Open Geospatial Consortium

Date: 2011-03-28

Reference number of this document: **OGC 09-000**

OGC name of this OGC® project document: **http://www.opengis.net/doc/IS/SPS/2.0**

Version: 2.0

Category: OpenGIS® Implementation Standard

Editor(s): Ingo Simonis, Johannes Echterhoff

# OGC® Sensor Planning Service Implementation Standard

**Warning**

This document is an OGC Member approved international standard. This document is available on a royalty free, non-discriminatory basis. Recipients of this document are invited to submit, with their comments, notification of any relevant patent rights of which they are aware and to provide supporting documentation.

| | |
|---|---|
| Document type: | OpenGIS® Standard |
| Document subtype: | Interface |
| Document stage: | Approved |
| Document language: | English |

## License Agreement

# Contents

Page

# Figures

Page

# Tables

Page

## i. Abstract

The OpenGIS® Sensor Planning Service Interface Standard (SPS) defines interfaces for queries that provide information about the capabilities of a sensor and how to task the sensor. The standard is designed to support queries that have the following purposes: to determine the feasibility of a sensor planning request; to submit and reserve/commit such a request; to inquire about the status of such a request; to update or cancel such a request; and to request information about other OGC Web services that provide access to the data collected by the requested task. This is one of the OGC Sensor Web Enablement (SWE) [http://www.opengeospatial.org/ogc/markets-technologies/swe] suite of standards.

## ii. Keywords

ogcdoc, sps, swe, swes, gml

## iii. Preface

This standard is part of OGC's Sensor Web Enablement (SWE) activity. It is the successor of SPS version 1.0.0 (OGC 07-014r3).

Suggested additions, changes, and comments on this report are welcome and encouraged. Such suggestions may be submitted using the OGC online change request application:

http://portal.opengeospatial.org/public_ogc/change_request.php

## iv. Document terms and definitions

This document uses the standard terms defined in Subclause 5.3 of [OGC 06-121r3], which is based on the ISO/IEC Directives, Part 2. Rules for the structure and drafting of International Standards. In particular, the word "shall" (not "must") is the verb form used to indicate a requirement to be strictly followed to conform to this standard.

## v. Submitting organizations

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

a) International Geospatial Services Institute GmbH (iGSI)

b) Spot Image, S.A.

c) SeiCorp, Inc.

## vi. Document contributor contact points

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

| CONTACT | COMPANY | EMAIL |
|---|---|---|
| Johannes Echterhoff (editor) | iGSI | johannes.echterhoff@igsi.eu |
| Ingo Simonis (editor) | iGSI | ingo.simonis@igsi.eu |
| Alexandre Robin | Spot Image, S.A. | alexandre.robin@spotimage.fr |
| Jim Greenwood | SeiCorp, Inc. | jgreenwood@Seicorp.com |

## vii. Issues

Any 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)]

## viii. Changes to the OGC Abstract Specification

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

## ix. Future work

Future Work will mainly address the abstraction of the currently operation-based specification to a behavior-based specification. Then, all binding approaches, such as SOAP or REST, will be defined in extensions to the core specification.

Direct subscriptions together with a tasking request are currently out-of-scope for the standard. This can lead to situations in which a client interested in receiving notifications about that tasking request or implied task misses published notifications. Functionality to enable performing a tasking request and directly subscribing for notification of related events should be realized in the future – either in a revision of the standard itself or as an extension.

Conditional dependencies between parameters (example: if parameter A has value Y then parameter B may only have value Z etc) can be supported in future versions of this standard. The functionality could also be added through extensions.

During the development of this standard, the OGC has changed its specification document template and development policies. This standard reflects those changes as much as possible, but full compliancy to the new OGC specification model needs to be achieved in future releases.

## Foreword

This SPS 2.0 standard replaces version 1.0 of the SPS standard (OGC 07-014r3). Version 2 revises and extends version 1. Though the general functionality of the service is preserved, the interface defined in this document is not backwards compatible to that of SPS version 1.0.0

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

This document includes three annexes. Annexes A and B are normative, and Annex C is informative.

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

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

## Introduction

The Sensor Planning Service (SPS) is intended to provide a standard interface to task collection assets (i.e., satellites, other sensors, and other information gathering assets) and to the support systems that surround them. Not only will different kinds of assets with differing capabilities be supported, but also different kinds of request processing systems, which may or may not provide access to the different stages of planning, scheduling, tasking, collection, processing, archiving, and distribution of requests and the resulting observation data and information that is the result of the requests. The SPS is designed to be flexible enough to handle such a wide variety of configurations.

This standard begins with an abstract overview of the SPS interface before describing the information model for operation requests and responses in a platform-neutral manner and subsequently applying this model to a specific binding (SOAP in this case).

# OpenGIS® Sensor Planning Service Implementation Standard

## 1 Scope

This OGC™ standard establishes the baseline of Sensor Planning Service functionality and requirements describing this functionality.

This document defines service interfaces for parameterizing – also called tasking – of taskable devices, such as sensors or actuators.

It defines terms and their synonyms relevant to the device control domain (task, tasking, sensor, asset etc).

The interfaces defined in this document provide functionality to:

- Retrieve metadata about the service (to understand service capabilities)
- Describe the parameterization options available for the sensor
- Check if the service is capable of performing a planned task (feasibility check)
- Reserve resources required to perform a planned task for a certain amount of time (useful for handling combined tasking of multiple sensors)
- Instruct the service to execute a task for a sensor
- Retrieve the status of a task
- Update a task
- Retrieve information about access to the data collected by a sensor – also on a per-task basis
- Cancel a task

This document leverages functionality defined by other standards, which enables:

- Provision and management of sensor descriptions
- Publication of and subscription for information on events recognized by the service – for example to automatically notify clients of new information on their task (that new data is available, that it was completed etc.)

The first sections of this document describe the theoretical background to understand SPS functionalities. After that, the common information and communication model for SPS is specified.

This OGC™ standard is applicable to all use cases in which one or more sensors or sensor systems can or need to be parameterized in order to influence the measurement process and therefore the information gathered by assets or systems.

# 2 Compliance

## 2.1 Specification identifier

All requirements and conformance-classes described in this document are owned by the standard identified as http://www.opengis.net/spec/SPS/2.0.

## 2.2 Conformance Classes

The following Table 1 specifies the conformance classes defined by this standard.

Compliance with a given conformance class shall be checked using the relevant tests specified in Annex A (normative).

**Table 1 — SPS Conformance Classes**

| Conformance class name | Conformance class identifier | Operation and/or behavior |
|---|---|---|
| Core | http://www.opengis.net/spec/SPS/2.0/conf/Core | The server implements the GetCapabilities, DescribeTasking, Submit, GetStatus, GetTask and DescribeResultAccess operations as defined by this standard as well as the conformance classe(s) that this conformance class depends upon (see Figure 1). |
| State Logger | http://www.opengis.net/spec/SPS/2.0/conf/StateLogger | The server implements state logger functionality as defined by this standard as well as the conformance classe(s) that this conformance class depends upon (see Figure 1). |
| Feasibility Controller | http://www.opengis.net/spec/SPS/2.0/conf/FeasibilityController | The server implements the GetFeasibility operation as well as the conformance classe(s) that this conformance class depends upon (see Figure 1). |
| Reservation Manager | http://www.opengis.net/spec/SPS/2.0/conf/ReservationManager | The server implements the Reserve and Confirm operations as well as the conformance classe(s) that this conformance class depends upon (see Figure 1). |
| Task Updater | http://www.opengis.net/spec/SPS/2.0/conf/TaskUpdater | The server implements the Update operation as well as the conformance classe(s) that this conformance class depends upon (see Figure 1). |
| Task Canceller | http://www.opengis.net/spec/SPS/2.0/conf/TaskCanceller | The server implements the Cancel operation as well as the conformance classe(s) that this conformance class depends upon (see Figure 1). |
| Basic PubSub | http://www.opengis.net/spec/SPS/2.0/conf/BasicPubSub | The server implements publish/subscribe functionality and publish SPS events as defined in this standard as well as the conformance classe(s) that this conformance class depends upon (see Figure 1). |
| ChannelBased PubSub | http://www.opengis.net/spec/SPS/2.0/conf/ChannelBasedPubSub | The server implements publish/subscribe functionality and publish SPS events on the SPS channels/topics as defined by this standard as well as the conformance classe(s) that this conformance class depends upon (see Figure 1). |
| XML Encoding | http://www.opengis.net/spec/SPS/2.0/conf/XMLEncoding | The server encodes the data types from the conceptual model in XML as defined by this standard as well as the conformance classe(s) that this conformance class depends upon (see Figure 1). |
| SOAP | http://www.opengis.net/spec/SPS/2.0/conf/SOAP | The server implements the SOAP binding as defined in this standard as well as the conformance classe(s) that this conformance class depends upon (see Figure 1). |

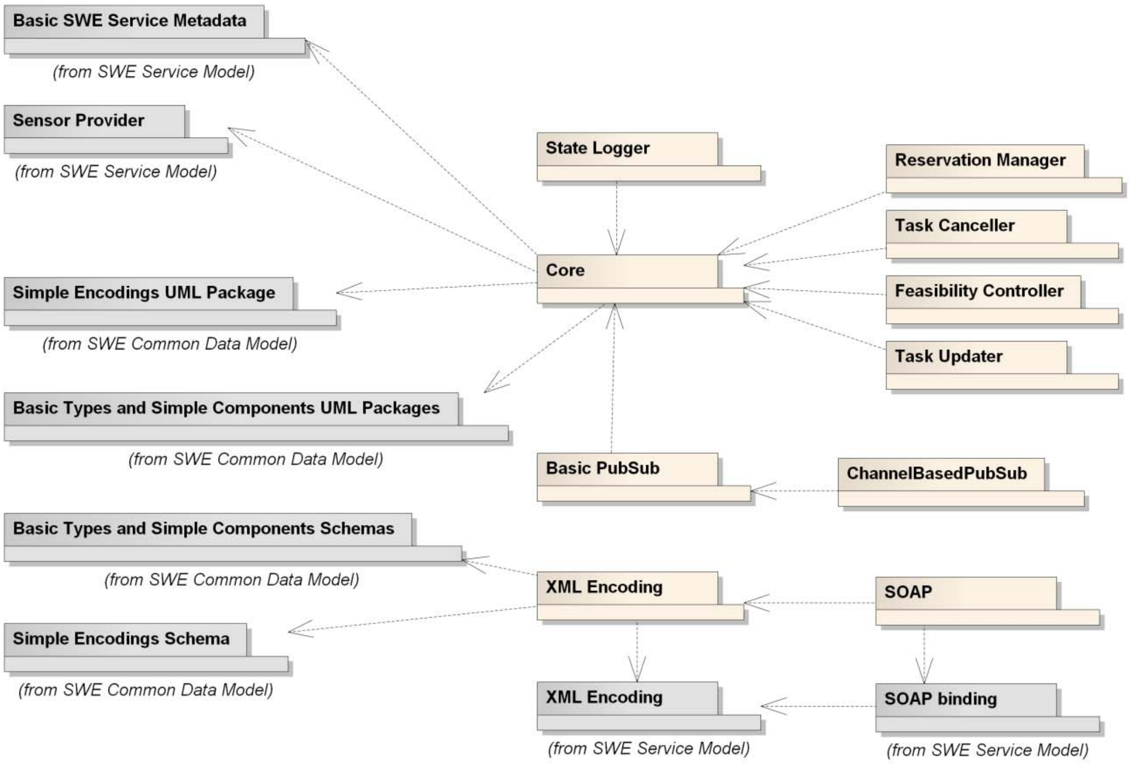


**Figure 1 - SPS Conformance Classes and their dependencies**

## 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*

ISO 19108:2002, *Geographic information — Temporal schema*

ISO 19136:2007, *Geographic information -- Geography Markup Language (GML)*

(see also: OpenGIS® Encoding Standard *Geography Markup Language*, OGC document 07-036)

ISO DIS 19156:2010, *Geographic information — Observations and Measurements*

OGC 06-121r3, *OpenGIS® Web Services Common Specification*

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

OpenGIS® Encoding Standard, *SWE Common Data Model*, OGC document number 08-094

OpenGIS® Implementation Standard, *SWE Service Model*, OGC document number 09-001

NOTE This SWE Service Model standard contains a list of normative references that are also applicable to this Implementation Standard.

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

## 4 Terms and definitions

For the purposes of this standard, the terms and definitions specified in clause 4 of [OGC 06-121r3] shall apply, as well as the terms and definitions specified in clause 4 of [09-001]. In addition, the following terms and definitions apply.

### 4.1 Asset

**synonyms: sensor, simulation**
an available means of collecting information

### 4.2 Asset Management System

**synonyms: acquisition system, asset support system**
system for controlling the effective utilization of an asset

### 4.3 Collection

act of gathering information

NOTE In the context of SPS, the term is usually perceived having the process of gathering information in mind. Another interpretation is the aggregation of the results of one or more collection processes.

### 4.4 Requirement

something that is necessary in advance

### 4.5 Simulation

use of models to investigate time dependent processes

### 4.6 Task

(conceptual) resource that represents a SPS assignment. It includes the (possibly empty) set of tasking parameters.

### 4.7 Tasking

parameterizing an asset; can be done by sending one or more tasking requests

### 4.8 Tasking request

request with certain tasking semantics that contains tasking parameters

NOTE In the context of SPS, the GetFeasibility, Reserve, Submit and Update requests are tasking requests.

### 4.9 Tasking Parameter

parameter that has an influence on the parameterization of an asset

# 5 Conventions

## 5.1 Abbreviated terms

Most of the abbreviated terms listed in Subclause 5.1 of the OWS Common Implementation Specification [OGC 06-121r3] apply to this document, plus the following abbreviated terms.

| | |
|---|---|
| AOI | Area Of Interest |
| FES | Filter Encoding Specification |
| AM | Asset Management |
| O&M | Observation and Measurement |
| SensorML | Sensor Model Language |
| SOS | Sensor Observation Service |
| SPS | Sensor Planning Service |
| SWE | Sensor Web Enablement |
| SWE Common | SWE Common Data Model |
| SWES | SWE Service Model |
| WCS | Web Coverage Service |
| WMS | Web Map Service |
| WNS | Web Notification Service |

### 5.2 UML notation

Diagrams that appear in this standard are presented using the Unified Modeling Language (UML) static structure diagram, as described in Subclause 5.2 of [OGC 06-121r3].

NOTE Packages and data types from foreign namespaces or data types from packages other than the one under consideration are shown with grey background unless they are given only as types of attributes from classes in the model defined in this specification. Interfaces are shown with light turquoise background.

### 5.3 Platform-neutral and platform-specific standards

For compliance with Clause 10 of OGC Topic 12 and ISO 19119, this standard follows the pattern defined in subclause 5.4 of [OGC 06-121r3]. That is, model elements are specified in platform-neutral fashion first, using tables that serve as data dictionaries for the UML model (see clause 5.4 of this document). Platform-specific encodings of these model elements are provided in separate clauses or documents. The XML Schema encoding has automatically been generated using the rules defined in clause 24 of [OGC 09-001].

This document specifies platform-specific encodings appropriate for a SOAP/WSDL operation binding. However, the model as well as its XML Schema encoding (and other data) can be used by other bindings as well, like REST(ful) or POX (Plain Old XML) over HTTP (using XML or KVP encoding).

### 5.4 Data dictionary tables

The UML model data dictionary is specified herein in a series of tables. The contents of the columns in these tables are described in table 1 of [OGC 06-121r3]. The contents of these data dictionary tables are normative, including any table footnotes.

### 5.5 Classes imported from other specifications with predefined XML encoding

This specification uses an automatic mapping approach from the UML model to the XML Schema encoding. The approach is described in chapter 24 of [OGC 09-001]. As shown in Figure 13, this standard uses types defined by other standards. For the mapping to XML Schema, the implementation instructions listed in table D.2 of [OGC 07-036] are used together with the instructions listed in Table 2, Table 3 and Table 4 in this standard and Table 4 from [OGC 09-001].

For an explanation of the table columns, see clause D.2.1 in OGC 07-036.

**Table 2 — Implementation of types from OWS Common [OGC 06-121r3]**

| UML class | object element | type | property type |
|---|---|---|---|
| AbstractMetadata | ows:AbstractMetaData | - | - |
| GetCapabilities | - | ows:GetCapabilitiesType | - |
| LanguageString | - | ows:LanguageStringType | - |
| OWSServiceMetadata | - | ows:CapabilitiesBaseType | - |
| ReferenceGroup | ows:ReferenceGroup | ows:ReferenceGroupType | - |

**Table 3 — Implementation of types from SWE Common Data Model [OGC 08-094]**

| UML class | object element | type | property type |
|---|---|---|---|
| AbstractDataComponent | swe:AbstractDataComponent | swe:AbstractDataComponentType | swe:AbstractDataComponentPropertyType |
| AbstractEncoding | swe:AbstractEncoding | swe:AbstractEncodingType | swe:AbstractEncodingPropertyType |

**Table 4 — Implementation of types from SWE Service Model [OGC 09-001]**

| UML class | object element | type | property type |
|---|---|---|---|
| AbstractContents | swes:AbstractContents | swes:AbstractContentsType | swes:AbstractContentsPropertyType |
| AbstractOffering | swes:AbstractOffering | swes:AbstractOfferingType | swes:AbstractOfferingPropertyType |
| ExtensibleRequest | swes:ExtensibleRequest | swes:ExtensibleRequestType | swes:ExtensibleRequestPropertyType |
| ExtensibleResponse | swes:ExtensibleResponse | swes:ExtensibleResponseType | swes:ExtensibleResponsePropertyType |
| NotificationProducerMetadata | swes:NotificationProducerMetadata | swes:NotificationProducerMetadataType | swes:NotificationProducerMetadataPropertyType |

### 5.6 Namespace Conventions

This standard uses a number of namespace prefixes throughout; they are listed in Table 5. Note that the choice of any namespace prefix is arbitrary and not semantically significant.

**Table 5 — Prefixes and Namespaces used in this standard**

| Prefix | Namespace |
|---|---|
| gml | http://www.opengis.net/gml/3.2 |
| ows | http://www.opengis.net/ows/1.1 |
| soap11 | http://schemas.xmlsoap.org/soap/ |
| soap12 | http://www.w3.org/2003/05/soap-envelope |
| swe | http://www.opengis.net/swe/2.0 |
| swes | http://www.opengis.net/swes/2.0 |
| wsa | http://www.w3.org/2005/08/addressing |
| wsn-b | http://docs.oasis-open.org/wsn/b-2 |
| xs | http://www.w3.org/2001/XMLSchema |

# 6 Sensor Planning Service – Abstract Overview

## 6.1 Introduction

The operational context of the SPS is abstracted from, and therefore applies to, several areas of interest. In the scientific area there is a constant interplay between facts, and theories that explain the facts, which then gives rise to the need for more information in order to confirm and extend the theories. Similarly, in the medical area symptoms give rise to a need for information that calls for tests that support diagnosis. In the military area there is always a great deal that is unknown about a battle space, or about a theatre of operations other than war, which gives rise to needs for specific useful information. In the business area corporations and other non-governmental organizations have a need for global economic intelligence.

All of these areas have information needs, and the SPS is used to task assets to satisfy those needs. The SPS provides an interface to parameterize assets and asset management systems. It can be applied whenever a client is allowed to influence the internal processes of such a system. The SPS does not provide direct access to the information gathered by the system itself. This will be done via a SOS or some other (OGC) Web service. It rather serves as an interface layer to the parameterization interface of the underlying system (see Figure 2).

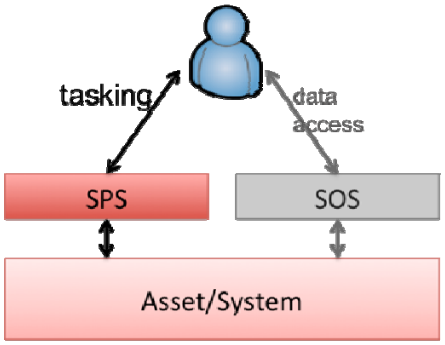


**Figure 2 — SWE Interface of an Asset (Management System)**

The SPS is an interface to a system of any complexity. The system itself is considered as a black box. In this black box, some sort of process gets executed that can be manipulated by setting specific parameters.

