Open Geospatial Consortium Inc.

Date: 2006-05-13

Reference number of this document: **OGC 06-028r3**

Version: 0.9

Category: OpenGIS® Best Practices

Editor: Ingo Simonis

OGC® Sensor Alert Service Candidate Implementation Specification

Warning

This is an OGC Best Practices document. The document is not an OGC Standard and may not be referred to as an OGC Standard. It is subject to change without notice. However, this document is an official position of the OGC membership on this particular technology topic.

Document type:	OGC Best Practices
Document subtype:	if applicable
Document stage:	Candidate OpenGIS® Implementation Specification
Document language:	English

License Agreement

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

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

THIS LICENSE IS A COPYRIGHT LICENSE ONLY, AND DOES NOT CONVEY ANY RIGHTS UNDER ANY PATENTS THAT MAY BE IN FORCE ANYWHERE IN THE WORLD.

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

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

Except as contained in this notice, the name of LICENSOR or of any other holder of a copyright in all or part of the Intellectual Property shall not be used in advertising or otherwise to promote the sale, use or other dealings in this Intellectual Property without prior written authorization of LICENSOR or such copyright holder. LICENSOR is and shall at all times be the sole entity that may authorize you or any third party to use certification marks, trademarks or other special designations to indicate compliance with any LICENSOR standards or specifications.

This Agreement is governed by the laws of the Commonwealth of Massachusetts. The application to this Agreement of the United Nations Convention on Contracts for the International Sale of Goods is hereby expressly excluded. In the event any provision of this Agreement shall be deemed unenforceable, void or invalid, such provision shall be modified so as to make it valid and enforceable, and as so modified the entire Agreement shall remain in full force and effect. No decision, action or inaction by LICENSOR shall be construed to be a waiver of any rights or remedies available to it.

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

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)	University of Muenster
Johnny Tolliver	Oak Ridge National Laboratory
Alexander Walkowski	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	1.0.0	Ingo Simonis	All	SAS redesigned to fulfil the requirements set in SAS IE.

vi. Changes to the OGC Abstract Specification

The OGC® Abstract Specification does not require changes to accommodate the technical contents of this ument[IS1][jst2].

vii. Future work

This work depicts the initial version of the Sensor Alert Service. Future modifications are likely to appear in the near future.

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.

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

Introduction

The Sensor Alert Service (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 must 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 and those sensors that are able to send alerts like "above a HIGH threshold" and "below a LOW threshold" can register at the SAS. This allows more “clever” sensors 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 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 is exceeded/felt short of.

All alert types get registered, or advertised, that they can potentially be sent. If an event occurs the node will send it to the SAS via the *publish* operation. A consumer (interested party) may *subscribe* to alerts disseminated by the SAS. If an event occurs the SAS will *notify* all clients subscribed to this event type.

Figure 1: Event Notification System

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 \cite{Group:uq}, 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 (RFC 2222) and Transport Layer Security (RFC 2246). 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) \cite{XMPP.org:fk}.

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 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*

This OWS Common Specification contains a list of normative references that are also applicable to this Implementation Specification.

OGC 05-114 *OpenGIS®* Web Notification Service

OGC 05-090 SWE Architecture

OGC 05-088 SOS

OGC 05-087 O&M

OGC 05-086 SensorML

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.

4.1 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 do always contain a time and location value.

4.2 Notification

A message sent to a receiver indicating some kind of event.

4.3 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. A message contains an alert in its body. A message sent to a receiver indicating some kind of event is a notification.

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.

SOS	Sensor Observation Service
SPS	Sensor Planning Service
WNS	Web Notification Service
SAS	Sensor Alert Service
SWE	Sensor Web Enablement
O&M	Observation and Measurement
SensorML	Sensor Model Language
TML	Transducer Markup 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 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.

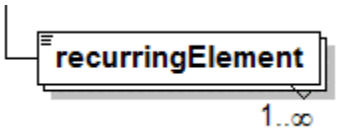
This is a mandatory simple element. Note the upper left corner of the rectangle indicates that data is contained in this element.

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.

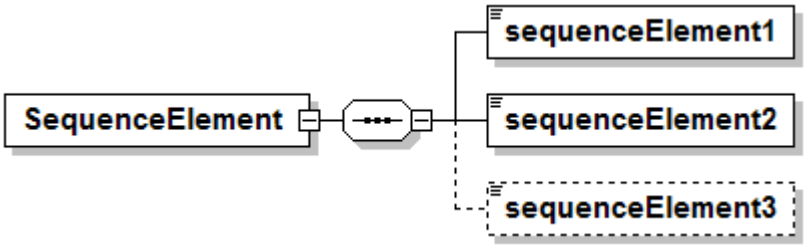


This example shows a recurring element that must occur at least once but can occur an unlimited amount of times. The upper bound here is shown with the infinity symbol.

5.3.4 Sequence Connector

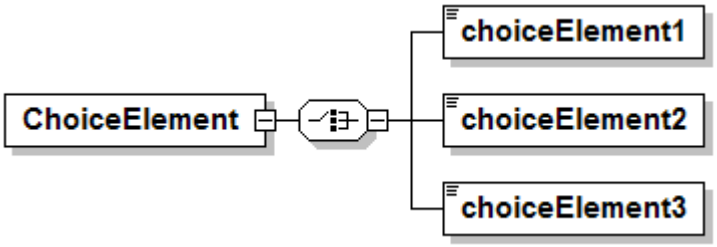
The connection box, called a sequence indicator, indicates that the "SequenceElement" data is made up of three elements. In this example, the first two elements are mandatory and the third element is optional

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



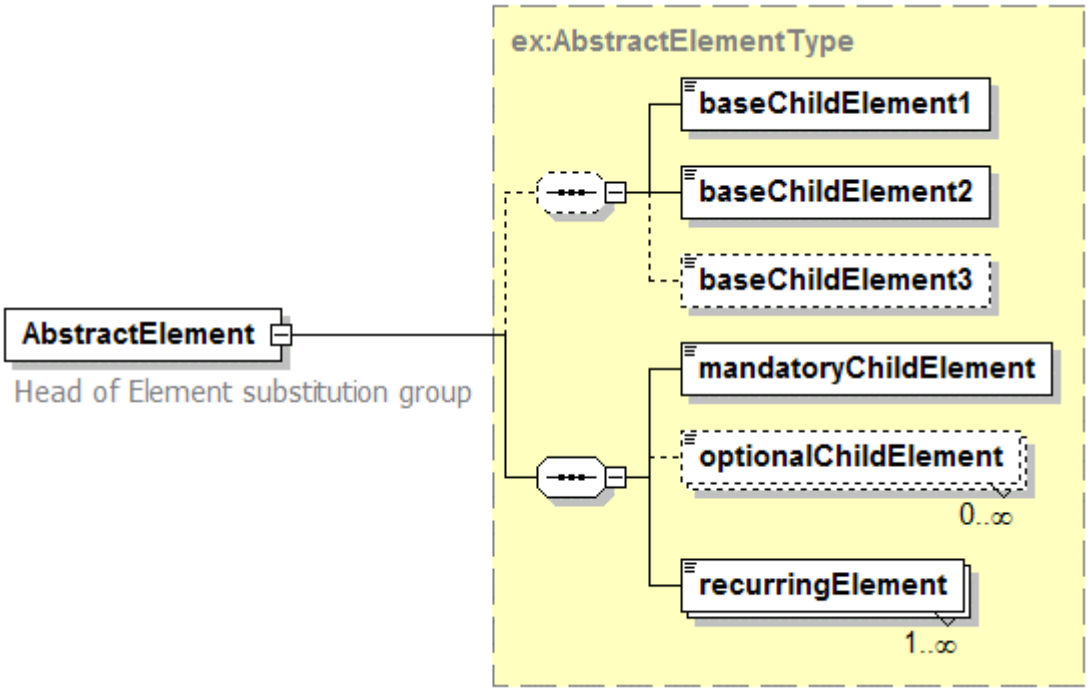
5.3.5 Choice Connector

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.



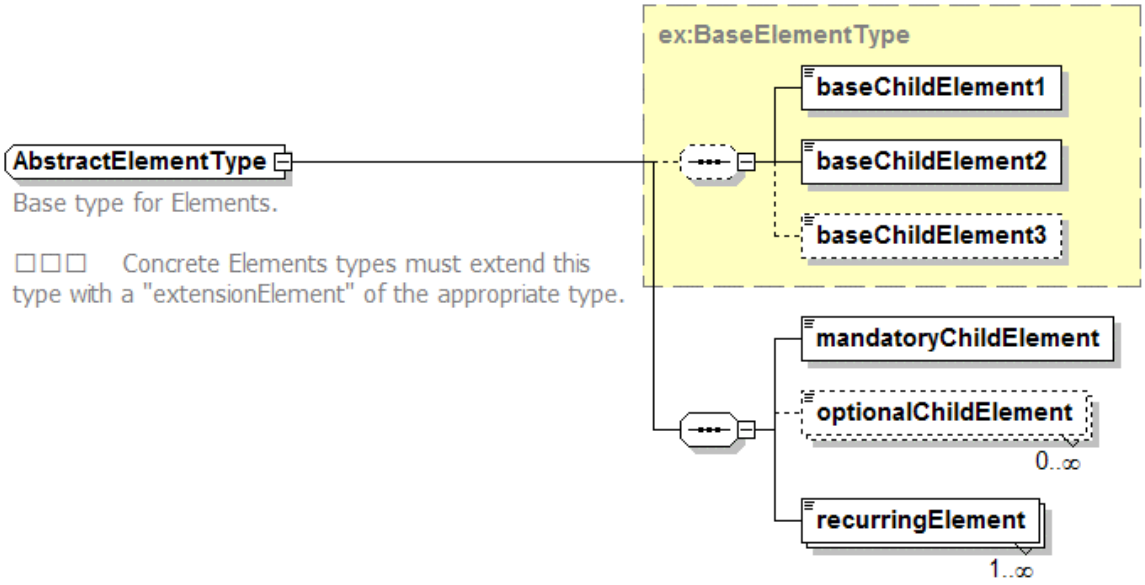
5.3.6 Definition with Complex Type

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 in Annex C, and thus specify the UML model data type and multiplicity of each listed item.

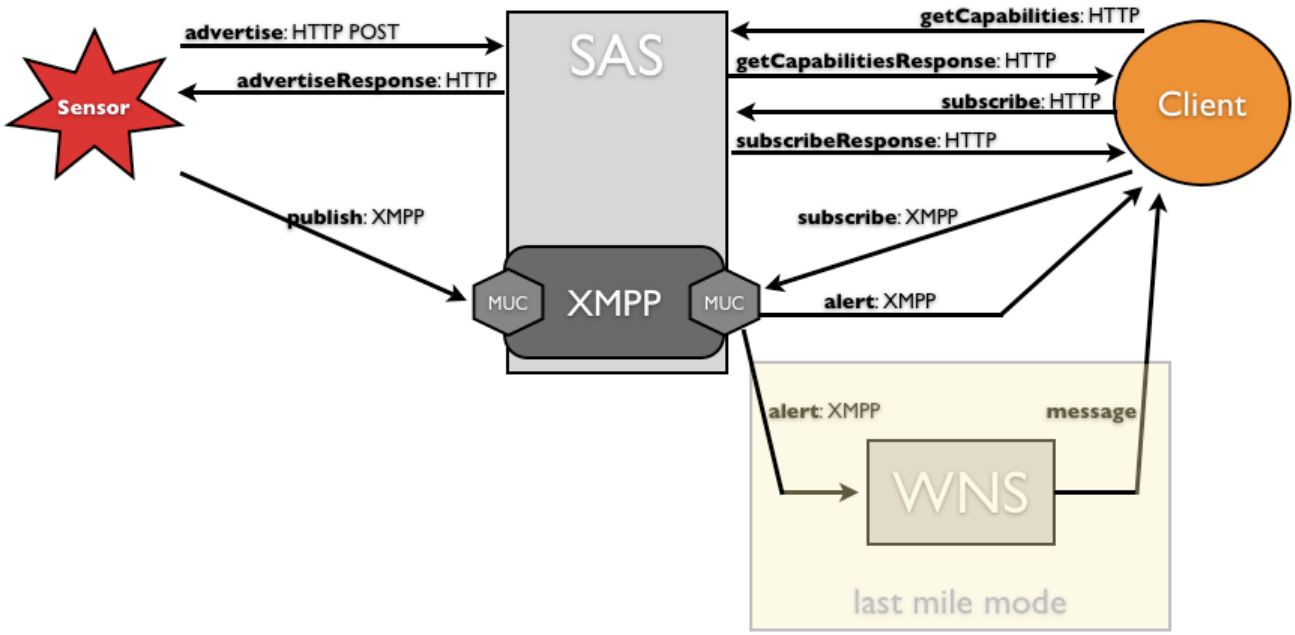
The specified platform-neutral data could be encoded in many alternative ways, each appropriate to one or more specific distributed computing platforms (DCPs). This

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 operations requests (using XML encoding).

6 SAS overview

The specified SAS specifies an interface that allows nodes to advertise and publish observational data or its describing metadata respectively. It is important to emphasize that the SAS itself acts rather like a registry than an event notification system!

The following figure illustrates a high level view on the SAS and the protocols used at the different steps.



Sensors or other data producers do advertise their offers to the Sensor Alert Service using HTTP POST requests. The SAS will then instruct the Messaging Server to create a new MUC (Multi User Chat) 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 any communication with the Messaging Server. SAS contain some kind of look up table internally 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.

Please notice that the differentiation between Messaging Server and SAS is pure logically! 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.

After the advertisement is received by the SAS, it will return the MUC using HTTP. The sensor will then register itself with this MUC to be able to publish data. This registration is in fact a subscription to the MUC and uses XMPP.

On the client side (the client might be a human user or a machine, even another SAS), the client learns about the capabilities of this SAS by sending a *getCapabilities* request using HTTP. The HTTP-base *getCapabilitiesResponse* basically contains all information that was provided by the sensor in its advertisement plus a SAS controlled *SubscriptionOfferingID* (it has to be mentioned that it is up to the SAS to integrate any number of advertisements in a single SubscriptionOfferingID). The SubscriptionOfferingID identifies a single MUC.

To subscribe to a specific SubscriptionOfferingID, the client has to send two (!) times the subscribe request. The first one is HTTP based and will return the MUC. Thus it is more a look up rather than a real subscription. The real subscription takes place after the XMPP based subscription process has been performed. There is one exception: This applies to cases when the client wants to be informed via a Web Notification Service. In and only this case, the subscription process ends after the first, HTTP based request. Instead of 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 the last mile mode, the SAS will not forward messages to an outgoing MUC, but will forward the message to a WNS instead!

The following figures will illustrate the entire SAS usage cycle by means of an example. Given a river segmented that is observed by two sensors. Each sensor has a certain area of operation. The sensors (red stars) advertise their capabilities to the SAS, a client will make use of it.

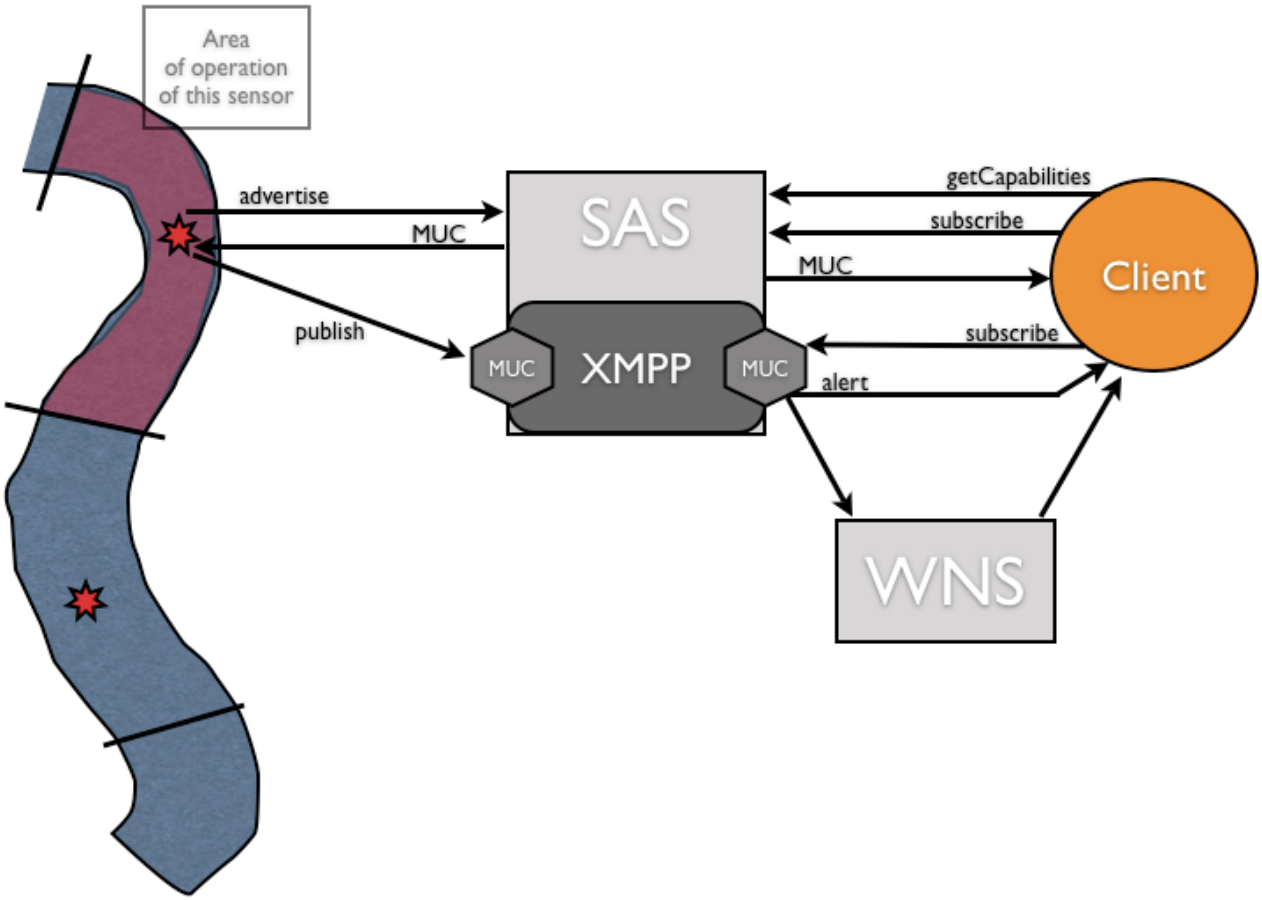


Figure 2: SAS example part 1, overview

The next figures focuses on the advertisement process only. We see that the advertisement contains the following information:

- Feature of Interest (given as a simple string value)
- Operation area (provided as a gml:Point, gml:Curve, or gml:Polygon)
- Alert message structure. The alert message consists of a number of mandatory and optional fields. See published message further down.

