

Open GIS Consortium Inc.

Date: 2003-04-21

Reference number of this OpenGIS® Project Document: **OGC 03-008r2**

Version: 0.1.0

Category: OpenGIS Discussion Paper

Editors: Ingo Simonis, Andreas Wytzisk

Web Notification Service

Copyright notice

This OGC document is a draft and is copyright-protected by OGC. While the reproduction of drafts in any form for use by participants in the OGC standards development process is permitted without prior permission from OGC, neither this document nor any extract from it may be reproduced, stored or transmitted in any form for any other purpose without prior written permission from OGC.

Warning

This document is not an OGC Standard. It is distributed for review and comment. It represents work in progress being completed through the OGC Interoperability Program and Specification Program. This document does not represent the official position of the OGC nor of the OGC Technical Committee. It is subject to change without notice and may not be referred to as an OGC Standard or Specification. However, the intent of the authors is to evolve the document until it becomes an OpenGIS Implementation Specification. The authors welcome feedback. Recipients of this document are invited to submit their comments, and if applicable, notification and supporting documentation pertaining to relevant patent rights.

Document type: OpenGIS Discussion Paper
Document subtype: Engineering Specification
Document stage: Publicly Available
Document language: English

Table of Contents

i.	Preface.....	iv
ii.	Submitting organizations	iv
iii.	Document Contributor Contact Points.....	iv
iv.	Revision history	v
v.	Future Work.....	v
	Foreword.....	vii
	Introduction.....	viii
1	Scope.....	1
2	Conformance	1
3	Normative References.....	1
4	Terms and definitions	2
5	Conventions	2
5.1	Symbols (and abbreviated terms).....	2
5.2	UML Notation	3
6	Operational Context	4
6.1	Introduction.....	4
6.2	Sensor Web.....	4
7	Design and Specification for Web Notification	5
7.1	One-way notification.....	7
7.2	Two-way notification	8
8	Communication Model	9
8.1	doNotification Schema.....	9
8.2	doCommunication Schema	10
8.3	Communication Response Model.....	11
9	Operations	12
9.1	Introduction.....	12
9.2	The getCapabilities Operation (required)	13
9.2.1	Request Overview	13
9.2.2	Request Parameters	14
9.2.3	Request Format.....	15
9.2.4	Request Response.....	15
9.3	The registerUser Operation (required).....	15
9.3.1	Request Overview	15
9.3.2	Request Parameters	15
9.3.3	Request Format.....	16
9.3.4	Request Response.....	16
9.4	The doNotification Operation (required)	17
9.4.1	Request Overview	17
9.4.2	Request Parameters	17
9.4.3	Request Format.....	18

9.4.4	Request Response.....	18
9.5	The doCommunication Operation	19
9.5.1	Request Overview	19
9.5.2	Request Parameters.....	19
9.5.3	Request Format.....	21
9.5.4	Request Response.....	21
9.6	The doReply Operation	21
9.6.1	Request Overview	21
9.6.2	Request Parameters.....	22
9.6.3	Request Format.....	23
9.6.4	Response Format.....	23
10	Parameter Metadata.....	23
	Annex A: WSDL Definitions (normative).....	24
	Annex B: Core XML Schema (normative).....	26
	Annex C: Message XML Schema (normative).....	31
	Annex D: Miscellaneous XML Schema (informative).....	34
	Bibliography	38

i. Preface

This draft specification is one of five engineering specifications produced under OGC's Sensor Web Enablement (SWE) activity, which is being executed under OGC's Interoperability Program. This latest version was produced under the OGC Web Services (OWS) 1.2 Initiative, conducted June 2002 - January 2003.

ii. Submitting organizations

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

- a. Institute for Geoinformatics, University of Muenster, Germany

iii. Document Contributor Contact Points

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

CONTACT	COMPANY	ADDRESS	PHONE/FAX	EMAIL
Ingo Simonis	University of Muenster	Robert-Koch-Str. 26-28 48149 Muenster Germany	+49-251-83-30057	simonis@uni-muenster.de
Andreas Wytzisk	University of Muenster	Robert-Koch-Str. 26-28 48149 Muenster Germany	+49-251-83-30057	wytzisk@uni-muenster.de

iv. Revision history

Date	Release	Author	Paragraph modified	Description
05. Aug. 2002	0.0.1	Simonis, Wytzisk		Initiating version
19. Aug. 2002	0.0.2	Simonis, Wytzisk	div.	WNS-core schema added, minor changes
21. Aug. 2002	0.0.3	Jeffrey Simon	div.	Minor changes
21. Aug. 2002	0.0.4	Simonis, Wytzisk	7; Appendix	Examples added; message schema added; core schema overdone
22. Aug. 2002	0.0.5	Simonis, Wytzisk	Appendix	Schemata overdone
30. Aug. 2002	0.0.6	Simonis, Wytzisk	6	Registration sequence diagram revised; paragraph status management added
06. Sept. 2002	0.0.7	Simonis, Wytzisk	8.3; Appendix	InstantMessaging as a possible communication channel added; WNS core schema revised
20. Sept. 2002	0.0.8	Simonis, Wytzisk	Appendix	WSDL added, schemata overdone
31. Oct. 2002	0.0.9	Simonis, Wytzisk	6, 7, Appendix	Schemas modified
10. Nov. 2002	0.0.10	Simonis, Wytzisk	div.	Minor changes
02. Jan. 2003	v0.0.11 (03-008)	Simonis, Wytzisk	Future work	new
18 Jan 2003	v0.1.0 (03-008r1)	Harry Niedzwiadek	Various	The final OWS 1.2 review and edit. Produced OGC 03-008r1. Version level incremented to 0.1.0 to reflect successful implementation under OWS 1.2.

The issues in this specification are captured in the following format:

Issue Name: [Issue Name goes here. (Your Initials, Date)] Issue Description: [Issue Description.] Resolution: [Insert Resolution Details and History.] (Your Initials, Date)]

v. Future Work

The Web Notification Service (WNS) is the first asynchronous messaging service specified by OGC. At the moment, the WNS message schema is optimized to fulfil the needs of services supporting the use of sensors, like “Sensor Planning Service”. Future work activities should include the adaptation of the message schema to the needs of other services.

Whereas WNS is actually used to send and receive notifications, it is potentially extendable to a “*Work Flow Service*”. This extension is possible because WNS can forward any kind of message using HTTP POST. An enhanced version would allow proper service chaining by forwarding job orders defined in a Work Flow Language (WFL). The specification of a work flow language would also be necessary. The following figure illustrates the possible role of a Work Flow Service:

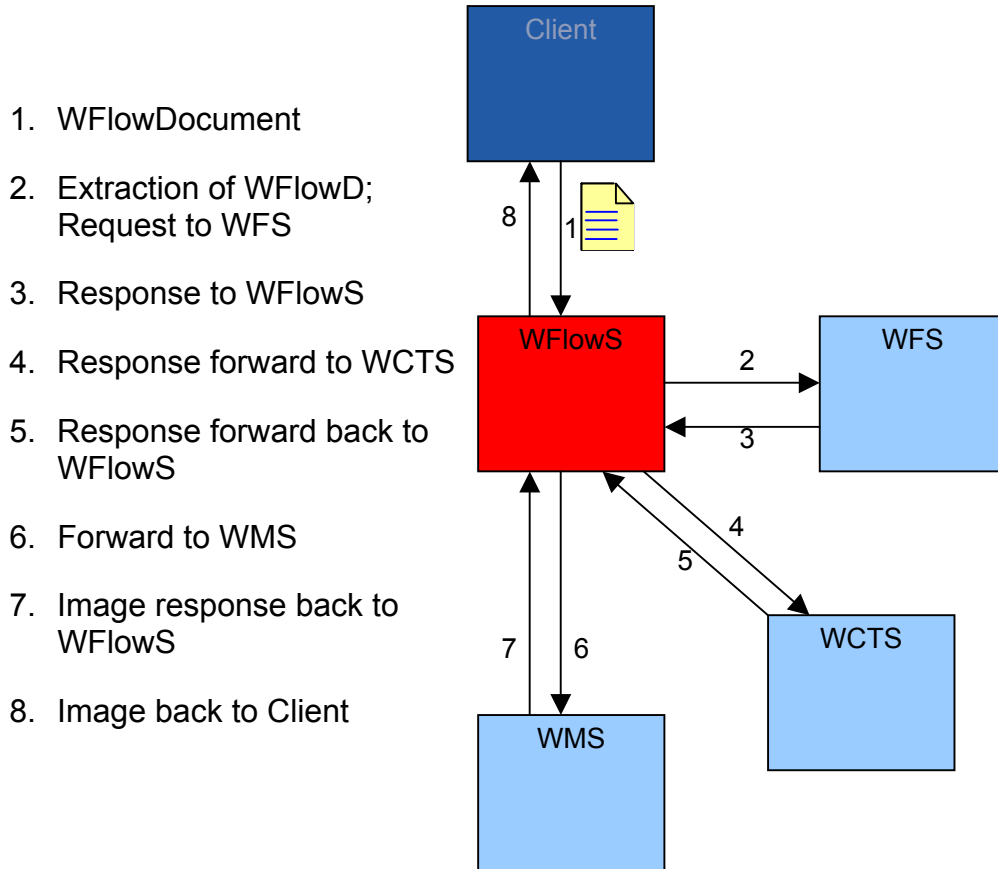


Figure 1 WFlowS: The Possible Role of a Future Extended WNS

Foreword

This specification was developed under the OWS 1.2 initiative. The work is totally new and does not replace other documents in whole or in part.

