType>
 </xs:element>
 <xs:element name="SensorID" type="xs:token">
   <xs:annotation>
     <xs:documentation>Unique ID for every registered sensor. Usually
set by SAS.</xs:documentation>
   </xs:annotation>
 </xs:element>
 <xs:element name="SubscriptionOfferingID" type="xs:token">
   <xs:annotation>
     <xs:documentation>ID of a collection of advertised sensors having
the same message format. Useful for grouping of and requesting data
from a set of sensors all having the same sensor type. Note that it is
up to the service to create these offerings, so it would be valid for
the service to group devices according to their type and geographic
location.</xs:documentation>
   </xs:annotation>
 </xs:element>
 <xs:element name="Timestamp" type="xs:dateTime">
   <xs:annotation>
     <xs:documentation>Point in time when the sensor generated the
event.</xs:documentation>
   </xs:annotation>
 </xs:element>
 <xs:element name="XMPPCredentials" type="xs:token">
   <xs:annotation>
     <xs:documentation>The JID of a sensor. Will be created by SAS or
is given by a sensor. SAS can use the JID for identifying clients that
```



```
join a MUC. JID is of the form node@domain e.g.
jondoe@foo.bar.com.</xs:documentation>
   </xs:annotation>
 </xs:element>
 <xs:complexType name="XMPPResponseType">
   <xs:sequence>
     <xs:element ref="sas:XMPPURI"/>
   </xs:sequence>
 </xs:complexType>
 <xs:element name="XMPPURI" type="xs:anyURI"/>
</xs:schema>
```

sasContents.xsd

```
<?xml version="1.0" encoding="UTF-8"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema"
xmlns:sas="http://www.opengis.net/sas/0.0"
xmlns:ows="http://www.opengis.net/ows"
targetNamespace="http://www.opengis.net/sas/0.0"
elementFormDefault="qualified" attributeFormDefault="unqualified"
version="0.0.0" xml:lang="en">
 <xs:annotation>
   <xs:appinfo>sasContents.xsd</xs:appinfo>
   <xs:documentation>
     <description>This XML Schema encodes the Contents section of the
SAS GetCapabilities operation response.</description>
   </xs:documentation>
 </xs:annotation>
 <!-- ================================================================
      and imports
      ================================================================ -->
 <xs:import namespace="http://www.opengis.net/ows"
schemaLocation="http://schemas.opengis.net/ows/1.0.0/owsGetCapabilities
.xsd"/>
 <xs:include schemaLocation="./sasCommon.xsd"/>
 <!-- ================================================================
      and types
      ================================================================ -->
 <xs:element name="Contents">
   <xs:annotation>
     <xs:documentation/>
```

```
   </xs:annotation>
   <xs:complexType>
     <xs:sequence>
       <xs:element name="AcceptAdvertisements" type="xs:boolean">
         <xs:annotation>
           <xs:documentation>Defines if advertisements can be sent to
SAS, i.e. if SAS operates in transactional mode.</xs:documentation>
         </xs:annotation>
       </xs:element>
       <xs:element name="ReliableCommunicationSupported"
type="xs:boolean">
         <xs:annotation>
           <xs:documentation>SAS indicates support of acknowledgements
sent by alert receivers to confirm reception of
alerts.</xs:documentation>
         </xs:annotation>
       </xs:element>
       <xs:element name="SubscriptionOfferingList" minOccurs="0">
         <xs:annotation>
           <xs:documentation>root element for all
SubscriptionOfferings</xs:documentation>
         </xs:annotation>
         <xs:complexType>
           <xs:sequence>
             <xs:element name="SubscriptionOffering"
type="sas:SubscriptionOfferingType" maxOccurs="unbounded"/>
           </xs:sequence>
         </xs:complexType>
       </xs:element>
     </xs:sequence>
   </xs:complexType>
 </xs:element>
 <xs:complexType name="SubscriptionOfferingType">
   <xs:sequence>
     <xs:element ref="sas:SubscriptionOfferingID"/>
     <xs:element ref="sas:messageStructure"/>
     <xs:element name="member" maxOccurs="unbounded">
       <xs:complexType>
         <xs:sequence>
```

```
            <xs:element ref="sas:SensorID"/>
            <xs:element ref="sas:reportingFrequency" minOccurs="0"/>
            <xs:element ref="sas:Location" minOccurs="0"/>
          </xs:sequence>
        </xs:complexType>
      </xs:element>
    </xs:sequence>
 </xs:complexType>
</xs:schema>
```

sasMessageSchema.xsd