Example: a webcam takes pictures every minute. The SPS interface to this webcam allows modifying this time interval to anything between 10sec and 1hr.

Example: A more complex example is that of a satellite. The SPS interface allows to set a number of parameters, such as region of interest, time of interest, incidence angle with azimuth and elevation, ground resolution etc.

It is up to the SPS provider to define which parameterization options are available to clients via the SPS interface of the given service.

A system operator may even decide to have a chain of SPS instances to provide different capabilities to different types of users for the very same asset.

Example: Consider the webcam again. Authorized users may change the looking angle and the zoom value, whereas non-authorized users can only chose between three pre-defined settings.

This concept of abstraction levels is described in more detail in section 6.5.

### 6.2 Client Server Interaction

This section explains the typical interaction between an SPS client and service.

The interaction starts with the *GetCapabilities* request to explore what the service can offer. If additional information about a sensor is required, the *DescribeSensor* operation is used to retrieve all available information about the sensor (see Figure 3).

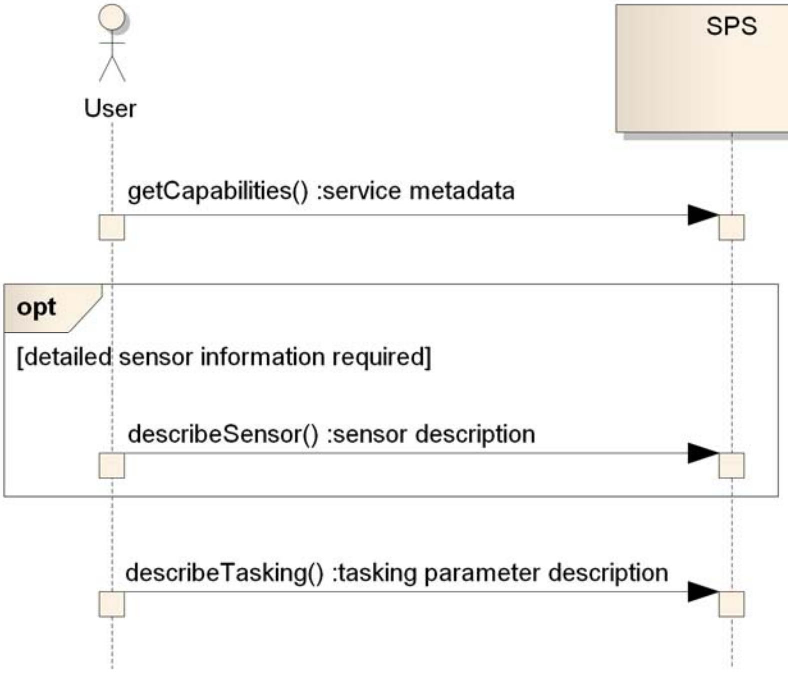

User
SPS
getCapabilities() :service metadata
opt
[detailed sensor information required]
describeSensor() :sensor description
describeTasking() :tasking parameter description

**Figure 3 — client server interaction part 1**

Next, the client needs to learn which parameters have to be set in order to task the sensor. The client sends a *DescribeTasking* request and receives a *DescribeTaskingResponse,* which defines syntax and semantic of each tasking parameter, including choices between different parameter settings, default values, and value ranges.

Note: For complex missions, a huge number of parameters might need to be set by clients. Alternatively, the service might only provide a choice between five preconfigured missions, and then there might only be a single parameter to be set by clients, even though the missions are very complex in nature. It fully depends on the service provider to define the parameters the client shall or may set. The definition of tasking profiles is encouraged to reflect the specific requirements of different communities in a consistent way. Nevertheless, tasking parameters are encoded using SWE Common and the SPS provider should add semantic annotation to them. This allows generic SPS clients to display more specific parameter descriptions including their semantic annotations so that a client can still meaningfully task an asset even if the client software does not provide any other support for this activity (which client software that was specifically developed to support certain tasking profiles will most probably do).

After the client learned about the tasking parameters, it can choose to either submit a tasking request (*Submit* operation) or to perform a feasibility check (*GetFeasibility* operation) – see Figure 4. Both operations create – if valid and accepted – a SPS assigment called task. Other operations allow to reserve and update a task, which will be discussed later on.

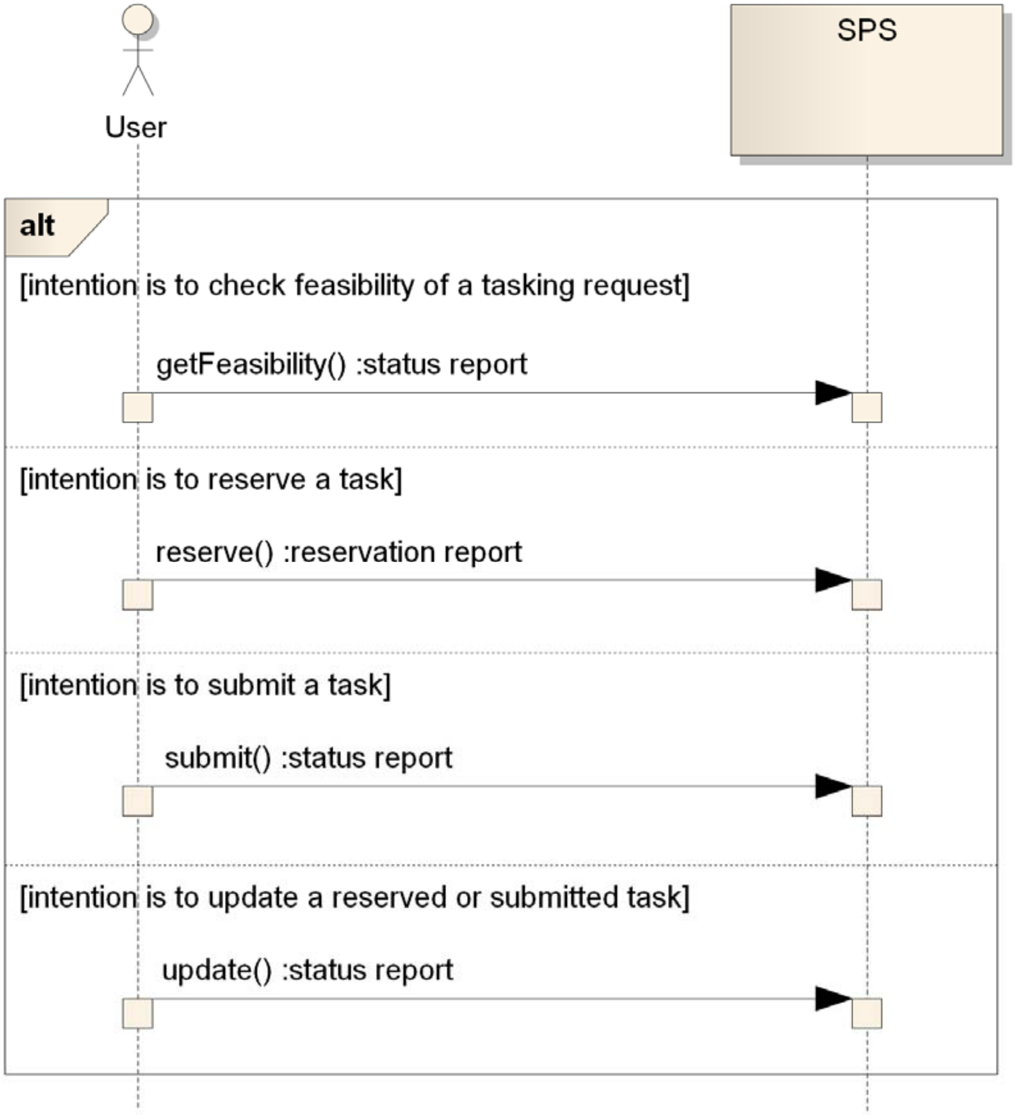

User
SPS
alt
[intention is to check feasibility of a tasking request]
getFeasibility() :status report
[intention is to reserve a task]
reserve() :reservation report
[intention is to submit a task]
submit() :status report
[intention is to update a reserved or submitted task]
update() :status report

**Figure 4 — client server interaction part 2**

Note: Before being accepted, each tasking request is checked for feasibility by the service. Even though a tasking request has been reported previously as feasible, it does not mean that this task is still feasible at the time of submitting the task. The façaded asset might have been tasked by someone else in the meantime or became unavailable (see clause 6.3.4 for further details).

The *GetFeasibilityResponse* contains a *StatusReport*, which indicates that the tasking request is or is not feasible. Optionally, the report lists alternative sets of tasking parameters that might help the client in formulating a tasking request that is feasible and that satisfies his information needs.

Independent of a prior *GetFeasibility* request, clients always send *Submit*/*Reserve* tasking requests with all required tasking parameters to the service. There is no option to use the identifier of a previous *GetFeasibility* tasking request in a subsequent *Submit*/*Reserve* tasking request. This lifts the burden from the service to store all *GetFeasibility* request payloads.[1]

If a task defined by the client is submitted to the service and is feasible, it is executed by the service.

A client may reserve a task using the *Reserve* operation. All resources required to execute the task are blocked by the service but execution does not start until the client explicitly confirms it (via the *Confirm* operation) – see Figure 5.

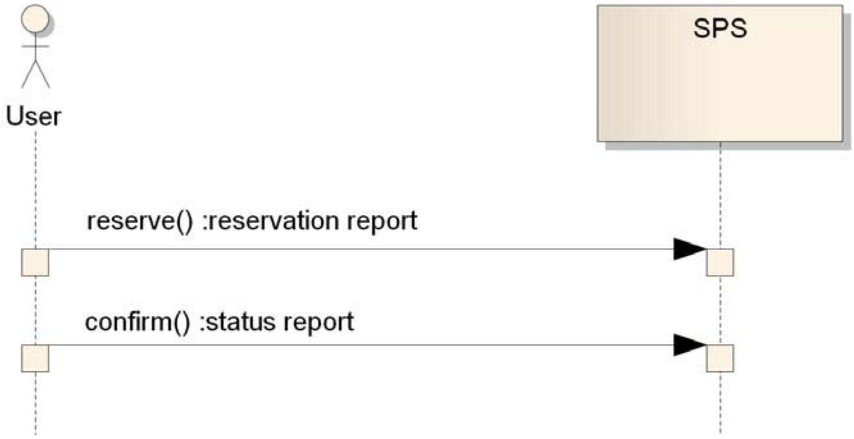


**Figure 5 — client server interaction part 3**

A reservation expires at a defined point in time at which a service can reclaim all resources blocked by the reservation.

Once a task is submitted/reserved, the client can *Update* or *Cancel* it. If a service cannot reserve/execute a request as provided by the client, it can provide a list of alternative parameter settings. A client can always ask for the current status of a task / tasking request via the *GetStatus* operation – see Figure 6.

[1] However, such behavior can be defined in an extension of this specification.

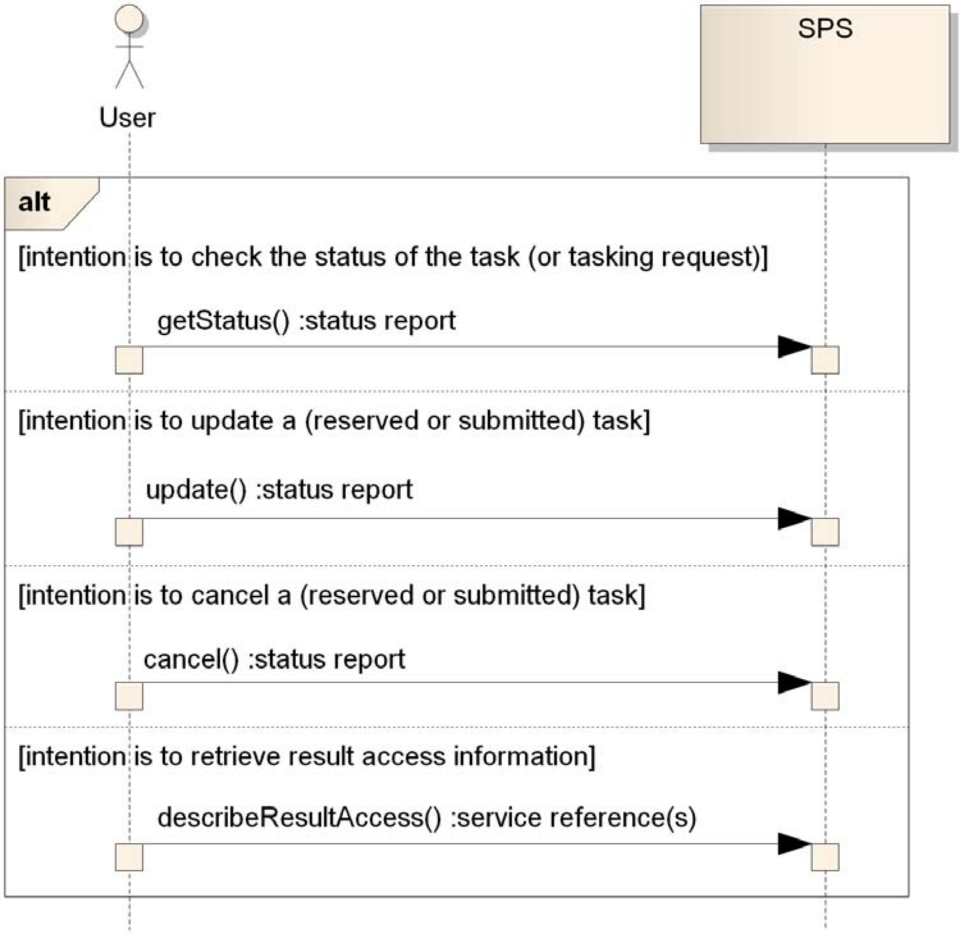

User
SPS
alt
[intention is to check the status of the task (or tasking request)]
getStatus() :status report
[intention is to update a (reserved or submitted) task]
update() :status report
[intention is to cancel a (reserved or submitted) task]
cancel() :status report
[intention is to retrieve result access information]
describeResultAccess() :service reference(s)

**Figure 6 — client server interaction part 4**

The SPS responds to *DescribeResultAccess* requests with references to all data that was produced for a given task, even if the task was cancelled or has failed. Clients can explore the references and retrieve the data gathered for this task.

The SPS service can also send notifications including *StatusReports* to inform interested clients about specific events, for example that new data has been published for a task, that a task was completed or has failed. See clause 8 for further details on asynchronous notification behavior.

## 6.3 Task – Concept and Handling

### 6.3.1 Introduction

The following sections discuss relevant aspects of tasking assets via SPS. The terms *task*, *tasking*, *tasking request* and *tasking parameter* are defined in clause 4 and thus are not defined again in this section.

### 6.3.2 Tasking Parameters

In order to parameterize an asset (management system), clients need to provide tasking parameters that influence the parameterization of the asset. Tasking parameters need to:

- describe full syntax as well as semantic of the parameter
- be extensible with metadata
- support optional parameters
- support choices between parameterization options
- support default values
- support provision of value constraints
- indicate whether the parameter can be used in a task update

The data types defined in SWE Common Data Model [OGC 08-094] satisfy these requirements and thus are used by SPS for defining tasking parameters and for encoding their values.

### 6.3.3 Tasking requests

To parameterize an asset, a client first has to initialize a set of tasking parameters, which constitute the task that the client is interested in getting executed by the SPS. The definition of the tasking parameters for a given asset can be retrieved via the SPS *describeTasking* operation.

Then, the client has required all information to formulate and send a *tasking request* to the SPS. Four types of tasking request are differentiated.

- *getFeasibility* – to determine whether the task (remember, a tasking request contains tasking parameters) can be executed by the service or not, depending upon its current state (see clause 6.3.4). This operation can also be used to check if an update of an existing task is feasible.
- *reserve* – to block all resources required to execute the task (if it is feasible) for a certain amount of time; this is useful to ensure that assets from different services can be tasked together (see clause 6.3.5). The reserved task either expires or gets confirmed to be executed by the client.
- *submit* – to instruct the service to execute the task (if it is feasible).
- *update* – to update (if feasible) the tasking parameters for a task that is already reserved or in execution.

Determining if a tasking request is feasible can take a long time, depending on the procedures executed by the service to evaluate the feasibility. In simple cases a trivial syntax check of the tasking parameters might be sufficent, while in other cases the service might have to wait for human approval.

However, clients may require information to be collected until a certain point in time and thus need the feasibility check to be completed some time before. Clients define this latest time when the response to a *getFeasibility* request must be available using the *lastestResponseTime* property in their request (see clause 7.3.1.3). If the service is not able to determine the feasibility of a tasking request until then, both the service and the client consider the tasking request as not feasible. This decision cannot be changed later on, i.e. any response sent by the service at a later stage is void.

A feasible tasking request means that it is accepted by the service. Depending on the request, the SPS either schedules a new task or provides a positive feasibility response without doing any further activity internally (see clause 6.3.6 for further details on state handling). *Tasking* in general can involve a sequence of tasking requests to have an asset gather the desired information.

Example: A client first checks the feasibility of a task (via the getFeasibility operation) – once a feasible set of tasking parameters has been determined, the task is submitted (via the submit operation) and is then updated multiple times to adjust the way the asset is gathering information. This can for example be a switch of the sampling frequency, or orientation of a remote sensor.

Clause 6.2 explained the client/server interactions for tasking an asset via the SPS in more detail.

### 6.3.4 Feasibility of a Task

To task a certain asset or system, tasking parameters have to be provided by the client. The definition of these parameters depends on the given asset and the parameterization abstraction level chosen by the service provider (see clause 6.5 for further information). A set of tasking parameters – or better: the set of values for these parameters – constitutes a tasking request (see clause 6.3.3). Before an SPS can accept such a tasking request, it has to check whether that task can be performed or not. This is called a feasibility check.

Feasibility of a task (or tasking request) shall be checked:

- by client request, i.e. if a client needs a pre-check for an intended task.
- if a client wants to reserve a task; a reserved task can be set to operational state by the client at any time until the reservation expires. The service has to ensure the full feasibility of the task during the reservation time. Under certain conditions, the service cannot maintain the feasibility of a reserved task, e.g. if the asset was tasked by someone else with higher priority. If the service supports publish/subscribe functionality as described in this specification then it informs the client that the reserved task has failed.
- when a task is submitted; the task can only be executed if it is feasible.
- whenever the client wants to perform an update of a reserved or submitted task.

The feasibility check performed by the service shall proof that the asset is capable of executing the intended task or task update. As such, during a feasibility study a service can check the items on the following (not exhaustive) list:

- syntax of tasking parameters
- presence of mandatory parameters
- validity of parameter configuration
- asset availability
- parameterization update is valid according to current execution state

The result of a feasibility check depends on the current state of the service and associated resources (e.g. the asset itself but also operators, support units, radio links, etc).

As an example, imagine a task intended to be performed during a certain interval of time by a specific asset (see Figure 7).