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

Introduction

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

1. **Sensor Model Language (SensorML)** – The general models and XML encodings for sensors. SensorML originated under OWS 1.1, was significantly enhanced under OWS 1.2 and is now available as a public discussion paper.
2. **Observations & Measurements (O&M)** - The general models and XML encodings for sensor observations and measurements. O&M originated under OWS 1.1 and was significantly enhanced under OWS 1.2.
3. **Sensor Collection Service (SCS)** – A service by which a client can obtain observations from one or more sensors/platforms (can be of mixed sensor/platform types). Clients can also obtain information that describes the associated sensors and platforms. This service originated under OWS 1.1 and was significantly enhanced under OWS 1.2.
4. **Sensor Planning Service (SPS)** – A service by which a client can determine collection feasibility for a desired set of collection requests for one or more mobile sensors/platforms, or a client may submit collection requests directly to these sensors/platforms. This service was defined under OWS 1.2.
5. **Web Notification Service (WNS)** – A service by which a client may conduct asynchronous dialogues (message interchanges) with one or more other services. This service is useful when many collaborating services are required to satisfy a client request, and/or when significant delays are involved in satisfying the request. This service was defined under OWS 1.2 in support of SPS operations. WNS has broad applicability in many such multi-service applications.

This document specifies Web Notification Service. The other components are specified under separate cover.

The WNS is a general purpose messaging service. It is an asynchronous and stateful service. It is a service that sends notifications consisting of well-structured content to a client. To enable any kind of dialogue between the user and an invoking service, functionality has to be provided that enables the user to asynchronously answer with any kind of structured content.

Web Notification Service: A Draft Implementation Specification

1 Scope

This document specifies the interfaces for the Web Notification Service. These interfaces provide the means to:

- Request information describing WNS capabilities,
- Register users,
- Submit simple, synchronous user notification requests, and
- Submit requests for an asynchronous user-service or service-service dialogue.

2 Conformance

Conformance and Interoperability Testing for WNS may be checked using all the relevant tests specified in Annexes A, B, and C (normative). The framework, concepts, and methodology for testing, and the criteria to be achieved to claim conformance are specified in ISO 19105: Geographic information — Conformance and Testing.

3 Normative References

The following normative documents contain provisions, which through reference in this document constitute provisions of this architecture. For dated references, subsequent amendments to these publications or revisions of any of these publications do not apply. However, parties to agreements based on this document are encouraged to investigate the possibility of applying the most recent editions of the normative documents indicated below. For undated references, the latest edition of the documents applies.

OpenGIS® <i>Abstract Specification Topic 0: Overview</i> , OGC document 99-100r1.

<i>Namespaces in XML</i> . W3C Recommendation (14 January 1999). Available [Online]: < http://www.w3.org/TR/1999/REC-xml-names-19990114/ >
--

<i>XML Schema Part 1: Structures</i> . W3C Recommendation (2 May 2001). Available [Online]: < http://www.w3.org/TR/xmlschema-1/ >

<i>XML Schema Part 2: Datatypes</i> . W3C Recommendation (2 May 2001). Available [Online]: < http://www.w3.org/TR/xmlschema-2/ >
--

4 Terms and definitions

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

4.1 Communication

Dialogue sense (default for this document): the act of virtually verbal communication between a user and a service or two services respectively. The aim of communication is a server driven “notification – answer” correlation.

4.2 User

A user could be a human or another service (acting as client).

5 Conventions

5.1 Symbols (and abbreviated terms)

IN	Information Need
HTTP	Hypertext transport protocol
IR	Information Request
OGC	Open GIS Consortium
OSI	Open Systems Interconnection
OWS	OGC Web Services
O&M	Observations and Measurements
SCS	Sensor Collection Service
SensorML	Sensor Model Language
SMS	Short Message Service
SPS	Sensor Planning Service
SWE	Sensor Web Enablement
UML	Unified Modeling Language
WFS	Web Feature Server
WMS	Web Map Server

WNS Web Notification Service

XML eXtended Markup Language

5.2 UML Notation

The diagrams that appear in this document are presented using the Unified Modeling Language (UML). The UML notations used in this document are described in the diagram below (Figure 2).

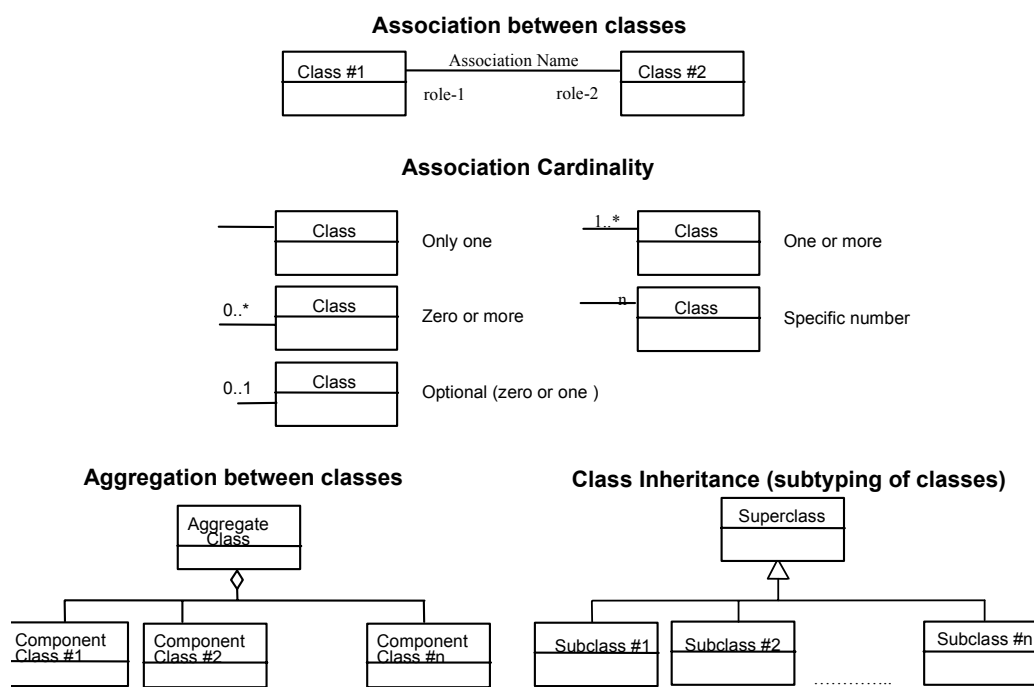


Figure 2 UML notation

In this diagram, the following three stereotypes of UML classes are used:

- a) <<Interface>> A definition of a set of operations that is supported by objects having this interface. An Interface class cannot contain any attributes.
- b) <<DataType>> A descriptor of a set of values that lack identity (independent existence and the possibility of side effects). A DataType is a class with no operations whose primary purpose is to hold the information.
- c) <<CodeList>> is a flexible enumeration that uses string values for expressing a list of potential values.

In this document, the following standard data types are used:

- a) `CharacterString` – A sequence of characters
- b) `Integer` – An integer number
- c) `Double` – A double precision floating point number
- d) `Float` – A single precision floating point number

6 Operational Context

6.1 Introduction

Web Service environments provide a suitable method to gather requested information in an appropriate way. Synchronous transport protocols such as HTTP provide the necessary functionalities to post requests and to receive the respective responses. HTTP is a reliable protocol in the way it ensures the packet delivery, in order, and with a definitive acknowledgement for each delivery or failure. In case of a simple Web Map Service, a user will receive visualized geographic information after a negligible amount of time, or the user will receive an exception message. HTTP satisfies the needs for this kind of processing, without the need for further functionality.

As services become more complex, basic request-response mechanisms need to contend with delays/failures. For example, mid-term or long-term (trans-) actions demand functions to support asynchronous communications between a user and the corresponding service, or between two services, respectively. The Web Notification Service fulfils these needs.

6.2 Sensor Web

A Web Notification Service is required to fulfil the needs of Sensor Web Enablement (SWE). SWE provides interfaces that allow the planning of observations and the respective gathering of the collected sensor data.

Independent of any considered use case (in situ sensors, earth observation etc.), two different utilizations of an SWE environment can be distinguished. The first one usually does not require any establishment of an asynchronous user-service/service-service communication, as all the processing can be handled purely synchronous and follows the service trading (publish – find – bind) paradigm. This case mainly pertains to collections of in-situ sensors (due to the complexity of remote sensors).

Observations that require preceding collection feasibility studies, complex control and management activities, or intermediate and/or subsequent user notifications, are not

conducive to synchronous operations, but instead favour asynchronous operations. For these cases, especially when a Sensor Planning Service comes into play, asynchronous communications need to be supported.

For example, in a request for a satellite image, the user submits a collection feasibility request through a Sensor Planning Service and then subsequently requests collection of the desired observations [1]. For this case, if any procedures are finished, interrupted, delayed, timed out, or cancelled, the user must be notified.