```
<?xml version="1.0" encoding="UTF-8"?>
<xs:schema xmlns:sas="http://www.opengis.net/sas/0.0"
xmlns:xs="http://www.w3.org/2001/XMLSchema"
targetNamespace="http://www.opengis.net/sas/0.0"
elementFormDefault="qualified" attributeFormDefault="unqualified"
version="0.0.0">
 <xs:include schemaLocation="./sasCommon.xsd"/>
 <xs:element name="SASMessage">
   <xs:annotation>
     <xs:documentation>A SASMessage defines the structure of a message
which is used when alerting via a communication protocol other than
XMPP (i.e., if a NotificationTarget was provided in the Subscribe
request). The SubscriptionID is used so that a client that uses one
communication endpoint for all alerts coming from a SAS can determine
the subscription the alert belongs to.</xs:documentation>
   </xs:annotation>
   <xs:complexType>
     <xs:sequence>
       <xs:element ref="sas:Alert"/>
     </xs:sequence>
     <xs:attribute name="SubscriptionID" type="xs:token"
use="required"/>
   </xs:complexType>
 </xs:element>
</xs:schema>
```

sasSubscribe.xsd

```
<?xml version="1.0" encoding="UTF-8"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema"
xmlns:sas="http://www.opengis.net/sas/0.0"
```

```
xmlns:wns="http://www.opengis.net/wns/0.0"
xmlns:swe="http://www.opengis.net/swe/1.0"
targetNamespace="http://www.opengis.net/sas/0.0"
elementFormDefault="qualified" attributeFormDefault="unqualified"
version="0.0.0" xml:lang="en">
 <!-- ================================================================

      and imports

      ================================================================ -->
 <xs:import namespace="http://www.opengis.net/wns/0.0"
schemaLocation="../../wns/0.0.0/wnsShared.xsd"/>
 <xs:import namespace="http://www.opengis.net/swe/1.0"
schemaLocation="../../sweCommon/1.0.0/swe.xsd"/>
 <xs:include schemaLocation="./sasCommon.xsd"/>
 <!-- ================================================================


      ================================================================ -->
 <xs:element name="Subscribe">
   <xs:annotation>
     <xs:documentation>Request to a SAS to subscribe to
alerts.</xs:documentation>
   </xs:annotation>
   <xs:complexType>
     <xs:complexContent>
       <xs:extension base="sas:RequestBaseType">
         <xs:sequence>
           <xs:choice>
             <xs:annotation>
               <xs:documentation>Either the client chose to subscribe
for all events of a single sensor (in which case he should be referred
directly to the MUC where the sensor publishes its data) or he can
specify in which events he is interested in.</xs:documentation>
             </xs:annotation>
             <xs:element ref="sas:SensorID" minOccurs="0"/>
             <xs:element name="EventFilter" minOccurs="0">
               <xs:complexType>
                 <xs:sequence>
                   <xs:annotation>
                     <xs:documentation>There is an implicit AND between
the conditions defined by the following elements.</xs:documentation>
                   </xs:annotation>
```

```
                    <xs:element ref="sas:SubscriptionOfferingID"
minOccurs="0"/>
                    <xs:element ref="sas:Location" minOccurs="0"/>
                    <xs:element name="ValueFilterList" minOccurs="0">
                      <xs:complexType>
                        <xs:sequence>
                          <xs:element name="member"
maxOccurs="unbounded">
                            <xs:complexType>
                              <xs:sequence>
                                <xs:element name="ValueFilter">
                                  <xs:annotation>
                                    <xs:documentation>Used to indicate
in which observed properties (indicated by the definition attribute)
the user is really interested. Also allows the definition of simple
value filters and provision of uom.</xs:documentation>
                                  </xs:annotation>
                                  <xs:complexType>
                                    <xs:sequence>
                                      <xs:element name="filterCriteria"
minOccurs="0">
                                        <xs:complexType>
                                          <xs:choice>
                                            <xs:element
name="isLessThan" type="xs:double"/>
                                            <xs:element
name="isGreaterThanOrEqualTo" type="xs:double"/>
                                            <xs:element
name="isGreaterThan" type="xs:double"/>
                                            <xs:element
name="isLessThanOrEqualTo" type="xs:double"/>
                                            <xs:element name="isEqual">
                                              <xs:simpleType>
                                                <xs:union
memberTypes="xs:string xs:double xs:boolean"/>
                                              </xs:simpleType>
                                            </xs:element>
                                            <xs:element
name="isNotEqualTo">
                                              <xs:simpleType>
                                                <xs:union
memberTypes="xs:string xs:double"/>
```

```
                                        </xs:simpleType>
                                      </xs:element>
                                      <xs:element
name="isBetween">
                                        <xs:complexType>
                                          <xs:sequence>
                                            <xs:element
name="lowerBoundary" type="xs:double"/>
                                            <xs:element
name="upperBoundary" type="xs:double"/>
                                          </xs:sequence>
                                        </xs:complexType>
                                      </xs:element>
                                    </xs:choice>
                                  </xs:complexType>
                                </xs:element>
                                <xs:element name="uom"
type="swe:UomPropertyType" minOccurs="0">
                                  <xs:annotation>
                                    <xs:documentation>Used to
indicate which uom the value provided in the filterCriteria has, so
that the service can transform it to the uom of the actual event
property (or vice versa). This only makes sense if the definition
provided in the ResultFilter references a swe:Quantity property
(swe:Count, swe:Boolean, swe:Category and swe:Text do not have a uom -
swe:Time has, but SAS does not perform temporal filtering right
now).</xs:documentation>
                                  </xs:annotation>
                                </xs:element>
                              </xs:sequence>
                              <xs:attribute name="definition"
type="xs:anyURI" use="required">
                                <xs:annotation>
                                  <xs:documentation>Points to
semantics information defining the precise nature of the
component</xs:documentation>
                                </xs:annotation>
                              </xs:attribute>
                            </xs:complexType>
                          </xs:element>
                        </xs:sequence>
                      </xs:complexType>
```