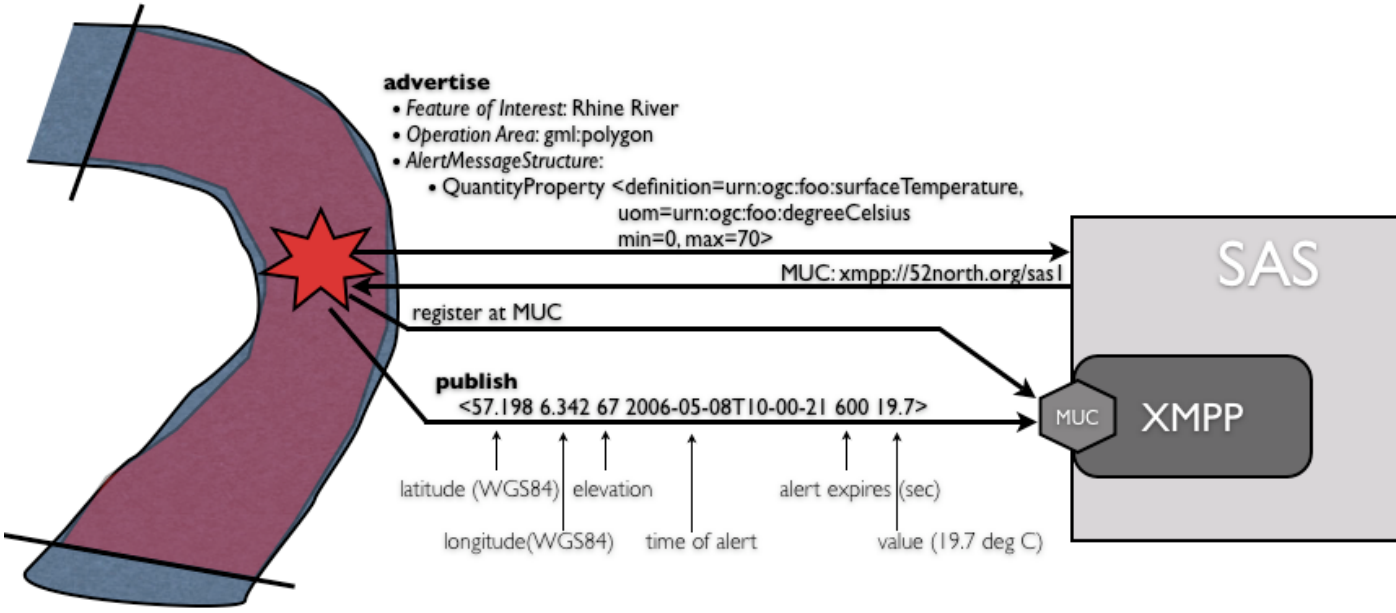


Figure 3: SAS example part 2, advertise

The advertisement contains only the optional part of the publication message that will be sent to the Messaging Server later. The advertisement as well as the response sent by SAS are HTTP based. The SAS simply provides a MUC. The sensor registers to this MUC and is now ready to send observation event messages in the form of "publications". Each publish message contains the mandatory fields

- Latitude (WGS84)
- Longitude (WGS84)
- Elevation (meters above sea level)
- Time of alert
- Time the alert will expire (in sec). In case that this value is 0, the alert has no temporal validity. It depends on the context if this results in a simple event with zero of infinite temporal validity.

- The observationResult; as it was defined in the advertisement. The advertisement clearly defines the phenomenon and the unit of measure as well as the datatype. Possible values are Boolean, Category, Count, and Quantity.

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.

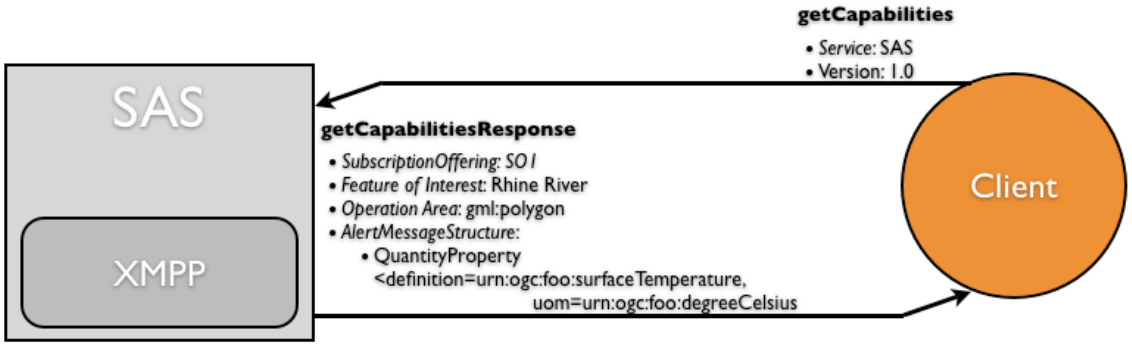


Figure 4: SAS example part 3, getCapabilities

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 getCapabilitiesResponse. This response contains the following information:

- Given the fact that a only a single sensor has advertised its data so far, the SAS will provide a single SubscriptionOfferingID. The IDs must be unique for each SAS.
- Feature of Interest: like advertised by the sensor
- Operation Area: like advertised by the sensor
- AlertMessageStructure: like advertised by the sensor

As we can see, most of the information is simply forwarded from the sensor to the client. It becomes a little bit more complex if more than one sensor is registered as it is up to the SAS to group (or even not to group) more than one sensor under a single SubscriptionOfferingID. If we look at the message structure, we can see that the observationResult values will be of type “Quantitiy”, which indicates that a decimal number will be provided that indicates the surfaceTemperature in degree Celsius at the current position of the sensor. This current version of SAS allows only point based observations. Thus, defining a polygon as operation area indicates the sensor is mobile and will operate in this area. No further information is given. It is intended to extend the SAS specification in the future to support grid based or other than point based observations also. The SAS makes use of the URN-Syntax to properly define the observed phenomenon.

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

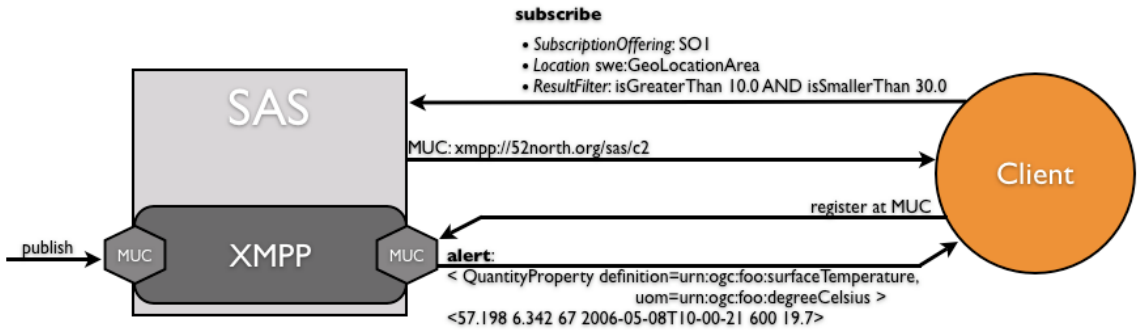


Figure 5: SAS example part 5, subscription and alerting

As mentioned before, the client sends a HTTP based and XML encoded subscribe request to the SAS. Though it is somewhat up to 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. This would require that 1) it is some kind of previously configured MUC, or 2) somebody else has subscribed with the same parameters before. Either way, the SAS will respond 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 that allow detecting abandoned MUCs.

If a sensor is publishing data, the Messaging Server will receive this data at one MUC, checks if necessary the registered conditions and forwards the alert message to those MUCs where the condition is fulfilled. Only if we deal with predefined alerts or alerts that do not allow further constraints (e.g. “battery is low” alerts), the SAS provides the same MUC for incoming as well as outgoing messages. Again, this is implementation specific; particularly as certain security constraints may apply.

The last part of our short example highlights the case that the client does not want to receive the 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 logic, both systems may run on the same machine or the even the same memory space. The following figure illustrates the situation.

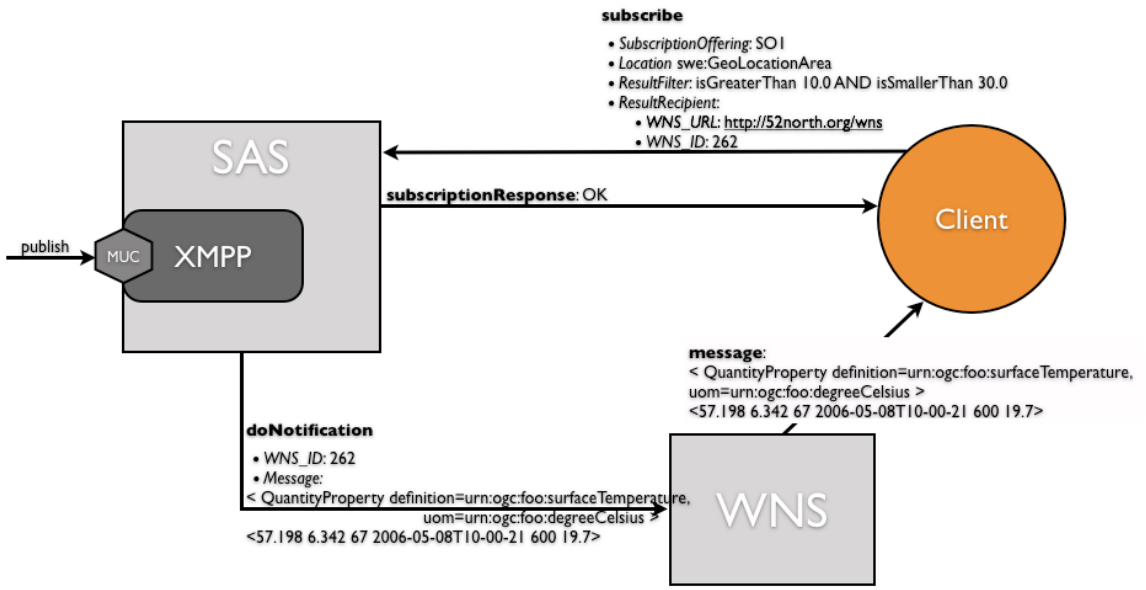


Figure 6: SAS example, part 6, subscription and alerting in last-mile-mode

The subscription request sent by the client looks slightly different as in the previous example. Again, the client defines a location (the area the observation has to be performed in) and an observation result filter (the observation result must be between 10 and 30 degrees Celsius), but provides the WNS data additionally: 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: The old WNS specification defines the message structure for incoming messages. This is void for the current specification, where the WNS capabilities are reduced to protocol transducing only. The messages will not be touched by the WNS anymore. It is the responsibility of the calling service to make sure that message fits into the necessary protocols. This is particularly true for those protocols having specific constraints concerning message length (e.g. SMS with a maximum of 160 characters).

7 Alert Messages

7.1 Sensor Alert Messages

Alert messages are those messages sent from a sensor to the SAS and from the SAS to the client; independently if the message contains an alert of the type "my battery is low" or the simple value of an observation. All alert messages follow the schema given below. To keep the protocol as lightweight as possible, all alert messages use the following structure:

```
<Latitude Longitude Altitude TimeofAlert AlertExpires Value>
```


`Latitude` and `Longitude` are given as WGS84 coordinates. The `Altitude` defines the ellipsoidal height and uses the unit *meters*. `TimeOfAlert` defines the point in time the alert was triggered. `AlertExpires` is rather an indicator for the client and should not be seen as an absolute value. It is given in *seconds*.

The number of `Value` elements is unbounded and depends on the number of observables supported by the sensor. The syntax is as follows:

```
<Double Double Double YYYY-MM-DDTHH-MM-SS+-hh:mm Integer Double>
```

Example:

```
<57.982 7.8335 291.0 2006-09-04T10-00-00+2 0 21.8>
```

All elements are separated by spaces. The `AlertTime` instance is a restricted version of ISO 8601:2000. To overcome ISO pitfalls, the given structure is the only one allowed. The time is always represented as Coordinated Universal Time (UTC) plus/minus possible offsets in hours and minutes: +-hh:mm.

`AlertExpires` defines the number in seconds before the alert expires. The value "`0`" indicates that the alert expiring date is undefined and not that it expires immediately.

The structure of the alert message is partly defined by the sensor or SAS respectively. As the first five elements are fixed, the sensor defines the number of `Value` elements and their type. The element `AlertMessageStructure` looks as illustrated in the following figure.

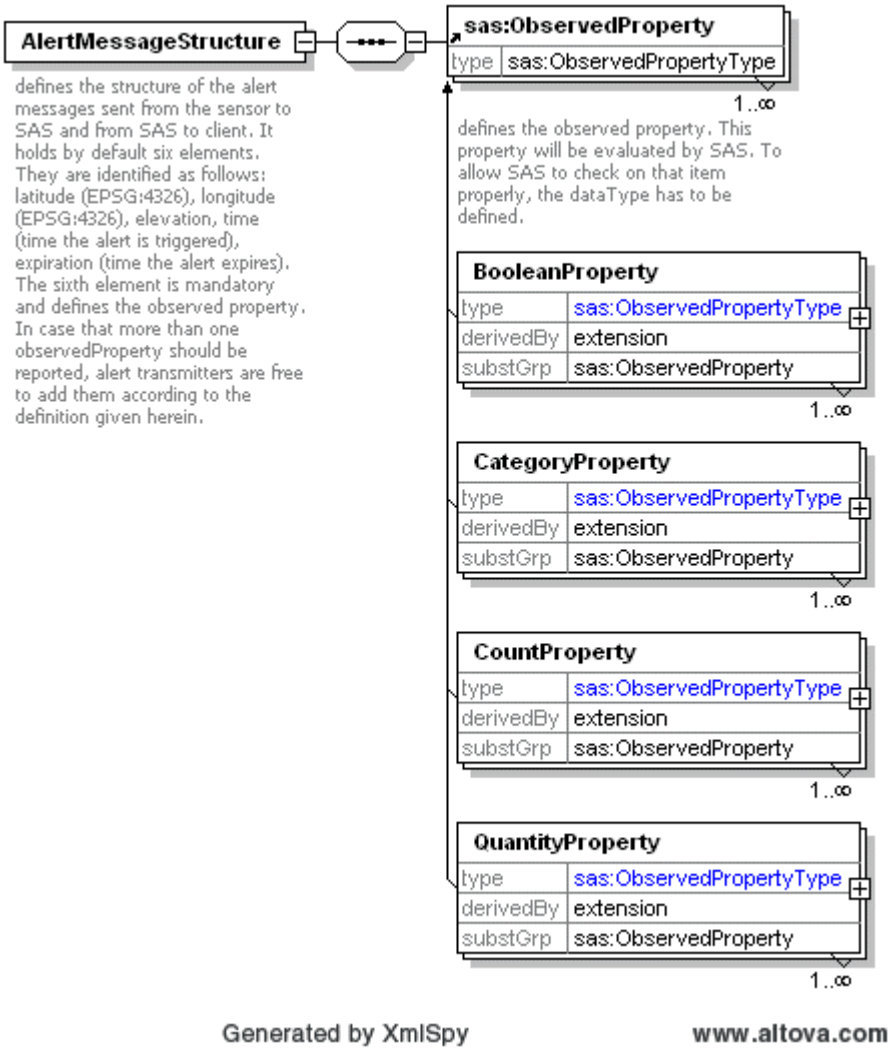


Figure 7: AlertMessageStructure in XMLSpy notation

As we can see, the ObservedProperty (which is the observable the `Value` element provides a value for) can be one out of the four types

- `BooleanProperty` to define true/false types (e.g. if the observable is doorIsOpen, wich can be true or false only).
- `CategoryPropery` to define ordinal structured values, e.g. an observable with the possible values low, medium, heigh. The possible categories for a given observable can be found at the observable definition source or are linked from there.
- `CountProperty` to define integer values. The `CountProperty` defines min and max values optionally.
- `QuantityProperty` to define double values. The `QuantityProperty` defines min and max values optionally.

We will demonstrate this with an example. Say a sensor publishes temperature and cloud coverage data. It would use the following XML fragment:

```
<sas:AlertMessageStructure ...>
    <sas:CountProperty>
        <sas:Content max="50" min="0" definition="urn:ogc:def:temperature"
        uom="urn:ogc:units:degreeCelsius"/>
    </sas:CountProperty>
    <sas:CategoryProperty>
        <sas:Content definition="urn:ogc:def:cloudCoverage"/>
    </sas:CategoryProperty>
</sas:AlertMessageStructure>
```

Listing 1: AlertMessageStructure example

A possible alert from the sensor may look like the following:

```
<57.3632 8.2398 232.0 2006-05-12T23-23-22 3600 12.2 overclouded>
```

7.2 SAS Alert Messages

The alerts send by the SAS server look slightly different from the ones send by the sensor. This becomes necessary as messages that are forwarded by the SAS may use a different alert message structure. We will explain this using an example: Imagine a client that subscribes to all alerts in a specific bounding box. Let's say two sensors do publish alerts. Whereas the first sensor provides temperature data, the second sensor provides temperature and humidity data. As the client cannot know by any means the message structure used by the different sensors, the alert message structure is included in the message. **Note**: A possible alternative would be that the message contains an unique identifier that allows the client to resolve the message structure. We decided against this option to keep the communication patterns as simple as possible.

The element `SASAlert` looks like illustrated in the next figure.

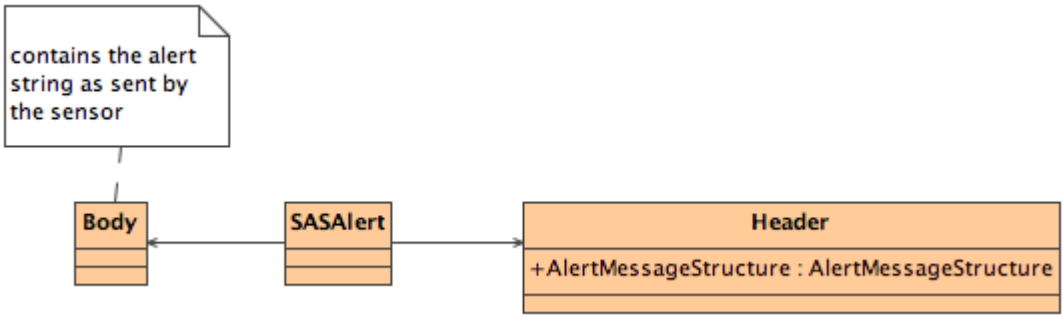


Figure 8: SASAlert element in UML notation

An alert sent by SAS may look like the following:

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

```
<SASAlert xmlns="http://www.opengis.net/sas"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://www.opengis.net/sas ../sasAll.xsd">
   <Header>
      <AlertMessageStructure>
         <CategoryProperty>
            <Content definition="urn:ogc:def:security:window_status"/>
         </CategoryProperty>
      </AlertMessageStructure>
   </Header>
   <Body>51.96 7.607 70.0 2006-09-04T02:27:04 0 window_broken</Body >
</SASAlert>
```

Listing 2: Example of an SASAlert sent by SAS

8 SAS Operations

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

a) 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.

b) Advertise (required 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!

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

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

e) 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).

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

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

Figure 1 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.) A more complete UML model of the SAS interface is provided in Annex C (informative).

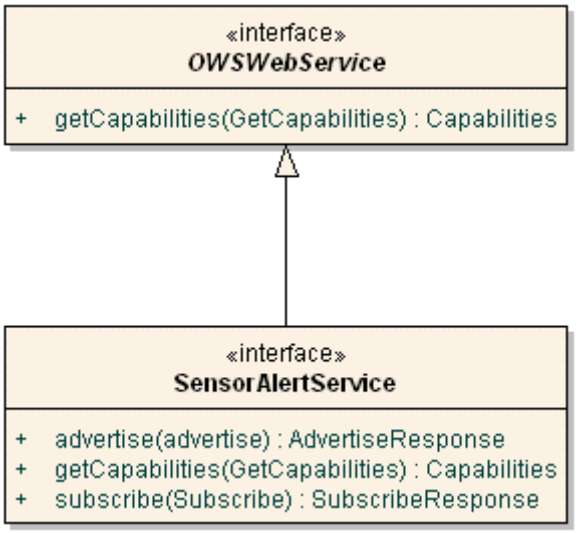


Figure 9: SAS interface UML diagram

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. The UML classes modelling these data structures are included in the complete UML model in Annex C.

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

9 Shared aspects

9.1 Introduction

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

9.2 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
AlertFrequency	Frequency used by the sensor to send alerts. Unit: Alerts per second.	double
AlertMessageStructure	Defines the structure of the alert	complex

Name	Definition	Data type and value
	message. Notice: The first six elements of any alert message are predefined. This element only defines the observedProperties. **See chapter** 7 above.	
DesiredPublicationExpiration	Point in time when the sensor will not publish alerts anymore. Format follows ISO8601:2000 and is specified as: YYYY-MM-DDTHH-MM-SS+-hh:mm	string
FeatureOfInterest	Specifies target feature for which alerts are published. Mostly a helper for in-situ sensors, since geo-location has to be done on the server side. The supported area should be listed in the selected offering capabilities.	string
ObservedProperty	Observable associated with the alerts.	anyURI
OperationArea	Area of operation of the sensor	complex
PublicationID	ID administered by SAS server to uniquely identify advertisement offerings.	ID
SubscriptionID	Unique Identifier for the subscription. Provided by SAS server.	ID
SubscriptionOfferingID	This unique identifier identifies the SubscriptionOffering. It is part of the Capabilities document and can be used by clients in Subscribe operation requests.	token

9.2.1 Operation Area

The `OperationArea` element provides a simple point or bounding box that represents all possible locations a sensor can be placed at. In case of a stationary sensor, usually a point location will be provided. It is intended to extend this element in the future to allow more specific definitions, including curves and polygon geometries. **Note**: The operation area defines the area the sensor can be located. This area is not necessarily identical with the area the sensor observes. It is most likely that we will add an additional element in an upcoming version that allows the definition of the observed area. The following figure illustrates the element.