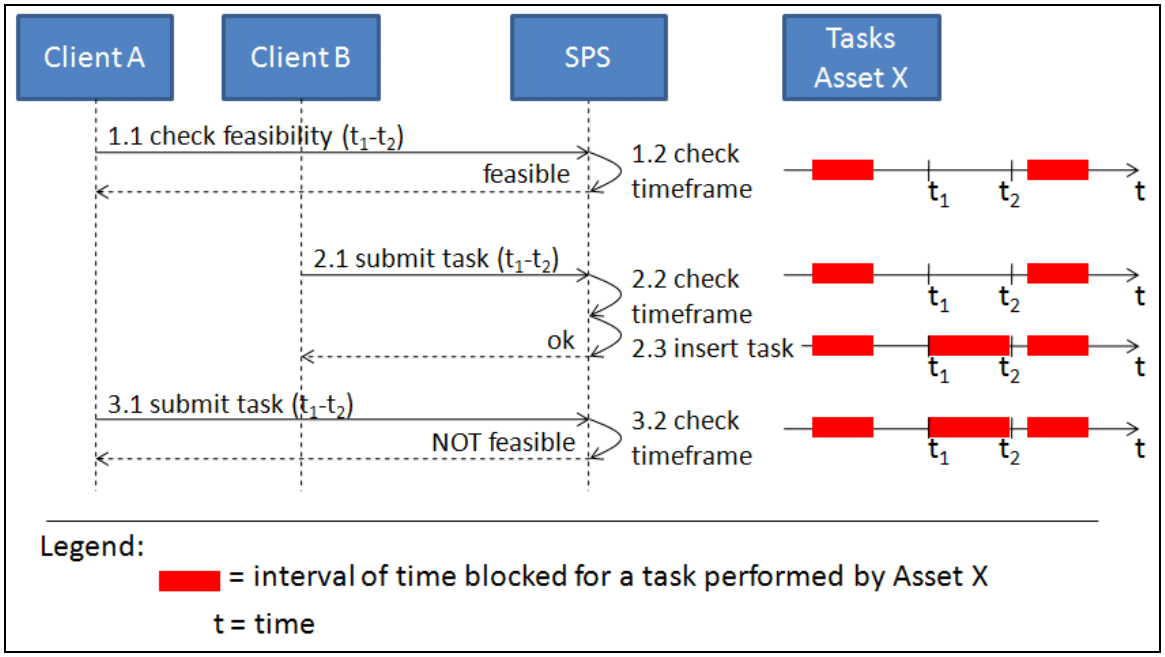

Client A
Client B
SPS
Tasks Asset X
1.1 check feasibility ($t_1$-$t_2$)
feasible
1.2 check timeframe
2.1 submit task ($t_1$-$t_2$)
2.2 check timeframe
ok
2.3 insert task
3.1 submit task ($t_1$-$t_2$)
NOT feasible
3.2 check timeframe
$t_1$
$t_2$
t
Legend:
🟥 = interval of time blocked for a task performed by Asset X
t = time

**Figure 7 — dynamics of a feasibility study result**

Client A checks the feasibility of a task to be executed in the time interval $t_1$-$t_2$ (1.1). The SPS checks the internal schedule for asset X (1.2) and recognizes that the time frame is not blocked by any other task. The SPS therefore responds that the task is feasible.

Before client A acts again, client B submits a task for asset X with the time interval $t_1$-$t_2$ (2.1). Again the SPS checks if the time frame is not already blocked by another task (2.2) and – as this is not the case – adds the task to the schedule of asset X (2.3). The time interval $t_1$-$t_2$ in the schedule of asset X is now blocked by the task from client B. The SPS has accepted the submission of the task from client B and sends an according response.

Now client A submits its tasking request (3.1). The SPS checks the internal schedule of asset X and recognizes that the time frame $t_1$-$t_2$ is already blocked by another task (from client B). It thus rejects the submission.

### 6.3.5 Reserving a Task

Clients can reserve tasks. This is useful e.g. if a client needs to task several assets (that are useful to him only if tasked in one go) via different services. A task can also be reserved before being submitted. Such a reservation actually represents a task for which all required resources are allocated by the service but which shall not be executed until the client confirms it.

In other words, the client puts a reserved task "on hold". This can be compared to a transaction in which the client first provides all parameterization details and finally confirms his task. The service shall check the feasibility of the task before it accepts the reservation. When the client confirms the reserved task, it is executed by the service.

The confirmation does not involve an additional feasibility check by the service because a reserved task shall be feasible until it expires. If a service can no longer guarantee the feasibility of a reserved task for any reason, the reservation shall fail.

The expiration time of a reserved task is defined by the service (optionally in agreement with the expiration time the client requested), thus making sure that resources are not blocked forever. The client can cancel a reservation if the service supports the cancel operation.

A reserved task can be updated if tasking parameters are updatable (see clause 6.3.2). Each update is subject to a feasibility check by the service.

### 6.3.6 State Handling

This section explains in more detail how an SPS handles the states of a tasking request and a task. This is done via two state machine diagrams. A formal documentation of the diagrams is given in clause 10.

NOTE: One or more of the transitions shown in the following state diagrams are triggered by events and have a specific effect, which is to notify interested clients about the event (*/Notify*). A service that implements publish/subscribe functionality can inform clients about these events.

When a client sends a tasking request to the service, it initiates the behavior shown in Figure 8.

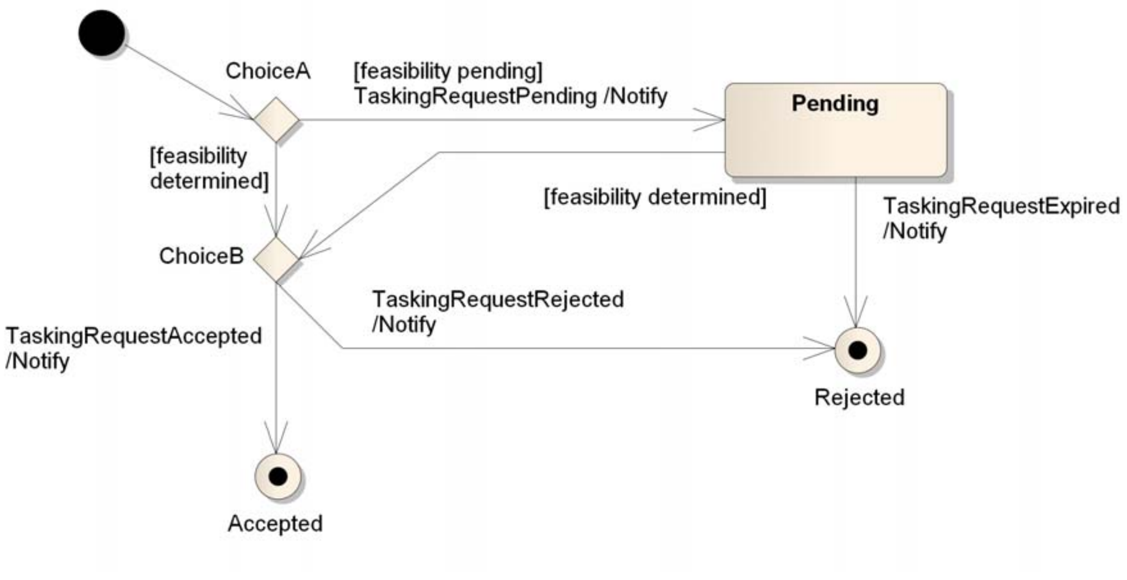


**Figure 8 — tasking request state machine diagram**

The decision if a tasking request is feasible or not is either directly available (*feasibility determined*), or requires more time (*feasibility pending*). The latter causes the tasking request to transition into the *Pending* state. If the feasibility of the (pending) tasking request cannot be determined before the request expires, then the tasking request automatically transitions from *Pending* into the *Rejected* state. Otherwise the tasking request gets back into the decision cycle: if the tasking request is feasible, then it shall be *Accepted* by the service – otherwise it shall be *Rejected*.

A task shall be scheduled by the service if the client reserved or submitted it. The following state diagram illustrates the state handling for such a task.

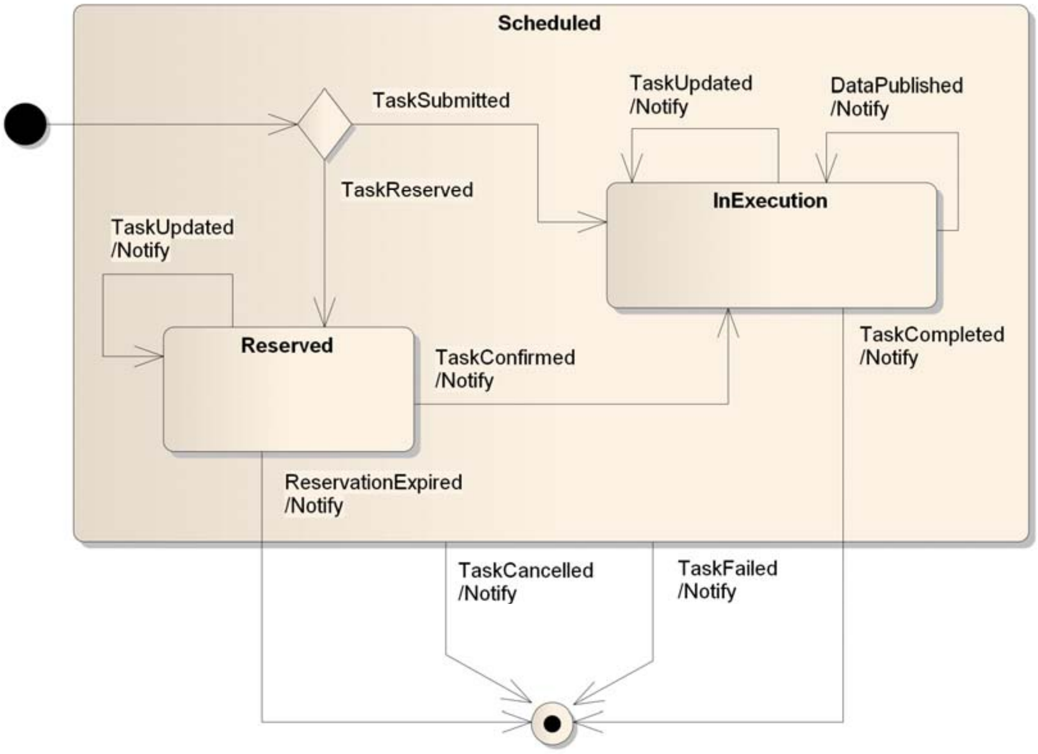


**Figure 9 — task state machine diagram**

Based on the user's intention, the task automatically transitions either into the state *Reserved* or *InExecution*.

A *Reserved* task can be updated by the client but shall not change to state *InExecution* unless the client confirms it. If the client does not confirm a *Reserved* task before it expires, the task (automatically) transitions into the final state (category *Expired)*.

The client can update a task that is *InExecution* at any time. If such a task produces new data that is made available to the client, the SPS can send a notification. The task itself remains in its current state (*InExecution* or a substate thereof) – or more specifically: it transitions (back) into its current state.

If the task is completed it transitions into the final state (category *Completed*). This implies that all data gathered for the task has been published.

A client can cancel a scheduled task at any time. The task then transitions into the final state (category *Cancelled*).

If the server fails to complete a scheduled task as planned, then the task transitions into the final state (category *Failed*).

### 6.4 Status Reporting

Status reports provide information about the status of a task and tasking request but also about the outcome of the cancellation/confirmation of a task. Status reports are contained in operation responses but are also used to encode event information, which can be published to subscribed clients.

Note: The conceptual model of a status report (see clause 7.3.1.5) is therefore quite flexible. Which features of the tasking report are used depends on the specific functionality that was invoked. This is described in detail in the according clauses.

Whenever an SPS receives a tasking request, it assigns a unique identifier to it and provides this identifier in the status report (concerning the status of the request) that is contained in the tasking response. The identifier is created even though the decision on the request may still be pending (i.e. the request is not accepted yet). Once the request gets accepted, in the case of a *Submit* or *Reserve* request a task will be scheduled. The identifier remains the same, i.e. the initially created identifier for a request now becomes an identifier for a task. This allows clients to use the identifier received with the response for subsequent requests to retrieve information about such tasks, e.g. via *GetStatus*.

Example: the state history of a feasible Submit request and resulting task can thus be the sequence of the states *Pending* → *Accepted*/*InExecution* (this state can be entered more than once, e.g. when the task was updated or data was published) → *Completed*

Example: the state history for a feasible Reserve request can be the sequence of the states *Accepted*/*Reserved* (the *Reserved* state can also be entered more than once if the reservation was updated) → *Expired*; another possible sequence is *Pending* → *Accepted*/*Reserved* → *InExecution* → *Completed*

The SPS reports the current status of a tasking request or task following the state types defined in the state machines (see clauses 6.3.6 and chapter 10). A service includes only final and non-final states. Whenever a task has reached a state, even the same state yet again (e.g. after an update the state is still *inExecution*) after being triggered by an event recognized by this standard (see clauses 10.1.3 and 10.2.3) or an extension of this standard, the code for this event (see clause 7.3.1.9) shall be added to the report as well. This helps clients keeping track of the current status of a task/tasking request and the reason why a certain state was (re-)entered.

By default, an SPS therefore logs the information about the latest state transition that a tasking request/task made. An SPS can also support provision of the full state history, i.e. all state transitions. This capability is indicated in the service's metadata. If this capability is not supported by a service instance then such a service can discard information about all state transitions of a task/tasking request except for the latest one. In any case, an SPS is only obliged to provide status information for a certain period of time after a tasking request/task was finalized. How long exactly this period is depends on the given service instance.

### 6.5 Levels of Abstraction – SPS Chains

The functionality offered to a client through the SPS by the asset owner can range from full blown, detailed parameterization options to just a small set of very abstract parameters. Asset owners usually define the tasking parameters of their system according to which functionality they want to make available to their clients. Several abstraction layers can be put in place to make tasking more intuitive for end users – thereby hiding system complexity – while still allowing experts to take advantage of the full set of parameterization options (see following figure).

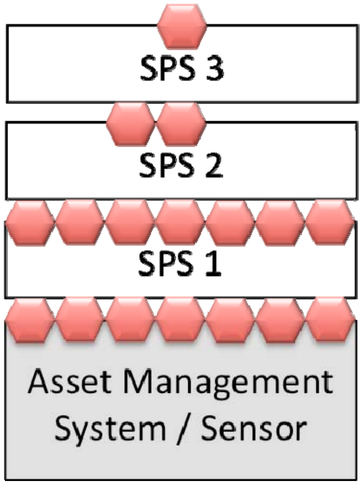


**Figure 10 — tasking on various abstraction levels**

The figure above shows an example of an asset management system (or a concrete sensor), which has a number of SPS instances assigned to it. The system has a number of parameters to be set (red hexagons). In order to task the system, all parameters have to be defined, which is handled by the first SPS instance (SPS 1). The abstraction interface on top (SPS 2) only shows two parameters and a second abstraction interface (SPS 3) shows a single parameter only.

As an example for such an SPS chain, imagine a satellite control system. The system itself requires the parameters region of interest, time of interest, min/max of azimuth and elevation as well as coverage type to be set. The base SPS interface therefore describes seven tasking parameters. At the next abstraction level, the clients can only define region of interest, time of interest and coverage type. The SPS instance at that level will take care for the missing parameters azimuth and elevation. At the highest abstraction level, the SPS interface describes only a single parameter: region of interest. Thus, clients cannot define the time of interest or any of the other parameters, but need to accept what is offered by the service. The different SPS instances simply forward the information provided by a client from the highest level to the asset management system and define the missing parameters with their own data. Clients are usually not aware of this “request enrichment”; it is opaque to them.

The same chain of SPS instances is conceivable for different types of SPS, like simulation systems, processing systems, fusion systems and real physical assets.

### 6.6 Asynchronous Communication

The Sensor Planning Service interface often facades complex asset management systems that do not provide an immediate response to operation requests or which need a long time to gather needed information. The former can be due to the fact that the request has to be analyzed first, which might be a time consuming task. The latter can be due to the fact that the asset – for example a reconnaissance drone or satellite – is not located above the area of interest and therefore has to be moved there, first. Another example is to have the service itself inform the client about a situation of interest. In either case, this shows that the SPS needs to have functionality to support an asynchronous interaction pattern.

### 6.7 Information Access

The service functionality of SPS does not encompass operations for direct access to the information gathered or produced by an asset. Data retrieval services like SOS (Sensor Observation Service), WMS (Web Map Service), or WCS (Web Coverage Service), or even FTP or REST-based services are much more suited to perform this functionality. The SPS interface provides references to the information gathered by an asset. The reference data contains enough information to retrieve the complete set of data output by an asset for a certain task.

# 7 Sensor Planning Service – Implementation Model

## 7.1 Interface Overview

The SPS operations can be divided into informational and functional operations. The informational operations include *GetCapabilities*, *DescribeTasking*, *DescribeResultAccess*, *GetTask* and *GetStatus* operation. The functional operations are the *GetFeasibility*, *Reserve*, *Confirm*, *Submit*, *Update* and *Cancel* operations. All functional operations have an effect on the asset management system.

The SPS defines five interfaces with eleven operations that can be requested by a client and performed by an SPS server. In addition, it incorporates two interfaces from the SWE Common Service Model [OGC 09-001] – these interfaces define two more operations. Figure 11 is a UML diagram showing these interfaces (grey interfaces are defined by OGC 09-001).

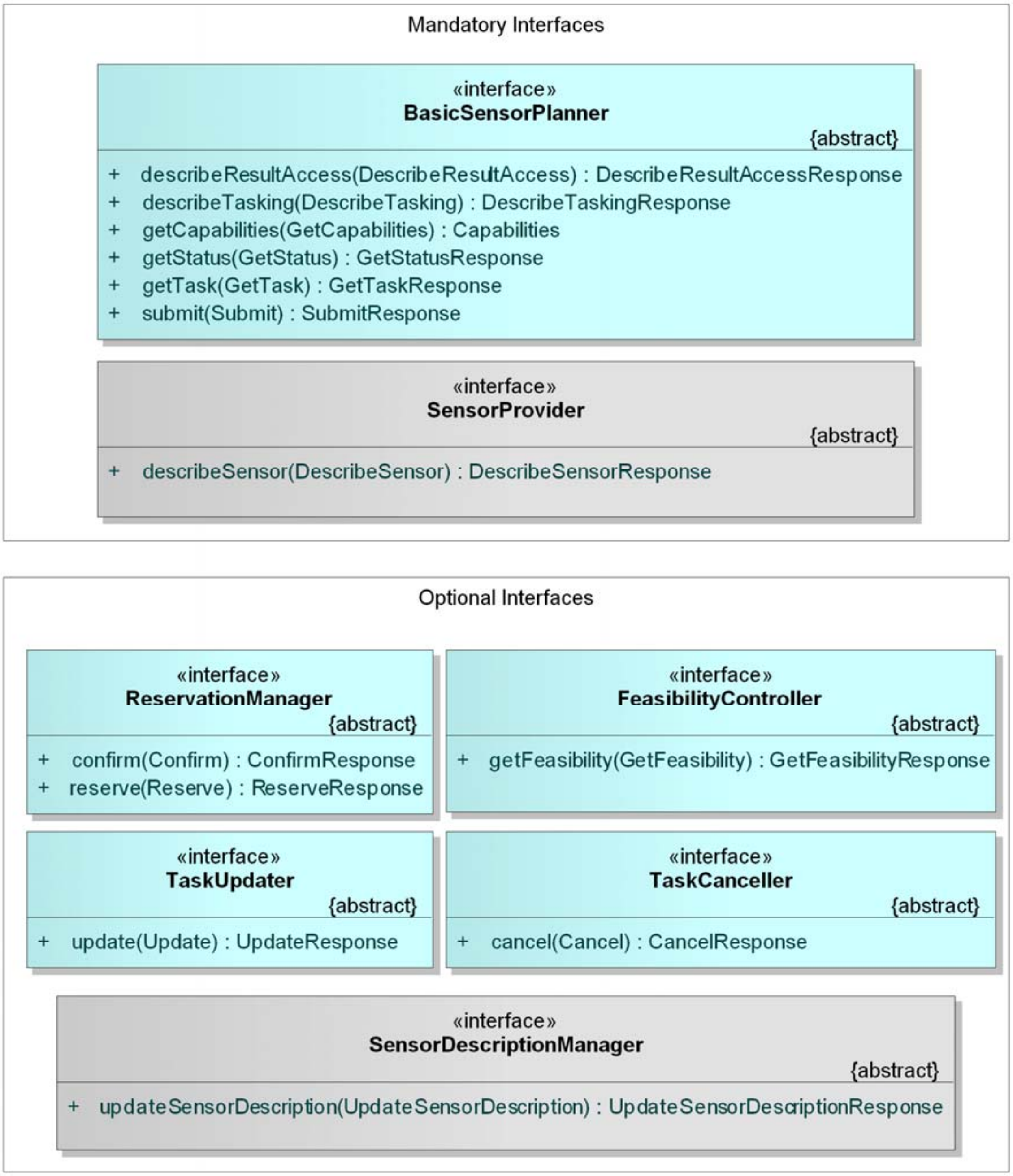

Mandatory Interfaces
«interface»
BasicSensorPlanner
{abstract}
+ describeResultAccess(DescribeResultAccess) : DescribeResultAccessResponse
+ describeTasking(DescribeTasking) : DescribeTaskingResponse
+ getCapabilities(GetCapabilities) : Capabilities
+ getStatus(GetStatus) : GetStatusResponse
+ getTask(GetTask) : GetTaskResponse
+ submit(Submit) : SubmitResponse
«interface»
SensorProvider
{abstract}
+ describeSensor(DescribeSensor) : DescribeSensorResponse
Optional Interfaces
«interface»
ReservationManager
{abstract}
+ confirm(Confirm) : ConfirmResponse
+ reserve(Reserve) : ReserveResponse
«interface»
FeasibilityController
{abstract}
+ getFeasibility(GetFeasibility) : GetFeasibilityResponse
«interface»
TaskUpdater
{abstract}
+ update(Update) : UpdateResponse
«interface»
TaskCanceller
{abstract}
+ cancel(Cancel) : CancelResponse
«interface»
SensorDescriptionManager
{abstract}
+ updateSensorDescription(UpdateSensorDescription) : UpdateSensorDescriptionResponse

**Figure 11 — SPS interfaces 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. These structures are discussed in subsequent clauses. The UML classes modeling these data structures are included in the following clauses.