If the feasibility request returns a positive response, the user would then request the observations. The requested data will probably not be ready for immediate retrieval — there will likely be a delay. It is also possible that the service would not be able to provide an exact retrieval date-time. Thus, a notification mechanism becomes necessary.

The client application or service (e.g., SPS) is typically not a continuously accessible Web service. Hence the client cannot be reached by an http-call from the respective service when data finally becomes available. A self-contained, Web accessible notification service provides the solution to this condition.

7 Design and Specification for Web Notification Service

The Web Notification Service Model includes two different kinds of notifications. First, the “one-way-communication” provides the user with information without expecting a response. Second, the “two-way-communication” provides the user with information and expects some kind of asynchronous response. This differentiation implies the differences between simple and sophisticated WNS. A simple WNS provides the capability to notify a user and/or service that a specific event occurred. In addition, the latter is able to receive a response from the user.

The basis on which notifications will be sent is free to the service and will be described in its capabilities. The “way-of-notification” palette may include:

- e-mail
- http-call (as HTTP POST: in case of sophisticated clients that act as web services themselves)
- SMS
- Instant Message
- phone call
- letter

- fax

A WNS has to provide at least one of the described notification mechanisms. To make use of the notification capabilities, users have to be registered beforehand. This registration will be performed by either a user, or by an OGC Service that can act as a proxy for the user, which makes use of the notification functionality (e.g., a SPS). Figure 3 demonstrates this behaviour:

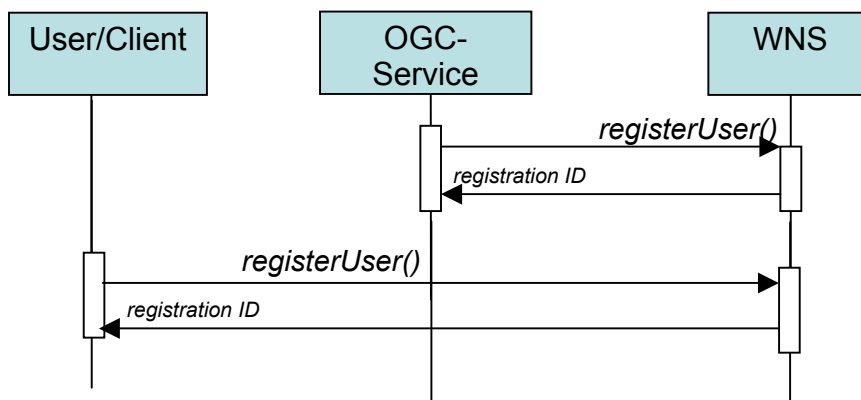


Figure 3 Sequence Diagram of the Registration Process

The OGC Service that uses the WNS must provide the registration information about the user (identification) and it must provide the notification target (e.g., the user's e-mail address, phone number, etc). In return, the service will receive the registration ID or an exception as acknowledgement.

Independently of the registration requestor (the user of service that acts as a proxy), there is no user data verification mechanism available yet. For example, if a user (client service) requests an expensive data collection, he (the service) cannot get notified if the provided email address was misspelled during registration. In a next evolutionary step, the WNS will be equipped with internal status management and will provide the necessary interface that allows users/services to check if an error has occurred during previous operations.

Issue Name: [User verification mechanism. (HAN, 18 Jan 2003)]

Issue Description: [Need to extend the interface to support user confirmation between client service and WNS.]

Resolution: [Insert Resolution Details and History.] (Your Initials, Date)]

7.1 One-way notification

The “one-way-communication” defines the simplest WNS. This version of WNS provides the facilities to handle the registration of users, as described above, and the sending of notifications.

Once registered, the calling service is enabled to send notifications to the user/client by calling the `doNotification()` operation, as shown in Figure 4.

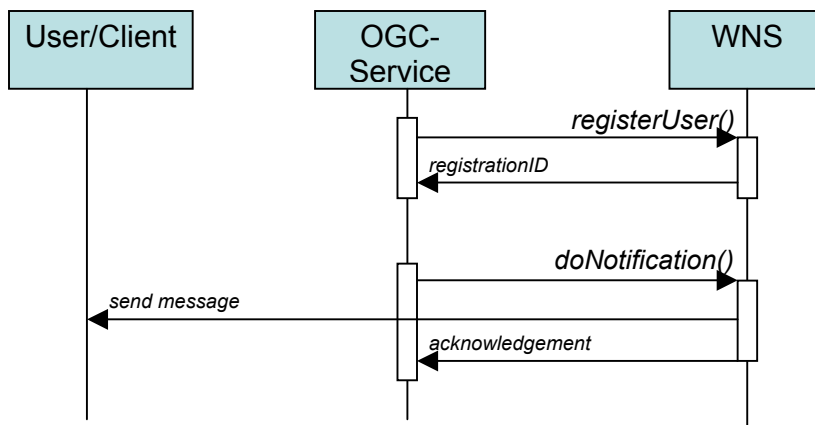


Figure 4 Sequence Diagram for Simple WNS

The acknowledgement of the `doNotification()`-request includes a status report. The states are:

- notification sent successfully,
- notification sent unsuccessfully, or
- notification timed out.

The acknowledgement does not provide any information about receiving the notification, that is,:

- “Notification sent successfully” does not indicate that the user has received the notification for all events. It just signals that the notification was successfully sent on its way by the WNS.
- “Notification sent unsuccessfully” indicates that the notification was sent but returned to the WNS, meaning that it was undeliverable (e.g., due to a misspelled e-mail address).
- “Notification timed out” indicates that the WNS was not able to send the notification.

7.2 Two-way notification

The “two-way-notification” constitutes enhanced WNS. These services provide all the capabilities described above. Additionally, the enhanced WNS provides the interface to receive notification responses sent by users. Hence, the service provides the capabilities to establish a virtual asynchronous communication with the user. Figure 5 demonstrates this procedure.

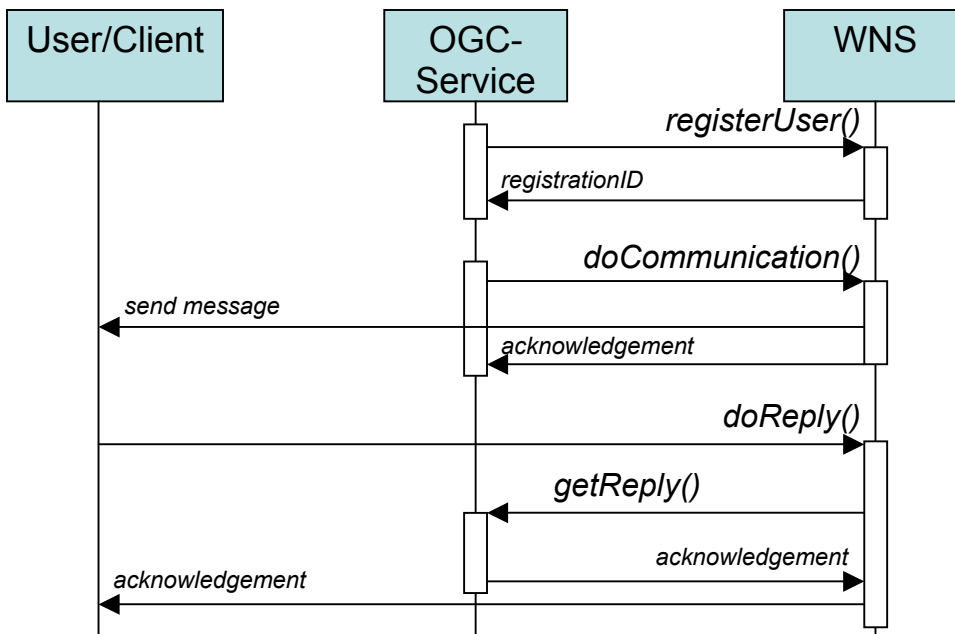


Figure 5 Sequence Diagram of an Enhanced WNS

To simplify the operational use in OWS Testbed 1.2, the control sequence shown above will not be supported entirely. User responses are currently sent back to the calling OGC Service directly. This implies that the OGC Service, or a downstream component, must be able to receive the responses.

To free the calling service from the burden of providing an interface for different notification protocols, the user response will be sent back to the WNS in a future version, supporting the full sequence as shown in Figure 5. Hence the user will be able to reply through all the communication protocols provided by the WNS (e.g., SMS, HTTP commands, e-mails...).

This highly advanced feature will be defined in more detail in the future.

8 Communication Model

Notifications are messages that are sent from a service to a client using arbitrary protocols. Generally, two kinds have to be distinguished: simple notifications that solely inform the client that some event has occurred, without requesting a client reply; and complex (virtual) notifications. The latter requires that the client must provide input to some specific questions. Two different requests exist for these cases: The doNotification request fulfils the needs of the simple notification, whereas the doCommunication request invokes the complex notification/reply procedure.

The notification message itself is a well-defined structure, depending on the capabilities of the WNS. It is device dependent. In the case of an SMS-notification, the WNS will send a rather tiny fragment, whereas a huge XML-file could be sent to an Internet server client. The conversion of the XML-message code provided by the calling service to a suitable notification is the task of the Web Notification Service.

Callers to OGC Web Services generally have to know which protocols are available to send notifications to a specific user, due to their responsibility to register a user with a WNS. Practically, they just forward the information provided by the user to the WNS, during the registration process, which means that they have no idea how a specific user could be reached generally. This knowledge is stored at the WNS, exclusively. To simplify the use of WNS for other Web Services, the only way to notify users is by sending XML-message code to the WNS.