```
                        </xs:element>
                      </xs:sequence>
                    </xs:complexType>
                  </xs:element>
                </xs:sequence>
              </xs:complexType>
            </xs:element>
          </xs:choice>
          <xs:element name="ResultRecipient" minOccurs="0">
            <xs:annotation>
              <xs:documentation>Allows to specify where to the SAS
should send the data . If this element is used, the SAS will not open a
MUC and provide the data to the client, but will forward any alert to
the specified remote MUC or uses the WNS (in which case SASMessages
will be sent to the specified target(s)). This allows to use the SAS in
"last-mile-mode".</xs:documentation>
            </xs:annotation>
            <xs:complexType>
              <xs:choice>
                <xs:element name="XMPPURI" type="xs:anyURI">
                  <xs:annotation>
                    <xs:documentation>The excplicit URI where the xmpp
messages should be send to. Also implies that simple Alerts, not
SASMessages are published on this MUC.</xs:documentation>
                  </xs:annotation>
                </xs:element>
                <xs:element ref="wns:NotificationTarget"/>
              </xs:choice>
            </xs:complexType>
          </xs:element>
          <xs:element ref="sas:ReliableCommunication" minOccurs="0"/>
        </xs:sequence>
      </xs:extension>
    </xs:complexContent>
  </xs:complexType>
 </xs:element>
 <xs:element name="SubscribeResponse">
   <xs:complexType>
     <xs:sequence>
       <xs:element ref="sas:AcknowledgementChannel" minOccurs="0"/>
```

```
      <xs:element ref="sas:AlertChannel" minOccurs="0"/>
    </xs:sequence>
    <xs:attribute name="SubscriptionID" type="xs:token"
use="required">
      <xs:annotation>
        <xs:documentation>ID of the new
subscription.</xs:documentation>
      </xs:annotation>
    </xs:attribute>
    <xs:attribute name="expires" type="xs:dateTime" use="required">
      <xs:annotation>
        <xs:documentation>Point in time when the subscription will
expire.</xs:documentation>
      </xs:annotation>
    </xs:attribute>
  </xs:complexType>
 </xs:element>
 <xs:element name="CancelSubscription">
  <xs:complexType>
    <xs:complexContent>
      <xs:extension base="sas:RequestBaseType">
        <xs:sequence>
          <xs:element name="SubscriptionID" type="xs:token"/>
        </xs:sequence>
      </xs:extension>
    </xs:complexContent>
  </xs:complexType>
 </xs:element>
 <xs:element name="CancelSubscriptionResponse">
  <xs:complexType>
    <xs:sequence>
      <xs:element name="SubscriptionID" type="xs:token"/>
    </xs:sequence>
  </xs:complexType>
 </xs:element>
 <xs:element name="RenewSubscription">
  <xs:complexType>
    <xs:complexContent>
```

```
      <xs:extension base="sas:RequestBaseType">
        <xs:sequence>
          <xs:element name="SubscriptionID" type="xs:token"/>
        </xs:sequence>
      </xs:extension>
    </xs:complexContent>
  </xs:complexType>
 </xs:element>
 <xs:element name="RenewSubscriptionResponse">
  <xs:complexType>
    <xs:sequence>
      <xs:element name="SubscriptionID" type="xs:token"/>
    </xs:sequence>
    <xs:attribute name="expires" type="xs:dateTime" use="required"/>
  </xs:complexType>
 </xs:element>
</xs:schema>
```

sasDescribeAlert.xsd

```
<?xml version="1.0" encoding="UTF-8"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema"
xmlns:sas="http://www.opengis.net/sas/0.0"
xmlns:swe="http://www.opengis.net/swe/1.0"
targetNamespace="http://www.opengis.net/sas/0.0"
elementFormDefault="qualified" attributeFormDefault="unqualified"
version="0.0.0" xml:lang="en">
 <!-- ================================================================
      and imports
      ================================================================ -->
 <xs:include schemaLocation="./sasCommon.xsd"/>
 <!-- ================================================================


      ================================================================ -->
 <xs:element name="DescribeAlert">
   <xs:annotation>
     <xs:documentation>Used to request an observation template for the
alert referenced by the given ID</xs:documentation>
   </xs:annotation>
   <xs:complexType>
```

```
      <xs:complexContent>
        <xs:extension base="sas:RequestBaseType">
          <xs:sequence>
            <xs:element ref="sas:SensorID"/>
          </xs:sequence>
        </xs:extension>
      </xs:complexContent>
    </xs:complexType>
  </xs:element>
  <xs:element name="DescribeAlertResponse">
    <xs:complexType>
      <xs:sequence>
        <xs:element ref="sas:messageStructure"/>
      </xs:sequence>
    </xs:complexType>
  </xs:element>
</xs:schema>
```

sasDescribeSensor.xsd