The SPS interfaces are:

a) *BasicSensorPlanner* (mandatory) – This interface represents the core functionality of an SPS. It contains the following operations:

    a. *GetCapabilities* – This operation allows a client to request and receive service metadata documents that describe the capabilities of the specific server implementation. This operation also supports negotiation of the specification version being used for client-server interactions.

    b. *DescribeTasking* – This operation allows a client to request the information that is needed in order to prepare a tasking request targeted at the assets that are supported by the SPS and that are selected by the client. The server will return information about all parameters that have to be set by the client in order to create a task.

    c. *Submit* – This operation submits a task. Depending on the façaded asset, it may perform a simple modification of the asset or start a complex mission.

    d. *GetStatus* – This operation allows a client to receive information about the current status of the requested task.

    e. *GetTask* – This operation returns complete information about the requested task.

    f. *DescribeResultAccess* – This operation allows a client to retrieve information, which enables access to the data produced by the asset. The server response may contain references to any kind of data accessing OGC Web services such as SOS, WMS, WCS or WFS.

b) *SensorProvider* (mandatory) – It specifies the following operation:

    a. *DescribeSensor* – This operation allows a client to request a detailed description of a sensor. The request can be targeted at a description that was valid at a certain point in or during a certain period of time in the past [OGC 09-001 clause 11].

c) *ReservationManager* (optional) – This optional interface enables clients to reserve a task instead of directly submitting it. This facilitates tasking of a group of SPSs. Reserved tasks have a finite lifetime before they expire. During this lifetime such a task can be confirmed so that the service starts execution. The interface contains the following operations:

    a. *Reserve* – This operation reserves a task. A reservation lasts for a certain amount of time and can be confirmed during this timeframe

    b. *Confirm* – This operation is used to confirm a reserved task. By confirming a reserved task the SPS executes the task.

d) *FeasibilityController* (optional) – SPS implementing this interface are capable of evaluating the feasibility of a task. This allows clients to pre-check their tasking request. The interface contains the following operation:

   a. *GetFeasibility* – This operation checks whether a tasking request is feasible based on the current state of the service and façaded asset(s). It can be used to provide alternative tasking requests to the client. Depending on the asset type façaded by the SPS, the SPS server action may be as simple as checking that the request parameters are valid, and are consistent with certain business rules, or it may be a complex operation that calculates the utilizability of the asset to perform a specific task at the defined location, time, orientation, calibration etc.

e) *TaskUpdater* (optional) – A service that implements this interface allows clients to update a reserved or accepted task. The interface contains the following operation:

   a. *Update* – This operation is used to request a modification of a reserved or accepted task.

f) *TaskCanceller* (optional) – This interface, if implemented, enables clients to cancel a reserved or accepted task. The interface contains the following operation:

   a. *Cancel* – This operation allows a client to cancel a previously reserved or accepted task.

g) *SensorDescriptionManager* (optional). It specifies the following operation:

   a. *UpdateSensorDescription* - This operation allows clients to update the description of a sensor [OGC 09-001 clause 12].

The operations of those interfaces decribed above have many similarities with other OGC Web Services operations/interfaces. Aspects that are common with other OWS specifications are thus specified in the OpenGIS® Web Services Common Implementation Specification [OGC 06-121r3]. Many of these common aspects are normatively referenced herein, instead of being repeated in this specification.

The operations in each of the SPS interfaces will be described in subsequent clauses.

| Requirement | |
|---|---|
| http://www.opengis.net/spec/SPS/2.0/req/interfaces | |
| REQ 1. | Each SPS instance shall implement the interfaces *BasicSensorPlanner* and *SensorProvider*. |

## 7.2 SPS Exceptions

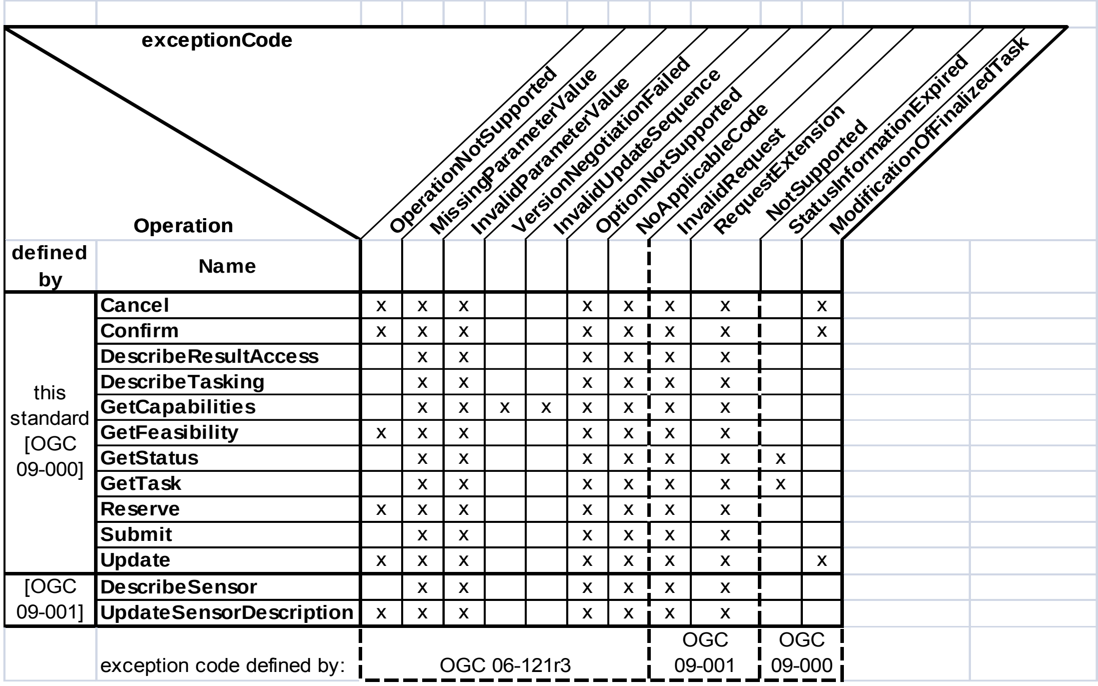
| Requirement | |
|---|---|
| http://www.opengis.net/spec/SPS/2.0/req/exceptions | |
| REQ 2. | Whenever an SPS server encounters an error while performing one of its operations, it shall return an exception message according to the model/schema defined in chapter 8 of [OGC 06-121r3]. |

| Requirement | |
|---|---|
| http://www.opengis.net/spec/SPS/2.0/req/exceptions/codes | |
| REQ 3. | The allowed standard exception codes shall include those defined in clause 15 of [OGC 09-001] and those defined in Table 6 in this standard. They shall be used according to Figure 12. Only, the *OperationNotSupported* exception shall not apply for the *DescribeSensor* operation implemented by an SPS, because that operation is mandatory for an SPS implementation. |

**Table 6 — Exception (code) defined by SPS**

| exceptionCode value | Meaning of code | "locator" value |
|---|---|---|
| StatusInformationExpired | The service already discarded status information for the requested task / tasking request. | None, omit "locator" parameter |
| ModificationOfFinalizedTask | The client attempted to modify (e.g. cancel, update or confirm) a task that was already finalized. | None, omit "locator" parameter |

NOTE: Each SPS operation may define additional requirements with respect to exception handling for that operation.



| Operation: defined by | Operation: Name | OperationNotSupported | MissingParameterValue | InvalidParameterValue | VersionNegotiationFailed | InvalidUpdateSequence | OptionNotSupported | NoApplicableCode | InvalidRequest | RequestExtensionNotSupported | StatusInformationExpired | ModificationOfFinalizedTask |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| this standard [OGC 09-000] | Cancel | x | x | x | | | x | x | x | x | | x |
| | Confirm | x | x | x | | | x | x | x | x | | x |
| | DescribeResultAccess | | x | x | | | x | x | x | x | | |
| | DescribeTasking | | x | x | | | x | x | x | x | | |
| | GetCapabilities | | x | x | x | x | x | x | x | x | | |
| | GetFeasibility | x | x | x | | | x | x | x | x | | |
| | GetStatus | | x | x | | | x | x | x | x | x | |
| | GetTask | | x | x | | | x | x | x | x | x | |
| | Reserve | x | x | x | | | x | x | x | x | | |
| | Submit | | x | x | | | x | x | x | x | | |
| | Update | x | x | x | | | x | x | x | x | | x |
| [OGC 09-001] | DescribeSensor | | x | x | | | x | x | x | x | | |
| | UpdateSensorDescription | x | x | x | | | x | x | x | x | | |
| | exception code defined by: | OGC 06-121r3 | | | | | | | OGC 09-001 | | OGC 09-000 | |

**Figure 12 — SPS operations with applicable exceptionCodes**

| Requirement | |
|---|---|
| http://www.opengis.net/spec/SPS/2.0/req/exceptions/UnknownIdentifier | |
| REQ 4. | If the value of an identifier used in a request is unknown to the service, it shall return an *InvalidParameterValue* exception, with the exception locator naming the property of the request that contained the unknown value ("task", "procedure" etc. – lookup the actual name in the UML model/table describing the properties of the request type). |

SPS may drop all information about a finalized task after the minimum storage time for that information has passed (see documentation on minStatusTime provided in clause 7.3.3.3). In consequence, a previously valid task identifier in a GetStatus request can cause an InvalidParameterValue exception once the task information is no longer available at SPS.

| Requirement | |
|---|---|
| http://www.opengis.net/spec/SPS/2.0/req/exceptions/InvalidTaskingParameters | |
| REQ 5. | If a service encounters in a TaskingRequest that either<br>• the tasking parameters sent in the request are not structured according to the description provided in the *DescribeTasking* response,<br>• the encoding used by the client is not supported by the service, or<br>• the provided values are not encoded correctly,<br>an *InvalidParameterValue* exception with locator *taskingParameters* shall be returned. |

| Requirement | |
|---|---|
| http://www.opengis.net/spec/SPS/2.0/req/exceptions/ModificationOfFinalizedTask | |
| REQ 6. | If a client attempts to perform an operation on a finalized task (like updating, confirming or cancelling it) then the service shall return a *ModificationOfFinalizedTask* exception. |

### 7.3 Package Overview

This standard defines 13 packages that correspond to the operations introduced in clause 7.1. Each package contains a number of data types and definitions.

In addition, SPS makes use of two packages defined in other standards: The *Common* package, which contains data types shared by several operations, and the *Contents* package, which contains data types used in the *GetCapabilities* operation, are defined in OGC 08-094 and OGC 09-001 respectively.

All SPS packages use data types specified in other standards. Those data types are normatively referenced herein, instead of being repeated in this standard.

Figure 13 shows a UML diagram summarizing the external dependencies of the SPS.

Note: The *InsertSensor* and *DeleteSensor* operations, defined in OGC 09-001 (SWE Service Model), are not specified in this version of SPS.

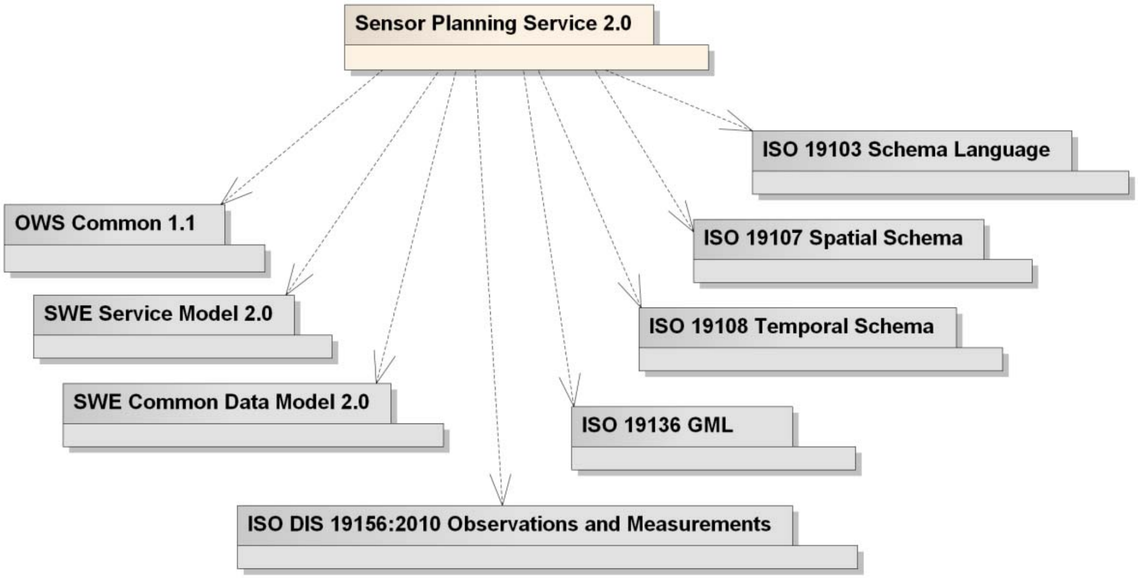

Sensor Planning Service 2.0
OWS Common 1.1
SWE Service Model 2.0
SWE Common Data Model 2.0
ISO DIS 19156:2010 Observations and Measurements
ISO 19136 GML
ISO 19108 Temporal Schema
ISO 19107 Spatial Schema
ISO 19103 Schema Language

**Figure 13 — SPS model external dependencies**

Figure 14 shows a UML diagram summarizing the package dependencies of the SPS.

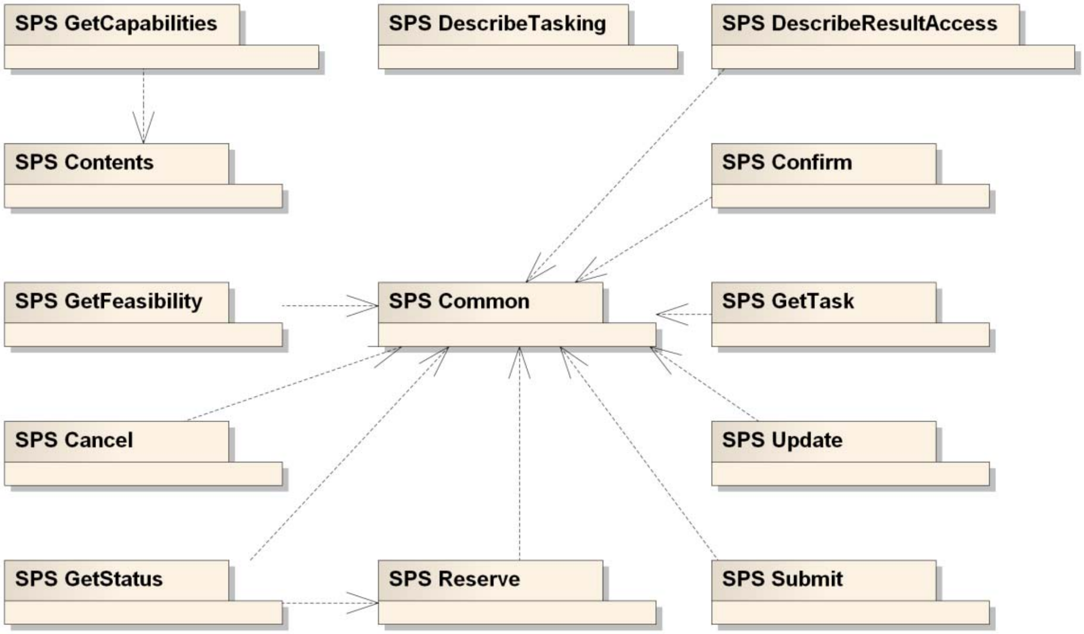


**Figure 14 — SPS package dependencies**

The following clauses describe each package in more detail.

Each operation request type defined in the following sections requires to set the service and version properties.

| Requirement | |
|---|---|
| http://www.opengis.net/spec/SPS/2.0/req/ServiceAndVersion | |
| REQ 7. | For each operation request data type, the *service* property shall have the value “SPS” and the *version* property shall have the value “2.0.0”. |

### 7.3.1 Common Package

#### 7.3.1.1 Introduction

This package contains all data types used by two or more service operations.

#### 7.3.1.2 Data Types

The conceptual model of the Common package is shown in the following UML diagram.

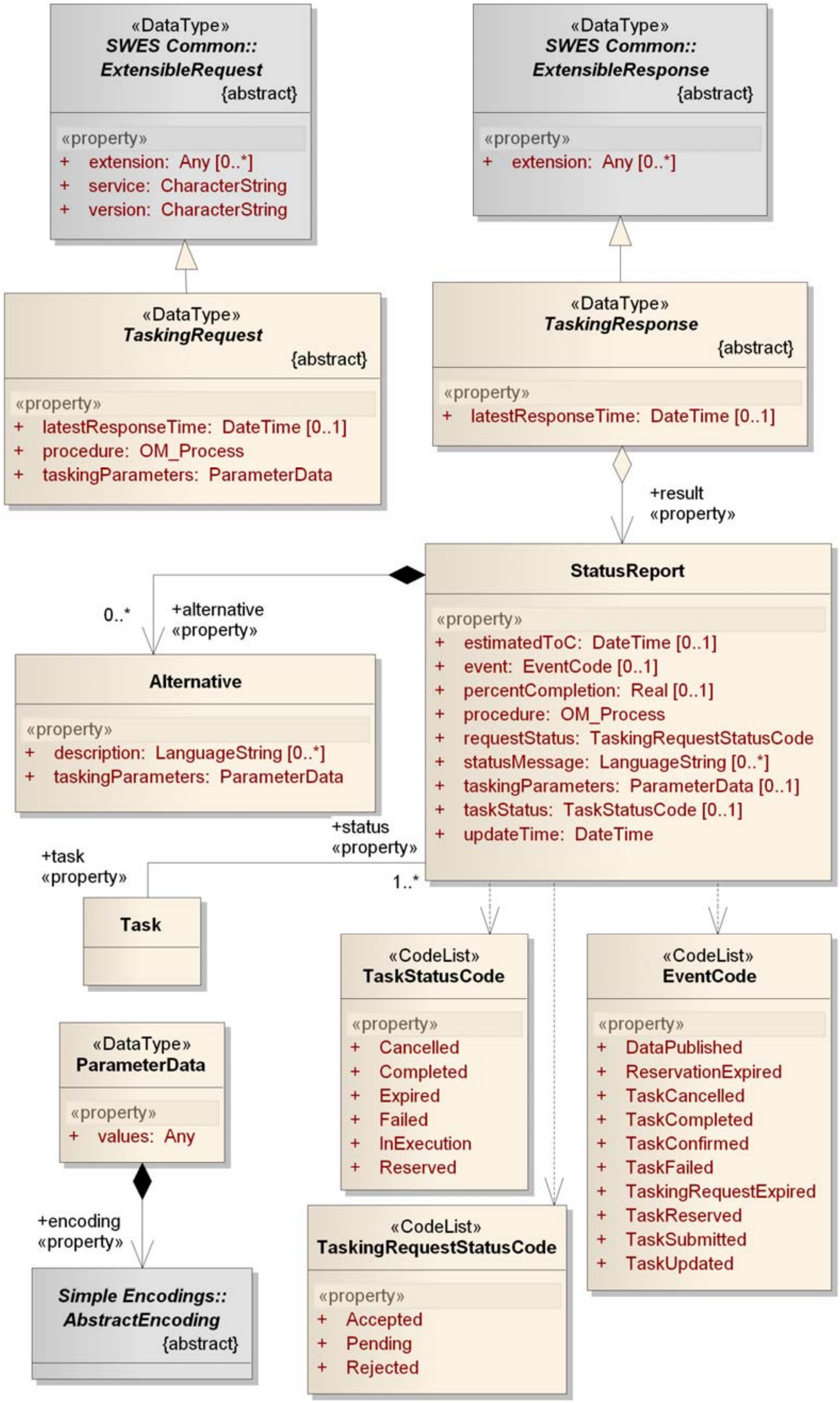


**Figure 15 — Data types contained in the Common package**

The details of each class contained in the package are explained in the following subclauses.