8.1 doNotification Schema

In case of simple notifications, the client (calling) service has to adopt the common NotificationEventSchema, as shown in Annex C.1. This schema defines the possible list of user describable properties that may be sent from a service to a WNS in a doNotification request. The client service has to define which notification events it supports within its capabilities. This has to be harmonized with the other OGC Web Services that utilize the WNS, e.g. the SPS. The schema will be stored at a central location and/or on each Web Notification Server. Because it is optional, no need arises to modify existing OGC Web Services.

For example, for SPS, the events that could be reported are 

Property	Description
Operation completed	The data collection is completed; you can retrieve the data at http://a.specific.url/xyz
Operation cancelled	The observing sensor crashed; your request is not feasible anymore.
Operation delayed	Due to hardware failure, the requested observations need two more weeks to be extracted from storage.
Operation failed	The data collection failed.
New Data available	An update to the data previously requested can be retrieved from http://a.specific.url/xyz
Further events	To be completed.

Table 1 Representative Notification Events

Example:

```
<?xml version="1.0" encoding="UTF-8"?>
<NotificationMessage xmlns="http://www.opengis.net/wns"
mlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://www.opengis.net/wns ../wnsMessage.xsd">
  <Type>Operation failed</Type>
  <MessageParameter>
    <CorrID>2147483647</CorrID>
    <Key>The satellite data from the following areas is not available</Key>
    <String>Area1, Area 7, Area 23</String>
  </MessageParameter>
</NotificationMessage>
```

8.2 doCommunication Schema

In the case of a complex notification, called a communication, each client service has to define its own keys. The keys shall be described in its capabilities, in order to provide a common semantic understanding.

The XML-message code sent to the WNS in a doCommunication request mainly persists on the following parameters:

- **Action:** The action defines the operation that is needed by the service, e.g., that it needs further information from the user.
- **CorrID:** The correlation ID has to be provided by the client services and allows the mapping of user replies to threads.

- **Key:** The key describes the parameter and what kind of interaction has to be undertaken, e.g., “accuracy needed”.
- **Unit:** The unit is an optional parameter which defines the unit of the requested input.
- **Option:** The option is an optional parameter which provides a list of possible input options, e.g., “yes” and “no”; list of figures: “2.0”, “2.5”, “3.0”, “3.5”.
- **Value:** The value part consists of a choice block of the simple data types which could be used for further description, based on the ones given in the “Key” node.

For the normative schema for the communication message, see appendix C2.

Example:

```
<?xml version="1.0" encoding="UTF-8"?>
<!--Sample XML file generated by XML Spy v4.4 U (http://www.xmlspy.com)-->
<CommunicationRequestMessage xmlns="http://www.opengis.net/wns"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://www.opengis.net/wns
D:\Projekte\OGC_HLA\SWE\WNS\XML\wnsMessage.xsd">
  <Action>Information needed</Action>
  <MessageParameter>
    <CorrID>2147483647</CorrID>
    <Key>Lookangle has to be defined</Key>
    <Unit>degree</Unit>
    <Options>-10 -5 0 5 10</Options>
  </MessageParameter>
</CommunicationRequestMessage>
```

8.3 Communication Response Model

Part one of the dialogue, the initiation of a dialogue notification by an OGC Web Service, and the subsequent message sent from the Web Notification Service, is discussed in clause 8. Once notified, it is up to the user to take further steps.

There are multiple ways a Web Notification Service may send a notification. The supported protocols are described in its capabilities. On the other hand, there are multiple ways to receive the reply from a user. Although the common protocol used for replies will certainly be HTTP (calling the doReply interface of the WNS), other protocols might be supported, as well. A WNS may provide a callback interface for SMS, phone calls, fax, etc., as well. The standardization concerning how these protocols shall be used is a highly advanced feature and is therefore postponed.

Currently, the only supported callback interface used for user replies is HTTP. This restriction does not constrain the WNS from using other protocols to send its initial notification.

The XML-part of a doCommunication response from a user, explicitly the doReply operation, differs from the doCommunication request only in the replacement of the Unit and Options nodes by a Value node that contains the user provided answer for the request. This is clarified the following figure:

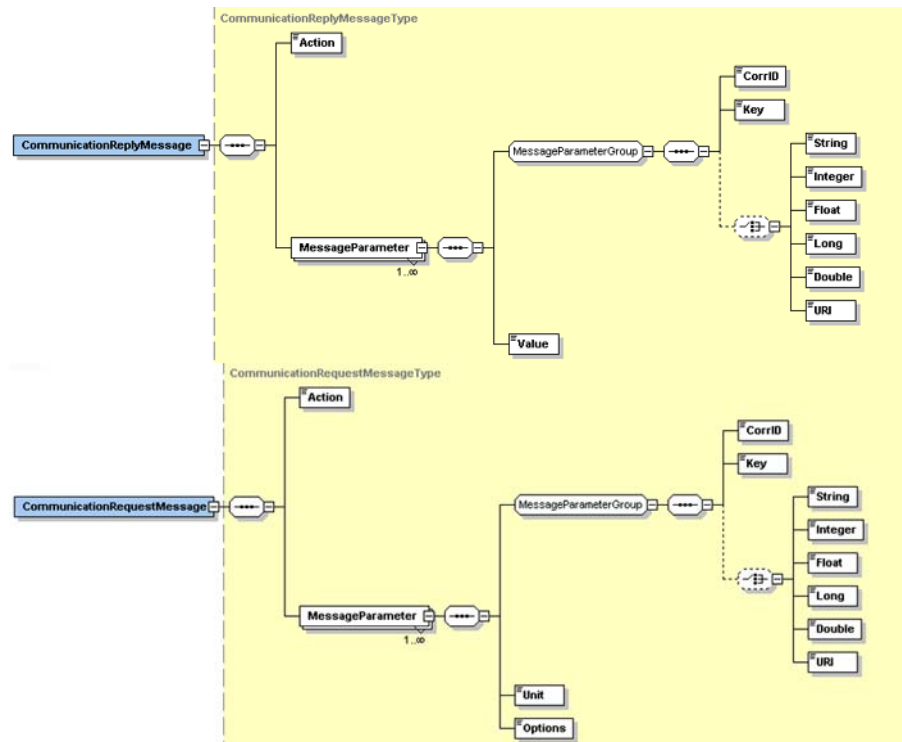


Figure 6 Comparison of a CommunicationReplyMessage (above) and a CommunicationRequestMessage (below)

9 Operations

9.1 Introduction

The WNS operations can be divided into registration and notification operations. The doReply-operation is a rather advanced feature and will not be defined entirely yet.

Request	Response Type
getCapabilities	Capabilities XML-file
registerUser	RegistrationID
doNotification	Notification status indicating the success of the notification sending procedure
doCommunication	Notification status indicating the success of the notification sending procedure
doReply	Status indicating the success of forwarding the reply to the calling service

Table 2 WNS Request Types

9.2 The getCapabilities Operation (required)

The purpose of the getCapabilities operation is to describe the capabilities of a service. It follows the guidelines provided in the OWS 1.2 Service Information Model [2]. In the particular case of a Web Notification Service, the response of a getCapabilities request is general information about the service itself and specific information about the available notification protocols and mandatory operational parameters. The following questions have to be answered in the capabilities:

- What notification protocols are supported?
- Which parameters are mandatory for registering?
- What kinds of response protocols are supported?

9.2.1 Request Overview

Request Parameter	Required/ Optional	Description
VERSION=version	O	Request version
SERVICE=WNS	R	Service type
REQUEST=getCapabilities	R	Request name
UPDATESEQUENCE=string	O	Sequence number or string for cache control

Table 3 The Parameters of a getCapabilities Request URL

9.2.2 Request Parameters

9.2.2.1 *VERSION*

The **optional** VERSION parameter, and its use in version negotiation, is specified in the Basic Service Elements section.

9.2.2.2 *SERVICE*

The **required** SERVICE parameter indicates which of the available service types at a particular service instance is being invoked. This parameter allows the same URL prefix to offer Capabilities XML for multiple OGC Web Services.

When invoking getCapabilities on a WNS that implements this version of the specification, or a later one, the service_name value "WNS" **shall** be used.

9.2.2.3 *REQUEST*

This nature of the **required** REQUEST parameter is specified in the Basic Service Elements section. To invoke the getCapabilities operation, the value "getCapabilities" **shall** be used.

9.2.2.4 *UPDATESEQUENCE*

The **optional** UPDATESEQUENCE parameter is for maintaining cache consistency. Its value can be an integer, a timestamp in [ISO 8601:1988(E)] format, or any other number or string. The server **may** include an UpdateSequence value in its Capabilities XML. If present, this value **should** be increased when changes are made to the Capabilities (e.g., when sensors, observables or observable values are added to the service). The server is the sole judge of lexical ordering sequence. The client **may** include this parameter in its getCapabilities request. The response of the server, based on the presence and relative value of UpdateSequence, in the client request and the server metadata, **shall** be according to Table 4:

Client Request UpdateSequence Value	Server Metadata UpdateSequence Value	Server Response
none	any	most recent Capabilities XML
any	none	most recent Capabilities XML
equal	equal	Exception: code=CurrentUpdateSequence
lower	higher	most recent Capabilities XML
higher	lower	Exception: code=InvalidUpdateSequence

Table 4 Use of UpdateSequence Parameter

9.2.3 Request Format

For http-post operation, an appropriate xml-schema is defined. See Annex B.