```
<?xml version="1.0" encoding="UTF-8"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema"
xmlns:sas="http://www.opengis.net/sas/0.0"
xmlns:sml="http://www.opengis.net/sensorML/1.0"
targetNamespace="http://www.opengis.net/sas/0.0"
elementFormDefault="qualified" attributeFormDefault="unqualified"
version="0.0.0" xml:lang="en">
  <!-- ================================================================
       and imports
       ================================================================ -->
  <xs:include schemaLocation="./sasCommon.xsd"/>
  <!-- ================================================================


       ================================================================ -->
  <xs:element name="DescribeSensor">
    <xs:annotation>
      <xs:documentation>Used to request detailed sensor
metadata.</xs:documentation>
    </xs:annotation>
    <xs:complexType>
```

```
      <xs:complexContent>
        <xs:extension base="sas:RequestBaseType">
          <xs:sequence>
            <xs:element ref="sas:SensorID"/>
          </xs:sequence>
        </xs:extension>
      </xs:complexContent>
    </xs:complexType>
  </xs:element>
  <!--The response is a SensorML document or a reference to it.-->
  <xs:element name="DescribeSensorResponse">
    <xs:annotation>
      <xs:documentation>Response to DescribeSensor</xs:documentation>
    </xs:annotation>
    <xs:complexType>
      <xs:sequence>
        <xs:element ref="sas:sensorDescription"/>
      </xs:sequence>
    </xs:complexType>
  </xs:element>
</xs:schema>
```

Annex C
(informative)

Example XML documents

A.3 C.1 Capabilities example

```
<?xml version="1.0" encoding="UTF-8"?>

<Capabilities xmlns="http://www.opengis.net/sas/0.0"
xmlns:xlink="http://www.w3.org/1999/xlink"
xmlns:ows="http://www.opengis.net/ows"
xmlns:wns="http://www.opengis.net/wns/0.0"
xmlns:swe="http://www.opengis.net/swe/1.0"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" version="1.0.0">

 <ows:ServiceIdentification>

   <ows:Title>IFGI SAS</ows:Title>

   <ows:Abstract>Sensor alert service for environment
sensors</ows:Abstract>

   <ows:Keywords>

     <ows:Keyword>SAS</ows:Keyword>

     <ows:Keyword>environment</ows:Keyword>

     <ows:Keyword>monitoring</ows:Keyword>

   </ows:Keywords>

   <ows:ServiceType
codeSpace="http://opengis.net">OGC:SAS</ows:ServiceType>

   <ows:ServiceTypeVersion>1.0.0</ows:ServiceTypeVersion>

   <ows:Fees>NONE</ows:Fees>

   <ows:AccessConstraints>NONE</ows:AccessConstraints>

 </ows:ServiceIdentification>

 <ows:ServiceProvider>

   <ows:ProviderName>Institute for Geoinformatics, University of
Muenster</ows:ProviderName>

   <ows:ProviderSite xlink:href="http://ifgi.uni-muenster.de"/>

   <ows:ServiceContact>

     <ows:IndividualName>Johannes Echterhoff</ows:IndividualName>

     <ows:PositionName>research associate</ows:PositionName>

     <ows:ContactInfo>

       <ows:Phone>

         <ows:Voice>+49 251 83 39761</ows:Voice>

         <ows:Facsimile>+49 251 83 39763</ows:Facsimile>
```

```
       </ows:Phone>
       <ows:Address>
         <ows:DeliveryPoint>Institute for Geoinformatics, University of
Muenster</ows:DeliveryPoint>
         <ows:City>Muenster</ows:City>
         <ows:AdministrativeArea>NRW</ows:AdministrativeArea>
         <ows:PostalCode>48149</ows:PostalCode>
         <ows:Country>Germany</ows:Country>
         <ows:ElectronicMailAddress>echterhoff@uni-
muenster.de</ows:ElectronicMailAddress>
       </ows:Address>
     </ows:ContactInfo>
   </ows:ServiceContact>
 </ows:ServiceProvider>
 <ows:OperationsMetadata>
   <ows:Operation name="GetCapabilities">
     <ows:DCP>
       <ows:HTTP>
         <ows:Get xlink:href="http://mars.uni-
muenster.de/52nSAS/SAS?"/>
         <ows:Post xlink:href="http://mars.uni-
muenster.de/52nSAS/SAS"/>
       </ows:HTTP>
     </ows:DCP>
   </ows:Operation>
   <ows:Operation name="GetWSDL">
     <ows:DCP>
       <ows:HTTP>
         <ows:Get xlink:href="http://mars.uni-
muenster.de/52nSAS/SAS?"/>
       </ows:HTTP>
     </ows:DCP>
   </ows:Operation>
   <ows:Operation name="Advertise">
     <ows:DCP>
       <ows:HTTP>
         <ows:Post xlink:href="http://mars.uni-
muenster.de/52nSAS/SAS"/>
       </ows:HTTP>
```