#### 7.3.1.3 TaskingRequest

This abstract data type serves as the super class for all tasking requests such as *GetFeasibility*, *Reserve*, *Submit* and *Update* requests.

Usually, tasking requests contain one or more tasking parameters (see clauses 6.3.2 and 7.4).

| Requirement | |
|---|---|
| http://www.opengis.net/spec/SPS/2.0/req/TaskingRequest/parameters | |
| REQ 8. | The tasking parameters for tasking a given procedure shall be structured according to the tasking parameter description for that procedure. |

Any valid tasking request leads to a *TaskingResponse*. Although the SPS is supposed to send a *TaskingResponse* when receiving a *TaskingRequest*, the decision whether to accept or reject a tasking request might not be available immediately (or takes longer than the timeout of the used communication protocol allows). This leads to a *Pending* state.

| Requirement | |
|---|---|
| http://www.opengis.net/spec/SPS/2.0/req/TaskingRequest/pending | |
| REQ 9. | If a task acceptance or rejection decision is not available immediately, the state of the tasking request shall be set to *Pending*. |

To avoid tasking requests on *Pending* for time periods longer than acceptable for clients, the SPS provides a mechanism allowing clients to constrain this period of time. A client can define a *latestResponseTime* for a tasking request. If the server does not provide a *TaskingResponse* with final result until then, the requested tasking is agreed both by the client and service as being rejected. The tasking request is expired (see definition of the event *TaskingRequestExpired* in clauses 6.3.6, 7.3.1.9 and 10.2.3.1).

| Requirement | |
|---|---|
| http://www.opengis.net/spec/SPS/2.0/req/TaskingRequest/updateTaskingExpirationHandling | |
| REQ 10. | In case that the intention of the tasking request was to update a reserved or submitted task but the tasking request expired, the task remains in its current state. The service shall set the tasking request status to *Rejected*. |

| Requirement | |
|---|---|
| http://www.opengis.net/spec/SPS/2.0/req/TaskingRequest/synchronousTasking | |
| REQ 11. | If synchronous request-response handling is taking place, the service shall provide an immediate tasking response with request status *Pending* if the service cannot bring about a decision directly. |

In such a situation, the default mechanism for the client to retrieve the result is to perform a *GetStatus* request (see clause 7.3.6), possibly involving multiple GetStatus requests until the final result is provided by the service.

| Requirement | |
|---|---|
| http://www.opengis.net/spec/SPS/2.0/req/TaskingRequest/asynchronousTasking | |
| REQ 12. | If asynchronous request-response handling is taking place, and the client provided an endpoint address, the service shall send any state transition to that address. The first state transition might be the transfer into state *Pending*. |

Note: this way, the entity at the endpoint (where the response shall be delivered to asynchronously) gets the information (task identifier) required to pull for the status of the tasking request The defined behavior also ensures consistency of the tasking request handling regardless if synchronous or asynchronous request-response is in use.

The abstract TaskingRequest data type is derived from the *ExtensibleRequest* data type specified in clause 9 of [OGC09-001] and therefore inherits all the properties contained in that data type. *TaskingRequest* does not restrict the content model of *ExtensibleRequest*.

| Requirement | |
|---|---|
| http://www.opengis.net/spec/SPS/2.0/req/TaskingRequest/dataType | |
| REQ 13. | The *TaskingRequest* data type shall contain the properties defined for *ExtensibleRequest*. In addition, it shall contain the properties according to Table 7. |

**Table 7 — Properties in the TaskingRequest data type**

| Name | Definition | Data type and values | Multiplicity and use |
|---|---|---|---|
| latestResponseTime | point in time at which the definite decision about the tasking request (the requested tasking action being accepted or rejected) has to be provided by the SPS. | DateTime<br>(see ISO 19103 and OGC 07-036 Table D.2)<br>shall be a point in time in the future (compared to server time when the tasking request was received) | Zero or one (optional) |
| procedure | Pointer to the procedure that is to be tasked. | OM_Process [id]<br>(see ISO DIS 19156) | One (mandatory) |
| taskingParameters | parameter values required to task the sensor | ParameterData, see clause 7.3.1.11 | One (mandatory)<br>values for tasking parameters shall be provided in one of the encodings supported by the service, see clause 7.3.3.3 |
| id) Note: the primary use of this property is to provide a pointer/identifier – see OGC 09-001 clause 16.3.1 for further details. | | | |

#### 7.3.1.4 TaskingResponse

##### 7.3.1.4.1 TaskingResponse – Content and StatusCodes

A tasking response is sent as the direct response to a tasking request.

| Requirement | |
|---|---|
| http://www.opengis.net/spec/SPS/2.0/req/TaskingResponse/content | |
| REQ 14. | The tasking response shall contain a (subclass of the) *StatusReport* (see clause 7.3.1.5), which indicates if the requested tasking action was *accepted*, *rejected* or if the decision is *pending*. |

| Requirement | |
|---|---|
| http://www.opengis.net/spec/SPS/2.0/req/TaskingResponse/statusCodes | |
| REQ 15. | In the *StatusReport*, the SPS shall use the status codes defined in Table 8. |

The following table defines the valid status codes (see clauses 7.3.1.6 and 7.3.1.8) of a *StatusReport* (see clause 7.3.1.5) in response to a specific request.

**Table 8 — Status Codes, usage and meaning in *TaskingResponse* specializations**

| **requestStatus Code**[3] | **taskStatus Code**[3] | **taskStatus code usage and overall meaning in** | | | |
|---|---|---|---|---|---|
| | | **Get Feasibility Response** | **Reserve Response** | **Submit Response** | **Update Response** |
| **Pending** | -[1] | tasking request is pending | | | |
| **Rejected** | -[1] | task is not feasible | task is not reserved | task is rejected | update is rejected |
| **Accepted** | -[2] | task is feasible | NA (taskStatus is mandatory here, use value “Reserved“) | NA (taskStatus is mandatory here, use,value “InExecution“ or “Completed“) | task was updated |
| | **Reserved** | NA | task is reserved | NA | NA<br>(do not set taskStatus) |
| | **InExecution** | NA | NA | task is submitted and the service executes it | NA<br>(do not set taskStatus) |
| | **Completed** | NA | NA | the task was submitted and the service already completed its execution | NA |
| | **Cancelled** | NA | NA | NA | NA |
| | **Failed** | NA | NA | NA | NA |
| | **Expired** | NA | NA | NA | NA |

NA = taskStatus value not applicable

Notes:

1) If requestStatus is Pending or Rejected then taskStatus is not set by the service
2) If requestStatus is Accepted then taskStatus is not used in GetFeasibilityResponse/UpdateResponse – however, taskStatus is then required in ReserveResponse/SubmitResponse
3) or any other sub code

#### 7.3.1.4.2 TaskingResponse – Data Type

The abstract data type *TaskingResponse* serves as the super class for the *GetFeasibilityResponse*, *ReserveResponse*, *SubmitResponse,* and *UpdateResponse* types.

The abstract *TaskingResponse* data type is derived from the *ExtensibleResponse* data type specified in clause 9 of [OGC09-001] and therefore inherits all the properties contained in that data type. *TaskingResponse* does not restrict the content model of *ExtensibleResponse*.

| Requirement | |
|---|---|
| http://www.opengis.net/spec/SPS/2.0/req/TaskingResponse/dataType | |
| REQ 16. | The *TaskingResponse* data type shall contain the properties defined for *ExtensibleResponse*. In addition, it shall contain the properties according to Table 9. |

**Table 9 — Properties in the TaskingResponse data type**

| Name | Definition | Data type and values | Multiplicity and use |
|---|---|---|---|
| latestResponseTime | Point in time at which the definite decision about the tasking request (the requested action being accepted or rejected) will be provided by the SPS. The parameter allows clients to understand how long the decision process (accept or reject the tasking request) might take. | DateTime<br>(see ISO 19103 and OGC 07-036 Table D.2)<br>shall be a point in time in the future | Zero or one (optional)<br>shall be included by the service if the client included a latestResponseTime in the tasking request – the service shall then use that time (as a confirmation of the response time requested by the client) or use a time that is before the one requested by the client (the earlier time is per definition the latestResponseTime that both client and server agree upon) |
| result | provides the outcome of the tasking request | StatusReport *or subclass*, see clause 7.3.1.5 | One (mandatory) |

#### 7.3.1.5 StatusReport

This data type provides information about the status of a given task/tasking request. In addition, it is the super class of *ReservationReport*. The status report identifies the sensor that is tasked (*procedure*) and the task itself (*task*). It contains status codes to indicate the status of a tasking request (*requestStatus*) and task (*taskStatus*) as well as optionally a server defined status message (*statusMessage*) in addition to the time when a certain status was entered (*updateTime*). If an event known to the service (see state machine diagram in Figure 9 and event definitions in Table 14) caused the transition into the new status, the code for the *event* can also be provided. The status message can be provided in any number of languages. Further on, the *StatusReport* provides an estimation of the time to completion of the task (*estimatedToC*) and information about the overall progress of an executed task (*percentCompletion*). The *StatusReport* can also contain *Alternatives* and the *taskingParameters* that were provided by the client when submitting, reserving or updating a task.

| Requirement | |
|---|---|
| http://www.opengis.net/spec/SPS/2.0/req/StatusReport/taskingParameters | |
| REQ 17. | By default, *taskingParameters* are only provided in *StatusReports* of *GetStatus* and *GetTask* responses. By default, *StatusReports* in responses to tasking requests do not reflect the *taskingParameters* used in the request.<br>Note: This behaviour may get overwritten in an extension to this standard, if focus is more on verification of tasking parameters than on lightweight response messages. |

| Requirement | |
|---|---|
| http://www.opengis.net/spec/SPS/2.0/req/StatusReport/announcement | |
| REQ 18. | SPS servers shall announce in their Capabilities if all state changes are tracked for non-finalized tasks and tasking requests (see clause 7.3.2.4.3). |

If supported by the server, clients can request this status history of a task or tasking request by using a *GetStatus* request with *since* parameter (see clause 7.3.6.1 for further information).

If supported by the service, status reports caused by certain events can also be published to a list of interested consumers (see clause 6.4).

| Requirement | |
|---|---|
| http://www.opengis.net/spec/SPS/2.0/req/StatusReport/dataType | |
| REQ 19. | The *StatusReport* type shall contain the properties according to Table 10. |

Note: the usage of the *StatusReport* in the *Cancel*, *Confirm*, *GetFeasibility*, *Reserve*, *Submit* and *Update* operations is further defined in the according clauses.

**Table 10 — Properties in the StatusReport data type**

| Name | Definition | Data type and values | Multiplicity and use |
|---|---|---|---|
| task | Pointer to the task that this status report belongs to. | Task type [id]<br>see clause 7.3.1.6 | One (mandatory) |
| estimatedToC | estimated completion time of the task | DateTime<br>(see ISO 19103 and OGC 07-036 Table D.2) | Zero or one (optional)<br>Include if estimation makes sense and can be provided. |
| event | signifies the event that caused the transition into the new state/status | EventCode<br>see clause 7.3.1.9 | Zero or one (optional)<br>Shall be included if transition to current state was triggered by a known event (one of those listed in the EventCode code list, see clause 7.3.1.9 – or extensions thereof) |
| percentCompletion | indicates the progress made in executing the task | Real (see ISO 19103)<br>value shall be in the range of 0-100 | Zero or one (optional)<br>Shall only be used for StatusReports with taskStatus 'InExecution', a following state or a substate thereof. |
| procedure | Pointer to the process that is the subject of the task for which the report was generated. | OM_Process [id]<br>see ISO DIS 19156 | One (mandatory) |
| requestStatus | identifies the state of the request (that may have led to the scheduling of the task) | TaskingRequestStatusCode, see clause 7.3.1.7 | One (mandatory) |
| statusMessage | Server defined free text that further describes the status. | LanguageString, see clause 10.7 in [OGC 06-121r3] | Zero to many (optional) |
| taskingParameters | Parameters used in a tasking request that led to the current status. | ParameterData, see clause 7.3.1.11 | Zero or one (optional) |
| taskStatus | identifies the state of a scheduled task | TaskStatusCode, see clause 7.3.1.8 | Zero or one (optional) |
| updateTime | point in time at which the task entered the reported state | DateTime<br>(see ISO 19103 and OGC 07-036 Table D.2) | One (mandatory) |
| alternative | alternative set of tasking parameters that would be feasible at the time of report generation | Alternative type, see clause 7.3.1.10 | Zero to many (optional) |
| id) Note: the primary use of this property is to provide a pointer/identifier – see OGC 09-001 clause 16.3.1 for further details. | | | |

| Requirement | |
|---|---|
| http://www.opengis.net/spec/SPS/2.0/req/StatusReport/completionRatio | |
| REQ 20. | SPS servers shall put the *percentCompletion* property to 0% when the task entered the *InExecution* state. Only when the task is *InExecution* shall the percentCompletion be increased. |

Whenever a task *InExecution* was cancelled or failed, the last status report show the progress of the task made until then. A completed task has 100% completion. Note that the *percentCompletion* value can in fact decrease in two consecutive *GetStatus* requests. This happens for example if the SPS receives an *Update* request and needs to start over the requested activites.

#### 7.3.1.6 Task

The *Task* type represents the complete information about a task. This encompasses information about the current and – optionally – also previous statuses the task was in. The according status reports also include the tasking parameters that were used when reserving/submitting and updating the task.

| Requirement | |
|---|---|
| http://www.opengis.net/spec/SPS/2.0/req/Task/uniqueIdentifier | |
| REQ 21. | An SPS shall assign a unique identifier for each task (including tasking requests) it creates (using the identifier property it automatically inherits as defined in OGC 09-001 clause 24.2.4.1). |

| Requirement | |
|---|---|
| http://www.opengis.net/spec/SPS/2.0/req/Task/pendingRequestTaskIdentifier | |
| REQ 22. | An SPS shall assign a unique identifier for each tasking request if the tasking request enters *Pending* state. Thus, the *Task* identifier can identify a task or a pending tasking request. |

| Requirement | |
|---|---|
| http://www.opengis.net/spec/SPS/2.0/req/Task/identifierPassing | |
| REQ 23. | If an SPS defines a unique identifier for a tasking request and no task identifier has beeen created already, the same identifier shall be used for the task that will be created as a result of this request.<br>Thus, the passing of identifiers applies to all tasking requests except for the *Update* request, where a task identifier was already created. |

| Requirement | |
|---|---|
| http://www.opengis.net/spec/SPS/2.0/req/Task/dataType | |
| REQ 24. | The *Task* type shall contain the properties according to Table 11. |

**Table 11 — Properties in the Task data type**

| Name | Definition | Data type and values | Multiplicity and use |
|---|---|---|---|
| status | Status information of the task. | StatusReport, see clause 7.3.1.5 | One or more (mandatory)<br>At least the current status shall be available for a task. |

### 7.3.1.7 TaskingRequestStatusCode

The *TaskingRequestStatusCode* code list defines the different status codes for tasking requests. The states that a tasking request can transition through are discussed in clause 6.3.6 and in detail in clause 10.

| Requirement | |
|---|---|
| http://www.opengis.net/spec/SPS/2.0/req/TaskingRequestStatusCode/list | |
| REQ 25. | The *TaskingRequestStatusCode* code list shall contain the properties/code values according to Table 12. |

**Table 12 — Properties in the TaskingRequestStatusCode code list**

| Code | Definition | Value |
|---|---|---|
| Accepted | See clause 10.2.2.2 – Tasking request was accepted; this is a final state for a tasking request. | “Accepted” |
| Pending | See clause 10.2.2.1 – Tasking request is pending. | “Pending” |
| Rejected | See clause 10.2.2.6 – Tasking request was rejected; this is a final state for a tasking request. | “Rejected” |

This code list is extensible. SPS profiles/extensions or implementations may add additional codes that define sub states of those defined in this specification. The concrete SPS implementation using sub states defines when to send which notifications to the clients if publish/subscribe functionality is supported by the service.

| Requirement | |
|---|---|
| http://www.opengis.net/spec/SPS/2.0/req/TaskingRequestStatusCode/codeSyntax | |
| REQ 26. | New TaskingRequestStatusCodes shall conform to the following syntax:<br>`other: <existing_code>_<new_code_for_substate>`<br>Code names shall only use the characters A-Z, a-z and 0-9. By adhering to this syntax, clients can ignore sub states but will still understand the main state. |

EXAMPLE: A valid new sub state would be “`other: Pending_OperatorInformed`”.

#### 7.3.1.8 TaskStatusCode

The *TaskStatusCode* code list defines the different status codes for tasks. The states and the transition between the states are discussed in clause 6.3.6 and in detail in clause 10.

| Requirement | |
|---|---|
| http://www.opengis.net/spec/SPS/2.0/req/TaskStatusCode/list | |
| REQ 27. | The *TaskStatusCode* code list shall contain the properties/code values according to Table 13. |

**Table 13 — Properties in the TaskStatusCode code list**

| Code | Definition | Value |
|---|---|---|
| Cancelled | See clause 10.1.2.5 – Task was cancelled; this code identifies a subcategory of the final state in the task state machine. | “Cancelled” |
| Completed | See clause 10.1.2.5 – Task was completed as planned; this code identifies a subcategory of the final state in the task state machine. | “Completed” |
| Expired | See clause 10.1.2.5 – Task reservation expired; this code identifies a subcategory of the final state in the task state machine. | “Expired” |
| Failed | See clause 10.1.2.5 – Task failed; this code identifies a subcategory of the final state in the task state machine. | “Failed” |
| InExecution | See clause 10.1.2.2 – Task is executed by the service. | “InExecution” |
| Reserved | See clause 10.1.2.3 – Task is reserved at the service. | “Reserved” |

This code list is extensible. SPS profiles/extensions or implementations may add additional codes that define sub states of those defined in this specification. The concrete SPS implementation using sub states defines when to send which notifications to the clients if publish/subscribe functionality is supported by the service.

| Requirement | |
|---|---|
| http://www.opengis.net/spec/SPS/2.0/req/TaskStatusCode/codeSyntax | |
| REQ 28. | New TaskStatusCodes shall conform to the following syntax:<br>`other: <existing_code>_<new_code_for_substate>`<br>Code names shall only use the characters A-Z, a-z and 0-9. By adhering to this syntax, clients can ignore sub states but will still understand the main state. |

EXAMPLE: A valid new sub state would be “`other: InExecution_SensorInitialized`”.

### 7.3.1.9 EventCode

The *EventCode* type is a list of codes signifying events that happen in SPSs and are identified in this standard.

| Requirement | |
|---|---|
| http://www.opengis.net/spec/SPS/2.0/req/EventCode/list | |
| REQ 29. | The *EventCode* code list shall contain the properties/code values according to Table 14. |

The events defined in this table can trigger state transitions – see clause 10 for further details.