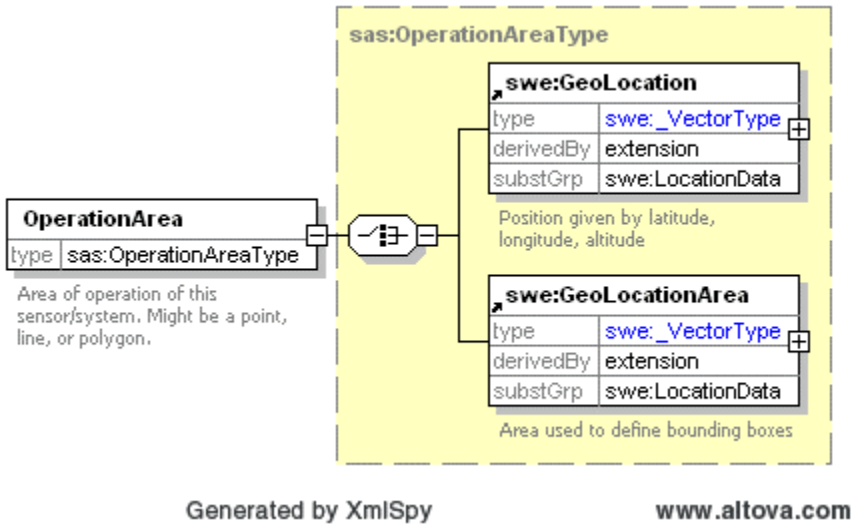


Figure 10: OperationArea element in XMLSpy notation

The `OperationArea` element is defined in swe:Common. Please review the SWE Architecture document for further information.

9.3 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 Service operations and their encoding methods defined in this specification.

Table 2 — Operation request encoding

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

10 GetCapabilities operation (mandatory)

10.1 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 must return to describe its capabilities.

10.2 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.

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)		
AcceptVersions	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) [b]	Each parameter may be implemented by each client [b] 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 [a] If parameter not implemented or not received, shall provide default response If parameter implemented and received, shall provide specified response
updateSequence	Zero or one (optional) [b]		
AcceptFormats	Zero or one (optional) [b]		

a A specific OWS is allowed to make mandatory server implementation of any of these three parameters.

b If a specific OWS makes mandatory server implementation of any of these three parameters, that parameter can also be made mandatory in the operation request, also requiring client implementation of this parameter.

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.

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" xmlns:ows="http://www.opengeospatial.net/ows"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xsi:schemaLocation="http://www.opengis.net/sas
http://mars.uni-muenster.de/swerep/trunk/sas/0.0.1/sasGetCapabilities.xsd" service="SAS">
</GetCapabilities>
```

10.3 GetCapabilities operation response

10.3.1 Normal response

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 about subscriptions and advertisements offered at this SAS interface, as specified in following Subclauses below.

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].

10.3.2 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 in **Error! Reference source not found.** shall be included or not depending on whether that operation is implemented by that server. In Table 6 and **Error! Reference source not found.**, 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 right 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.
	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.

10.3.3 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. The two necessary sections provide all information a client needs to perform its intended task.

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` elements. The only possible values are true and false. In case the SAS accepts advertisements, the next step would be to send an advertise request.

In case that a client wants to subscribe to alerts, the `SubscriptionOffering` Element contains all information how the final (at the messaging server) subscription has to be performed. Multiple `SubcriptionOffering` elements are pooled in a

`SubscriptionOfferingList`. The following figure shows the structure of the Contents element.

Figure 11: Contents element in XMLSpy notation

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

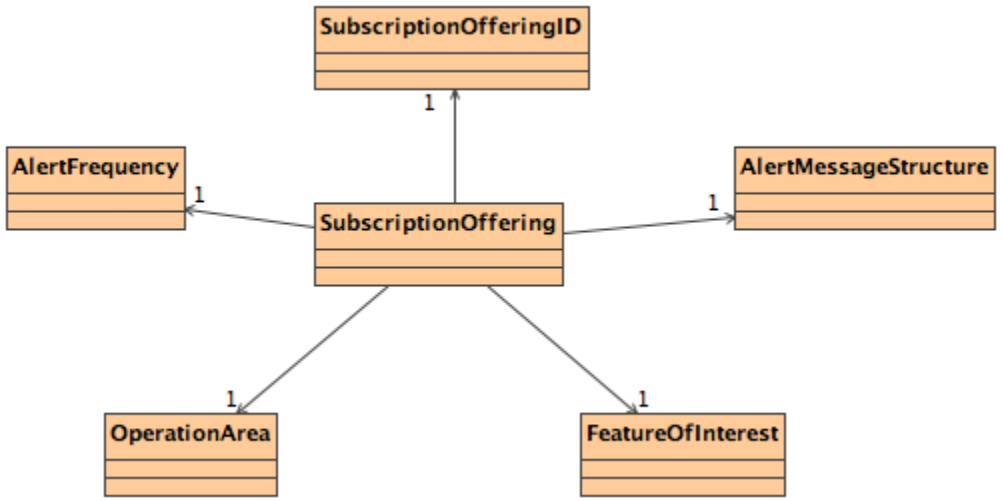


Figure 12: SubscriptionOffering element in UML notation

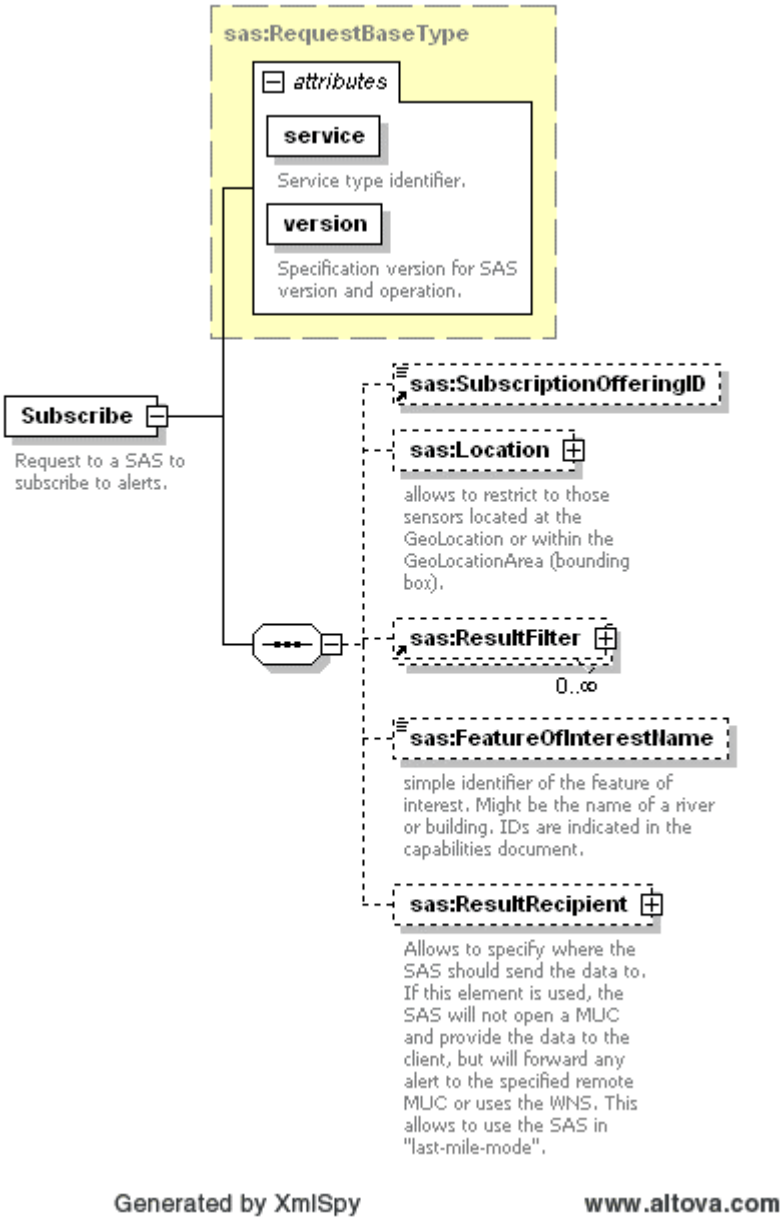


Figure 13: SubscriptionOffering in XMLSpy notation

10.3.4 Capabilities document XML encoding

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

10.3.5 Capabilities document example

In response to GetCapabilities operation request, a SAS server might generate a document that looks like (See Annex D1 for an entire capabilities document):

Example will follow

10.3.6 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.

11 Advertise operation (mandatory)

11.1 Introduction

The advertise operation allows SAS clients to advertise what kind of data could be published. The operation might be used by sensors able to launch a 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 feature of interest, its operation area, the frequency it will send the alerts and about the alert itself (as part of the `AlertMessageStructure` element).

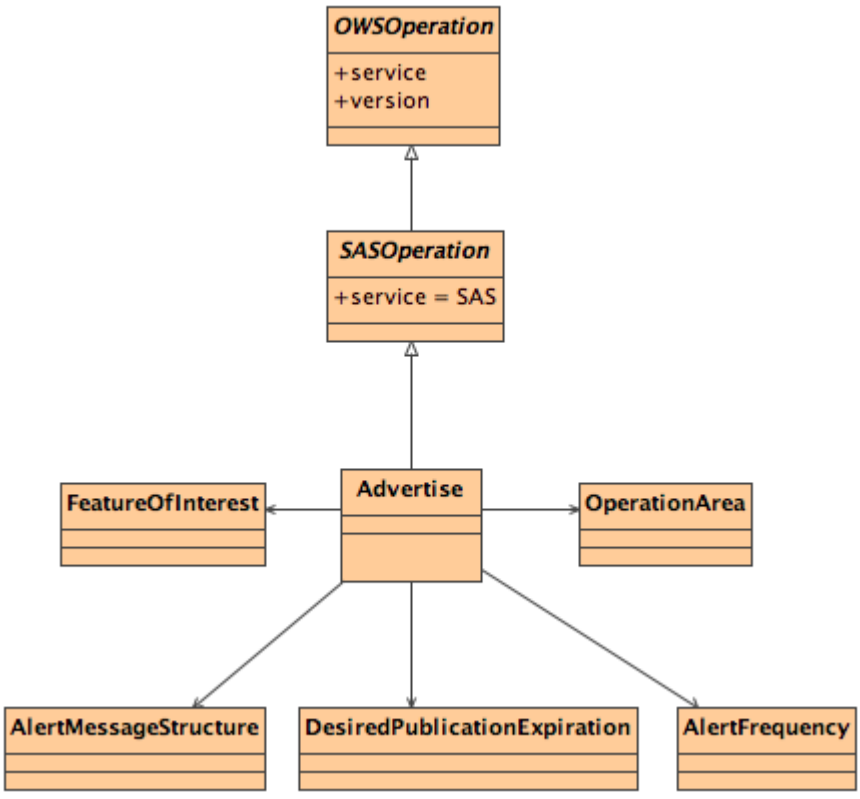


Figure 14: Advertise element in UML notation

Figure 15: Advertise element in XMLSpy notation

11.2 Advertise operation request

11.2.1 Advertise 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 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 each Implementation Specification and Schemas version	One (mandatory)
FeatureOfInterest	see Subclause 9	String	One (mandatory)
OperationArea	see Subclause 9	complex	One (mandatory)
AlertFrequency	Frequency the alerts will be send in "alerts per second"	double	One (mandatory)
DesiredPublicationExpiration	see Subclause 9	String	One (mandatory)
AlertMessageStructure	see Subclause 9	complex	One (mandatory)

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).

11.2.2 Advertise request KVP encoding

KVP encoding is not supported.

11.2.3 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).

EXAMPLE An example Advertise operation request XML encoded for HTTP POST is:

```
<?xml version="1.0" encoding="UTF-8"?>
<Advertise xmlns="http://www.opengis.net/sas" xmlns:xsi="http://www.w3.org/2001/XMLSchema-
instance" xsi:schemaLocation="http://www.opengis.net/sas ../sasAll.xsd" xmlns:swe
="http://www.opengis.net/swe" service="SAS" version="1.0.0">
    <FeatureOfInterest>
        <Name>Muenster</Name>
        <Description>The temperature is measured by a sensor which is mounted at a weather
        station located on the roof of the IFGI in Muenster
        </Description>
    </FeatureOfInterest>
    <OperationArea>
        <swe:GeoLocation>
            <swe:longitude>
                <swe:Quantity>51.96</swe:Quantity>
                </swe:longitude>
            <swe:latitude>
                <swe:Quantity>7.607</swe:Quantity>
                </swe:latitude>
            <swe:altitude>
                <swe:Quantity>78</swe:Quantity>
            </swe:altitude>
        </swe:GeoLocation>
    </OperationArea>
    <AlertMessageStructure>
        <QuantityProperty>
            <Content definition="urn:ogc:temperature" uom="urn:ogc:foo:degree_Celsius"
            min="0" max="40"/>
        </QuantityProperty>
    </AlertMessageStructure>
    <AlertFrequency>0.1</AlertFrequency>
    <DesiredPublicationExpiration>2007-10-30T00:00:00Z</DesiredPublicationExpiration >
</Advertise>
```

Listing 3: Example Advertise

11.3 Advertise operation response

11.3.1 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 figure.

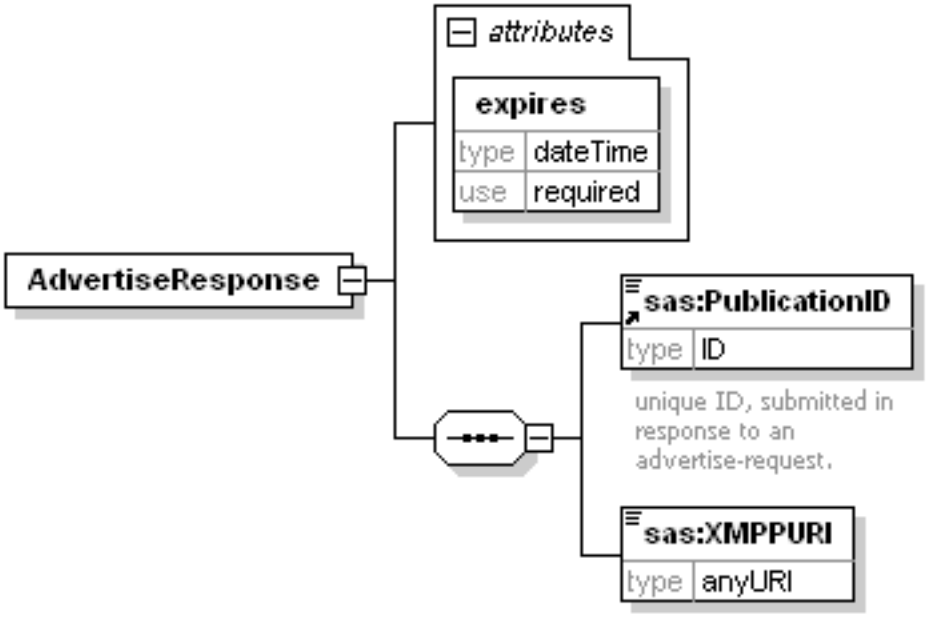


Figure 16: AdvertiseResponse element in XMLSpy notation

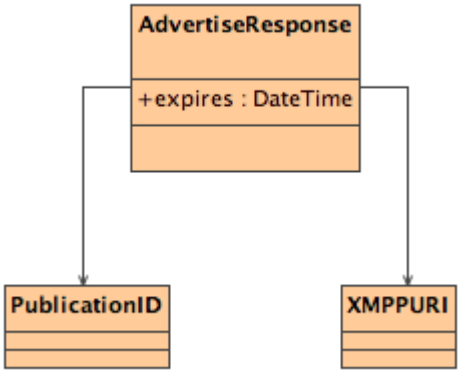


Figure 17: AdvertiseResponse element in UML notation

Table 8: 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.	Double; the value follows ISO 8601:2000, i.e. YYYY-MM-DDTHH-MM-SS+-hh:mm	One (mandatory)
PublicationID	Unique identifier, provided by SAS	ID	One (mandatory)

XMPPURI	Subscription endpoint where the sensor has to register using XMPP to publish alerts	URI	One (mandatory)

11.3.2 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).

11.3.3 Advertise response example

```
<?xml version="1.0" encoding="UTF-8"?>
<AdvertiseResponse … expires ="2007-01-01T00:00:00Z">
    <PublicationID>Pub301</PublicationID>
    <XMPPURI>xmpp://52North.org/sas/publications/c301</XMPPURI >
</AdvertiseResponse>
```

Listing 4: Example advertise response

11.3.4 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 9. For each listed exceptionCode, the contents of the "locator" parameter value shall be as specified in the right column of Table 9.

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 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

12 RenewAdvertisement operation (mandatory)

12.1 Introduction

The RenewAdvertisement operation allows SAS clients to renew 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.

12.2 RenewAdvertisement operation request

12.2.1 RenewAdvertisement request parameters

A request to perform the RenewAdvertisement operation shall include the parameters listed and defined inTable 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.

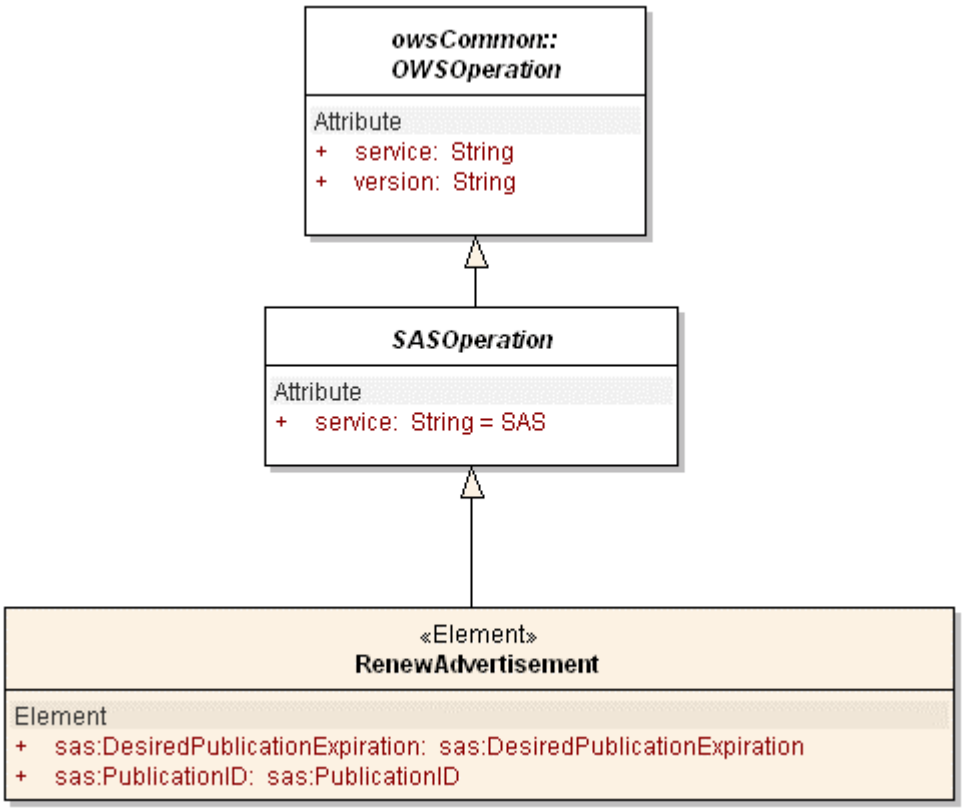


Figure 18: RenewAdvertisement operation in UML notation

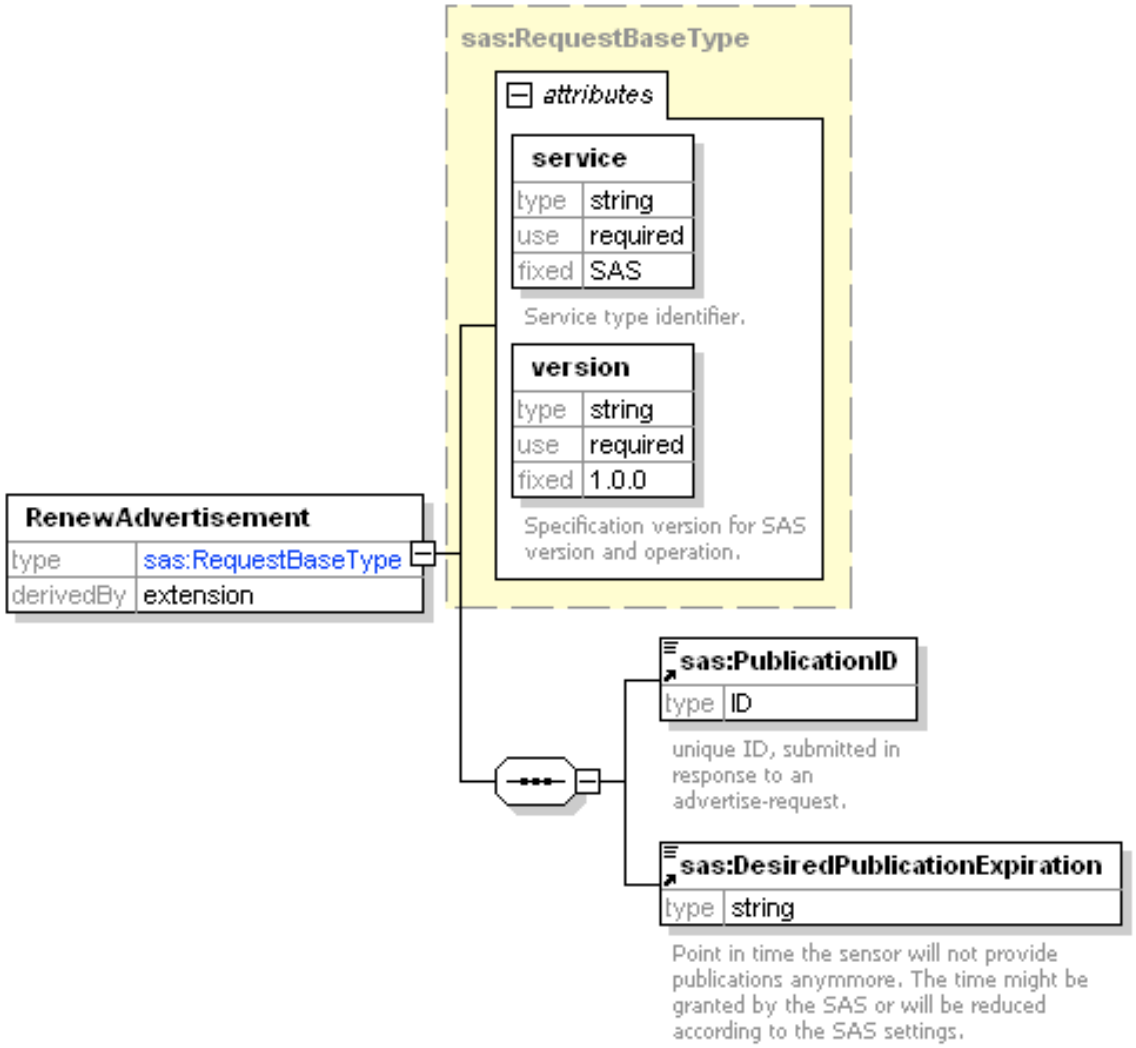


Figure 19: RenewAdvertisement operation in XMLSpy notation

Table 10 — 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 (e.g., “WMS”, “WFS”)	One (mandatory)
version	Specification version for operation	Character String type, not empty Value is specified by each Implementation Specification and Schemas version	One (mandatory)
PublicationID	ID administered by SAS server to uniquely identify advertisement offerings.	ID	One (mandatory)
DesiredPublicationExpiration	see Subclause 9	String	One (mandatory)

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 10 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).

12.2.2 RenewAdvertisement request KVP encoding

KVP encoding not supported!

12.2.3 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).

EXAMPLE An example RenewAdvertisement operation request XML encoded for HTTP POST is:

```
<?xml version="1.0" encoding="UTF-8"?>
<RenewAdvertisement … service ="SAS" version="1.0.0">
    <PublicationID>Pub301</PublicationID>
    <DesiredPublicationExpiration>
        2007-10-30T00:00:00Z
    </DesiredPublicationExpiration >
</RenewAdvertisement>
```


Listing 5: Example renew advertisement

12.3 RenewAdvertisement operation response

12.3.1 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 11. This table also specifies the UML model data type plus the multiplicity and use of each listed part.

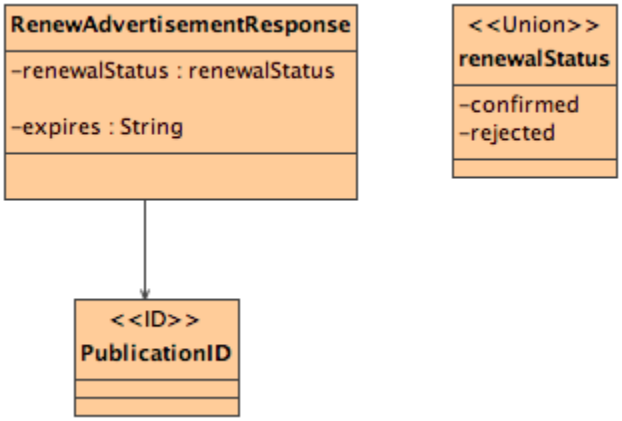


Figure 20: RenewAdvertisementResponse in UML notation

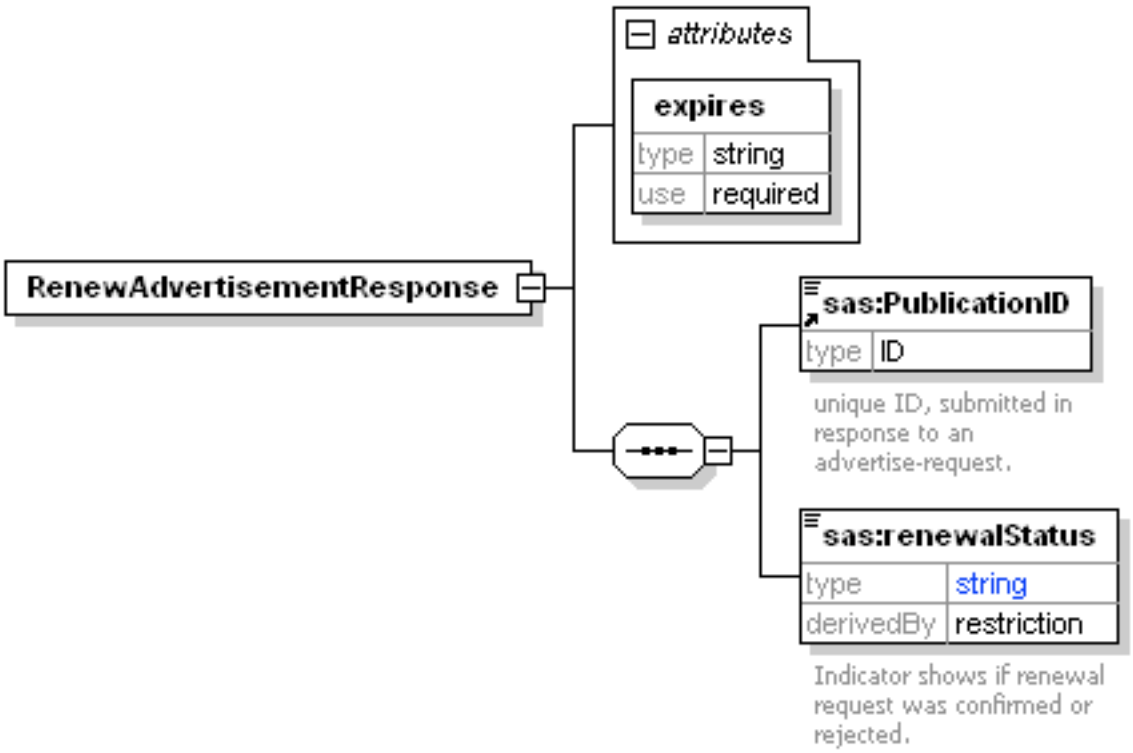


Figure 21: RenewAdvertisementResponse in XMLSpy notation

Table 11 — Parts of RenewAdvertisement operation response

Name	Definition	Data type and values	Multiplicity and use
PublicationID	ID administered by SAS server to uniquely identify advertisement offerings.	ID	one (mandatory)
expires	Defines the time this advertisement will expire.	String, follows ISO 8601:2000	One (mandatory)
renewalStatus	Identifier if renewal was successful	String “confirmed” or “rejected”	one (mandatory)

12.3.2 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).

12.3.3 RenewAdvertisement response example

A RenewAdvertisement operation response for SAS can look like this encoded in XML:

```
<?xml version="1.0" encoding="UTF-8"?>
<RenewAdvertisementResponse … expires="2007-06-01T00:00:00Z">
```

```
    <PublicationID>Pub301</PublicationID>
    <renewalStatus>confirmed</renewalStatus>
</RenewAdvertisementResponse>
```

Listing 6: Example RenewAdvertisementResponse

12.3.4 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 12. For each listed exceptionCode, the contents of the "locator" parameter value shall be as specified in the right 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 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

13 CancelAdvertisement operation (mandatory)

13.1 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 PublicationID that was provided by the SAS server.

13.2 CancelAdvertisement operation request

13.2.1 CancelAdvertisement request parameters

A request to perform the CancelAdvertisement operation shall include the parameters listed and defined in Table 7. 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.

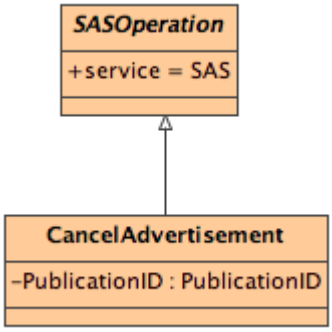


Figure 22: CancelAdvertisement in UML notation

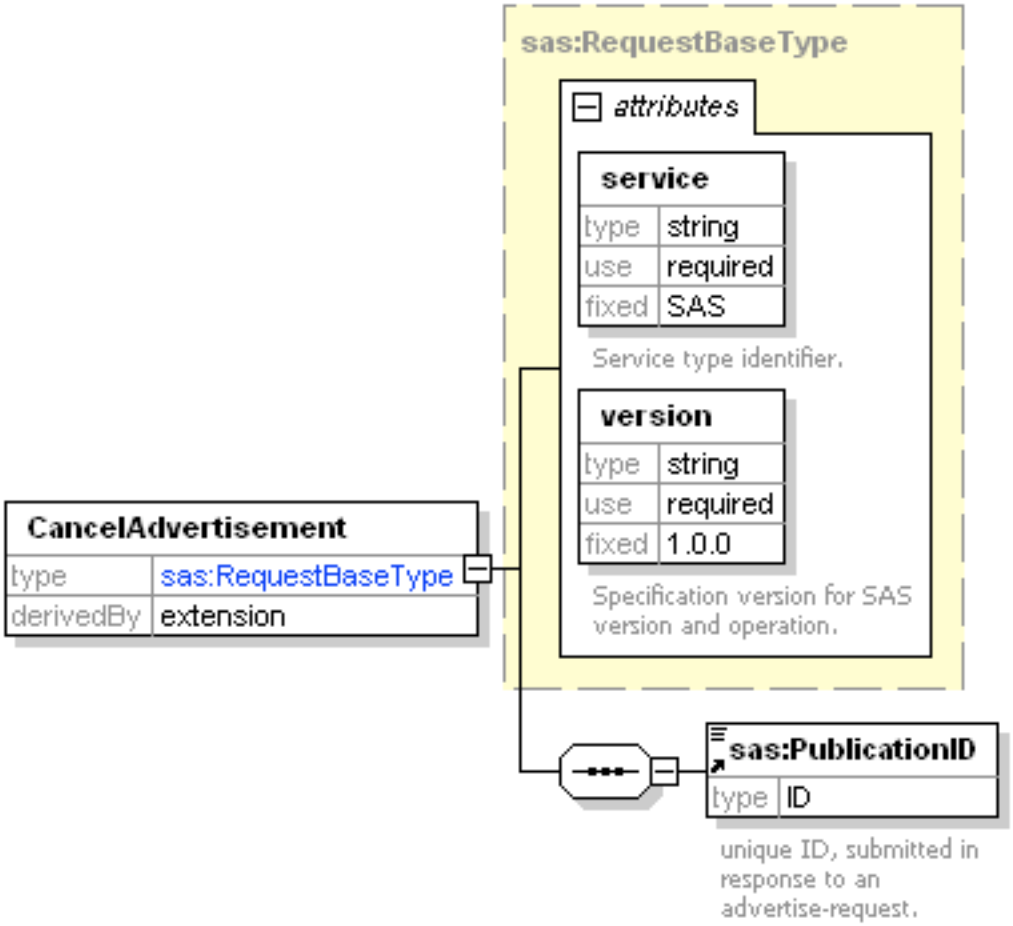


Figure 23: CancelAdvertisement in XMLSpy notation

Table 13 — 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 (e.g., “WMS”, “WFS”)	One (mandatory)
version	Specification version for operation	Character String type, not empty Value is specified by each Implementation Specification and Schemas version	One (mandatory)
PublicationID	ID administered by SAS server to uniquely identify advertisement offerings.	ID	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 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 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).

13.2.2 CancelAdvertisement request KVP encoding

KVP encoding not supported!

13.2.3 CancelAdvertisement request XML encoding (mandatory)

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 (sasAdvertisement.xsd).

EXAMPLE An example CancelAdvertisement operation request XML encoded for HTTP POST is:

```
<?xml version="1.0" encoding="UTF-8"?>
<CancelAdvertisement service="SAS" version="0.0.1" xmlns="http://www.opengis.net/sas"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xsi:schemaLocation="http://www.opengis.net/sas
http://mars.uni-muenster.de/swerep/trunk\SAS\0.0.1\sasAdvertise.xsd">
    <PublicationID>pub-1</PublicationID>
</CancelAdvertisement>
```

13.3 CancelAdvertisement operation response

13.3.1 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 14. This table also specifies the UML model data type plus the multiplicity and use of each listed part.

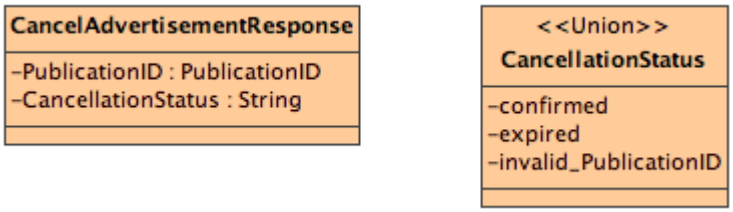


Figure 24: CancelAdvertisementResponse in UML notation

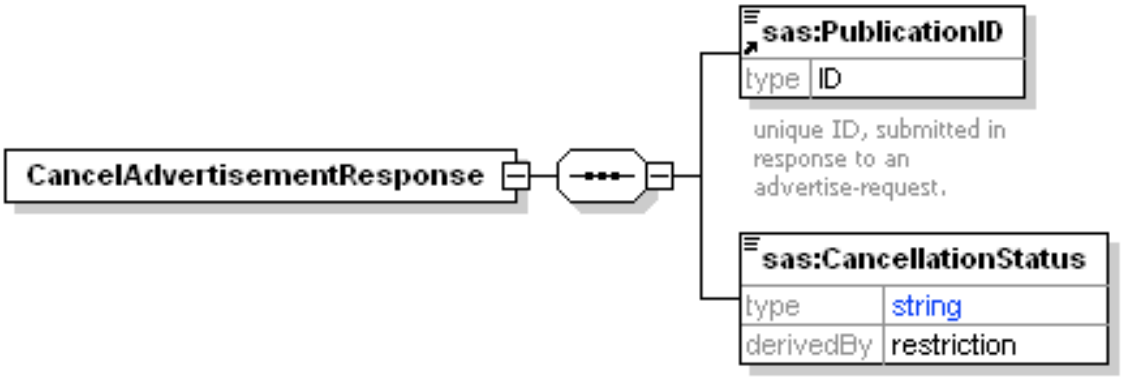


Figure 25: CancelAdvertisementResponse in XMLSpy notation

Table 14 — Parts of CancelAdvertisement operation response

Name	Definition	Data type and values	Multiplicity and use
PublicationID	Unique identifier provided by SAS upon advertise or renewAdvertisement request	ID	one (mandatory)
cancellationStatus	Identifies the status of the cancellation request	String “confirmed” “expired” or “invalid_PublicationID”	one (mandatory)

13.3.2 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).

13.3.3 CancelAdvertisement response example

A CancelAdvertisement operation response for SAS can look like this encoded in XML:

```
<?xml version="1.0" encoding="UTF-8"?>
<CancelAdvertisementResponse xmlns="http://www.opengis.net/sas"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://www.opengis.net/sas
http://mars.uni-muenster.de/swerep/trunk/SAS/0.0.1/sasAdvertise.xsd">
   <SubscriptionID>pub-1</ SubscriptionID >
   <Status>confirmed</ Status >
</CancelAdvertisementResponse>
```

Listing 7: Example CancelAdvertisement response

13.3.4 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 15. For each listed exceptionCode, the contents of the "locator" parameter value shall be as specified in the right 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 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

14 Subscribe operation (mandatory)

14.1 Introduction

The Subscribe operation allows SAS clients to subscribe to alerts that are advertised and published at this SAS. The consumer can define custom alerts, based on the advertised and published alerts. This is done by using constraining elements like result filter, location areas etc.

14.2 Subscribe operation request

14.2.1 Subscribe request parameters

A request to perform the Subscribe operation shall include the parameters listed and defined in Table 16. 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.

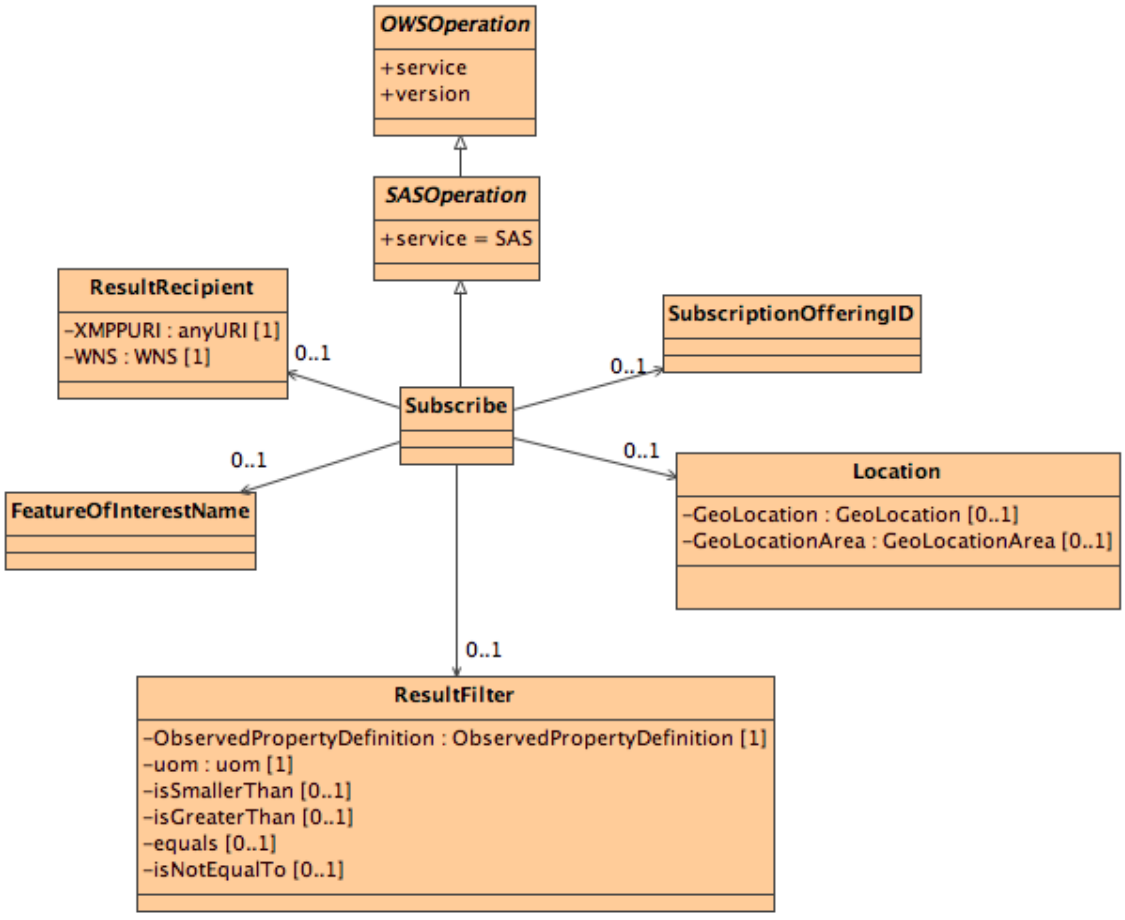


Figure 26: Subscribe in UML notation

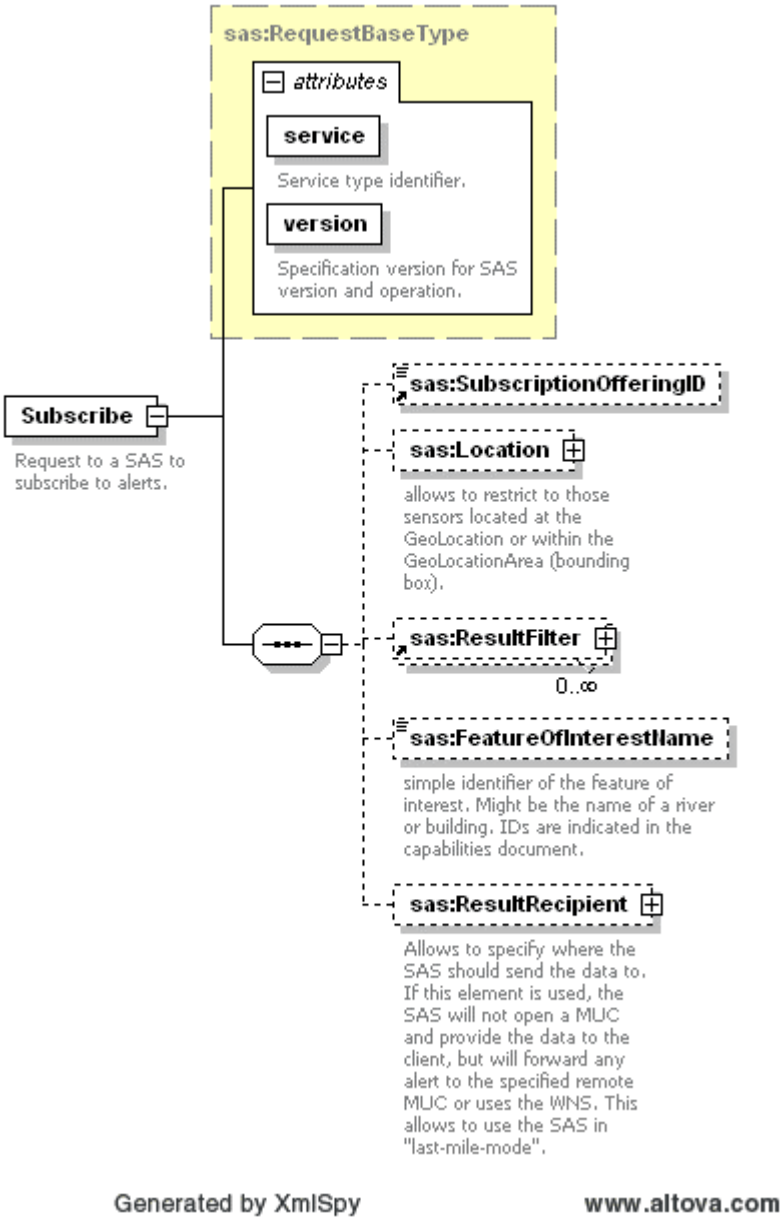


Figure 27: Subscribe in XMLSpy notation

Table 16 — 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 (e.g., “WMS”, “WFS”)	One (mandatory)
version	Specification version for operation	Character String type, not empty Value is specified by each Implementation Specification and Schemas version	One (mandatory)
SubscriptionOfferingID	see Subclause 9	Token	One (optional)
Location	Defines the position or bounding box from where the alert has to be triggered	swe:GeoLocation or swe:GeoLocationArea (see SWE Architecture IPR)	One (optional)
ResultFilter	Allows constraining alert values. Possible filters are isSmallerThan, isGreaterThan, equals, and isNotEqualTo	Type depends on the advertised type of the alert message structure element. Is of type double if isSmallerThan or isGreaterThan is used. Can be of type double or string if one of the other two filters is used.	One (optional)
FeatureOfInterestName	Defines the feature of interest using its name, e.g. “RhineRiver”	String	One (optional)
ResultRecipient	Used to instruct the SAS to send alerts to a specific XMPP-MUC or to use a WNS for alert message forwarding. If not ResultRecipient is 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 (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 16 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.

14.2.2 Subscribe request KVP encoding

KVP encoding not supported!

14.2.3 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).

EXAMPLE An example Subscribe operation request XML encoded for HTTP POST is:

```
<?xml version="1.0" encoding="UTF-8"?>
<Subscribe … service ="SAS" version="1.0.0">
    <ResultFilter ObservedPropertyDefinition="urn:ogc:temperature"
    uom="degree Celsius">
        <isGreaterThan>27.5</isGreaterThan>
    </ResultFilter>
    <FeatureOfInterestName>Muenster</FeatureOfInterestName >
</Subscribe>
```

Listing 8: Example subscribe request. All alerts from FeatureOfInterest "Muenster" with temperature vales greater than 27.5 degrees Celsius are subscribed

14.3 Subscribe operation response

14.3.1 Normal response parameters

The normal response to a valid Subscribe operation request shall be SubscribeResponse. More precisely, a response from the Subscribe 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.

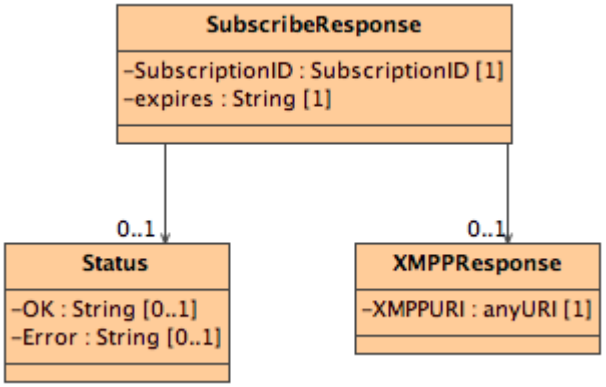


Figure 28: SubscribeResponse in UML notation

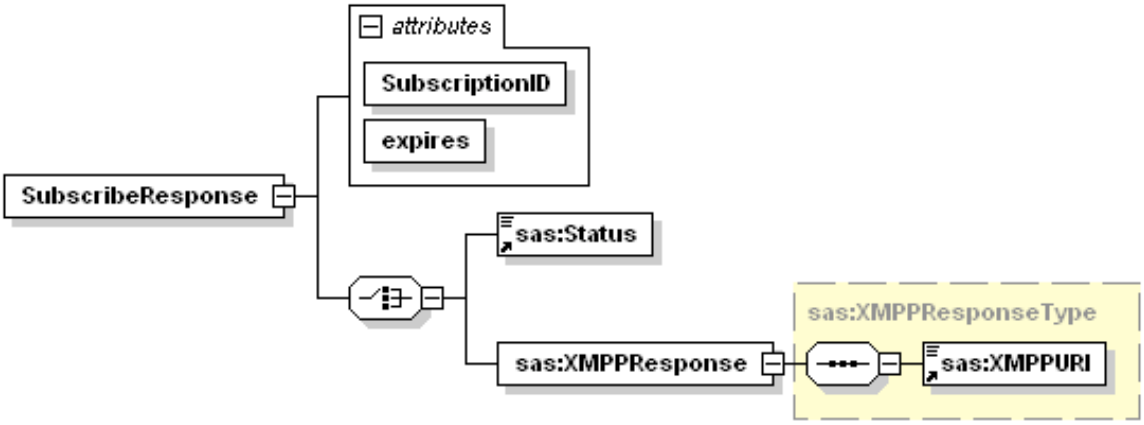


Figure 29: SubscribeResponse in XMLSpy notation

Table 17 — 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 maximal time a subscription remains valid. After this period, the client has to send a SubscriptionRenewal request.	String, follows ISO 8601:2000 and is restricted to YYYY-MM-DDTHH:MM:SS+-hh:mm	one (mandatory)
Status	Indicates if everything worked fine.	String ("OK" or "Error")	one (optional)
XMPPResponse	In case that the client didn't provide any ResultRecipient data in its request, the SAS supposes that a MUC has to be provided. All alerts matching all conditions will be sent to this MUC. If the response contains the XMPPResponse, the Status element is omitted.	anyURI	one (mandatory)

14.3.2 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).

14.3.3 Subscribe response example

A Subscribe operation response for SAS can look like this encoded in XML:

```
<?xml version="1.0" encoding="UTF-8"?>
<SubscribeResponse …
    SubscriptionID ="Sub101" expires="2007-01-01T00:00:00Z">
    <XMPPResponse>
        <XMPPURI>xmpp://52North.org/sas/publications/c1</XMPPURI >
    </XMPPResponse>
</SubscribeResponse>
```

Listing 9: Example SubscribeResponse

14.3.4 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 18. For each listed exceptionCode, the contents of the "locator" parameter value shall be as specified in the right 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 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

15 RenewSubscription operation (mandatory)

15.1 Introduction

The RenewSubscription operation allows SAS clients to renew the subscription that has expired. This operation becomes necessary as SAS servers determine the maximal time before a subscription expires automatically. In case that the former MUC is already deleted by the SAS, a new MUC might be provided (or the old one reopened).

15.2 RenewSubscription operation request

15.2.1 RenewSubscription request parameters

A request to perform the RenewSubscription 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.

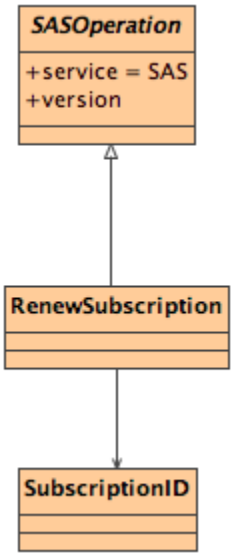


Figure 30: RenewSubscription in UML notation

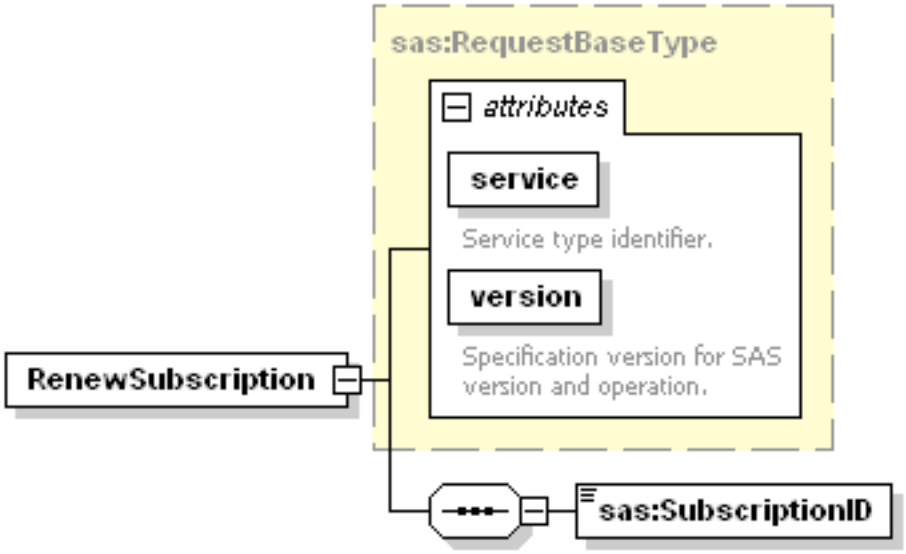


Figure 31: RenewSubscription in XMLSpy notation

Table 19 — Parameters in RenewSubscription 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 (e.g., “WMS”, “WFS”)	One (mandatory)
version	Specification version for operation	Character String type, not empty Value is specified by each Implementation Specification and Schemas version	One (mandatory)
SubscriptionID	Identifier provided by SAS as part of a SubscribeResponse (see Subclause 14)	ID	one (mandatory)

The “Multiplicity and use” column in Table 19 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).

15.2.2 RenewSubscription request KVP encoding

KVP encoding not supported!

15.2.3 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).

EXAMPLE An example RenewSubscription operation request XML encoded for HTTP POST is:

```
<?xml version="1.0" encoding="UTF-8"?>
<RenewSubscription … service ="SAS" version="1.0.0">
    <SubscriptionID>Sub103</SubscriptionID>
</RenewSubscription>
```

Listing 10: Example RenewSubscription

15.3 RenewSubscription operation response

15.3.1 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 20. This table also specifies the UML model data type plus the multiplicity and use of each listed part.

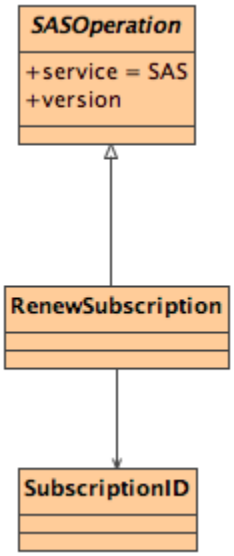


Figure 32: RenewSubscriptionResponse in UML notation

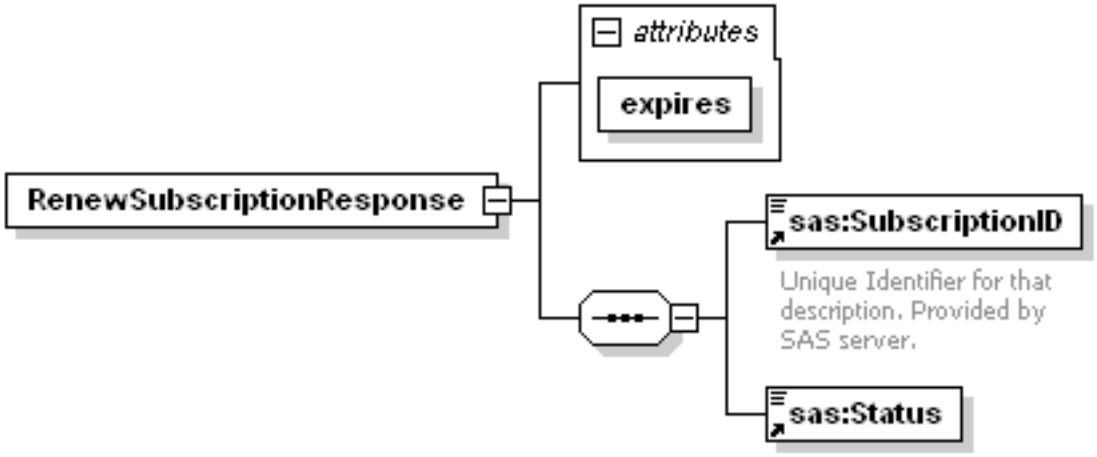


Figure 33: RenewSubscriptionResponse in XMLSpy notation

Table 20 — 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 14)	ID	one (mandatory)
expires	The time that this subscription automatically terminates. Each SAS is free to choose the maximal time a subscription remains valid. After this period, the client has to send a SubscriptionRenewal request.	String, follows ISO 8601:2000 and is restricted to YYYY-MM-DDTHH:MM:SS+-hh:mm	one (mandatory)
Status	Indicates if everything worked fine	String ("OK" or "Error")	one (optional)

15.3.2 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).

15.3.3 RenewSubscription response example

A RenewSubscription operation response for SAS can look like this encoded in XML:

```
<?xml version="1.0" encoding="UTF-8"?>
<RenewSubscriptionResponse … expires ="2007-06-01T00:00:00Z">
    <SubscriptionID>Sub103</SubscriptionID>
    <Status>OK</Status>
</RenewSubscriptionResponse>
```

Listing 11: Example RenewSubscription response

15.3.4 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 21. For each listed exceptionCode, the contents of the "locator" parameter value shall be as specified in the right 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 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

16 CancelSubscription operation (mandatory)

16.1 Introduction

The CancelSubscription operation allows SAS clients to cancel a subscription.

16.2 CancelSubscription operation request

16.2.1 CancelSubscription request parameters

A request to perform the CancelSubscription 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.

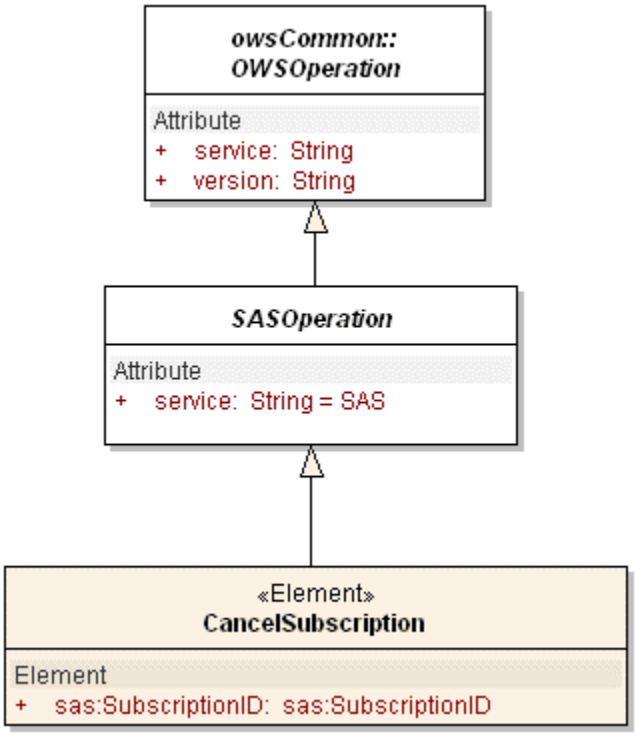


Figure 34: CancelSubscription in UML notation

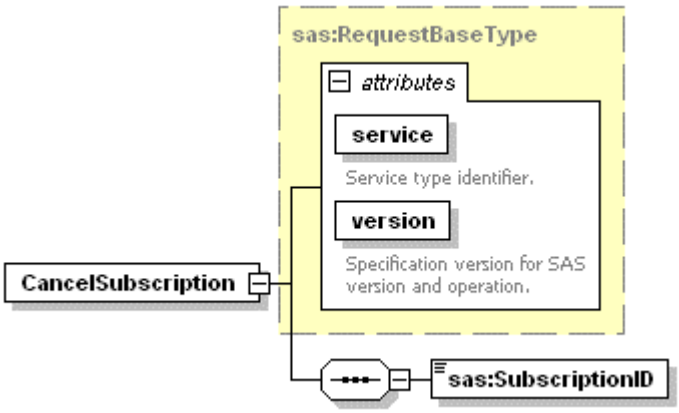


Figure 35: CancelSubscription in XMLSpy notation

Table 22 — 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)
request	Operation name	Character String type, not empty Value is "CancelSubscription"	One (mandatory)
version	Specification version for operation	Character String type, not empty Value is specified by each 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 22 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).

16.2.2 CancelSubscription request KVP encoding

KVP encoding not supported!

16.2.3 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 (sasSubscription.xsd).

EXAMPLE An example CancelSubscription operation request XML encoded for HTTP POST is:

```
<?xml version="1.0" encoding="UTF-8"?>
<CancelSubscription service="SAS" version="0.0.1" xmlns="http://www.opengis.net/sas"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xsi:schemaLocation="http://www.opengis.net/sas
http://mars.uni-muenster.de/swerep\trunk\SAS\0.0.1\sasSubscribe.xsd">
    <SubscriptionID>sub-1</SubscriptionID>
</CancelSubscription>
```

16.3 CancelSubscription operation response

16.3.1 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 23. This table also specifies the UML model data type plus the multiplicity and use of each listed part.

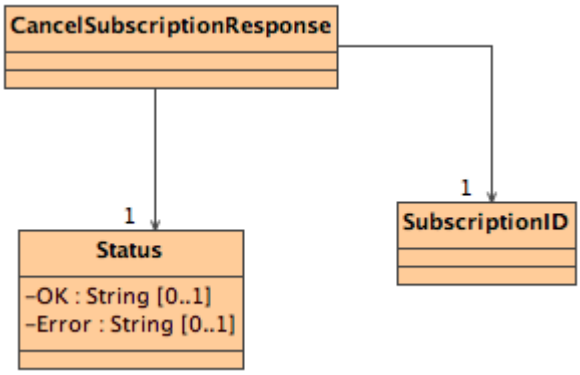


Figure 36: CancelSubscriptionResponse in UML notation

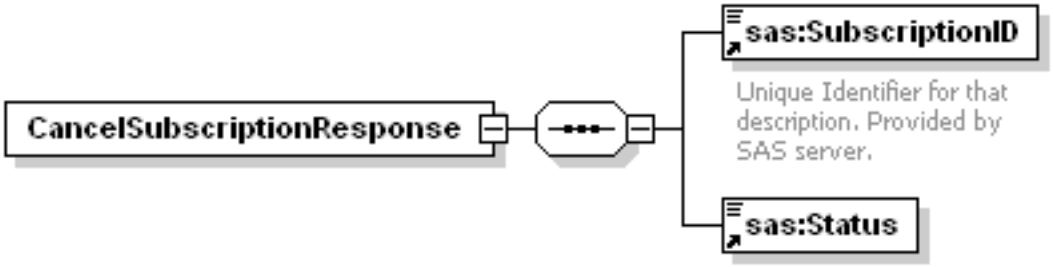


Figure 37: CancelSubscriptionResponse in XMLSpy notation

Table 23 — 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 his superficial.	ID	One (mandatory)
Status	Indicates success of the operation	String, value is “OK” or “error”	One (mandatory)

16.3.2 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).

16.3.3 CancelSubscription response example

A CancelSubscription operation response for SAS can look like this encoded in XML:

```
<?xml version="1.0" encoding="UTF-8"?>
<CancelSubscriptionResponse xmlns="http://www.opengis.net/sas"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xsi:schemaLocation="http://www.opengis.net/sas
http://mars.uni-muenster.de/swerep\trunk\SAS\0.0.1\sasSubscribe.xsd">
    <SubscriptionID>sub-1</SubscriptionID>
    <Status>OK</CancellationStatus>
</CancelSubscriptionResponse>
```

Listing 12: Example CancelSubscription response

16.3.4 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 24. For each listed exceptionCode, the contents of the "locator" parameter value shall be as specified in the right 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 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

17 Alerting via WNS

This chapter describes sending alerts to clients via WNS. DoNotification is the only operation a SAS has to perform when sending an alert. As the SAS is not designed to receive feedback from the client the DoCommunication operation is not used. The information provided by the client when subscribing for an alert (WNSUserID and WNSURL) are sufficient for a SAS to alert the client via WNS. Figure 38 shows the structure of a DoNotification request.

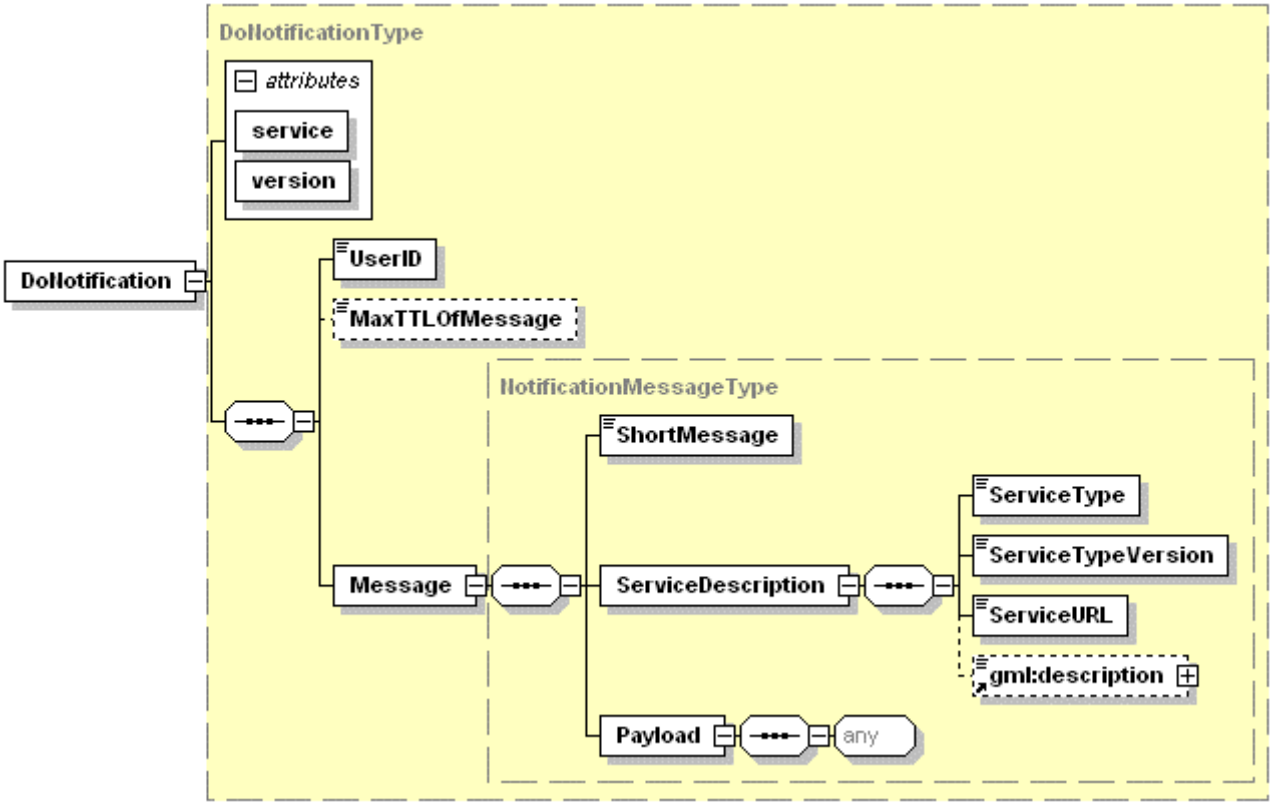


Figure 38: DoNotification request in XMLSpy notation

Such a request will be sent by a SAS to the service determined by the WNSURL using the WNSUserID for the UserID element. The optional MaxTTLOfMessage is used by the WNS as a hint to determine how long an incoming message should be cached if it cannot be delivered at once. The SAS should not make use of this element because it shall only filter incoming sensor readings for the client and not make decisions of whether the information will get outdated in the future or not.

The Message element contains information used by the WNS when forwarding a notification to the client. The ServiceDescription contains information needed for identifying and parsing incoming messages on the client side. ServiceType and ServiceTypeVersion ('SAS' and '1.0.0' for this specification) are used by the client to determine how to parse the message. The ServiceURL contains the URL of the service sending the request. It is used by the client when differentiating incoming messages from different services with same ServiceType and ServiceTypeVersion. The optional

gml:description may be used for providing additional information concerning the message.

The Payload element holds the real message content. For a SAS the structure of this content is defined by the SASMessage element (see Figure 39).

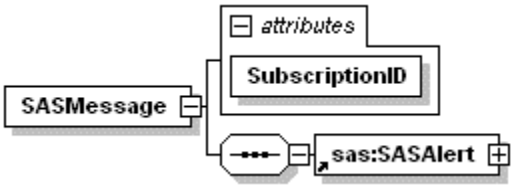


Figure 39: SASMessage in XMLSpy notation

The SASMessage simply contains a SASAlert (see chapter 7) which would otherwise be issued via XMPP. The SubscriptionID is used by clients who are using the same WNS account for receiving alerts from different subscriptions.

Listing 17-1: exemplary DoNotification request

```
<?xml version="1.0" encoding="UTF-8"?>
<DoNotification xmlns="http://www.opengis.net/wns"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://www.opengis.net/wns
../../../wns/0.1.1/wns.xsd" service="WNS" version="0.1.1">
  <UserID>wns100</UserID>
  <Message>
     <ShortMessage>Sub103 alert generated</ShortMessage>
     <ServiceDescription>
        <ServiceType>SAS</ServiceType>
        <ServiceTypeVersion>1.0.0</ServiceTypeVersion>
        <ServiceURL>http://52north.org/sas</ServiceURL>
     </ServiceDescription>
     <Payload>
        <SASMessage xmlns=http://www.opengis.net/sas
SubscriptionID="Sub103">
           <SASAlert>
              <Header>
                 <AlertMessageStructure>
                    <CategoryProperty>
                       <Content
definition="urn:ifgi:security:building"/>
                    </CategoryProperty>
                 </AlertMessageStructure>
              </Header>
              <Body>51.96 7.607 70.0 2006-09-04T02:27:04Z 0
window_broken</Body>
           </SASAlert>
        </SASMessage>
     </Payload>
  </Message>
</DoNotification>
```

The service sending a DoNotification request does not know which channel(s) the client has registered for being notified by WNS. It may be a channel like SMS or phone with

constraints concerning the length and format of a message. Sending XML via such a channel does not make sense. Thus a ShortMessage must be provided by the caller which contains a short description of the message content in human-readable form. This ShortMessage should not contain more than 100 tokens (this constraint is set by WNS!). For SAS an exemplary ShortMessage might be: "Subscription103 alert generated".

When notifiying via SMS or phone the WNS will cache the message and provide a specific ID with which the client can retrieve the complete message later on via a GetMessage request (as long as the WNS caches the message). In this case the SMS (or phone call) would contain a message in the following format: *<messageID> <wnsURL> <ServiceType> <ShortMessage>*.

An SMS generated for the DoNotification request of Listing 17-1 might thus contain the following text: "*M40726 http://52north.org/wns SAS Sub103 alert generated*".

However, the complete message will be sent by WNS to the client in a NotificationMessage. The structure of a NotificationMessage is depicted in Figure 40 while Listing 17-2 presents an exemplary message a SAS client might receive from WNS.

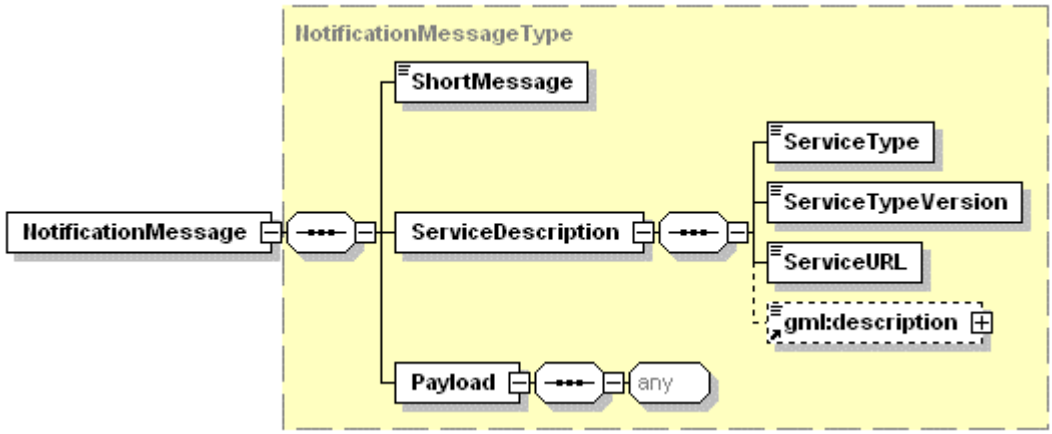


Figure 40: NotificationMessage in XMLSpy notation

Listing 17-2: exemplary NotificationMessage

```
<?xml version="1.0" encoding="UTF-8"?>
<NotificationMessage xmlns="http://www.opengis.net/wns"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://www.opengis.net/wns
../../../wns/0.1.1/wnsMessage.xsd">
  <ShortMessage>Sub103 alert generated</ShortMessage>
  <ServiceDescription>
     <ServiceType>SAS</ServiceType>
     <ServiceTypeVersion>1.0.0</ServiceTypeVersion>
     <ServiceURL>http://52north.org/sas</ServiceURL>
  </ServiceDescription>
  <Payload>
     <SASMessage xmlns="http://www.opengis.net/sas"
SubscriptionID="Sub103">
```

```
        <SASAlert>
           <Header>
              <AlertMessageStructure>
                 <CategoryProperty>
                    <Content definition="urn:ifgi:security:building"/>
                 </CategoryProperty>
              </AlertMessageStructure>
           </Header>
           <Body>51.96 7.607 70.0 2006-09-04T02:27:04Z 0
window_broken</Body>
        </SASAlert>
     </SASMessage>
  </Payload>
</NotificationMessage>
```

18 Running Example

For a better understanding of the SAS and its functionality we will develop a running example. This chapter will show you how sensors can be registered at the SAS via the Advertise operation, how a client can subscribe for alerts and how these alerts will be sent to the client.

18.1 Registering, renewing and cancelling advertisements

The first step to get information about a SAS instance would be to request its Capabilities, especially the Contents section of its Capabilities. The Contents section provides information about the sensors offered by the specific SAS instance: their specific ID which can be used for identifying a sensor in a Subscribe request, their operational area, the phenomenon or phenomena measured, the structure these measurements will be provided in, the name of the observed feature and the approximate frequency measurements are produced. Additionally each SAS instance declares whether it accepts new advertisements or not. Listing 18-1 shows an exemplary GetCapabilities request.

Listing 18-1: GetCapabilities request instance for retrieving only the Contents section

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

Initially the Contents will not contain any SubscriptionOfferings because no sensors have been advertised (see Listing 18-2).

Listing 18-2: Capabilities instance without SubscriptionOfferings

```
<?xml version="1.0" encoding="UTF-8"?>
<Capabilities xmlns="http://www.opengis.net/sas"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://www.opengis.net/sas ../sasCapabilities.xsd"
version="1.0.0">
  <Contents>
     <AcceptAdvertisements>true</AcceptAdvertisements>
  </Contents>
</Capabilities>
```

As we can see the SAS accepts new advertisements. So now SAS clients – sensor operators or even the sensors themselves – can send Advertise requests to the SAS for registering alert messages. In this example we will register three different sensors.

At first a sensor measuring temperature will be registered. The Advertise request is shown in Listing 18-3.

Listing 18-3: Advertise request instance for registering a temperature sensor

```
<?xml version="1.0" encoding="UTF-8"?>
<Advertise xmlns="http://www.opengis.net/sas"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://www.opengis.net/sas ../sasAll.xsd"
xmlns:swe="http://www.opengis.net/swe" service="SAS" version="1.0.0">
  <FeatureOfInterest>
     <Name>Muenster</Name>
     <Description>The temperature is measured by a sensor which is
mounted at a weather station located on the roof of the IFGI in
Muenster</Description>
  </FeatureOfInterest>
  <OperationArea>
     <swe:GeoLocation>
        <swe:longitude>
           <swe:Quantity>51.96</swe:Quantity>
        </swe:longitude>
        <swe:latitude>
           <swe:Quantity>7.607</swe:Quantity>
        </swe:latitude>
        <swe:altitude>
           <swe:Quantity>78</swe:Quantity>
        </swe:altitude>
     </swe:GeoLocation>
  </OperationArea>
  <AlertMessageStructure>
     <QuantityProperty>
        <Content definition="urn:ogc:temperature" uom="degree Celsius"
min="0" max="40"/>
     </QuantityProperty>
  </AlertMessageStructure>
  <AlertFrequency>0.1</AlertFrequency>
  <DesiredPublicationExpiration>
       2007-10-30T00:00:00Z
    </DesiredPublicationExpiration>
</Advertise>
```

The request contains - besides the FeatureOfInterest with which the measurements provided by the sensor can be associated – the location of the sensor plus information about the phenomenon being measured (urn:ogc:temperature in degree celsius), the structure of alert messages and the frequency new measurements are supposed to be generated by the sensor (one alert every ten seconds). These pieces of information will be added to the SAS's Capabilities document if the request is accepted. The DesiredPublicationExpiration states that the sensor is supposed to stop publishing alerts after 2007-10-30T00:00:00Z.

If the SAS accepts the request a response as shown in Listing 18-4 will be issued.

Listing 18-4: AdvertiseResponse instance

```
<?xml version="1.0" encoding="UTF-8"?>
<AdvertiseResponse xmlns="http://www.opengis.net/sas"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://www.opengis.net/sas ../sasAll.xsd"
expires="2007-01-01T00:00:00Z">
  <PublicationID>Pub301</PublicationID>
  <XMPPURI>xmpp://52North.org/sas/publications/c301</XMPPURI>
</AdvertiseResponse>
```

The response contains a URI, which identifies the MUC where the sensor can publish alerts via XMPP. In addition the advertisement will automatically expire at 2007-01-01T00:00:00Z. As we can see this date is way before the desired publication expiration provided in the request. This is because the SAS must be able to determine at which point in time MUCs can be closed automatically. Otherwise there could exist unused MUCs where no alerts are published anymore, e.g. if the sensor associated with that MUC runs out of battery or gets destroyed before the desired publication expiration.

Therefore the advertisement identified via the given PublicationID must be renewed before it expires. This can be done via a RenewAdvertisement request (see Listing 18-5).

Listing 18-5: RenewAdvertisement instance

```
<?xml version="1.0" encoding="UTF-8"?>
<RenewAdvertisement xmlns="http://www.opengis.net/sas"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://www.opengis.net/sas ../sasAll.xsd"
service="SAS" version="1.0.0">
  <PublicationID>Pub301</PublicationID>
  <DesiredPublicationExpiration>2007-10-
30T00:00:00Z</DesiredPublicationExpiration>
</RenewAdvertisement>
```

The request contains the ID of the advertisement which shall be renewed and the DesiredPublicationExpiration. In this case it is the same as in the Advertise request, but it might as well be a new date – e.g. depending on new calculations concerning battery depletion.

If the request is accepted the SAS should provide a new expiration date in the response (see Listing 18-6). This date should not be (too) near to the old expiration date so that the sensor does not have to renew its advertisement too frequently.

Listing 18-6: RenewAdvertisementResponse instance

```
<?xml version="1.0" encoding="UTF-8"?>
<RenewAdvertisementResponse xmlns="http://www.opengis.net/sas"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://www.opengis.net/sas ../sasAll.xsd"
expires="2007-06-01T00:00:00Z">
  <PublicationID>Pub301</PublicationID>
  <renewalStatus>confirmed</renewalStatus>
</RenewAdvertisementResponse>
```

In addition to the temperature sensor two other sensors are registered at the SAS, publishing humidity measurements on the one hand (see Listing 18-7) and readings from a building security system on the other hand (see Listing 18-8).

Listing 18-7: Advertise request instance for a sensor publishing humidity readings

```
<?xml version="1.0" encoding="UTF-8"?>
<Advertise xmlns="http://www.opengis.net/sas"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://www.opengis.net/sas ../sasAll.xsd"
xmlns:swe="http://www.opengis.net/swe" service="SAS" version="1.0.0">
  <FeatureOfInterest>
     <Name>Muenster</Name>
     <Description>humidity is measured by a sensor which is mounted at a
weather station located on the roof of the IFGI in
Muenster</Description>
  </FeatureOfInterest>
  <OperationArea>
     <swe:GeoLocation>
        <swe:longitude>
           <swe:Quantity>51.96</swe:Quantity>
        </swe:longitude>
        <swe:latitude>
           <swe:Quantity>7.607</swe:Quantity>
        </swe:latitude>
        <swe:altitude>
           <swe:Quantity>77.8</swe:Quantity>
        </swe:altitude>
     </swe:GeoLocation>
  </OperationArea>
  <AlertMessageStructure>
     <QuantityProperty>
        <Content definition="urn:ogc:humidity" uom="percent" min="0"
max="99"/>
     </QuantityProperty>
  </AlertMessageStructure>
  <AlertFrequency>2</AlertFrequency>
  <DesiredPublicationExpiration>2008-04-
01T00:00:00Z</DesiredPublicationExpiration>
</Advertise>
```

Listing 18-8: Advertise request instance for a sensor publishing readings of a building security system

```
<?xml version="1.0" encoding="UTF-8"?>
<Advertise xmlns="http://www.opengis.net/sas"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://www.opengis.net/sas ../sasAll.xsd"
xmlns:swe="http://www.opengis.net/swe" service="SAS" version="1.0.0">
  <FeatureOfInterest>
     <Name>IFGI</Name>
     <Description>The alerts are triggered if an event occures in the
building Robert-Koch Strasse 26-28</Description>
  </FeatureOfInterest>
  <OperationArea>
     <swe:GeoLocationArea>
        <swe:Envelop>
           <swe:lowerCorner>
              <swe:GeoLocation>
                 <swe:longitude>
                    <swe:Quantity>51.9599</swe:Quantity>
                 </swe:longitude>
                 <swe:latitude>
                    <swe:Quantity>7.6069</swe:Quantity>
                 </swe:latitude>
              </swe:GeoLocation>
           </swe:lowerCorner>
           <swe:upperCorner>
              <swe:GeoLocation>
                 <swe:longitude>
                    <swe:Quantity>51.9604</swe:Quantity>
                 </swe:longitude>
                 <swe:latitude>
                    <swe:Quantity>7.6076</swe:Quantity>
                 </swe:latitude>
              </swe:GeoLocation>
           </swe:upperCorner>
        </swe:Envelop>
     </swe:GeoLocationArea>
  </OperationArea>
  <AlertMessageStructure>
     <CategoryProperty>
        <Content definition="urn:ifgi:security:building"/>
     </CategoryProperty>
  </AlertMessageStructure>
  <AlertFrequency>1</AlertFrequency>
  <DesiredPublicationExpiration>2007-06-
30T13:00:00Z</DesiredPublicationExpiration>
</Advertise>
```

Of course an advertisement can also be cancelled. An exemplary request is shown in Listing 18-9 and the corresponding response in Listing 18-10.

Listing 18-9: CancelAdvertisement request instance