9.2.4 Request Response

As described in the OGC Service Information Model, [2].

9.2.4.1 *Response Format*

As described in the OGC Service Information Model, [2].

9.3 The registerUser Operation (required)

The registerUser operation allows registering users to receive further notification. The user address and the communicationProtocol have to be provided. The WNS must provide a UserID-management. UserIDs are provided as Long values.

9.3.1 Request Overview

The parameters listed in Table 5 satisfy the needs of the communicationProtocols-types http, e-mail, phone, fax, and SMS. To make use of further notification protocols, such as letter post, further parameters must be added. By sticking to the parameter=value paradigm, mail addresses will be difficult to describe. XML-requests (HTTP-Post) will not have these problems, the specific needs of national addresses could be solved by inheritance.

Parameter	Required/ Optional	Description
VERSION	O	Request version
SERVICE	R	Service type (WNS)
REQUEST	R	Request name (registerUser)
NAME	R	Name of the user
COMMUNICATIONPROTOCOL	R	Enumeration of communication protocols, used for sending notifications

Table 5 The Parameters of a registerUser Request URL

9.3.2 Request Parameters

9.3.2.1 *VERSION*

The **optional** VERSION parameter, and its use in version negotiation, is specified in the Basic Service Elements section.

9.3.2.2 SERVICE

The **required** SERVICE parameter indicates which of the available service types at a particular service instance is being invoked. This parameter allows the same URL prefix to offer Capabilities XML for multiple OGC Web Services.

When invoking getCapabilities on a WNS that implements this version of the specification or a later one, the service_name value "WNS" **shall** be used.

9.3.2.3 REQUEST

This nature of the **required** REQUEST parameter is specified in the Basic Service Elements section. To invoke the registerUser operation, the value "registerUser" **shall** be used.

9.3.2.4 NAME

The required NAME parameter defines the name of the user to be registered.

9.3.2.5 COMMUNICATIONPROTOCOL

The required COMMUNICATIONPROTOCOL parameter indicates the protocol that has to be used to send notifications. Currently, URL (for responses to other servlets using HTTP POST), e-mail, SMS, fax, phone, and Instant Messaging are supported.

Example:

```
<?xml version="1.0" encoding="UTF-8"?>
<!--Sample XML file generated by XML Spy v4.4 U (http://www.xmlspy.com)-->
<RegisterUser xmlns="http://www.opengis.net/wns" xmlns:xsi="http://www.w3.org/2001/XMLSchema-
instance" xsi:schemaLocation="http://www.opengis.net/wns ..\wns.xsd" Version="0.0.1" Service="WNS">
  <Name>Ingo Simonis</Name>
  <CommunicationProtocol>
    <email>simonis@uni-muenster.de</email>
  </CommunicationProtocol>
</RegisterUser>
```

9.3.3 Request Format

For http-post operation, an appropriate xml-schema is defined. See Annex B.

9.3.4 Request Response

The response will be a unique user identifier. It is a Long value.

9.3.4.1 registerUser Response Format

[See Annex B.]

9.4 The doNotification Operation (required)

The doNotification request is called to initiate the notification of a user.

9.4.1 Request Overview

Parameter	Required/ Optional	Description
VERSION	O	Request version
SERVICE	R	Service type (WNS)
REQUEST	R	Request name (doNotification)
USERID	R	UserID, given by the WNS beforehand. Identifies the user to be notified.
CORRID	O	Correlation ID
MSG	R	Message to be sent.

Table 6 The Parameters of a doNotification Request URL

9.4.2 Request Parameters

9.4.2.1 VERSION

The **optional** VERSION parameter, and its use in version negotiation, is specified in the Basic Service Elements section.

9.4.2.2 SERVICE

The **required** SERVICE parameter indicates which of the available service types at a particular service instance is being invoked. This parameter allows the same URL prefix to offer Capabilities XML for multiple OGC Web Services.

When invoking getCapabilities on a WNS that implements this version of the specification, or a later one, the service_name value "WNS" **shall** be used.

9.4.2.3 REQUEST

This nature of the **required** REQUEST parameter is specified in the Basic Service Elements section. To invoke the doNotification operation, the value "doNotification" **shall** be used.

9.4.2.4 USERID

The required USERID defines the user to be notified. It is a Long value identifier which is provided by the WNS during the registration procedure.

9.4.2.5 *CORRID*

The required CORRID parameter defines the CorrelationID that maps a user reply to a specific process. It has to be provided and administrated from the client service. The WNS only forwards this CorrelationID to the user by including it into the notification message. It has no significance to the WNS.

The CORRID has to be included in any reply from the user to allow the originally calling service to map the reply to a specific process.

9.4.2.6 *MSG*

The required MSG parameter defines the message that shall be sent to the user. For details see 8.1.

Example:

```
<?xml version="1.0" encoding="UTF-8"?>
<!--Sample XML file generated by XML Spy v4.4 U (http://www.xmlspy.com)-->
<DoNotification xmlns="http://www.opengis.net/wns" xmlns:xsi="http://www.w3.org/2001/XMLSchema-
instance" xsi:schemaLocation="http://www.opengis.net/wns ../wns.xsd" Version="0.0.1" Service="WNS">
  <UserID>4294967295</UserID>
  <Message>
    <Type>Operation completed</Type>
    <MessageParameter>
      <CorrID>2147483647</CorrID>
      <Key>Requested data available at</Key>
      <URI>http://a.data.source/data.xyz</URI>
    </MessageParameter>
    <MessageParameter>
      <CorrID>2147483647</CorrID>
      <Key>Costs</Key>
      <String>USD 2000.-</String>
    </MessageParameter>
  </Message>
</DoNotification>
```

9.4.3 Request Format

For http-post operation, an appropriate xml-schema is defined. See Annex B.

9.4.4 Request Response

The response to a doNotification request is an Integer, indicating the status of the Notification, see 8.1.

9.4.4.1 *doNotification Response Format*

[See Annex B.]

9.5 The doCommunication Operation

The doCommunication request is called to initiate a communication with a user. The communication contributes an asynchronous dialogue structure, means that the WNS will send a notification to the user. This notification indicates to the user that further action has to be taken (which can be done automatically).

9.5.1 Request Overview

A doCommunication request requires an http-post statement. All the XML-message code discussed in 8.2 has to be transmitted to the WNS. It might be possible to place this code inline, in an http-get statement, but due to the 1024 character restriction of http-get statements, this approach was abandoned. Table 7 provides an overview of a DoCommunication request. The appropriate request format is given under 9.5.2.

Parameter	Required/ Optional	Description
VERSION	O	Request version (0.0.1)
SERVICE	R	Service type (WNS)
REQUEST	R	Request name (doCommunication)
CALLBACK	R	Callback interface where replies from the user will be sent to.
USERID	R	UserID, given by the WNS beforehand. Identifies the user to be notified.
CORRID	R	CorrelationID, given by the requesting service to assign the message to a unique process.
MSG	R	Message to be sent. For details see 8.2 and 9.5.2.

Table 7 doCommunication Request Overview

9.5.2 Request Parameters

9.5.2.1 *VERSION*

The **optional** VERSION parameter, and its use in version negotiation, is specified in the Basic Service Elements section.

9.5.2.2 *SERVICE*

The **required** SERVICE parameter indicates which of the available service types at a particular service instance is being invoked. This parameter allows the same URL prefix to offer Capabilities XML for multiple OGC Web Services.

When invoking getCapabilities on a WNS that implements this version of the specification, or a later one, the service_name value "WNS" **shall** be used.

9.5.2.3 *REQUEST*

This nature of the **required** REQUEST parameter is specified in the Basic Service Elements section. To invoke the doCommunication operation, the value "doCommunication" **shall** be used.

9.5.2.4 *CALLBACK*

The required CALLBACK parameters define the interface where the user reply is to be sent. A callback interface can be part of the calling service, or it can be part of any downstream component (which will be treated as a black-box model).

9.5.2.5 *USERID*

The required USERID defines the user to be notified. It is a Long value identifier which is provided by the WNS during the registration procedure.

9.5.2.6 *CORRID*

The required CORRID parameter defines the CorrelationID that maps a user reply to a specific process. It has to be provided and administrated from the client service. The WNS only forwards this CorrelationID to the user by including it in the notification message. It has no significance to the WNS.

The CORRID has to be included in any reply from the user to allow the original client service to map the reply to a specific process.

9.5.2.7 *MSG*

The required MSG parameter defines the message that shall be sent to the user. For details see 8.1.

Example:

```
<?xml version="1.0" encoding="UTF-8"?>
<!--Sample XML file generated by XML Spy v4.4 U (http://www.xmlspy.com)-->
<DoCommunication xmlns="http://www.opengis.net/wns" xmlns:xsi="http://www.w3.org/2001/XMLSchema-
instance" xsi:schemaLocation="http://www.opengis.net/wns ../wns.xsd">
  <UserID>4294967295</UserID>
  <CallbackServerURL>http://www.xmlspy.com</CallbackServerURL>
  <Message>
    <Action>Information needed</Action>
    <MessageParameter>
      <CorrID>2147483647</CorrID>
      <Key>Number of cars within the simulation</Key>
      <Unit>Integer</Unit>
      <Options>1 2 5 10 100</Options>
    </MessageParameter>
  </Message>
</DoCommunication>
```

9.5.3 Request Format

[See Annex B.]