```
      </ows:DCP>
    </ows:Operation>
    <ows:Operation name="RenewAdvertisement">
      <ows:DCP>
        <ows:HTTP>
          <ows:Post xlink:href="http://mars.uni-
muenster.de/52nSAS/SAS"/>
        </ows:HTTP>
      </ows:DCP>
    </ows:Operation>
    <ows:Operation name="CancelAdvertisement">
      <ows:DCP>
        <ows:HTTP>
          <ows:Post xlink:href="http://mars.uni-
muenster.de/52nSAS/SAS"/>
        </ows:HTTP>
      </ows:DCP>
    </ows:Operation>
    <ows:Operation name="Subscribe">
      <ows:DCP>
        <ows:HTTP>
          <ows:Post xlink:href="http://mars.uni-
muenster.de/52nSAS/SAS"/>
        </ows:HTTP>
      </ows:DCP>
    </ows:Operation>
    <ows:Operation name="RenewSubscription">
      <ows:DCP>
        <ows:HTTP>
          <ows:Post xlink:href="http://mars.uni-
muenster.de/52nSAS/SAS"/>
        </ows:HTTP>
      </ows:DCP>
    </ows:Operation>
    <ows:Operation name="CancelSubscription">
      <ows:DCP>
        <ows:HTTP>
          <ows:Post xlink:href="http://mars.uni-
muenster.de/52nSAS/SAS"/>
```

```
        </ows:HTTP>
      </ows:DCP>
    </ows:Operation>
    <ows:Operation name="DescribeAlert">
      <ows:DCP>
        <ows:HTTP>
          <ows:Post xlink:href="http://mars.uni-
muenster.de/52nSAS/SAS"/>
        </ows:HTTP>
      </ows:DCP>
    </ows:Operation>
    <ows:Operation name="DescribeSensor">
      <ows:DCP>
        <ows:HTTP>
          <ows:Post xlink:href="http://mars.uni-
muenster.de/52nSAS/SAS"/>
        </ows:HTTP>
      </ows:DCP>
    </ows:Operation>
    <ows:Parameter name="service">
      <ows:Value>SAS</ows:Value>
    </ows:Parameter>
    <ows:Parameter name="version">
      <ows:Value>1.0.0</ows:Value>
    </ows:Parameter>
    <ows:Constraint name="PostEncoding">
      <ows:Value>XML</ows:Value>
      <ows:Value>SOAP</ows:Value>
    </ows:Constraint>
  </ows:OperationsMetadata>
  <Contents>
    <AcceptAdvertisements>true</AcceptAdvertisements>

<ReliableCommunicationSupported>true</ReliableCommunicationSupported>
    <SubscriptionOfferingList>
      <SubscriptionOffering>
        <SubscriptionOfferingID>sub1</SubscriptionOfferingID>
        <messageStructure>
```

```
          <swe:DataBlockDefinition>
            <swe:components name="sensor data structure">
              <swe:DataRecord>
                <swe:field name="component1">
                  <swe:Quantity definition="urn:x-
ogc:def:phenomenon:OGC:Temperature">
                    <swe:uom code="Cel"/>
                  </swe:Quantity>
                </swe:field>
                <swe:field name="component2">
                  <swe:Quantity definition="urn:x-
ogc:def:phenomenon:OGC:RelativeHumidity">
                    <swe:uom code="%"/>
                  </swe:Quantity>
                </swe:field>
                <swe:field name="component3">
                  <swe:Position definition="urn:x-
ogc:def:phenomenon:OGC:sampleLocation" referenceFrame="urn:x-
ogc:def:crs:EPSG:6.11:4326">
                    <swe:location>
                      <swe:Vector>
                        <swe:coordinate name="latitude">
                          <swe:Quantity>
                            <swe:uom code="deg"/>
                          </swe:Quantity>
                        </swe:coordinate>
                        <swe:coordinate name="longitude">
                          <swe:Quantity>
                            <swe:uom code="deg"/>
                          </swe:Quantity>
                        </swe:coordinate>
                      </swe:Vector>
                    </swe:location>
                  </swe:Position>
                </swe:field>
              </swe:DataRecord>
            </swe:components>
            <swe:encoding>
```

```
            <swe:TextBlock decimalSeparator="." blockSeparator="@@"
tokenSeparator=" "/>
          </swe:encoding>
        </swe:DataBlockDefinition>
      </messageStructure>
      <member>
        <SensorID>urn:x-ogc:object:sensor:IFGI:Temp:1</SensorID>
        <reportingFrequency>
          <swe:Quantity>
            <swe:uom code="Hz"/>
            <swe:value>0.1</swe:value>
          </swe:Quantity>
        </reportingFrequency>
        <Location>
          <swe:Envelope referenceFrame="urn:x-
ogc:def:crs:EPSG:6.11:4326">
            <swe:lowerCorner>
              <swe:Vector>
                <swe:coordinate name="latitude">
                  <swe:Quantity>
                    <swe:uom code="deg"/>
                    <swe:value>51.916945</swe:value>
                  </swe:Quantity>
                </swe:coordinate>
                <swe:coordinate name="longitude">
                  <swe:Quantity>
                    <swe:uom code="deg"/>
                    <swe:value>7.680686</swe:value>
                  </swe:Quantity>
                </swe:coordinate>
              </swe:Vector>
            </swe:lowerCorner>
            <swe:upperCorner>
              <swe:Vector>
                <swe:coordinate name="latitude">
                  <swe:Quantity>
                    <swe:uom code="deg"/>
                    <swe:value>51.974551</swe:value>
```