```
<?xml version="1.0" encoding="UTF-8"?>
<CancelAdvertisement xmlns="http://www.opengis.net/sas"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://www.opengis.net/sas ../sasAll.xsd"
service="SAS" version="1.0.0">
  <PublicationID>Pub301</PublicationID>
</CancelAdvertisement>
```

Listing 18-10: CancelAdvertisementResponse instance

```
<?xml version="1.0" encoding="UTF-8"?>
<CancelAdvertisementResponse xmlns="http://www.opengis.net/sas"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://www.opengis.net/sas ../sasAll.xsd">
  <PublicationID>Pub301</PublicationID>
  <CancellationStatus>confirmed</CancellationStatus>
</CancelAdvertisementResponse>
```

18.2 Subscribing, renewing and cancelling subscriptions

After accepting all advertisements the SAS contains three SubscriptionOfferings in its Capabilities (see Listing 18-11).

Listing 18-11: Capabilities instance containing three SubscriptionOfferings

```
<?xml version="1.0" encoding="UTF-8"?>
<Capabilities xmlns="http://www.opengis.net/sas"
xmlns:swe="http://www.opengis.net/swe"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://www.opengis.net/sas ../sasAll.xsd"
version="1.0.0">
  <Contents>
     <AcceptAdvertisements>true</AcceptAdvertisements>
     <SubscriptionOfferingList>
        <SubscriptionOffering>
           <SubscriptionOfferingID>1</SubscriptionOfferingID>
           <AlertMessageStructure>
              <QuantityProperty>
                 <Content definition="urn:ogc:temperature" uom="degree
Celsius" min="0" max="40"/>
              </QuantityProperty>
           </AlertMessageStructure>
           <FeatureOfInterest>
              <Name>Muenster</Name>
              <Description>The temperature is measured by a sensor which
is mounted at a weather station located on the roof of the IFGI in
Muenster</Description>
           </FeatureOfInterest>
           <OperationArea>
              <swe:GeoLocation>
                 <swe:longitude>
                    <swe:Quantity>51.96</swe:Quantity>
                 </swe:longitude>
```

```
                  <swe:latitude>
                     <swe:Quantity>7.607</swe:Quantity>
                  </swe:latitude>
                  <swe:altitude>
                     <swe:Quantity>78</swe:Quantity>
                  </swe:altitude>
               </swe:GeoLocation>
            </OperationArea>
            <AlertFrequency>0.1</AlertFrequency>
         </SubscriptionOffering>
         <SubscriptionOffering>
            <SubscriptionOfferingID>2</SubscriptionOfferingID>
            <AlertMessageStructure>
               <QuantityProperty>
                  <Content definition="urn:ogc:humidity" uom="percent"
min="0" max="99"/>
               </QuantityProperty>
            </AlertMessageStructure>
            <FeatureOfInterest>
               <Name>Muenster</Name>
               <Description>humidity is measured by a sensor which is
mounted at a weather station located on the roof of the IFGI in
Muenster</Description>
            </FeatureOfInterest>
            <OperationArea>
               <swe:GeoLocation>
                  <swe:longitude>
                     <swe:Quantity>51.96</swe:Quantity>
                  </swe:longitude>
                  <swe:latitude>
                     <swe:Quantity>7.607</swe:Quantity>
                  </swe:latitude>
                  <swe:altitude>
                     <swe:Quantity>77.8</swe:Quantity>
                  </swe:altitude>
               </swe:GeoLocation>
            </OperationArea>
            <AlertFrequency>2</AlertFrequency>
         </SubscriptionOffering>
         <SubscriptionOffering>
            <SubscriptionOfferingID>3</SubscriptionOfferingID>
            <AlertMessageStructure>
               <CategoryProperty>
                  <Content definition="urn:ifgi:security:building"/>
               </CategoryProperty>
            </AlertMessageStructure>
            <FeatureOfInterest>
               <Name>IFGI</Name>
               <Description>The alerts are triggered if an event occures
in the building Robert-Koch Strasse 26-28</Description>
            </FeatureOfInterest>
            <OperationArea>
               <swe:GeoLocationArea>
                  <swe:Envelop>
                     <swe:lowerCorner>
                        <swe:GeoLocation>
                           <swe:longitude>
                              <swe:Quantity>51.9599</swe:Quantity>
                           </swe:longitude>
                           <swe:latitude>
                              <swe:Quantity>7.6069</swe:Quantity>
                           </swe:latitude>
                        </swe:GeoLocation>
                     </swe:lowerCorner>
                     <swe:upperCorner>
```

```
                        <swe:GeoLocation>
                           <swe:longitude>
                              <swe:Quantity>51.9604</swe:Quantity>
                           </swe:longitude>
                           <swe:latitude>
                              <swe:Quantity>7.6076</swe:Quantity>
                           </swe:latitude>
                        </swe:GeoLocation>
                     </swe:upperCorner>
                  </swe:Envelop>
               </swe:GeoLocationArea>
            </OperationArea>
            <AlertFrequency>1</AlertFrequency>
         </SubscriptionOffering>
      </SubscriptionOfferingList>
   </Contents>
</Capabilities>
```

Clients can now subscribe for alerts. We will provide an exemplary Subscribe request for each SubscriptionOffering showing different semantics of the Subscribe request and its response.

Listing 18-12: Subscribe request instance (1)

```
<?xml version="1.0" encoding="UTF-8"?>
<Subscribe xmlns="http://www.opengis.net/sas"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://www.opengis.net/sas ../sasAll.xsd"
service="SAS" version="1.0.0">
   <ResultFilter ObservedPropertyDefinition="urn:ogc:temperature"
uom="degree Celsius">
      <isGreaterThan>27.5</isGreaterThan>
   </ResultFilter>
   <FeatureOfInterestName>Muenster</FeatureOfInterestName>
</Subscribe>
```

Listing 18-12 shows a Subscribe request where the client is interested in all temperature readings in degree Celsius for the feature ‘Muenster’ that are greater than 27.5. No other conditions have to be fulfilled and no target for receiving alerts has been provided, thus the SAS delivers the URI for a XMPP MUC where all alerts which fulfil the conditions set in the request are being sent to (see Listing 18-13).

Listing 18-13: SubscribeResponse instance (1)

```
<?xml version="1.0" encoding="UTF-8"?>
<SubscribeResponse xmlns="http://www.opengis.net/sas"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://www.opengis.net/sas ../sasAll.xsd"
SubscriptionID="Sub101" expires="2007-01-01T00:00:00Z">
   <XMPPResponse>
      <XMPPURI>xmpp://52North.org/sas/publications/c1</XMPPURI>
   </XMPPResponse>
```

```
</SubscribeResponse>
```

The client must register with that MUC to receive the alerts for his / her subscription identified via the ID (Sub101) provided in the response. An exemplary alert message received via that MUC and provided by the sensor associated with SubscriptionOffering '2' might look like the one in Listing 18-14.

Listing 18-14: exemplary alert published for subscription '2'

```
<?xml version="1.0" encoding="UTF-8"?>
<SASAlert xmlns="http://www.opengis.net/sas"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
  <Header>
     <AlertMessageStructure>
        <CategoryProperty>
           <Content definition="urn:ifgi:security:building"/>
        </CategoryProperty>
     </AlertMessageStructure>
  </Header>
  <Body>51.96 7.607 70.0 2006-09-04T02:27:04Z 0 window_broken</Body>
</SASAlert>
```

The SubscribeResponse also contains a date which shows when the subscription will expire (2007-01-01T00:00:00Z). This expiration date serves the same purpose as that provided for advertisements: the SAS will automatically remove all subscriptions after they expired - thus saving resources - unless those subscriptions are being renewed by the client. We will show how to renew a subscription later on.

Now suppose another Subscribe request like the one shown in Listing 18-15 is issued.

Listing 18-15: Subscribe request instance (2)

```
<?xml version="1.0" encoding="UTF-8"?>
<Subscribe xmlns="http://www.opengis.net/sas"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://www.opengis.net/sas ../sasAll.xsd"
service="SAS" version="1.0.0">
  <ResultFilter ObservedPropertyDefinition="urn:ogc:humidity"
uom="percent">
     <isGreaterThan>80.0</isGreaterThan>
  </ResultFilter>
  <FeatureOfInterestName>Muenster</FeatureOfInterestName>
  <ResultRecipient>
     <XMPPURI>xmpp://foo.bar.org/52NorthSasClient/c3</XMPPURI>
  </ResultRecipient>
</Subscribe>
```

Here the client is interested in all humidity readings associated to the feature ‘Muenster’ that are greater than 80. Additionally a URI to a MUC is provided where the SAS should publish all alerts for the corresponding subscription.

The response (see Listing 18-16) indicates that the SAS accepted the subscription and successfully connected to the MUC provided by the client.

Listing 18-16: SubscribeResponse instance (2)

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

<SubscribeResponse xmlns="http://www.opengis.net/sas"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://www.opengis.net/sas ../sasAll.xsd"
SubscriptionID="Sub102" expires="2007-01-01T00:00:00Z">
  <Status>OK</Status>
</SubscribeResponse>
```

The last exemplary Subscribe request (see Listing 18-17) requests sensor readings from a specific SubscriptionOffering.

Listing 18-17: Subscribe request instance (3)

```
<?xml version="1.0" encoding="UTF-8"?>
<Subscribe xmlns="http://www.opengis.net/sas"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://www.opengis.net/sas ../sasAll.xsd"
service="SAS" version="1.0.0">
  <SubscriptionOfferingID>3</SubscriptionOfferingID>
  <ResultFilter ObservedPropertyDefinition="urn:ifgi:security:building"
uom="urn:ogc:def:unit:none">
     <isNotEqualTo>status ok</isNotEqualTo>
  </ResultFilter>
  <ResultRecipient>
     <WNS>
        <WNSURL>http://52north.org/wns</WNSURL>
        <WNSUserID>wns100</WNSUserID>
     </WNS>
  </ResultRecipient>
</Subscribe>
```

The client is interested in every alert generated by SubscriptionOffering ‘3’ which contains a value that is not equal to ‘status ok’. As the SubscriptionOffering did not define a uom attribute the ResultFilter uses ‘urn:ogc:def:unit:none’ for the uom attribute which means that the client does not care about the uom used. All alerts that fulfil that condition shall be sent to the WNS user ‘wns100’ registered at WNS ‘http://52north.org/wns’.

The response (see Listing 18-18) indicates that the SAS accepted the new subscription.

Listing 18-18: SubscribeResponse instance (3)

```
<?xml version="1.0" encoding="UTF-8"?>
<SubscribeResponse xmlns="http://www.opengis.net/sas"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://www.opengis.net/sas ../sasAll.xsd"
SubscriptionID="Sub103" expires="2007-01-01T00:00:00Z">
  <Status>OK</Status>
</SubscribeResponse>
```

For a detailed description on how alerts are sent via WNS, see previous chapter. However, an exemplary NotificationMessage for subscription 'Sub103' which contains a SASMessage for the client might look like the one in Listing 18-19.

Listing 18-19: NotificationMessage from WNS containing a SASMessage

```
<?xml version="1.0" encoding="UTF-8"?>
<NotificationMessage xmlns="http://www.opengis.net/wns"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://www.opengis.net/wns
../../../wns/0.1.1/wnsMessage.xsd">
  <ShortMessage>Sub103 alert generated</ShortMessage>
  <ServiceDescription>
     <ServiceType>SAS</ServiceType>
     <ServiceTypeVersion>1.0.0</ServiceTypeVersion>
     <ServiceURL>http://52north.org/sas</ServiceURL>
  </ServiceDescription>
  <Payload>
     <SASMessage xmlns="http://www.opengis.net/sas"
SubscriptionID="Sub103">
        <SASAlert>
           <Header>
              <AlertMessageStructure>
                 <CategoryProperty>
                    <Content definition="urn:ifgi:security:building"/>
                 </CategoryProperty>
              </AlertMessageStructure>
           </Header>
           <Body>51.96 7.607 70.0 2006-09-04T02:27:04Z 0
window_broken</Body>
        </SASAlert>
     </SASMessage>
  </Payload>
</NotificationMessage>
```

As was said before a subscription is automatically removed by a SAS when the expiration date for that subscription has been reached. For renewing a subscription the client can issue a RenewSubscription request. An exemplary request for renewing subscription 'Sub103' (see listing above) is shown in Listing 18-20.

Listing 18-20: RenewSubscription request instance

```
<?xml version="1.0" encoding="UTF-8"?>
<RenewSubscription xmlns="http://www.opengis.net/sas"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://www.opengis.net/sas ../sasAll.xsd"
service="SAS" version="1.0.0">
  <SubscriptionID>Sub103</SubscriptionID>
</RenewSubscription>
```

If the SAS accepts the request it provides a new expiration date (see Listing 18-21).

Listing 18-21: RenewSubscriptionResponse instance

```
<?xml version="1.0" encoding="UTF-8"?>
<RenewSubscriptionResponse xmlns="http://www.opengis.net/sas"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://www.opengis.net/sas ../sasAll.xsd"
expires="2007-06-01T00:00:00Z">
  <SubscriptionID>Sub103</SubscriptionID>
  <Status>OK</Status>
</RenewSubscriptionResponse>
```

A subscription can also be cancelled (see Listing 18-22 and Listing 18-22).

Listing 18-22: CancelSubscription request instance

```
<?xml version="1.0" encoding="UTF-8"?>
<CancelSubscription xmlns="http://www.opengis.net/sas"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://www.opengis.net/sas ../sasAll.xsd"
service="SAS" version="1.0.0">
  <SubscriptionID>Sub103</SubscriptionID>
</CancelSubscription>
```

Listing 18-23: CancelSubscriptionResponse instance

```
<?xml version="1.0" encoding="UTF-8"?>
<CancelSubscriptionResponse xmlns="http://www.opengis.net/sas"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://www.opengis.net/sas ../sasAll.xsd">
  <SubscriptionID>Sub103</SubscriptionID>
  <Status>OK</Status>
</CancelSubscriptionResponse>
```

Annex A
(normative)

Abstract test suite

A.1 General

TBD

In each Implementation Specification document, Annex A shall specify the Abstract Test Suite, as specified in Clause 9 and Annex A of ISO 19105. That Clause and Annex specify the ISO/TC 211 requirements for Abstract Test Suites. Examples of Abstract Test Suites are available in an annex of most ISO 191XX documents, one of the more useful is in ISO 191TBD. Note that this guidance may be more abstract than needed in an OpenGIS® Implementation Specification.

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/0.30.0. 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].

gml4sas.xsd

```
<?xml version="1.0" encoding="UTF-8"?>
<xs:schema xmlns:gml="http://www.opengis.net/gml"
xmlns:xs="http://www.w3.org/2001/XMLSchema"
targetNamespace="http://www.opengis.net/gml"
elementFormDefault="qualified" attributeFormDefault="unqualified"
version="1.0.0">
    <xs:annotation>
        <xs:documentation>
     gml4sos.xsd
     Utility schema which simply includes the GML schema documents
required for the SOS schema</xs:documentation>
    </xs:annotation>
    <!-- ==============================================================
        includes and imports
    ============================================================== -->
    <xs:include
schemaLocation="http://schemas.opengis.net/gml/3.1.1/base/geometryAggre
gates.xsd"/>
    <xs:include
schemaLocation="http://schemas.opengis.net/gml/3.1.1/base/feature.xsd"/
>
</xs:schema>
```

ogc4sas.xsd

```
<?xml version="1.0" encoding="UTF-8"?>
<xs:schema xmlns:ogc="http://www.opengis.net/ogc"
xmlns:xs="http://www.w3.org/2001/XMLSchema"
targetNamespace="http://www.opengis.net/ogc"
elementFormDefault="qualified" attributeFormDefault="unqualified"
version="1.0.0">
   <xs:annotation>
      <xs:documentation>
   ogc4sas.xsd
    Utility schema which simply includes the OGC schema documents
required for the OGC SAS schemas</xs:documentation>
   </xs:annotation>
   <!-- ================================================================
       includes and imports
   ================================================================ -->
   <xs:include schemaLocation="../../filter/1.1.30/filter.xsd"/>
</xs:schema>
```

ows4sas.xsd

```
<?xml version="1.0" encoding="UTF-8"?>
<xs:schema xmlns:ows="http://www.opengeospatial.net/ows"
xmlns:xs="http://www.w3.org/2001/XMLSchema"
targetNamespace="http://www.opengeospatial.net/ows"
elementFormDefault="qualified" attributeFormDefault="unqualified"
version="1.0.0">
   <xs:annotation>
      <xs:documentation>
   ows4sas.xsd
    Utility schema which simply includes the OWS schema documents
required for the OGC SAS schemas</xs:documentation>
   </xs:annotation>
   <!-- ================================================================
       includes and imports
   ================================================================= -->
   <xs:include
schemaLocation="../../ows/1.0.30/owsGetCapabilities.xsd"/>
   <xs:include
schemaLocation="../../ows/1.0.30/owsExceptionReport.xsd"/>
</xs:schema>
```

sasAdvertise.xsd

```
<?xml version="1.0" encoding="UTF-8"?>
<!-- edited with XMLSpy v2006 sp2 U (http://www.altova.com) by Ingo
Simonis (ifgi) -->
<schema xmlns="http://www.w3.org/2001/XMLSchema"
xmlns:ogc="http://www.opengis.net/ogc"
xmlns:ows="http://www.opengeospatial.net/ows"
xmlns:sas="http://www.opengis.net/sas"
xmlns:swe="http://www.opengis.net/swe"
targetNamespace="http://www.opengis.net/sas"
elementFormDefault="qualified" attributeFormDefault="unqualified"
version="1.0.0" xml:lang="en">
   <!-- ================================================================
      includes and imports
   ================================================================ -->
   <include schemaLocation="./sasCommon.xsd"/>
   <!-- ================================================================
      request
   ================================================================ -->
   <element name="Advertise">
      <annotation>
         <documentation>Request to a SAS to allow a publisher to
publish alerts.</documentation>
      </annotation>
      <complexType>
         <complexContent>
            <extension base="sas:RequestBaseType">
               <sequence>
                  <element name="FeatureOfInterest"
type="sas:FeatureOfInterestIDType">
                     <annotation>
                        <documentation>Specifies target feature for
which alerts are published. Mostly a helper for in-situ sensors, since
geo-location has to be done on the server side. The supported area
should be listed in the selected offering capabilities.
                        </documentation>
                     </annotation>
                  </element>
                  <element ref="sas:OperationArea"/>
                  <element ref="sas:AlertMessageStructure"/>
                  <element ref="sas:AlertFrequency"/>
                  <element ref="sas:DesiredPublicationExpiration"/>
               </sequence>
            </extension>
         </complexContent>
      </complexType>
   </element>
   <element name="AdvertiseResponse">
      <complexType>
         <sequence>
            <element ref="sas:PublicationID"/>
            <element name="XMPPURI" type="anyURI"/>
         </sequence>
         <attribute name="expires" type="dateTime" use="required"/>
      </complexType>
   </element>
   <element name="CancelAdvertisement">
      <complexType>
```

```
            <complexContent>
               <extension base="sas:RequestBaseType">
                  <sequence>
                     <element ref="sas:PublicationID"/>
                  </sequence>
               </extension>
            </complexContent>
         </complexType>
      </element>
      <element name="CancelAdvertisementResponse">
         <complexType>
            <sequence>
               <element ref="sas:PublicationID"/>
               <element name="CancellationStatus">
                  <simpleType>
                     <restriction base="string">
                        <enumeration value="confirmed"/>
                        <enumeration value="expired"/>
                        <enumeration value="invalid_PublicationID"/>
                     </restriction>
                  </simpleType>
               </element>
            </sequence>
         </complexType>
      </element>
      <element name="RenewAdvertisement">
         <complexType>
            <complexContent>
               <extension base="sas:RequestBaseType">
                  <sequence>
                     <element ref="sas:PublicationID"/>
                     <element ref="sas:DesiredPublicationExpiration"/>
                  </sequence>
               </extension>
            </complexContent>
         </complexType>
      </element>
      <element name="RenewAdvertisementResponse">
         <complexType>
            <sequence>
               <element ref="sas:PublicationID"/>
               <element name="renewalStatus">
                  <annotation>
                     <documentation>Indicator shows if renewal request
was confirmed or rejected.</documentation>
                  </annotation>
                  <simpleType>
                     <restriction base="string">
                        <enumeration value="confirmed"/>
                        <enumeration value="rejected"/>
                     </restriction>
                  </simpleType>
               </element>
            </sequence>
            <attribute name="expires" type="string" use="required"/>
         </complexType>
      </element>
      <!---->
```

```
    <element name="PublicationID" type="ID">
       <annotation>
          <documentation>unique ID, submitted in response to an
advertise-request.</documentation>
       </annotation>
    </element>
    <element name="DesiredPublicationExpiration" type="string">
       <annotation>
          <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.</documentation>
       </annotation>
    </element>
</schema>
```