9.5.4 Request Response

[See Annex B.]

9.6 The doReply Operation

The doReply operation is one way for the user to respond to a dialogue notification. It may be the method that is most often used for this purpose. See chapter 8 for further details.

9.6.1 Request Overview

doReply requests shall use the http-post command.

[TBD. Completed in future effort.]

Parameter	Required/ Optional	Description
VERSION	O	Request version (0.0.1)
SERVICE	R	Service type (WNS)
REQUEST	R	Request name (doReply)
USERID	R	Callback interface where replies from the user will be sent.
CORRID	R	CorrelationID, supplied by the requesting service to assign the message to a unique process
MSG	R	Message

Table 8 doReply Parameter Overview

9.6.2 Request Parameters

9.6.2.1 *VERSION*

The **optional** VERSION parameter, and its use in version negotiation, is specified in the Basic Service Elements section.

9.6.2.2 *SERVICE*

The **required** SERVICE parameter indicates which of the available service types at a particular service instance is being invoked. This parameter allows the same URL prefix to offer Capabilities XML for multiple OGC Web Services.

When invoking getCapabilities on a WNS that implements this version of the specification, or a later one, the service_name value "WNS" **shall** be used.

9.6.2.3 *REQUEST*

This nature of the **required** REQUEST parameter is specified in the Basic Service Elements section. To invoke the doCommunication operation, the value "doCommunication" **shall** be used.

9.6.2.4 *USERID*

The required USERID identifies the replying user. It is a Long value identifier which was provided by the WNS during the registration procedure, and is included in the preceding dialogue notification.

9.6.2.5 *CORRID*

The required CORRID parameter defines the CorrelationID that maps a user reply to a specific process. It has to be provided and administrated from the client service. The

WNS only forwards this CorrelationID to the user by including it in the notification message. It has no significance to the WNS.

The CORRID has to be included in any reply from the user to allow the original client service to map the reply to a specific process again.

9.6.2.6 MSG

The required MSG parameter defines the message that shall be sent to the user.

Example (possible response to the request shown in [#DoCommunicationXMLExample](#)):

```
<?xml version="1.0" encoding="UTF-8"?>
<DoReply xmlns="http://www.opengis.net/wns" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://www.opengis.net/wns ../wns.xsd">
  <UserID>4294967295</UserID>
  <CorrID>4294967295</CorrID>
  <CallbackServerURL>http://www.xmlspy.com</CallbackServerURL>
  <Message>
    <Action>Information needed</Action>
    <MessageParameter>
      <CorrID>2147483647</CorrID>
      <Key>Number of cars within the simulation</Key>
      <Value>5</Value>
    </MessageParameter>
  </Message>
</DoReply>
```

9.6.3 Request Format

[See Annex B.]

9.6.4 Response Format

[See Annex B.]

10 Parameter Metadata

[TBD]

Annex A: WSDL Definitions (normative)

A.1 Conformance class: Web Notification Service

The following WSDL interface specification is to be shared by all implementations of the Web Notification Service conformance class. It makes use of the XML Schema listed in Annex B.

```
<?xml version="1.0" encoding="UTF-8"?>
<definitions xmlns="http://schemas.xmlsoap.org/wsdl/" xmlns:http="http://schemas.xmlsoap.org/wsdl/http/"
  xmlns:mime="http://schemas.xmlsoap.org/wsdl/mime/" xmlns:sps="http://www.opengis.net/sps"
  xmlns:ogc="http://www.opengis.net/ogc" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://schemas.xmlsoap.org/wsdl/
    ..\wsdl\wsdl-1.1.xsd">
  <types>
    <xsd:schema>
      <xsd:import namespace="http://www.opengis.net/wns" schemaLocation="WNS.xsd"/>
      <xsd:import namespace="http://www.opengis.net/ogc" schemaLocation="OGC-exception.xsd"/>
    </xsd:schema>
  </types>
  <!-- ***** -->
  <!-- * MESSAGES * -->
  <!-- ***** -->
  <message name="ServiceExceptionReport">
    <part name="body" element="ogc:ServiceExceptionReport"/>
  </message>
  <message name="GetCapabilities">
    <part name="body" element="wns:GetCapabilities"/>
  </message>
  <message name="RegisterUser">
    <part name="body" element="wns:RegisterUser"/>
  </message>
  <message name="DoNotification">
    <part name="body" element="wns:DoNotification"/>
  </message>
  <message name="DoCommunication">
    <part name="body" element="wns:DoCommunication"/>
  </message>
  <message name="DoReply">
    <part name="body" element="wns:DoReply"/>
  </message>
  <message name="GetCapabilitiesResponse">
    <part name="body" element="wns:GetCapabilitiesResponse"/>
  </message>
  <message name="RegisterUserResponse">
    <part name="body" element="wns:RegisterUserResponse"/>
  </message>
  <message name="DoNotificationResponse">
    <part name="body" element="wns:DoNotificationResponse"/>
  </message>
  <message name="DoCommunicationResponse">
    <part name="body" element="wns:DoCommunicationResponse"/>
  </message>
  <message name="DoReplyResponse">
    <part name="body" element="wns:DoReplyResponse"/>
  </message>
  <!-- ***** -->
```



```
<!-- * PORT TYPES * -->
<!-- ***** -->
<portType name="GetCapabilitiesPortType">
  <operation name="GetCapabilities">
    <input message="GetCapabilities"/>
    <output message="GetCapabilitiesResponse"/>
    <fault name="exception" message="ServiceExceptionReport"/>
  </operation>
</portType>
<portType name="RegisterUserPortType">
  <operation name="RegisterUser">
    <input message="RegisterUser"/>
    <output message="RegisterUserResponse"/>
    <fault name="exception" message="ServiceExceptionReport"/>
  </operation>
</portType>
<portType name="DoNotificationPortType">
  <operation name="DoNotification">
    <input message="DoNotification"/>
    <output message="DoNotificationResponse"/>
    <fault name="exception" message="ServiceExceptionReport"/>
  </operation>
</portType>
<portType name="DoCommunicationPortType">
  <operation name="DoCommunication">
    <input message="DoCommunication"/>
    <output message="DoCommunicationResponse"/>
    <fault name="exception" message="ServiceExceptionReport"/>
  </operation>
</portType>
<portType name="DoReplyPortType">
  <operation name="DoReply">
    <input message="DoReply"/>
    <output message="DoReplyResponse"/>
    <fault name="exception" message="ServiceExceptionReport"/>
  </operation>
</portType>
</definitions>
```

Annex B: WNS XML Schema (normative)