```
                    </swe:Quantity>
                  </swe:coordinate>
                  <swe:coordinate name="longitude">
                    <swe:Quantity>
                      <swe:uom code="deg"/>
                      <swe:value>7.716518</swe:value>
                    </swe:Quantity>
                  </swe:coordinate>
                </swe:Vector>
              </swe:upperCorner>
            </swe:Envelope>
          </Location>
        </member>
      </SubscriptionOffering>
    </SubscriptionOfferingList>
  </Contents>
  <wns:NotificationAbilities>
    <wns:SupportedCommunicationProtocols>
      <wns:XMPP>true</wns:XMPP>
      <wns:SMS>false</wns:SMS>
      <wns:Phone>false</wns:Phone>
      <wns:Fax>false</wns:Fax>
      <wns:Email>true</wns:Email>
      <wns:WSAddressing>false</wns:WSAddressing>
      <wns:WNS>true</wns:WNS>
    </wns:SupportedCommunicationProtocols>
    <wns:SupportedCommunicationFormats>
      <wns:NotificationFormat>basic</wns:NotificationFormat>
    </wns:SupportedCommunicationFormats>
  </wns:NotificationAbilities>
</Capabilities>
```

Annex D
(informative)

WSDL document

A WSDL document describes the interface of a Web service. A WSDL binding describes how the service is bound to a messaging protocol, particularly the SOAP messaging protocol. A WSDL SOAP binding can be either a Remote Procedure Call (RPC) style binding or a document style binding. A SOAP binding can also have an encoded use or a literal use. The following WSDL (version 1.1) document uses document/literal style to specify the service interface of the SAS.

A.4 D.1 WSDL example

```
<?xml version="1.0" encoding="UTF-8" standalone="no"?>
<definitions xmlns:n52sas="http://www.52north.org/sas/1.0"
xmlns:sas="http://www.opengis.net/sas/0.0"
xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/"
xmlns="http://schemas.xmlsoap.org/wsdl/"
xmlns:xsd="http://www.w3.org/2001/XMLSchema" name="sasDocumentLiteral"
targetNamespace="http://www.52north.org/sas/1.0"
xmlns:mime="http://schemas.xmlsoap.org/wsdl/mime/"
xmlns:http="http://schemas.xmlsoap.org/wsdl/http/">
 <types>
   <xsd:schema targetNamespace="http://www.52north.org/sas/1.0">
     <xsd:import namespace="http://www.opengis.net/sas/0.0"
schemaLocation="http://schemas.opengeospatial.net/sas/0.0.0/sasAll.xsd"
/>
   </xsd:schema>
 </types>
 <message name="GetCapabilitiesInput">
   <part name="parameter" element="sas:GetCapabilities"/>
 </message>
 <message name="GetCapabilitiesKVPInput">
   <part name="request" element="xsd:string"/>
   <part name="service" element="xsd:string"/>
 </message>
 <message name="AdvertiseInput">
   <part name="parameter" element="sas:Advertise"/>
 </message>
 <message name="RenewAdvertisementInput">
   <part name="parameter" element="sas:RenewAdvertisement"/>
```

```
</message>
<message name="CancelAdvertisementInput">
  <part name="parameter" element="sas:CancelAdvertisement"/>
</message>
<message name="SubscribeInput">
  <part name="parameter" element="sas:Subscribe"/>
</message>
<message name="RenewSubscriptionInput">
  <part name="parameter" element="sas:RenewSubscription"/>
</message>
<message name="CancelSubscriptionInput">
  <part name="parameter" element="sas:CancelSubscription"/>
</message>
<message name="DescribeAlertInput">
  <part name="parameter" element="sas:DescribeAlert"/>
</message>
<message name="DescribeSensorInput">
  <part name="parameter" element="sas:DescribeSensor"/>
</message>
<message name="GetCapabilitiesOutput">
  <part name="parameter" element="sas:Capabilities"/>
</message>
<message name="AdvertiseOutput">
  <part name="parameter" element="sas:AdvertiseResponse"/>
</message>
<message name="RenewAdvertisementOutput">
  <part name="parameter" element="sas:RenewAdvertisementResponse"/>
</message>
<message name="CancelAdvertisementOutput">
  <part name="parameter" element="sas:CancelAdvertisementResponse"/>
</message>
<message name="SubscribeOutput">
  <part name="parameter" element="sas:SubscribeResponse"/>
</message>
<message name="RenewSubscriptionOutput">
  <part name="parameter" element="sas:RenewSubscriptionResponse"/>
</message>
```