**Table 14 — Properties in the EventCode code list**

| Name [a] | Definition | Value [a] |
|---|---|---|
| DataPublished | New data was published for a task that is 'InExecution'. | "DataPublished" |
| ReservationExpired | A reserved task has expired (the expiration time set by the service is before now - "now" being the time measured by the service). | "ReservationExpired" |
| TaskCancelled | A scheduled task has been cancelled. [b] | "TaskCancelled" |
| TaskCompleted | A task that was 'InExecution' was completed as planned. Implies that all data gathered for the task has been published. | "TaskCompleted" |
| TaskConfirmed | A reserved task was confirmed. | "TaskConfirmed" |
| TaskFailed | A scheduled task has failed. [c] | "TaskFailed" |
| TaskingRequestExpired | A pending tasking request has expired. | "TaskingRequest Expired" |
| TaskReserved | A task was reserved. | "TaskReserved" |
| TaskSubmitted | A task was submitted. | "TaskSubmitted" |
| TaskUpdated | A task was updated. | "TaskUpdated" |

a Although some values listed in the column appear to contain spaces, they shall not contain spaces.

b Data gathered and published for the cancelled task should not automatically be deleted so that a client can retrieve the data that was gathered until the task was cancelled.

c Data gathered and published for the failed task should not automatically be deleted so that a client can at least retrieve the data that was gathered until the task failed.

This code list is extensible. SPS profiles/extensions or implementations can add additional event codes that can for example identify transition events in substates of the InExecution state.

| Requirement | |
|---|---|
| http://www.opengis.net/spec/SPS/2.0/req/EventCode/codeSyntax | |
| REQ 30. | New EventCodes shall conform to the following syntax:<br>`other: [A-Za-z0-9_]{2,}`<br>Code names shall only use the characters A-Z, a-z and 0-9. By adhering to this syntax, clients can ignore sub states but will still understand the main state. |

EXAMPLE: A valid new sub state would be "`other: OperatorInformed`".

### 7.3.1.10 Alternative

This data type represents a suggestion of a set of alternative tasking parameter values. An optional description may be used to provide further information on this alternative.

| Requirement | |
|---|---|
| http://www.opengis.net/spec/SPS/2.0/req/Alternative/dataType | |
| REQ 31. | The *Alternative* type shall contain the properties according to Table 15. |

**Table 15 — Properties in the Alternative data type**

| Name | Definition | Data type and values | Multiplicity and use |
|---|---|---|---|
| description | human readable description of the alternative | LanguageString, see clause 10.7 in [OGC 06-121r3] | Zero to many (optional) |
| taskingParameters | block of encoded values together with a description of the encoding | ParameterData, see clause 7.3.1.11 | One (mandatory) |

### 7.3.1.11 ParameterData

This data type contains properties to store (tasking) parameter values and a description of the encoding being used. It aggregates the required types from the SWE Common Data Model. This data type is used by SPS whenever data needs to be delivered to or from the service in an efficient way (see also clause 7.4 - SPS tasking parameters representation).

The *DescribeTasking* operation (see clause 7.3.4) provides the description of the tasking parameters and how they should be structured when encapsulated in the values attribute of the ParameterData object.

| Requirement | |
|---|---|
| http://www.opengis.net/spec/SPS/2.0/req/ParameterData/dataType | |
| REQ 32. | The *ParameterData* type shall contain the properties according to Table 16. |

**Table 16 — Properties in the ParameterData data type**

| Name | Definition | Data type and values | Multiplicity and use |
|---|---|---|---|
| encoding | description of the encoding used to encode the given values | AbstractEncoding, see clause 7.6 in [OGC 08-094]<br>shall provide details for one of the encodings supported by the service (see clause 7.3.3.3, Table 24) | One (mandatory) |
| values | block of values encoded as specified by the encoding (description) | Any type<br>value shall be as defined by the encoding | One (mandatory) |

### 7.3.2 GetCapabilities Operation

#### 7.3.2.1 Introduction

The mandatory *GetCapabilities* operation allows clients to retrieve service metadata from a server. The response to a *GetCapabilities* request contains service metadata about the server, including specific information about the sensors provided by the service, supported data encodings, and – if supported by the service – metadata about the supported notification functionality.

#### 7.3.2.2 Data Types

The conceptual model of the GetCapabilities operation is shown in the following UML diagram.

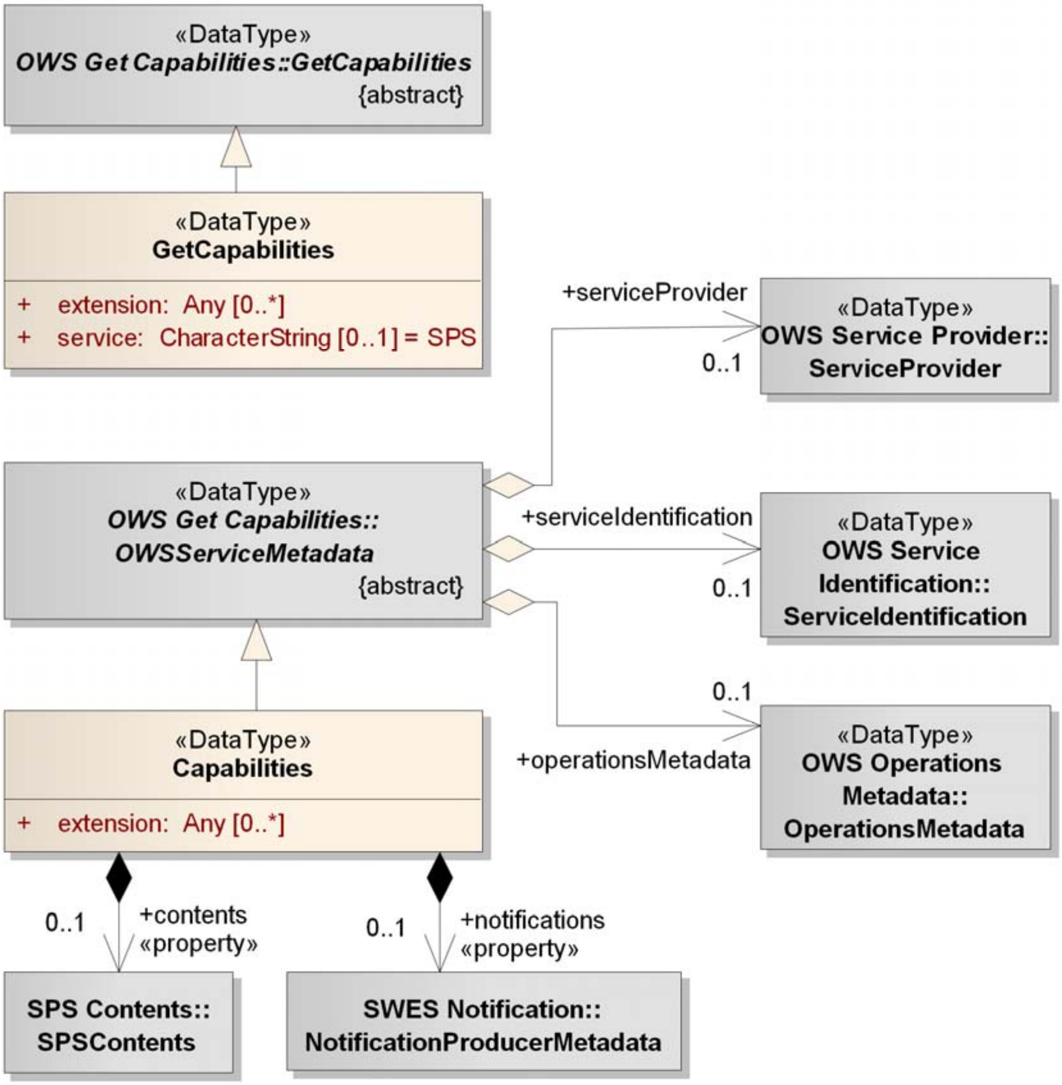


**Figure 16 — Data types of the GetCapabilities operation**

The details of the operation request and response are explained in the following subclauses.

### 7.3.2.3 Operation Request – GetCapabilities

Sending an instance of the GetCapabilities data type to the service performs an SPS *GetCapabilities* operation request.

The *GetCapabilities* data type is derived from the similarly named data type defined by OWS Common (see clauses 7.2 and 7.3 in [06-121r3]).

| Requirement | |
|---|---|
| http://www.opengis.net/spec/SPS/2.0/req/GetCapabilitiesRequest/dataType | |
| REQ 33. | The SPS *GetCapabilities* data type shall contain the properties of the OWS Common GetCapabilities data type from OWS Common (listed in table 3 of [06-121r3]). In addition, it shall contain the properties according to Table 17. |

**Table 17 — Properties in the GetCapabilities data type**

| Name | Definition | Data type and values | Multiplicity and use |
|---|---|---|---|
| extension | container for elements defined by extension specifications | Any type<br>value is defined by the extension specification | Zero or more (optional) |
| service | service type identifier | Character String type, not empty<br>value shall be "SPS" | Zero or one (optional)<br>default value is "SPS" |

NOTE The request property – derived from OWS Common GetCapabilities type – is explicit or implied by each specific binding of the GetCapabilities operation, so is not necessarily part of the request representation defined by that binding.

OWS operations usually do not allow the addition of elements. However, with respect to the core & extension pattern for service specifications (where the core service functionality is defined in the base specification and extension specifications may define further functionality that integrates with the existing one) it is desirable to have a place in service requests and responses where elements defined by extensions, for example policy assertions, can be added without the XML instances becoming invalid. The extension property of the *GetCapabilities* data type is the realization of such an extension point.

| Requirement | |
|---|---|
| http://www.opengis.net/spec/SPS/2.0/req/GetCapabilitiesRequest/sectionNames | |
| REQ 34. | The allowed set of service metadata (or Capabilities) section names and meanings shall be as specified in Tables 6 and 10 of [OGC 06-121r3], with the addition listed in Table 18 below. |

**Table 18 — Additional Section name value and meaning**

| Section name | Meaning |
|---|---|
| notifications | Return Notifications section in service metadata document |

The "Multiplicity and use" column in Table 3 of [OGC 06-121r3] and
Table 19 in this specification specifies the optionality of each listed parameter in the SPS GetCapabilities operation request.

| Requirement | |
|---|---|
| http://www.opengis.net/spec/SPS/2.0/req/GetCapabilitiesRequest/parameters | |
| REQ 35. | SPS clients and servers shall implement the *GetCapabilities* parameters as defined in Table 19. |

**Table 19 — Implementation of parameters in GetCapabilities operation request**

| Name | Multiplicity | Client implementation | Server implementation |
|---|---|---|---|
| service | Zero or one (optional) | May be implemented by all clients, using specified value<br>If parameter not provided, default value is to be assumed by service | Shall be implemented by all servers, checking that parameter is received with specified value<br>Default value shall be assumed if parameter is not provided in request |
| request | One (mandatory) | Shall be implemented by all clients, using specified value<br>In specific binding the value may be implied through encoded request structure | Shall be implemented by all servers, checking if parameter is received with specified value<br>In specific binding the value may be implied through encoded request structure |
| 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) | Each parameter may be implemented by each client<br>If parameter not provided, shall expect default response<br>If parameter provided, shall allow default or specified response | Each parameter may be implemented by each server<br>If parameter not implemented or not received, shall provide default response<br>If parameter implemented and received, shall provide specified response |
| updateSequence | Zero or one (optional) | | |
| acceptFormats | Zero or one (optional) | | |

### 7.3.2.4 Operation Response – Capabilities

The Capabilities data type defines the normal response returned by an SPS when a valid *GetCapabilities* request has been received.

It is derived from the OWSServiceMetadata data type defined by OWS Common (see clause 7.4 in [OGC 06-121r3]). It contains two more sections (contents and notifications) – depending upon the *GetCapabilities* request and the functionality supported by the service.

| Requirement | |
|---|---|
| http://www.opengis.net/spec/SPS/2.0/req/GetCapabilitiesResponse/dataType | |
| REQ 36. | The *Capabilities* data type shall include the properties of the OWSServiceMetadata data type (as defined in clauses 7.4.2 to 7.4.7 in [OGC 06-121r3]) with the additional properties according to Table 20. |

**Table 20 — Properties in the Capabilities data type**

| Name | Definition | Data type and values | Multiplicity and use |
|---|---|---|---|
| contents | metadata about the provided sensors and supported data encodings | SPSContents, see clause 7.3.3 | Zero or one (optional)<br>inclusion depends on the values in the Sections parameter of the GetCapabilities operation request |
| extension | container for elements defined by extension specifications | Any type<br>value is defined by the extension specification | Zero or more (optional)<br>use as explained for the extension property in the GetCapabilities operation request data type (see clause 7.3.2.3) |
| notifications | metadata about the supported notification functionality | NotificationProducerMetadata, see clause 8 in [OGC 09-001] | Zero or one (optional)<br>inclusion depends on the values in the Sections parameter of the GetCapabilities operation request |

| Requirement | |
|---|---|
| http://www.opengis.net/spec/SPS/2.0/req/GetCapabilitiesResponse/defaultVersion | |
| REQ 37. | A service implementing this standard shall at least be capable of providing a Capabilities document with version number “2.0.0” that is structured as defined in section 7.3.2.4. |

| Requirement | |
|---|---|
| http://www.opengis.net/spec/SPS/2.0/req/GetCapabilitiesResponse/sections | |
| REQ 38. | An SPS shall implement the sections of the Capabilities document listed in Table 21 according to the *Use* column in that table. |

| Requirement | |
|---|---|
| http://www.opengis.net/spec/SPS/2.0/req/GetCapabilitiesResponse/sectionListing | |
| REQ 39. | If the *Sections* parameter is supported for the *GetCapabilities* operation request the service shall list the supported section names as values of an accordingly named parameter in the metadata of the GetCapabilities operation. |

Clients can request any combination of the sections listed in the GetCapabilities operations metadata.

**Table 21 — SPS section name values and contents**

| Section name | Contents | Use |
|---|---|---|
| serviceIdentification | Metadata about this specific server (see clause 7.4.4 in [OGC 06-121r3]). | mandatory |
| serviceProvider | Metadata about the organization operating this server (see clause 7.4.5 in [OGC 06-121r3]). | mandatory |
| 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 (see clause 7.4.6 in [OGC 06-121r3]). | mandatory |
| contents | Metadata about the sensors provided by the SPS and supported data encodings (see clause 7.3.3 below). | mandatory |
| notifications | Metadata about the supported notification functionality (see clause 8 in [OGC 09-001]). | conditional<br>required if publish/subscribe functionality is realized by the service |

#### 7.3.2.4.1 OperationsMetadata section standard contents

For the SPS, the OperationsMetadata section is structured like for all OGC Web Services – as specified in Subclause 7.4.6 of [OGC 06-121r3].

#### 7.3.2.4.2 Advertising Implemented Operations

The parameter names and values to be used in the *OperationsMetadata* section, which indicate the implemented operations of an SPS instance, are specified in Table 22 and Table 23.

| Requirement | |
|---|---|
| http://www.opengis.net/spec/SPS/2.0/req/GetCapabilitiesResponse/implementedOperations | |
| REQ 40. | The implemented operations shall be listed in the *OperationsMetadata* by SPS instances according to the values defined in Table 22 and Table 23. |

In Table 22 and Table 23, 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 22 — Required values of OperationsMetadata section attributes**

| Attribute name | Attribute value | Meaning of attribute value |
|---|---|---|
| Operation.name | GetCapabilities | This server implements the GetCapabilities operation. |
| | DescribeSensor | This server implements the DescribeSensor operation. |
| | DescribeTasking | This server implements the DescribeTasking operation. |
| | Submit | This server implements the Submit operation. |
| | GetStatus | This server implements the GetStatus operation. |
| | GetTask | This server implements the GetTask operation. |
| | DescribeResultAccess | This server implements the DescribeResultAccess operation. |

**Table 23 — Optional values of OperationsMetadata section attributes**

| Attribute name | Attribute value | Meaning of attribute value |
|---|---|---|
| Operation.name | Reserve | This server implements the Reserve operation. |
| | Confirm | This server implements the Confirm operation. |
| | GetFeasibility | This server implements the GetFeasibility operation. |
| | Update | This server implements the Update operation. |
| | Cancel | This server implements the Cancel operation. |
| | UpdateSensorDescription | This server implements the UpdateSensorDescription operation. |

#### 7.3.2.4.3 Advertising Support for Status Logging

SPS instances can log the statuses of tasks and tasksing requests for any period of time.

| Requirement | |
|---|---|
| http://www.opengis.net/spec/SPS/2.0/req/GetCapabilitiesResponse/status-logging-supported | |
| REQ 41. | If an SPS service logs the complete state history of non-finalized tasks and tasking requests, it shall list the identifier of the state logger conformance class (see subclause 2.2) as (one of the) value(s) of the profile parameter in the Capabilities document's ServiceIdentification section. |

#### 7.3.2.4.4 Advertising Supported Operation Encodings

| Requirement | |
|---|---|
| http://www.opengis.net/spec/SPS/2.0/req/GetCapabilitiesResponse/supportedEncodings | |
| REQ 42. | SPS servers shall specify the supported encodings for HTTP POST based transfer of operation requests. Specifically, an *ows:Constraint* element shall be included with *PostEncoding* as the value of the *name* attribute supporting<br>a) the value "SOAP" to indicate that SOAP encoding is allowed, as specified in clause 8.<br>b) the value "XML" to indicate that XML encoding is allowed (without SOAP message encapsulation). |

#### 7.3.2.4.5 Advertising Other Operation Metadata

In addition to the optional values listed in Table 23, there are many optional values of the *name* attributes and *value* elements in the *OperationsMetadata* section. Most of these attributes and elements are for recording the domains of various parameters and quantities.

EXAMPLE 1 The domain of the *exceptionCode* parameter can record all the codes implemented for each operation by that specific server. Similarly, each of the *GetCapabilities* operation optional request parameters can have its domain recorded.

EXAMPLE 2 The domain of the Sections parameter in the *GetCapabilities* operation request can record all the sections implemented by that specific server.

#### 7.3.2.4.6 Advertising Supported Conformance Classes

| Requirement | |
|---|---|
| http://www.opengis.net/spec/SPS/2.0/req/GetCapabilitiesResponse/conformanceClass | |
| REQ 43. | Any SPS service shall document in its capabilities document the supported conformance classes. The identifier (a URI) of each supported conformance class shall be listed as a value of the *profile* property of the *ServiceIdentification* section. |

#### 7.3.2.5 Exceptions

| Requirement | |
|---|---|
| http://www.opengis.net/spec/SPS/2.0/req/GetCapabilitiesResponse/exception | |
| REQ 44. | When an SPS server encounters an error while performing a GetCapabilities operation, it shall return an exception message as specified in clause 7.2. |

If the *GetCapabilities* request contained the *Sections* parameter with value *notifications* but that value is not listed by the service for the *Sections* parameter (because the service does not implement publish/subscribe functionality) then an *InvalidParameterValue* exception with locator *Sections* or *sections* is returned.

#### 7.3.2.6 Examples

Clause 9.6 provides example XML instances for the GetCapabilities operation request and response.

### 7.3.3 Contents Package

#### 7.3.3.1 Introduction

This package contains the data types used to provide metadata about the sensors provided by an SPS and the supported data encodings.

In order to reduce the size of the Capabilities document by reducing the amount of redundant information in the contents section, the *property inheritance mechanism* defined in clause 22 of [OGC 09-001] is used.

#### 7.3.3.2 Data Types

The conceptual model of the Contents package is shown in the following UML diagram.