B.1 Web Notification Service Schema

The following XML Schema is intended to be shared by all implementations of the Web Notification Service. It makes use of the XML Message Schema in Annex C.

```
<!-- edited with XML Spy v4.4 U (http://www.xmlspy.com) by Institut für Geoinformatik (Institut für
Geoinformatik) -->
<xs:schema targetNamespace="http://www.opengis.net/wns" xmlns="http://www.opengis.net/wns"
xmlns:xs="http://www.w3.org/2001/XMLSchema" elementFormDefault="qualified"
attributeFormDefault="unqualified">
  <!--=====
Includes and Imports
=====-->
  <xs:include schemaLocation="wnsMessage.xsd"/>
  <!--=====
REQUEST MESSAGES
=====-->
  <!--GetCapabilities-->
  <xs:element name="GetCapabilities" type="GetCapabilitiesType"/>
  <!--RegisterUser-->
  <xs:element name="RegisterUser" type="RegisterUserType"/>
  <!--DoNotification-->
  <xs:element name="DoNotification" type="DoNotificationType"/>
  <!--DoCommunication-->
  <xs:element name="DoCommunication" type="DoCommunicationType"/>
  <!--DoReply-->
  <xs:element name="DoReply" type="DoReplyType"/>
  <!--=====
RESPONSES
=====-->
  <!--GetCapabilitiesResponse-->
  <xs:element name="GetCapabilitiesResponse" type="GetCapabilitiesResponseType"/>
  <!--RegisterUserResponse-->
  <xs:element name="RegisterUserResponse" type="RegisterUserResponseType"/>
  <!--DoNotificationResponse-->
  <xs:element name="DoNotificationResponse" type="DoNotificationResponseType"/>
  <!--DoCommunicationResponse-->
  <xs:element name="DoCommunicationResponse" type="DoCommunicationResponseType"/>
  <!--DoReplyResponse-->
  <xs:element name="DoReplyResponse" type="DoReplyResponseType"/>
  <!--=====
TYPES
=====-->
  <!--=====
MessageTypes
=====-->
  <!--GetCapabilitiesType-->
  <xs:complexType name="GetCapabilitiesType">
    <xs:sequence>
      <xs:element name="Service">
```

```

        <xs:complexType>
          <xs:sequence>
            <xs:element name="Name" type="xs:string"/>
            <xs:element name="Title" type="xs:string"/>
            <xs:element name="Keywordlist" type="Keywordlist"/>
            <xs:element name="ContactInformation" type="ContactInformationType"/>
          </xs:sequence>
        </xs:complexType>
      </xs:element>
      <xs:element name="Capabilities" type="RequestType"/>
    </xs:sequence>
    <xs:attribute name="Version" type="xs:string" use="optional" fixed="0.0.1"/>
    <xs:attribute name="Service" type="xs:string" use="required" fixed="WNS"/>
    <xs:attribute name="UpdateSequenceString" type="xs:string" use="optional"/>
  </xs:complexType>
<!--RegisterUser-->
<xs:complexType name="RegisterUserType">
  <xs:sequence>
    <xs:element name="Name" type="xs:string"/>
    <xs:element name="CommunicationProtocol" type="CommunicationProtocolType"/>
  </xs:sequence>
  <xs:attribute name="Version" type="xs:string" use="optional" fixed="0.0.1"/>
  <xs:attribute name="Service" type="xs:string" use="required" fixed="WNS"/>
</xs:complexType>
<!--DoNotification-->
<xs:complexType name="DoNotificationType">
  <xs:sequence>
    <xs:element name="UserID" type="xs:unsignedLong"/>
    <xs:element name="Message" type="NotificationMessageType"/>
  </xs:sequence>
  <xs:attribute name="Version" type="xs:string" use="optional" fixed="0.0.1"/>
  <xs:attribute name="Service" type="xs:string" use="required" fixed="WNS"/>
</xs:complexType>
<!--DoCommunication-->
<xs:complexType name="DoCommunicationType">
  <xs:sequence>
    <xs:element name="UserID" type="xs:unsignedLong"/>
    <xs:element name="CallbackServerURL" type="xs:anyURI"/>
    <xs:element name="Message" type="CommunicationRequestMessageType"/>
  </xs:sequence>
</xs:complexType>
<!--DoReply-->
<xs:complexType name="DoReplyType">
  <xs:sequence>
    <xs:element name="UserID" type="xs:unsignedLong"/>
    <xs:element name="CorrID" type="xs:unsignedLong"/>
    <xs:element name="CallbackServerURL" type="xs:anyURI"/>
    <xs:element name="Message" type="CommunicationReplyMessageType"/>
  </xs:sequence>
</xs:complexType>
<!--CommunicationProtocolType-->
<!--=====
ResponseTypes
=====-->
<!--GetCapabilitiesResponseType-->
<xs:complexType name="GetCapabilitiesResponseType">
  <xs:sequence>
    <xs:any/>
  </xs:sequence>

```

```

</xs:complexType>
<!--RegisterUserResponseType-->
<xs:complexType name="RegisterUserResponseType">
  <xs:annotation>
    <xs:documentation>unique user id, provided by the WNS</xs:documentation>
  </xs:annotation>
  <xs:sequence>
    <xs:element name="UserID" type="xs:unsignedLong"/>
  </xs:sequence>
</xs:complexType>
<!--DoNotificationResponseType-->
<xs:complexType name="DoNotificationResponseType">
  <xs:sequence>
    <xs:element name="Status">
      <xs:simpleType>
        <xs:restriction base="xs:string">
          <xs:enumeration value="Notification sending successful"/>
          <xs:enumeration value="Notification sending failed"/>
          <xs:enumeration value="Notification timed out"/>
        </xs:restriction>
      </xs:simpleType>
    </xs:element>
  </xs:sequence>
</xs:complexType>
<!--DoCommunicationResponseType-->
<xs:complexType name="DoCommunicationResponseType">
  <xs:sequence>
    <xs:element name="Status">
      <xs:simpleType>
        <xs:restriction base="xs:string">
          <xs:enumeration value="Communication sending successful"/>
          <xs:enumeration value="Communication sending failed"/>
          <xs:enumeration value="Communication sending timed out"/>
        </xs:restriction>
      </xs:simpleType>
    </xs:element>
  </xs:sequence>
</xs:complexType>
<!--DoReplyResponseType-->
<xs:complexType name="DoReplyResponseType">
  <xs:sequence>
    <xs:element name="Status">
      <xs:simpleType>
        <xs:restriction base="xs:string">
          <xs:enumeration value="Reply forwarding successfully"/>
          <xs:enumeration value="Reply forwarding failed"/>
          <xs:enumeration value="Reply forwarding timed out"/>
        </xs:restriction>
      </xs:simpleType>
    </xs:element>
  </xs:sequence>
</xs:complexType>
<!--=====
Further Types & Elements
=====-->
<!--CommunicationProtocolType-->
<xs:complexType name="CommunicationProtocolType">
  <xs:group ref="CommunicationProtocolGroup"/>
</xs:complexType>

```

```

<!--CommunicationProtocolGroup: let's you chose the protocol type-->
<xs:group name="CommunicationProtocolGroup">
  <xs:choice>
    <xs:element name="Email" type="mailtoType"/>
    <xs:element name="URL" type="xs:anyURI"/>
    <xs:element name="SMS" type="xs:unsignedLong"/>
    <xs:element name="Phone" type="xs:unsignedLong"/>
    <xs:element name="Fax" type="xs:unsignedLong"/>
    <xs:element name="InstantMessaging" type="xs:string"/>
  </xs:choice>
</xs:group>
<!--mailtoType-->
<xs:simpleType name="mailtoType">
  <xs:restriction base="xs:string">
    <xs:pattern value="\w[-.\w]*@((([A-Za-z0-9])|([A-Za-z0-9][A-Za-z0-9\-\-]*[A-Za-z0-9]))\.[A-Za-
z]{2,3})"/>
  </xs:restriction>
</xs:simpleType>
<!--Keywordlist-->
<xs:complexType name="Keywordlist">
  <xs:sequence>
    <xs:element name="Keyword" type="xs:string" minOccurs="0" maxOccurs="unbounded"/>
  </xs:sequence>
</xs:complexType>
<!--ContactInformation-->
<xs:complexType name="ContactInformationType">
  <xs:sequence>
    <xs:element name="ContactPerson" type="xs:string"/>
    <xs:element name="ContactOrganization" type="xs:string"/>
    <xs:element name="Email" type="mailtoType"/>
  </xs:sequence>
</xs:complexType>
<!--RequestType for getCapabilities-->
<xs:complexType name="RequestType">
  <xs:sequence>
    <xs:element name="GetCapabilities" type="RequestDescriptionType"/>
    <xs:element name="RegisterUser" type="RequestDescriptionType"/>
    <xs:element name="DoNotification" type="RequestDescriptionType"/>
    <xs:element name="DoCommunication" type="RequestDescriptionType" minOccurs="0"/>
    <xs:element name="DoReply" type="RequestDescriptionType" minOccurs="0"/>
  </xs:sequence>
</xs:complexType>
<!--RequestDescriptionType-->
<xs:complexType name="RequestDescriptionType">
  <xs:sequence>
    <xs:element name="Format" type="xs:string"/>
    <xs:element name="DCPType" type="DCPType"/>
  </xs:sequence>
</xs:complexType>
<!--DCPType-->
<xs:complexType name="DCPType">
  <xs:sequence>
    <xs:element name="HTTP">
      <xs:complexType>
        <xs:sequence>
          <xs:element name="Get" minOccurs="0"/>
          <xs:element name="Post" minOccurs="0"/>
        </xs:sequence>
      </xs:complexType>
    </xs:element>
  </xs:sequence>
</xs:complexType>

```

```
        </xs:element>
      </xs:sequence>
    </xs:complexType>
    <!--Subtypes of DCPTType-->
  </xs:schema>
```

Annex C: XML Message Schema (normative)

C.1 wnsMessageSchema

```

<!-- edited with XML Spy v4.4 U (http://www.xmlspy.com) by Institut für Geoinformatik (Institut für
Geoinformatik) -->
<xs:schema targetNamespace="http://www.opengis.net/wns"
xmlns:xs="http://www.w3.org/2001/XMLSchema" xmlns="http://www.opengis.net/wns"
elementFormDefault="qualified" attributeFormDefault="unqualified">
  <!--=====
  ELEMENTS
  =====>
  <!--NotificationMessage-->
  <xs:element name="NotificationMessage" type="NotificationMessageType"/>
  <!--CommunicationRequestMessage-->
  <xs:element name="CommunicationRequestMessage" type="CommunicationRequestMessageType"/>
  <!--CommunicationReplyMessage-->
  <xs:element name="CommunicationReplyMessage" type="CommunicationReplyMessageType"/>
  <!--=====
  GROUPS
  =====>
  <!--MessageParameterGroup-->
  <xs:group name="MessageParameterGroup">
    <xs:sequence>
      <xs:element name="CorrID" type="xs:long"/>
      <xs:element name="Key" type="xs:string"/>
      <xs:choice minOccurs="0">
        <xs:element name="String" type="xs:string"/>
        <xs:element name="Integer" type="xs:integer"/>
        <xs:element name="Float" type="xs:float"/>
        <xs:element name="Long" type="xs:long"/>
        <xs:element name="Double" type="xs:double"/>
        <xs:element name="URI" type="xs:anyURI"/>
      </xs:choice>
    </xs:sequence>
  </xs:group>
  <!--=====
  TYPES
  =====>
  <!--NotificationMessageType-->
  <xs:complexType name="NotificationMessageType">
    <xs:sequence>
      <xs:element name="Type">
        <xs:simpleType>
          <xs:restriction base="xs:string">
            <xs:whiteSpace value="preserve"/>
            <xs:enumeration value="Operation completed"/>
            <xs:enumeration value="Operation failed"/>
            <xs:enumeration value="Operation cancelled"/>
            <xs:enumeration value="Operation delayed"/>
            <xs:enumeration value="New data available"/>
          </xs:restriction>
        </xs:simpleType>
      </xs:element>
    </xs:sequence>
  </xs:complexType>