```
<message name="CancelSubscriptionOutput">
  <part name="parameter" element="sas:CancelSubscriptionResponse"/>
</message>
<message name="DescribeAlertOutput">
  <part name="parameter" element="sas:DescribeAlertResponse"/>
</message>
<message name="DescribeSensorOutput">
  <part name="parameter" element="sas:DescribeSensorResponse"/>
</message>
<portType name="kvpPortType">
  <operation name="GetCapabilities">
    <input message="n52sas:GetCapabilitiesKVPInput"/>
    <output message="n52sas:GetCapabilitiesOutput"/>
  </operation>
</portType>
<portType name="postPortType">
  <operation name="GetCapabilities">
    <input message="n52sas:GetCapabilitiesInput"/>
    <output message="n52sas:GetCapabilitiesOutput"/>
  </operation>
  <operation name="Advertise">
    <input message="n52sas:AdvertiseInput"/>
    <output message="n52sas:AdvertiseOutput"/>
  </operation>
  <operation name="RenewAdvertisement">
    <input message="n52sas:RenewAdvertisementInput"/>
    <output message="n52sas:RenewAdvertisementOutput"/>
  </operation>
  <operation name="CancelAdvertisement">
    <input message="n52sas:CancelAdvertisementInput"/>
    <output message="n52sas:CancelAdvertisementOutput"/>
  </operation>
  <operation name="Subscribe">
    <input message="n52sas:SubscribeInput"/>
    <output message="n52sas:SubscribeOutput"/>
  </operation>
  <operation name="RenewSubscription">
```

```
      <input message="n52sas:RenewSubscriptionInput"/>
      <output message="n52sas:RenewSubscriptionOutput"/>
    </operation>
    <operation name="CancelSubscription">
      <input message="n52sas:CancelSubscriptionInput"/>
      <output message="n52sas:CancelSubscriptionOutput"/>
    </operation>
    <operation name="DescribeAlert">
      <input message="n52sas:DescribeAlertInput"/>
      <output message="n52sas:DescribeAlertOutput"/>
    </operation>
    <operation name="DescribeSensor">
      <input message="n52sas:DescribeSensorInput"/>
      <output message="n52sas:DescribeSensorOutput"/>
    </operation>
  </portType>
  <binding name="httpPOSTBinding" type="n52sas:postPortType">
    <http:binding verb="POST"/>
    <operation name="GetCapabilities">
      <http:operation location="SAS"/>
      <input>
        <mime:mimeXml part="Body"/>
      </input>
      <output>
        <mime:mimeXml part="Body"/>
      </output>
    </operation>
    <operation name="Advertise">
      <input>
        <mime:mimeXml part="Body"/>
      </input>
      <output>
        <mime:mimeXml part="Body"/>
      </output>
    </operation>
    <operation name="RenewAdvertisement">
      <input>
```

```
      <mime:mimeXml part="Body"/>
    </input>
    <output>
      <mime:mimeXml part="Body"/>
    </output>
  </operation>
  <operation name="CancelAdvertisement">
    <input>
      <mime:mimeXml part="Body"/>
    </input>
    <output>
      <mime:mimeXml part="Body"/>
    </output>
  </operation>
  <operation name="Subscribe">
    <input>
      <mime:mimeXml part="Body"/>
    </input>
    <output>
      <mime:mimeXml part="Body"/>
    </output>
  </operation>
  <operation name="RenewSubscription">
    <input>
      <mime:mimeXml part="Body"/>
    </input>
    <output>
      <mime:mimeXml part="Body"/>
    </output>
  </operation>
  <operation name="CancelSubscription">
    <input>
      <mime:mimeXml part="Body"/>
    </input>
    <output>
      <mime:mimeXml part="Body"/>
    </output>
```

```
    </operation>
    <operation name="DescribeSensor">
      <input>
        <mime:mimeXml part="Body"/>
      </input>
      <output>
        <mime:mimeXml part="Body"/>
      </output>
    </operation>
    <operation name="DescribeAlert">
      <input>
        <mime:mimeXml part="Body"/>
      </input>
      <output>
        <mime:mimeXml part="Body"/>
      </output>
    </operation>
  </binding>
  <binding name="httpKVPBinding" type="n52sas:kvpPortType">
    <http:binding verb="GET"/>
    <operation name="GetCapabilities">
      <http:operation location="SAS"/>
      <input>
        <http:urlEncoded/>
      </input>
      <output>
        <mime:mimeXml part="Body"/>
      </output>
    </operation>
  </binding>
  <binding name="soapBinding" type="n52sas:postPortType">
    <soap:binding style="document"
transport="http://schemas.xmlsoap.org/soap/http"/>
    <operation name="GetCapabilities">
      <soap:operation
soapAction="http://www.52north.org/sas/1.0/GetCapabilities"/>
      <input>
        <soap:body use="literal"/>
```