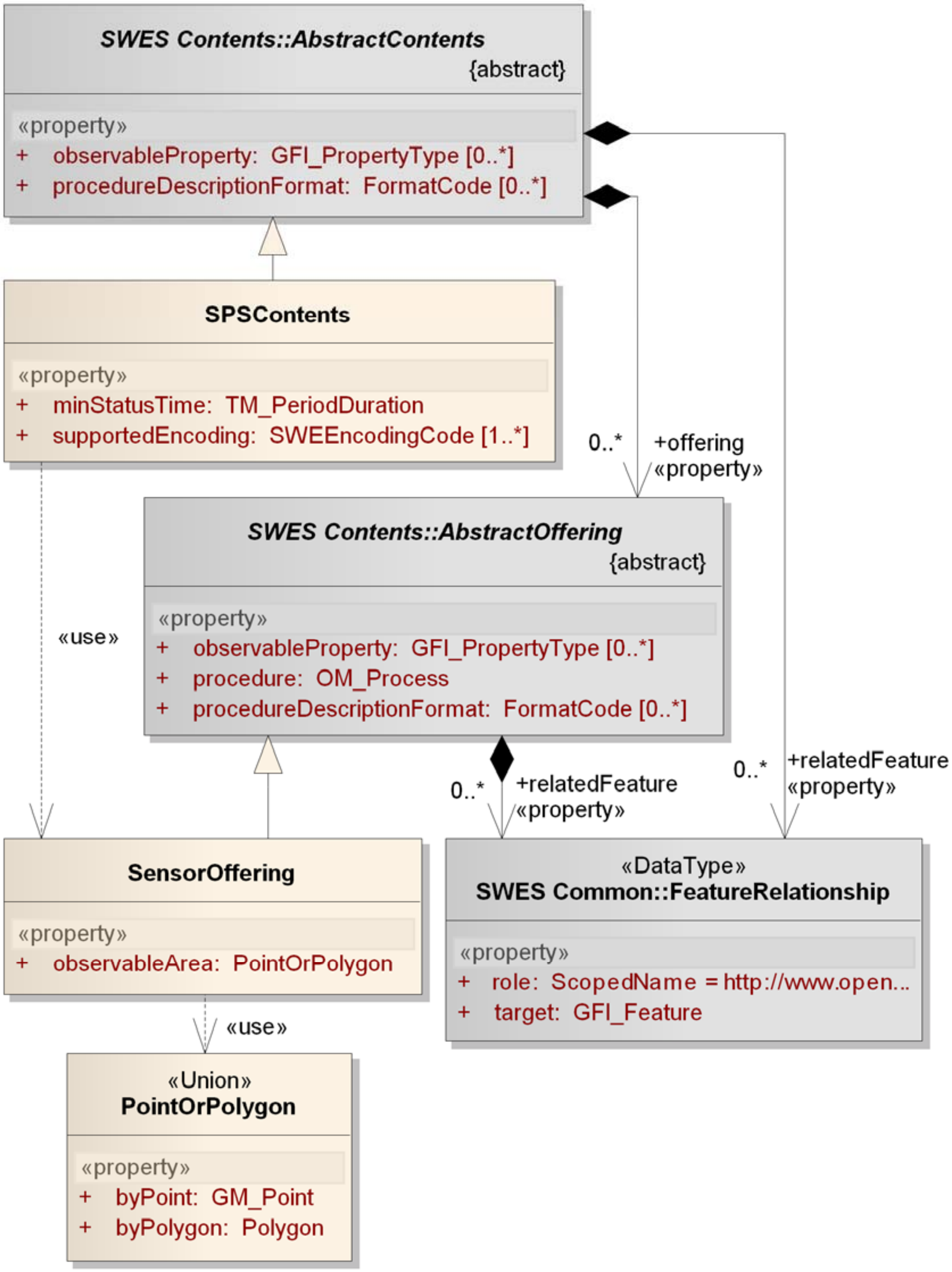

SWES Contents::AbstractContents
{abstract}
«property»
+ observableProperty: GFI_PropertyType [0..*]
+ procedureDescriptionFormat: FormatCode [0..*]
SPSContents
«property»
+ minStatusTime: TM_PeriodDuration
+ supportedEncoding: SWEEncodingCode [1..*]
0..*
+offering
«property»
«use»
SWES Contents::AbstractOffering
{abstract}
«property»
+ observableProperty: GFI_PropertyType [0..*]
+ procedure: OM_Process
+ procedureDescriptionFormat: FormatCode [0..*]
0..*
+relatedFeature
«property»
0..*
+relatedFeature
«property»
SensorOffering
«property»
+ observableArea: PointOrPolygon
«use»
«DataType»
SWES Common::FeatureRelationship
«property»
+ role: ScopedName = http://www.open...
+ target: GFI_Feature
«Union»
PointOrPolygon
«property»
+ byPoint: GM_Point
+ byPolygon: Polygon

**Figure 17 — Data types contained in the Contents package**

The details of each class contained in the package are explained in the following sub clauses.

#### 7.3.3.3 SPSContents

This data type defines the supported encodings for tasking parameter values and provides metadata about the sensors facaded by the service. In addition, it provides information on the storage time of task and task request status information. By default, SPS servers store the last status information only. After finalization of a task or task request, any SPS service stores this information until the *minStatusTime* has expired. This time period starts when the task or task request transitions into a final state. Optionally, SPS servers can store any additional historic status information. SPS servers indicate this capability by adding the *since* parameter to the metadata of the *GetStatus* operation in the *Capabilities* document (see clause 7.3.2.4.3 - Advertising Support for Status Logging on page 57).

The SPSContents acts as the *property provider* for a SensorOffering (see clause 22 in [OGC 09-001]).

The *SPSContents* (see Figure 17) type is derived from SWES *AbstractContents* type defined in clause 7 of [OGC09-001] and therefore inherits all the properties contained in that data type. SPSContents restricts the content model of AbstractContents in that it requires that the offering property is of type SensorOffering (see clause 7.3.3.4) or a subtype thereof.

| Requirement | |
|---|---|
| http://www.opengis.net/spec/SPS/2.0/req/GetCapabilitiesResponse/contents | |
| REQ 45. | The *SPSContents* data type shall contain the properties defined for SWES *AbstractContents*. In addition, it shall contain the properties according to Table 24 in combination with Table 25. |

**Table 24 — Properties in the SPSContents data type**

| Name | Definition | Data type and values | Multiplicity and use |
|---|---|---|---|
| minStatusTime | time period for which the services provides status information about finalized tasks or tasking requests | TM_PeriodDuration, see ISO 19108 | One (mandatory) |
| supportedEncoding | encoding supported by the service to encode tasking parameter values | SWEEncodingCode, see clause 10.2.3 in OGC 09-001<br>applicable code value(s) as defined in Table 25 | One or more (mandatory) |

**Table 25 — Code values applicable to the supportedEncoding property**

| Applicable Code Value(s) | Additional Note |
|---|---|
| http://www.opengis.net/swe/2.0/TextEncoding | best suited in most cases if ASCII encoded parameters are used |
| http://www.opengis.net/swe/2.0/XMLEncoding | generally applicable |
| http://www.opengis.net/swe/2.0/BinaryEncoding | suited for example for transferring image data to the service |

#### 7.3.3.4 SensorOffering

This data type contains metadata about a sensor provided by the service.

The *SensorOffering* (see Figure 17) type is derived from SWES *AbstractOffering* defined in clause 7 of [OGC 09-001] and therefore inherits all the properties contained in that type. *SensorOffering* does not restrict the content model of *AbstractOffering*.

| Requirement | |
|---|---|
| http://www.opengis.net/spec/SPS/2.0/req/GetCapabilitiesResponse/sensorOffering | |
| REQ 46. | The *SensorOffering* type shall contain the properties defined for SWES *AbstractOffering*. In addition, it shall contain the property according to Table 26. |

**Table 26 — Properties in the SensorOffering type**

| Name | Definition | Data type and values | Multiplicity and use |
|---|---|---|---|
| observable Area | the area that the sensor can observe | PointOrPolygon, see clause 7.3.3.5 | One (mandatory) |

The *SensorOffering* represents an *inheritor* of the properties contained in the *SPSContents*. The following table shows which of the properties defined in the content model of *SensorOffering* can be inherited and which cardinality is expected after the inheritance mechanism has been applied.

**Table 27 — Inheritance of SensorOffering properties (from SPSContents)**

| Property | Cardinality | Inheritance |
|---|---|---|
| procedure | 1 | NA |
| procedureDescriptionFormat | 1..* | replace |
| observableProperty | 1..* | replace |
| relatedFeature | 0..* | replace |
| observableArea | 1 | NA |

Thus, even though the UML model and schema encoding define the *observableProperty* and *procedureDescriptionFormat* properties as optional, they are mandatory in each *SensorOffering*. In other words, each offering has to include at least one value for these two properties after the property inheritance mechanism was applied.

#### 7.3.3.5 PointOrPolygon

This type represents a choice between the geometric types point or a polygon. Those two types are e.g. used to describe the point or the area observed by a sensor.

| Requirement | |
|---|---|
| http://www.opengis.net/spec/SPS/2.0/req/GetCapabilitiesResponse/PointOrPolygon | |
| REQ 47. | The PointOrPolygon union shall contain the properties/choices according to Table 28. |

**Table 28 — Properties in the PointOrPolygon union**

| Name | Definition | Data type and values | Multiplicity and use |
|---|---|---|---|
| byPoint | a point | GM_Point, see clause 10.3.1 in [OGC 07-036] | One (mandatory)<br>Because PointOrPolygon is a union, either a point or polygon shall be used (i.e. there is a choice between the properties) |
| byPolygon | a polygon | Polygon, see clause 10.5.4 in [OGC 07-036] | |

### 7.3.4 DescribeTasking Operation

#### 7.3.4.1 Introduction

The DescribeTasking operation allows SPS clients to retrieve the description of the data structures for the tasking parameters of a sensor. The data structure description is encoded in SWE Common (see clause 7.4 - SPS tasking parameters representation).

#### 7.3.4.2 Data Types

The conceptual model of the DescribeTasking operation is shown in the following UML diagram.

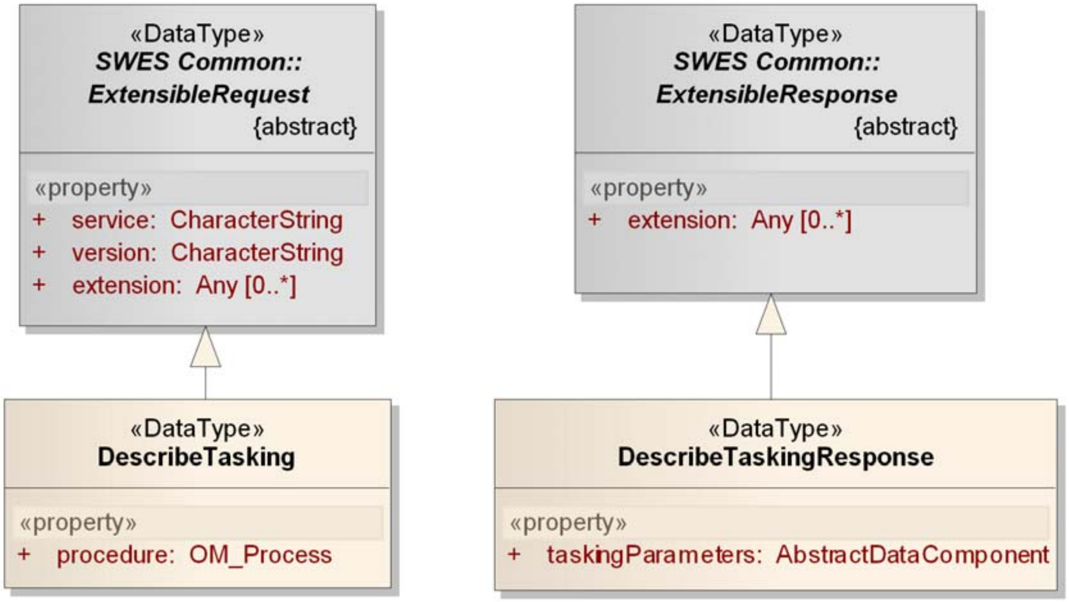


**Figure 18 — Data types of the DescribeTasking operation**

The details of the operation request and response are explained in the following subclauses.

#### 7.3.4.3 Operation Request - DescribeTasking

Sending an instance of the *DescribeTasking* data type to the service performs an SPS *DescribeTasking* operation request.

The *DescribeTasking* data type is derived from the SWES *ExtensibleRequest* data type specified in clause 9 of [OGC 09-001] and therefore inherits all the properties contained in that data type.

| Requirement | |
|---|---|
| http://www.opengis.net/spec/SPS/2.0/req/DescribeTaskingRequest/dataType | |
| REQ 48. | The *DescribeTasking* data type shall contain the properties defined for SWES *ExtensibleRequest*. In addition, it shall contain the property according to Table 29. |

**Table 29 — Property in the DescribeTasking data type**

| Name | Definition | Data type and values | Multiplicity and use |
|---|---|---|---|
| procedure | Pointer to the procedure (sensor) for which the tasking description is requested. | OM_Process [id]<br>see ISO DIS 19156 | One (mandatory) |
| id) Note: the primary use of this property is to provide a pointer/identifier – see OGC 09-001 clause 16.3.1 for further details. | | | |

#### 7.3.4.4 Operation Response - DescribeTaskingResponse

The *DescribeTaskingResponse* data type represents the response to an SPS *DescribeTasking* operation request.

The *DescribeTaskingResponse* data type is derived from the SWES *ExtensibleResponse* data type specified in clause 9 of [OGC 09-001] and therefore inherits all the properties contained in that data type. *DescribeTaskingResponse* does not restrict the content model of *ExtensibleResponse*.

| Requirement | |
|---|---|
| http://www.opengis.net/spec/SPS/2.0/req/DescribeTaskingResponse/dataType | |
| REQ 49. | The *DescribeTaskingResponse* data type shall contain the properties defined for SWES *ExtensibleResponse*. In addition, it shall contain the property according to Table 30. |

**Table 30 – Properties in the DescribeTaskingResponse data type**

| Name | Definition | Data type and values | Multiplicity and use |
|---|---|---|---|
| taskingParameters | description of tasking parameters for tasking the requested asset | AbstractDataComponent, see clause 7.2 in [OGC 08-094] | One (mandatory) |

The *taskingParameter* property shall be fully identified with a name defined by the SPS – see OGC 08-094 for further details. The XML Schema encoding of the *DescribeTaskingResponse* ensures (via the *soft-typed* tagged value) that such a name can be added by the service. The name is important for clients when certain encodings are used to encode the tasking paramter data, and the encodings require naming of SWE Common data components (like the XML encoding).

#### 7.3.4.5 Exceptions

| Requirement | |
|---|---|
| http://www.opengis.net/spec/SPS/2.0/req/DescribeTaskingResponse/exceptions | |
| REQ 50. | When an SPS server encounters an error while performing a DescribeTasking operation, it shall return an exception message as specified in clause 7.2. |

#### 7.3.4.6 Examples

Clause 9.6 provides example XML instances for the DescribeTasking operation request and response.

### 7.3.5 Submit Operation

#### 7.3.5.1 Introduction

The Submit operation allows SPS clients to submit a tasking request for an asset. The client encodes the tasking parameters according to the parameter description defined in the *DescribeTasking* response. SPS servers do a feasibility check of the request and perform the task if applicable.

#### 7.3.5.2 Data Types

The conceptual model of the Submit operation is shown in the following UML diagram.

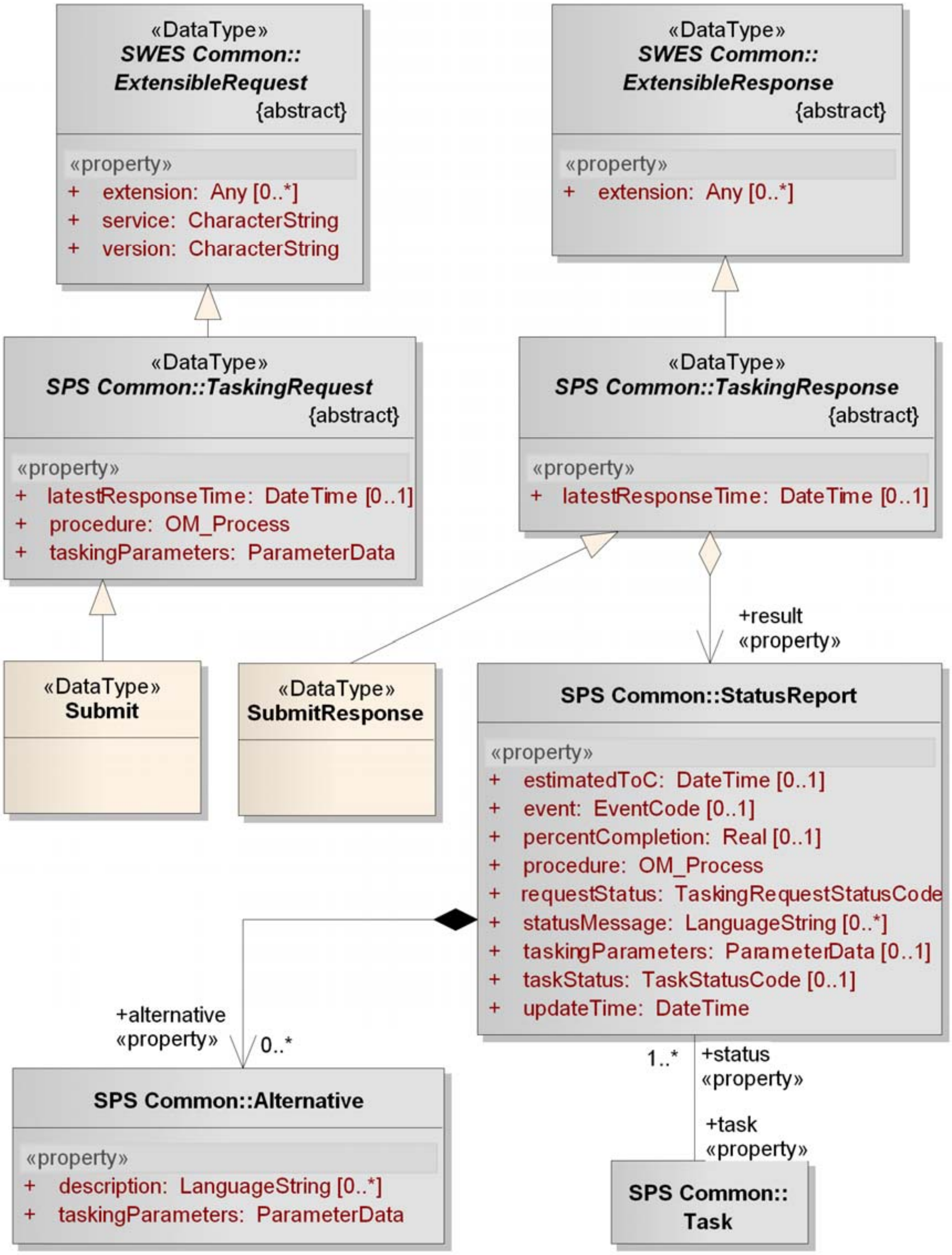

«DataType»
SWES Common::
ExtensibleRequest
{abstract}
«property»
+ extension: Any [0..*]
+ service: CharacterString
+ version: CharacterString
«DataType»
SWES Common::
ExtensibleResponse
{abstract}
«property»
+ extension: Any [0..*]
«DataType»
SPS Common::TaskingRequest
{abstract}
«property»
+ latestResponseTime: DateTime [0..1]
+ procedure: OM_Process
+ taskingParameters: ParameterData
«DataType»
SPS Common::TaskingResponse
{abstract}
«property»
+ latestResponseTime: DateTime [0..1]
+result
«property»
«DataType»
Submit
«DataType»
SubmitResponse
SPS Common::StatusReport
«property»
+ estimatedToC: DateTime [0..1]
+ event: EventCode [0..1]
+ percentCompletion: Real [0..1]
+ procedure: OM_Process
+ requestStatus: TaskingRequestStatusCode
+ statusMessage: LanguageString [0..*]
+ taskingParameters: ParameterData [0..1]
+ taskStatus: TaskStatusCode [0..1]
+ updateTime: DateTime
+alternative
«property»
0..*
1..*
+status
«property»
+task
«property»
SPS Common::Alternative
«property»
+ description: LanguageString [0..*]
+ taskingParameters: ParameterData
SPS Common::
Task

**Figure 19 — Data types of the Submit operation**

The details of the operation request and response are explained in the following sub clauses.

### 7.3.5.3 Operation Request - Submit

Sending an instance of the Submit data type to the service performs an SPS Submit operation request.

The *Submit* data type is derived from the *TaskingRequest* data type (see clause 7.3.1.3) and therefore inherits all the properties contained in that data type. *Submit* neither restricts the content model of *TaskingRequest* nor adds additional properties.

| Requirement | |
|---|---|
| http://www.opengis.net/spec/SPS/2.0/req/SubmitRequest/dataType | |
| REQ 51. | The *Submit* data type shall contain the properties defined for the *TaskingRequest* data type. |

### 7.3.5.4 Operation Response - SubmitResponse

The *SubmitResponse* data type represents the response to an SPS Submit operation request.