sasAll.xsd

```
<?xml version="1.0" encoding="UTF-8"?>
<schema xmlns:sas="http://www.opengis.net/sas"
xmlns="http://www.w3.org/2001/XMLSchema"
targetNamespace="http://www.opengis.net/sas"
elementFormDefault="qualified" attributeFormDefault="unqualified"
version="1.0.0" xml:lang="en">
   <annotation>
      <appinfo>sasAll.xsd 2006/05/11</appinfo>
      <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>
      </documentation>
   </annotation>
   <!-- ================================================================
       includes and imports
    ================================================================ -->
   <include schemaLocation="wns4sas.xsd"/>
   <include schemaLocation="sasCapabilities.xsd"/>
   <include schemaLocation="sasAdvertise.xsd"/>
   <include schemaLocation="sasSubscribe.xsd"/>
   <include schemaLocation="sasMessageSchema.xsd"/>
</schema>
```

sasCapabilities.xsd

```
<?xml version="1.0" encoding="UTF-8"?>
<schema xmlns="http://www.w3.org/2001/XMLSchema"
xmlns:ogc="http://www.opengis.net/ogc"
xmlns:ows="http://www.opengeospatial.net/ows"
xmlns:sas="http://www.opengis.net/sas"
targetNamespace="http://www.opengis.net/sas"
elementFormDefault="qualified" attributeFormDefault="unqualified"
version="1.0.0" xml:lang="en">
   <annotation>
      <appinfo>sosGetCapabilities.xsd 2005/05/11</appinfo>
      <documentation>
         <description>This XML Schema encodes the SAS GetCapabilities
operation request and response.</description>
      </documentation>
   </annotation>
   <!-- ============================================================
       includes and imports
    ============================================================= -->
   <include schemaLocation="./sasContents.xsd"/>
   <!-- ============================================================
       elements and types
    ============================================================= -->
   <element name="GetCapabilities">
      <annotation>
         <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. </documentation>
      </annotation>
      <complexType>
         <complexContent>
            <extension base="ows:GetCapabilitiesType">
               <attribute name="service" type="ows:ServiceType"
use="required" fixed="SAS"/>
            </extension>
         </complexContent>
      </complexType>
   </element>
   <!-- ========================================================== -->
   <element name="Capabilities">
      <annotation>
         <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.
</documentation>
      </annotation>
      <complexType>
```

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

sasCommon.xsd

```
<?xml version="1.0" encoding="UTF-8"?>
<!-- edited with XMLSpy v2006 sp2 U (http://www.altova.com) by Ingo
Simonis (ifgi) -->
<schema xmlns="http://www.w3.org/2001/XMLSchema"
xmlns:sas="http://www.opengis.net/sas"
xmlns:swe="http://www.opengis.net/swe"
xmlns:ogc="http://www.opengis.net/ogc"
xmlns:ns1="http://www.opengeospatial.net/ows"
targetNamespace="http://www.opengis.net/sas"
elementFormDefault="qualified" attributeFormDefault="unqualified"
version="1.0.0" xml:lang="en">
   <annotation>
      <appinfo>sasCommon.xsd</appinfo>
      <documentation>
         <description>This XML Schema encodes the elements and types
that are shared by muliple SAS operations.</description>
      </documentation>
   </annotation>
   <!-- ================================================================
      includes and imports
   ================================================================ -->
   <import namespace="http://www.opengis.net/swe"
schemaLocation="../../sweCommon/1.0.30/sweCommon.xsd"/>
   <!-- ================================================================
      elements and types
   ================================================================ -->
   <complexType name="RequestBaseType">
      <annotation>
         <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. </documentation>
      </annotation>
      <attribute name="service" type="string" use="required"
fixed="SAS">
         <annotation>
            <documentation>Service type identifier. </documentation>
         </annotation>
      </attribute>
      <attribute name="version" type="string" use="required"
fixed="1.0.0">
         <annotation>
            <documentation>Specification version for SAS version and
operation.</documentation>
         </annotation>
      </attribute>
   </complexType>
   <!--================================================================
        core elements for advertisement and capabilities

================================================================-->
   <element name="FeatureOfInterestID"
type="sas:FeatureOfInterestIDType">
      <annotation>
         <documentation>Simple String identifying the feature of
interest</documentation>
      </annotation>
```

```
    </element>
    <complexType name="FeatureOfInterestIDType">
        <sequence>
            <element name="Name" type="string"/>
            <element name="Description" type="string"/>
        </sequence>
    </complexType>
    <element name="OperationArea" type="sas:OperationAreaType">
        <annotation>
            <documentation>Area of operation of this sensor/system. Might
be a point, line, or polygon.</documentation>
        </annotation>
    </element>
    <complexType name="OperationAreaType">
        <choice>
            <element ref="swe:GeoLocation"/>
            <element ref="swe:GeoLocationArea"/>
        </choice>
    </complexType>
    <!--Alert message format-->
    <element name="AlertMessageStructure">
        <annotation>
            <documentation>defines the structure of the alert messages
sent from the sensor to SAS and from SAS to client. It holds by default
six elements. They are identified as follows: latitude (EPSG:4326),
longitude (EPSG:4326), elevation, time (time the alert is triggered),
expiration (time the alert expires). The sixth element is mandatory and
defines the observed property. In case that more than one
observedProperty should be reported, alert transmitters are free to add
them according to the definition given herein. </documentation>
        </annotation>
        <complexType>
            <sequence>
                <element ref="sas:ObservedProperty" maxOccurs="unbounded">
                    <annotation>
                        <documentation>defines the observed property. This
property will be evaluated by SAS. To allow SAS to check on that item
properly, the dataType has to be defined.</documentation>
                    </annotation>
                </element>
            </sequence>
        </complexType>
    </element>
    <!--Properties: alert messages elements from the seventh position
onwards-->
    <element name="ObservedProperty" type="sas:ObservedPropertyType"
abstract="true"/>
    <!--specific properties (support different data types). All types
are inherited from swe:fooType by restriction to ensure usage of uom
and definition-->
    <element name="BooleanProperty" abstract="false"
substitutionGroup="sas:ObservedProperty">
        <complexType>
            <complexContent>
                <extension base="sas:ObservedPropertyType">
                    <sequence>
                        <element name="Content"
type="sas:BooleanPropertyType"/>
```



```
                </sequence>
             </extension>
          </complexContent>
       </complexType>
   </element>
   <element name="CategoryProperty" abstract="false"
substitutionGroup="sas:ObservedProperty">
       <complexType>
          <complexContent>
             <extension base="sas:ObservedPropertyType">
                <sequence>
                   <element name="Content"
type="sas:CategoryPropertyType"/>
                </sequence>
             </extension>
          </complexContent>
       </complexType>
       <!--   <complexType>
         <sequence>
            <element name="Content" type="sas:CategoryPropertyType"/>
         </sequence>
      </complexType> -->
   </element>
   <element name="CountProperty" abstract="false"
substitutionGroup="sas:ObservedProperty">
       <complexType>
          <complexContent>
             <extension base="sas:ObservedPropertyType">
                <sequence>
                   <element name="Content">
                      <complexType>
                         <simpleContent>
                            <restriction
base="sas:CountPropertyType">
                               <attribute name="definition"
type="swe:definitionType" use="required"/>
                            </restriction>
                         </simpleContent>
                      </complexType>
                   </element>
                </sequence>
             </extension>
          </complexContent>
       </complexType>
       <!--  <complexType>
         <sequence>
            <element name="Content" type="sas:CountPropertyType"/>
         </sequence>
      </complexType> -->
   </element>
   <element name="QuantityProperty" abstract="false"
substitutionGroup="sas:ObservedProperty">
       <complexType>
          <complexContent>
             <extension base="sas:ObservedPropertyType">
                <sequence>
                   <element name="Content"
type="sas:QuantityPropertyType"/>
```

```
                </sequence>
             </extension>
          </complexContent>
       </complexType>
       <!--  <complexType>
          <sequence>
             <element name="Content" type="sas:QuantityPropertyType"/>
          </sequence>
       </complexType> -->
    </element>
    <!--corresponding types-->
    <complexType name="ObservedPropertyType"/>
    <complexType name="BooleanPropertyType">
       <simpleContent>
          <restriction base="swe:BooleanType">
             <attribute name="definition" type="swe:definitionType"
use="required"/>
          </restriction>
       </simpleContent>
    </complexType>
    <complexType name="CategoryPropertyType">
       <simpleContent>
          <restriction base="swe:CategoryType">
             <attribute name="definition" type="swe:definitionType"
use="required"/>
          </restriction>
       </simpleContent>
    </complexType>
    <complexType name="CountPropertyType">
       <simpleContent>
          <restriction base="swe:CountType">
             <attribute name="definition" type="swe:definitionType"
use="required"/>
             <attribute name="uom" type="swe:uomType" use="required"/>
          </restriction>
       </simpleContent>
    </complexType>
    <complexType name="QuantityPropertyType">
       <simpleContent>
          <restriction base="swe:QuantityType">
             <attribute name="definition" type="swe:definitionType"
use="required"/>
             <attribute name="uom" type="swe:uomType" use="required"/>
          </restriction>
       </simpleContent>
    </complexType>
    <!---->
    <element name="SubscriptionOfferingID" type="token"/>
    <element name="Status">
       <simpleType>
          <restriction base="string">
             <enumeration value="OK"/>
             <enumeration value="error"/>
          </restriction>
       </simpleType>
    </element>
    <element name="XMPPURI" type="anyURI"/>
    <element name="AlertFrequency" type="double">
```

```
      <annotation>
         <documentation>Defines the frequency alerts per
second.</documentation>
      </annotation>
   </element>
</schema>
```


sasContents.xsd

```
<?xml version="1.0" encoding="UTF-8"?>
<!-- edited with XMLSpy v2006 sp2 U (http://www.altova.com) by Ingo
Simonis (ifgi) -->
<schema xmlns="http://www.w3.org/2001/XMLSchema"
xmlns:sas="http://www.opengis.net/sas"
xmlns:ogc="http://www.opengis.net/ogc"
xmlns:ns1="http://www.opengeospatial.net/ows"
targetNamespace="http://www.opengis.net/sas"
elementFormDefault="qualified" attributeFormDefault="unqualified"
version="1.0.0" xml:lang="en">
   <annotation>
      <appinfo>sasContents.xsd</appinfo>
      <documentation>
         <description>This XML Schema encodes the Contents section of
the SAS GetCapabilities operation response.</description>
      </documentation>
   </annotation>
   <!-- ================================================================
      includes and imports
   ================================================================ -->
   <import namespace="http://www.opengeospatial.net/ows"
schemaLocation="./ows4sas.xsd"/>
   <include schemaLocation="./sasCommon.xsd"/>
   <!-- ================================================================
      elements and types
   ================================================================ -->
   <element name="Contents">
      <annotation>
         <documentation/>
      </annotation>
      <complexType>
         <sequence>
            <element name="AcceptAdvertisements" type="boolean"/>
            <element name="SubscriptionOfferingList" minOccurs="0">
               <annotation>
                  <documentation>root element for all
SubscriptionOfferings</documentation>
               </annotation>
               <complexType>
                  <sequence>
                     <element name="SubscriptionOffering"
type="sas:SubscriptionOfferingType" maxOccurs="unbounded"/>
                  </sequence>
               </complexType>
            </element>
         </sequence>
      </complexType>
   </element>
   <complexType name="SubscriptionOfferingType">
      <sequence>
         <element ref="sas:SubscriptionOfferingID"/>
         <element ref="sas:AlertMessageStructure"/>
         <element name="FeatureOfInterest"
type="sas:FeatureOfInterestIDType"/>
         <element ref="sas:OperationArea"/>
         <element ref="sas:AlertFrequency"/>
      </sequence>
```

```
    </complexType>
</schema>
```


sasMessageSchema.xsd

```
<?xml version="1.0" encoding="UTF-8"?>
<!-- edited with XMLSpy v2006 sp2 U (http://www.altova.com) by Ingo
Simonis (ifgi) -->
<xs:schema xmlns:sas="http://www.opengis.net/sas"
xmlns:xs="http://www.w3.org/2001/XMLSchema"
targetNamespace="http://www.opengis.net/sas"
elementFormDefault="qualified" attributeFormDefault="unqualified">
   <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 WNS. The SubscriptionID is used
for associating SASAlerts received via WNS. A client can use one WNS
account for receiving SASAlerts for various
subscriptions.</xs:documentation>
      </xs:annotation>
      <xs:complexType>
         <xs:sequence>
            <xs:element ref="sas:SASAlert"/>
         </xs:sequence>
         <xs:attribute name="SubscriptionID" use="required"/>
      </xs:complexType>
   </xs:element>
   <!---->
   <xs:element name="SASAlert">
      <xs:annotation>
         <xs:documentation>Defines the alert which will be issued by
the SAS. A client will receive SASAlerts via the channel he subscribed
for.</xs:documentation>
      </xs:annotation>
      <xs:complexType>
         <xs:sequence>
            <xs:element name="Header">
               <xs:annotation>
                  <xs:documentation>defines the message
structure</xs:documentation>
               </xs:annotation>
               <xs:complexType>
                  <xs:sequence>
                     <xs:element ref="sas:AlertMessageStructure"/>
                  </xs:sequence>
               </xs:complexType>
            </xs:element>
            <xs:element name="Body" type="xs:string">
               <xs:annotation>
                  <xs:documentation>contains the alert
string</xs:documentation>
               </xs:annotation>
            </xs:element>
         </xs:sequence>
      </xs:complexType>
   </xs:element>
</xs:schema>
```

sasSubscribe.xsd

```
<?xml version="1.0" encoding="UTF-8"?>
<!-- edited with XMLSpy v2006 sp2 U (http://www.altova.com) by Ingo
Simonis (ifgi) -->
<schema xmlns="http://www.w3.org/2001/XMLSchema"
xmlns:ogc="http://www.opengis.net/ogc"
xmlns:sas="http://www.opengis.net/sas"
targetNamespace="http://www.opengis.net/sas"
elementFormDefault="qualified" attributeFormDefault="unqualified"
version="1.0.0" xml:lang="en">
   <!-- ================================================================
      includes and imports
   ================================================================ -->
   <include schemaLocation="./sasCommon.xsd"/>
   <!-- ================================================================
      request
   ================================================================ -->
   <element name="Subscribe">
      <annotation>
         <documentation>Request to a SAS to subscribe to
alerts.</documentation>
      </annotation>
      <complexType>
         <complexContent>
            <extension base="sas:RequestBaseType">
               <sequence>
                  <element ref="sas:SubscriptionOfferingID"
minOccurs="0"/>
                  <element name="Location"
type="sas:OperationAreaType" minOccurs="0">
                     <annotation>
                        <documentation>allows to restrict to those
sensors located at the GeoLocation or within the GeoLocationArea
(bounding box).</documentation>
                     </annotation>
                  </element>
                  <element ref="sas:ResultFilter" minOccurs="0"
maxOccurs="unbounded"/>
                  <element name="FeatureOfInterestName" type="token"
minOccurs="0">
                     <annotation>
                        <documentation>simple identifier of the
feature of interest. Might be the name of a river or building. IDs are
indicated in the capabilities document.</documentation>
                     </annotation>
                  </element>
                  <element name="ResultRecipient" minOccurs="0">
                     <annotation>
                        <documentation>Allows to specify where the
SAS should send the data to. 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. This allows to use
the SAS in "last-mile-mode".</documentation>
                     </annotation>
                     <complexType>
                        <choice>
                           <element name="XMPPURI" type="anyURI">
                              <annotation>
```

```
                                        <documentation>The excplicit URI
where the xmpp messages should be send to</documentation>
                                    </annotation>
                                </element>
                                <element name="WNS" type="sas:WNSType"/>
                            </choice>
                        </complexType>
                    </element>
                </sequence>
            </extension>
        </complexContent>
    </complexType>
</element>
<element name="SubscribeResponse">
    <complexType>
        <choice>
            <element ref="sas:Status"/>
            <element name="XMPPResponse" type="sas:XMPPResponseType"/>
        </choice>
        <attribute name="SubscriptionID" type="ID" use="required"/>
        <attribute name="expires" type="dateTime" use="required"/>
    </complexType>
</element>
<element name="CancelSubscription">
    <complexType>
        <complexContent>
            <extension base="sas:RequestBaseType">
                <sequence>
                    <element name="SubscriptionID" type="ID"/>
                </sequence>
            </extension>
        </complexContent>
    </complexType>
</element>
<element name="CancelSubscriptionResponse">
    <complexType>
        <sequence>
            <element ref="sas:SubscriptionID"/>
            <element ref="sas:Status"/>
        </sequence>
    </complexType>
</element>
<element name="RenewSubscription">
    <complexType>
        <complexContent>
            <extension base="sas:RequestBaseType">
                <sequence>
                    <element name="SubscriptionID" type="ID"/>
                </sequence>
            </extension>
        </complexContent>
    </complexType>
</element>
<element name="RenewSubscriptionResponse">
    <complexType>
        <sequence>
            <element ref="sas:SubscriptionID"/>
            <element ref="sas:Status"/>
```

```
            </sequence>
            <attribute name="expires" type="dateTime" use="required"/>
        </complexType>
    </element>
    <!---->
    <element name="SubscriptionID" type="ID">
        <annotation>
            <documentation>Unique Identifier for that description.
Provided by SAS server.</documentation>
        </annotation>
    </element>
    <complexType name="XMPPResponseType">
        <sequence>
            <element ref="sas:XMPPURI"/>
        </sequence>
    </complexType>
    <complexType name="WNSType">
        <sequence>
            <annotation>
                <documentation>Use for the last mile a WNS, to do so
provide the UserID and the URL of the WNS</documentation>
            </annotation>
            <element name="WNSURL" type="anyURI"/>
            <element name="WNSUserID" type="token"/>
        </sequence>
    </complexType>
    <!---->
    <element name="ResultFilter">
        <complexType>
            <choice>
                <element name="isSmallerThan" type="double"
minOccurs="0"/>
                <element name="isGreaterThan" type="double"
minOccurs="0"/>
                <element name="equals" minOccurs="0">
                    <simpleType>
                        <union memberTypes="string double"/>
                    </simpleType>
                </element>
                <element name="isNotEqualTo" minOccurs="0">
                    <simpleType>
                        <union memberTypes="string double"/>
                    </simpleType>
                </element>
            </choice>
            <attribute name="ObservedPropertyDefinition" type="anyURI"
use="required"/>
            <attribute name="uom" type="anyURI" use="required"/>
        </complexType>
    </element>
</schema>
```

wns4sas.xsd

```
<?xml version="1.0" encoding="UTF-8"?>
<xs:schema xmlns:sas="http://www.opengis.net/sas"
xmlns:xs="http://www.w3.org/2001/XMLSchema"
xmlns:wns="http://www.opengis.net/wns"
targetNamespace="http://www.opengis.net/sas"
elementFormDefault="qualified" attributeFormDefault="unqualified"
version="1.0.0">
    <xs:import namespace="http://www.opengis.net/wns"
schemaLocation="../../wns/0.1.1/wns.xsd"/>
    <xs:element name="DoNotification" type="wns:DoNotificationType"/>
</xs:schema>
```


Annex C
(informative)

Example XML documents

D.1 Introduction

This annex provides more example XML documents than given in the body of this document.

Bibliography

[1] Guidelines for Successful OGC Interface Specifications, OGC document 00-014r1