```
      </input>
      <output>
        <soap:body use="literal"/>
      </output>
    </operation>
    <operation name="Advertise">
      <soap:operation
soapAction="http://www.52north.org/sas/1.0/Advertise"/>
      <input>
        <soap:body use="literal"/>
      </input>
      <output>
        <soap:body use="literal"/>
      </output>
    </operation>
    <operation name="RenewAdvertisement">
      <soap:operation
soapAction="http://www.52north.org/sas/1.0/RenewAdvertisement"/>
      <input>
        <soap:body use="literal"/>
      </input>
      <output>
        <soap:body use="literal"/>
      </output>
    </operation>
    <operation name="CancelAdvertisement">
      <soap:operation
soapAction="http://www.52north.org/sas/1.0/CancelAdvertisement"/>
      <input>
        <soap:body use="literal"/>
      </input>
      <output>
        <soap:body use="literal"/>
      </output>
    </operation>
    <operation name="Subscribe">
      <soap:operation
soapAction="http://www.52north.org/sas/1.0/Subscribe"/>
      <input>
```

```
        <soap:body use="literal"/>
      </input>
      <output>
        <soap:body use="literal"/>
      </output>
    </operation>
    <operation name="RenewSubscription">
      <soap:operation
soapAction="http://www.52north.org/sas/1.0/RenewSubscription"/>
      <input>
        <soap:body use="literal"/>
      </input>
      <output>
        <soap:body use="literal"/>
      </output>
    </operation>
    <operation name="CancelSubscription">
      <soap:operation
soapAction="http://www.52north.org/sas/1.0/CancelSubscription"/>
      <input>
        <soap:body use="literal"/>
      </input>
      <output>
        <soap:body use="literal"/>
      </output>
    </operation>
    <operation name="DescribeSensor">
      <soap:operation
soapAction="http://www.52north.org/sas/1.0/DescribeSensor"/>
      <input>
        <soap:body use="literal"/>
      </input>
      <output>
        <soap:body use="literal"/>
      </output>
    </operation>
    <operation name="DescribeAlert">
      <soap:operation
soapAction="http://www.52north.org/sas/1.0/DescribeAlert"/>
```

```
      <input>
        <soap:body use="literal"/>
      </input>
      <output>
        <soap:body use="literal"/>
      </output>
    </operation>
  </binding>
  <service name="sasService">
    <port name="sasHttpPost" binding="n52sas:httpPOSTBinding">
      <http:address location="http://www.52north.org/52nSAS/"/>
    </port>
    <port name="sasHttpKvp" binding="n52sas:httpKVPBinding">
      <http:address location="http://www.52north.org/52nSAS/"/>
    </port>
    <port name="sasSoap" binding="n52sas:soapBinding">
      <soap:address location="http://www.52north.org/52nSAS/SAS"/>
    </port>
  </service>
</definitions>
```

Bibliography

[1] BOTTS, M. & A. ROBIN (2007): *OpenGIS® Sensor Model Language (SensorML) Implementation Specification*. Category: OpenGIS® Implementation Specification. Version: 1.0.0. Reference number: OGC 07-000

[2] DIERKS, T. & C. ALLEN (1999): *The TLS Protocol*. online at: http://www.ietf.org/rfc/rfc2246.txt (accessed on: May, 11th, 2007)

[3] ISO (2002): *Geographic information - Temporal schema*. Category: Version: Reference number: Ref. No. ISO 19108:2002

[4] ISO (2007): *Geographic information - Geography Markup Language*. Category: Version: Reference number: Ref. No. ISO 19136:2007

[5] MYERS, J. (1997): *Simple Authentication and Security Layer (SASL)*. online at: http://www.ietf.org/rfc/rfc2222.txt (accessed on: May, 11th, 2007)

[6] OASIS (2006): *Web Services Notification (WSN), WS-BrokeredNotification and WS-Topics specifications.* online at: http://www.oasis-open.org/committees/tc_home.php?wg_abbrev=wsn (accessed on:

[7] SAINT-ANDRE, P. (2004): *Extensible Messaging and Presence Protocol (XMPP): Core*. online at: http://www.ietf.org/rfc/rfc3920.txt (accessed on: April 26, 2006)

[8] SAINT-ANDRE, P. (2006): *XEP-0045: Multi-User Chat*. online at: http://www.xmpp.org/extensions/xep-0045.html (accessed on: May, 11th, 2007)

[9] SIMONIS, I. & J. ECHTERHOFF (2006): *Web Notification Service Implementation Specification*. Category: OpenGIS® Implementation Specification. Version: 1.0.0. Reference number: OGC 06-069

[10] VRETANOS, P. A. (2005): *Filter Encoding Implementation Specification*. Category: OpenGIS Implementation Specification. Version: 1.1.0. (Adopted Specification) Reference number: OGC 04-095

[11] W3C (2001): *Web Services Description Language (WSDL)*. online at: http://www.w3.org/TR/wsdl.html (accessed on: May, 15th, 2007)

[12] WHITESIDE, A. (2005): *OpenGIS® Web Services Common Specification*. Category: OpenGIS® Implementation Specification. Version: 1.0.0. Reference number: OGC 05-008

[13] XMPP (2007): *Summary of XMPP*. online at: http://www.xmpp.org/about/summary.shtml (accessed on: May, 7th, 2007)