```

File name: 03-008r2

© OGC 2003 – All rights reserved.

```

    </xs:element>
    <xs:element name="MessageParameter" maxOccurs="unbounded">
      <xs:complexType>
        <xs:sequence>
          <xs:group ref="MessageParameterGroup"/>
        </xs:sequence>
      </xs:complexType>
    </xs:element>
  </xs:sequence>
</xs:complexType>
<!--CommunicationRequestMessageType-->
<xs:complexType name="CommunicationRequestMessageType">
  <xs:sequence>
    <xs:element name="Action" type="CommunicationAction"/>
    <xs:element name="MessageParameter" maxOccurs="unbounded">
      <xs:complexType>
        <xs:sequence>
          <xs:group ref="MessageParameterGroup"/>
          <xs:element name="Unit" type="xs:string"/>
          <xs:element name="Options" type="xs:anySimpleType"/>
        </xs:sequence>
      </xs:complexType>
    </xs:element>
  </xs:sequence>
</xs:complexType>
<!--CommunicationReplyMessageType-->
<xs:complexType name="CommunicationReplyMessageType">
  <xs:sequence>
    <xs:element name="Action" type="xs:string"/>
    <xs:element name="MessageParameter" maxOccurs="unbounded">
      <xs:complexType>
        <xs:sequence>
          <xs:group ref="MessageParameterGroup"/>
          <xs:element name="Value" type="xs:anySimpleType"/>
        </xs:sequence>
      </xs:complexType>
    </xs:element>
  </xs:sequence>
</xs:complexType>
<!--=====
SIMPLE TYPES
=====-->
<!--CommunicationAction-->
<xs:simpleType name="CommunicationAction">
  <xs:restriction base="xs:string">
    <xs:whiteSpace value="preserve"/>
    <xs:enumeration value="Information needed"/>
    <xs:enumeration value="Allowance to proceed"/>
    <xs:enumeration value="Allowance to abort"/>
  </xs:restriction>
</xs:simpleType>
</xs:schema>

```


Annex D: Miscellaneous XML Schema (informative)

D.1 OGC Exception Schema

The following XML Schema describes the OGC Exception format.

```
<?xml version="1.0" encoding="UTF-8"?>
<xsd:schema targetNamespace="http://www.opengis.net/ogc"
xmlns:ogc="http://www.opengis.net/ogc" xmlns:xsd="http://www.w3.org/2001/XMLSchema"
elementFormDefault="qualified">
  <xsd:element name="ServiceExceptionReport">
    <xsd:annotation>
      <xsd:documentation>
        The ServiceExceptionReport element contains one
        or more ServiceException elements that describe
        a service exception.
      </xsd:documentation>
    </xsd:annotation>
    <xsd:complexType>
      <xsd:sequence>
        <xsd:element name="ServiceException" type="ogc:ServiceExceptionType" minOccurs="0"
maxOccurs="unbounded">
          <xsd:annotation>
            <xsd:documentation>
              The Service exception element is used to describe
              a service exception.
            </xsd:documentation>
          </xsd:annotation>
        </xsd:element>
      </xsd:sequence>
      <xsd:attribute name="version" type="xsd:string" fixed="1.2.0"/>
    </xsd:complexType>
  </xsd:element>
  <xsd:complexType name="ServiceExceptionType">
    <xsd:annotation>
      <xsd:documentation>
        The ServiceExceptionType type defines the ServiceException
        element. The content of the element is an exception message
        that the service wished to convey to the client application.
      </xsd:documentation>
    </xsd:annotation>
    <xsd:simpleContent>
      <xsd:extension base="xsd:string">
        <xsd:attribute name="code" type="xsd:string">
          <xsd:annotation>
            <xsd:documentation>
              A service may associate a code with an exception
              by using the code attribute.
            </xsd:documentation>
          </xsd:annotation>
        </xsd:attribute>
      </xsd:extension>
    </xsd:simpleContent>
  </xsd:complexType>
</xsd:schema>
```

```

    </xsd:documentation>
  </xsd:annotation>
</xsd:attribute>
<xsd:attribute name="locator" type="xsd:string">
  <xsd:annotation>
    <xsd:documentation>
      The locator attribute may be used by a service to
      indicate to a client where in the client's request
      an exception was encountered. If the request included
      a 'handle' attribute, this may be used to identify the
      offending component of the request. Otherwise the
      service may try to use other means to locate the
      exception such as line numbers or byte offset from the
      beginning of the request, etc ...
    </xsd:documentation>
  </xsd:annotation>
</xsd:attribute>
</xsd:extension>
</xsd:simpleContent>
</xsd:complexType>
</xsd:schema>

```

D.2 XLink Schema

The following XML Schema is used by the XForms 1.0 schema.

```

<?xml version="1.0" encoding="UTF-8"?>
<xsd:schema targetNamespace="http://www.w3.org/1999/xlink"
  xmlns:xsd="http://www.w3.org/2001/XMLSchema" xmlns:xl="http://www.w3.org/1999/xlink"
  elementFormDefault="qualified" attributeFormDefault="unqualified">
  <!--
    This schema is in no way normative for XLink; it functions only as a part of the
    Schema for XForms to allow proper assessment of XForms documents and fragments.
  -->

```

See the XForms specification for details.

```

-->
  <xsd:attribute name="href" type="xsd:anyURI"/>
  <xsd:attribute name="type" type="xsd:string"/>
  <xsd:attribute name="role" type="xsd:anyURI"/>
  <xsd:attribute name="arcrole" type="xsd:anyURI"/>
  <xsd:attribute name="title" type="xsd:string"/>
  <xsd:attribute name="actuate">
    <xsd:simpleType>
      <xsd:restriction base="xsd:string">
        <xsd:enumeration value="onLoad"/>
        <xsd:enumeration value="onRequest"/>
        <xsd:enumeration value="other"/>
        <xsd:enumeration value="none"/>
      </xsd:restriction>
    </xsd:simpleType>
  </xsd:attribute>

```

```

<xsd:attribute name="show">
  <xsd:simpleType>
    <xsd:restriction base="xsd:string">
      <xsd:enumeration value="new"/>
      <xsd:enumeration value="replace"/>
      <xsd:enumeration value="embed"/>
      <xsd:enumeration value="other"/>
      <xsd:enumeration value="none"/>
    </xsd:restriction>
  </xsd:simpleType>
</xsd:attribute>
<xsd:attribute name="label" type="xsd:NCName"/>
<xsd:attribute name="from" type="xsd:NCName"/>
<xsd:attribute name="to" type="xsd:NCName"/>
</xsd:schema>

```

D.3 XML Events Schema

The following XML Schema is used by the XForms 1.0 schema.

```

<?xml version="1.0" encoding="UTF-8"?>
<!-- edited with XML Spy v4.0.1 U (http://www.xmlspy.com) by Micah Dubinko (XForms WG) -->
<xsd:schema targetNamespace="http://www.w3.org/2001/xml-events"
  xmlns:xsd="http://www.w3.org/2001/XMLSchema" xmlns:ev="http://www.w3.org/2001/xml-
  events" attributeFormDefault="unqualified">
  <!--

```

This schema is in no way normative for XML Events; it functions only as a part of the Schema for XForms to allow proper assessment of XForms documents and fragments.

See the XForms specification for details.

```

-->
  <xsd:attribute name="event" type="xsd:NMTOKEN"/>
  <xsd:attribute name="observer" type="xsd:IDREF"/>
  <xsd:attribute name="target" type="xsd:IDREF"/>
  <xsd:attribute name="handler" type="xsd:anyURI"/>
  <xsd:attribute name="phase">
    <xsd:simpleType>
      <xsd:restriction base="xsd:string">
        <xsd:enumeration value="capture"/>
        <xsd:enumeration value="default"/>
      </xsd:restriction>
    </xsd:simpleType>
  </xsd:attribute>
  <xsd:attribute name="propagate">
    <xsd:simpleType>
      <xsd:restriction base="xsd:string">
        <xsd:enumeration value="stop"/>
        <xsd:enumeration value="continue"/>
      </xsd:restriction>
    </xsd:simpleType>
  </xsd:attribute>

```

```
<xsd:attribute name="defaultAction">
  <xsd:simpleType>
    <xsd:restriction base="xsd:string">
      <xsd:enumeration value="cancel"/>
      <xsd:enumeration value="perform"/>
    </xsd:restriction>
  </xsd:simpleType>
</xsd:attribute>
</xsd:schema>
```

Bibliography

- [1] OGC(2003): Jeff Lansing [ed.], *Sensor Planning Service*, OGC 03-011r1.
- [2] OGC(2002): Josh Lieberman [ed.], *Service Information Model*, OGC 02-055.