The *SubmitResponse* data type is derived from the *TaskingResponse* data type (see clause 7.3.1.4) and therefore inherits all the properties contained in that data type. *SubmitResponse* neither restricts the content model of *TaskingResponse* nor adds additional properties.

| Requirement | |
|---|---|
| http://www.opengis.net/spec/SPS/2.0/req/SubmitResponse/dataType | |
| REQ 52. | The *SubmiResponse* data type shall contain the properties defined for the *TaskingResponse* data type. |

A *SubmitResponse* contains a *StatusReport* (see clause 7.3.1.5) to inform about the result of the requested operation. As a *Submit* request is a tasking request, the final result of that request might not be directly available and would then be *Pending*. The contents of the *StatusReport* properties after all possible state transitions are defined in table Table 31.

If the request is reported to be pending then a client needs to retrieve information about the final status of the request in another way, per default by the *GetStatus* operation.

**Table 31 – StatusReport usage for different state transitions of a Submit request**

| property name / cardinality | Submit Request State Transitions (From → To) | | | | | | | |
|---|---|---|---|---|---|---|---|---|
| | Initial → Pending | Initial → Accepted | | Initial → Rejected | Pending → Accepted | | Pending → Rejected | Pending → Rejected (request expired) |
| | | task is in execution | task is already completed 1 | | task is in execution | task is already completed 1 | | |
| task / 1 | new identifier provided by service | | | | identifier previously provided by service | | | |
| estimatedToC / 0..1 | NA | optional | NA | NA | optional | NA | NA | NA |
| event (code) / 0..1 | NA | TaskSubmitted | TaskCompleted | NA | TaskSubmitted | TaskCompleted | NA | TaskingRequestExpired |
| percentCompletion / 0..1 | NA | 0 | 100 | NA | 0 | 100 | NA | NA |
| procedure / 1 | identifier of procedure for which Submit request was made | | | | | | | |
| requestStatus (code) / 1 | Pending | Accepted | Accepted | Rejected | Accepted | Accepted | Rejected | Rejected |
| statusMessage / 0..* | service may provide additional information to client in human readable form | | | | | | | |
| taskingParameters / 0..1 | NA | | | | | | | |
| taskStatus (code) / 0..1 | NA | InExecution | Completed | NA | InExecution | Completed | NA | NA |
| updateTime / 1 | point in time when transition was made | | | | | | | |
| alternative | may be provided by service | | | | | | | |
| StatusReport encoded as ReservationReport | NA | | | | | | | |
| Applicable in Submit Response | yes | yes | yes | yes | no | no | no | no |

NA = not applicable, means element is not used in response

Notes:
1 this is a shortcut to convey information in the SubmitResponse that the Submit request was accepted and the submitted task already made the transitions Initial → InExecution and InExecution → Final (TaskCompleted)

| Requirement | |
|---|---|
| http://www.opengis.net/spec/SPS/2.0/req/SubmitResponse/taskAlreadyCompleted | |
| REQ 53. | If the *SubmitResponse* has taskStatus *Completed* then the service shall have performed state logging and notification – if supported – for the according task. That task then has made the transitions *Initial* → *InExecution* and *InExecution* → *Final* (*TaskCompleted*). |

### 7.3.5.5 Exceptions

| Requirement | |
|---|---|
| http://www.opengis.net/spec/SPS/2.0/req/SubmitResponse/exceptions | |
| REQ 54. | When an SPS server encounters an error while performing a Submit operation, it shall return an exception message as specified in clause 7.2. |

### 7.3.5.6 Examples

Clause 9.6 provides example XML instances for the Submit operation request and response.

**Issue Name**: After Submission Notification Gap (JE, Dec 17th,09)

**Issue Description:** When a service implements publish/subscribe functionality (see clause 8) and publishes notifications on status changes of submitted tasks, then a client might miss notifications for his task unless he subscribed for status changes of all tasks beforehand.

A client does not get the identifier for his task before it actually submitted it – only the SubmitResponse contains the task identifier, which can then be used in a subscription for notifications of that task. However, in the time it takes from the actual submission to a completed subscription the service might already have published notifications for the task. These notifications will be missed by the client.

The same issue applies for task reservations.

The client could try to retrieve the missing status information via a GetStatus request (see clause 7.3.6) using the *since* parameter in the request. However, implementation of that parameter is optional for an SPS (see clause 7.3.2.4.3).

A client might also reserve the task first, then subscribe for it and then confirm it. However, implementation of the Reserve operation is optional for SPS.

A solution could be to design an extension that allowed the inclusion of a subscription request directly in the submit request (as a request extension parameter), with the semantics that all notifications published for the task – if the request is accepted – are in the scope of that subscription.

**Resolution:** What usually is important to the client is to get the most updated status of his task. So if, right after submitting the task and subscribing for notifications about it, the client issues a GetStatus request, the response of this call fills the gap in the sense that the client is then aware of the latest status of the task and that he will be notified of any further changes.

### 7.3.6 GetStatus Operation

### 7.3.6.1 Introduction

The *GetStatus* operation allows SPS clients to retrieve status reports about a tasking request or a task. This operation is the default mechanism to retrieve status information about a task or tasking request.

As explained in clause 6.3.6, a task or tasking request makes one or more state transitions before reaching its final state (see Figure 8 and Figure 9 in clause 6.3.6). While not in the final state, a task can transition between several other states.

Clients can retrieve a status report via the *GetStatus* operation. The response to the operation either contains a (number of) *StatusReport(s)* or *ReservationReport(s)*.

By default, the *GetStatusResponse* contains a single status report. This report identifies the current/latest state of a task/tasking request. The *updateTime* parameter in the report defines when that state was entered.

SPS servers announce in their Capabilities if all state changes are tracked for non-finalized tasks and tasking requests (see clauses 7.3.2.4.3 and 7.3.3.3); this also defines how long a service needs to keep the according information before it can discard it.

| Requirement | |
|---|---|
| http://www.opengis.net/spec/SPS/2.0/req/GetStatus/service-metadata/since-parameter | |
| REQ 55. | If an SPS service logs the complete state history of non-finalized tasks and tasking requests and thus supports the state logger conformance class (see subclause 7.3.2.4.3), it shall add the parameter *since* to the metadata of the *GetStatus* operation in the *OperationsMetadata* section of the service's Capabilities document. The value of this parameter shall be *ows:AnyValue*. |

If supported by the server, clients can request the status history of a task/tasking request by using a *GetStatus* request with *since* parameter. If supported, the SPS shall return all status reports it has stored for the task, with an *updateTime* that is not before and not after the time period defined with the *since* time as begin position and the point in time when the *GetStatus* request was received by the service as end position.

If the GetStatusResponse contains multiple status reports then it is recommended that the service lists them in ascending temporal order regarding the update time of each report.

If the *since* parameter is used in a *GetStatus* request then the response may not contain any *StatusReport* in case that no status transition happened in that period. To retrieve the current status, an additional *GetStatus* request without *since* parameter becomes necessary. This standard behaviour could be overwritten in an extension to this standard. For example, it could be enforced that at least the last valid *StatusReport* would be returned. This behaviour was intentionally avoided here to allow for an operation that checks if any status updates happened in a given time period in the past.

#### 7.3.6.2 Data Types

The conceptual model of the GetStatus operation is shown in the following UML diagram.

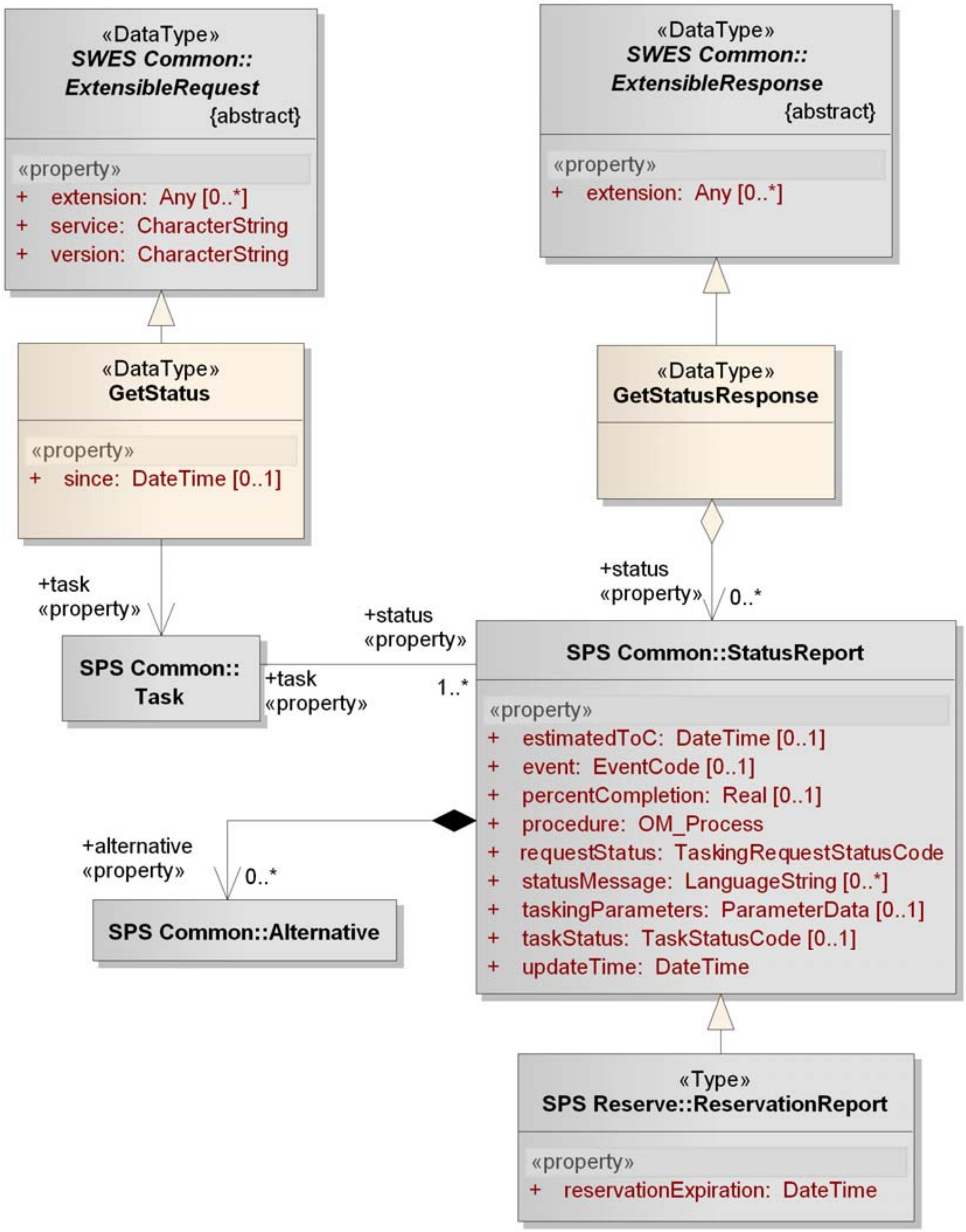


**Figure 20 — Data types of the GetStatus operation**

The details of the operation request and response are explained in the following subclauses.

### 7.3.6.3 Operation Request - GetStatus

Sending an instance of the GetStatus data type to the service performs an SPS *GetStatus* operation request.

The *GetStatus* data type is derived from the SWES *ExtensibleRequest* data type specified in clause 9 of [OGC 09-001] and therefore inherits all the properties contained in that data type. *GetStatus* does not restrict the content model of *ExtensibleRequest*.

| Requirement | |
|---|---|
| http://www.opengis.net/spec/SPS/2.0/req/GetStatusRequest/dataType | |
| REQ 56. | The *GetStatus* data type shall contain the properties defined for SWES *ExtensibleRequest*. In addition, it shall contain the properties according to Table 32. |

**Table 32 — Properties in the GetStatus data type**

| Name | Definition | Data type and values | Multiplicity and use |
|---|---|---|---|
| task | Pointer to the task for which status information is requested. | Task [id]<br>see clause 7.3.1.6<br>value as provided by SPS in response to a previous tasking request | One (mandatory) |
| since | point in time in the past that denotes the begin of the time period – ended by the time when the request was received by the service – for which status reports of the identified task are requested | DateTime<br>(see ISO 19103 and OGC 07-036 Table D.2)<br>value shall be a point in time in the past | One to zero (optional)<br>if not provided in the request only the latest state shall be reported (default behavior of the operation) |
| id) Note: the primary use of this property is to provide a pointer/identifier – see OGC 09-001 clause 16.3.1 for further details. | | | |

#### 7.3.6.4 Operation Response - GetStatusResponse

The *GetStatusResponse* data type represents the response to an SPS *GetStatus* operation request.

The *GetStatusResponse* data type is derived from the SWES *ExtensibleResponse* data type (see clause 9 of [OGC 09-001] and therefore inherits all the properties contained in that data type. *GetStatusResponse* does not restrict the content model of *ExtensibleResponse*.

| Requirement | |
|---|---|
| http://www.opengis.net/spec/SPS/2.0/req/GetStatusResponse/dataType | |
| REQ 57. | The *GetStatusResponse* data type shall contain the properties defined for SWES *ExtensibleResponse*. In addition, it shall contain the property according to Table 33. |

**Table 33 — Properties in the GetStatusResponse data type**

| Name | Definition | Data type and values | Multiplicity and use |
|---|---|---|---|
| status | status report providing information about the current or – if requested via the "since" parameter – a previous state of the requested task/tasking request | StatusReport (see 7.3.1.5)<br>Property usage for providing status information of a GetFeasibility or Update request as defined in Table 34, of a Reserve or Submit request as defined in<br>Table 35 and for a scheduled task as defined in Table 36 & Table 37; all tables also indicate when a ReservationReport rather than a StatusReport is used to encode the status information | Zero to many (mandatory)<br>whether zero, one or more reports are contained in the *GetStatus* response depends on the status of the task/tasking request (it could have made only one transition but also more), if the service supports the since parameter (if not then only the current status is returned) and that parameter is actually used in the request (even though clients may have the option to request more status information, without using the 'since' parameter they are only interested in the current status) |

The following tables define in more detail how status reports are used in a *GetStatus* operation response to provide status information of a task or tasking request.

With the *GetStatus* operation it is possible to retrieve the currently valid tasking parameter values. For performance reasons, the current tasking parameter settings are omitted from tasking request responses. In contrast, tasking request respones may contain alternative sets of tasking parameters, which is not possible in *GetStatus* responses.

| Requirement | |
|---|---|
| http://www.opengis.net/spec/SPS/2.0/req/GetStatusResponse/informationExtent | |
| REQ 58. | If the *since* option (i.e., the state logger conformance class) is supported by the service and a client uses the parameter in a *GetStatus* request to retrieve the status of a *Reserve* or *Submit* tasking request, then all state information shall be returned on that tasking request, including information about the task which was scheduled once the tasking request was accepted. |

| Requirement | |
|---|---|
| http://www.opengis.net/spec/SPS/2.0/req/GetStatusResponse/validTime | |
| REQ 59. | Status information is only reported if the according status/reservation report updateTime falls within the reporting period as defined in Figure 21and Figure 22. |

| Requirement | |
|---|---|
| http://www.opengis.net/spec/SPS/2.0/req/GetStatusResponse/informationExtent2 | |
| REQ 60. | If the *GetStatus* request was intended to retrieve the status of a task but the *since* parameter denotes a point in time when the tasking request that caused the task was not yet finalized, then the information on that tasking request shall also be included in the response, together with all status information about the task. |

The following figure depicts some exemplary tasks/tasking request in the left column and corresponding state transitions on a time axis running from left to right in the main window. The arrows indicate the lifetime of tasks and tasking requests. The vertical lines indicate state transitions.

Given a *GetStatus* request with *since* parameter defining the time period shown in the figure, the *StatusReport* contains information only about state transitions highlighted in red. In contrast, unreported state transitions that are illustrated as thin grey bars. As shown, only state transitions within the time period are reported.

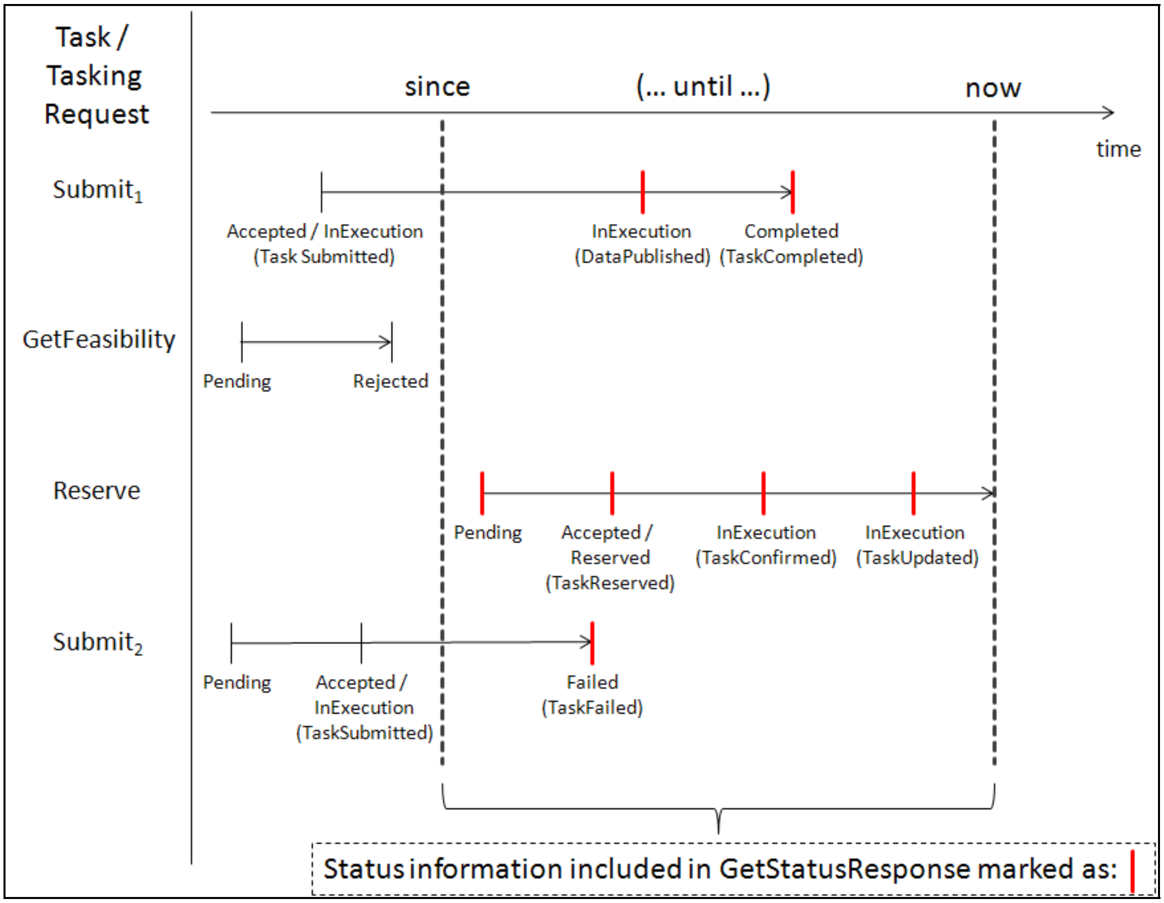

Task / Tasking Request
since
(… until …)
now
time
Submit1
Accepted / InExecution (Task Submitted)
InExecution (DataPublished)
Completed (TaskCompleted)
GetFeasibility
Pending
Rejected
Reserve
Pending
Accepted / Reserved (TaskReserved)
InExecution (TaskConfirmed)
InExecution (TaskUpdated)
Submit2
Pending
Accepted / InExecution (TaskSubmitted)
Failed (TaskFailed)
Status information included in GetStatusResponse marked as:

**Figure 21 – Status information returned for various exemplary tasks/tasking requests when the "since" parameter was used in GetStatus request**

Figure 22 illustrates the information returned for the same tasking situation, but the GetStatus operation is used without the *since* parameter.

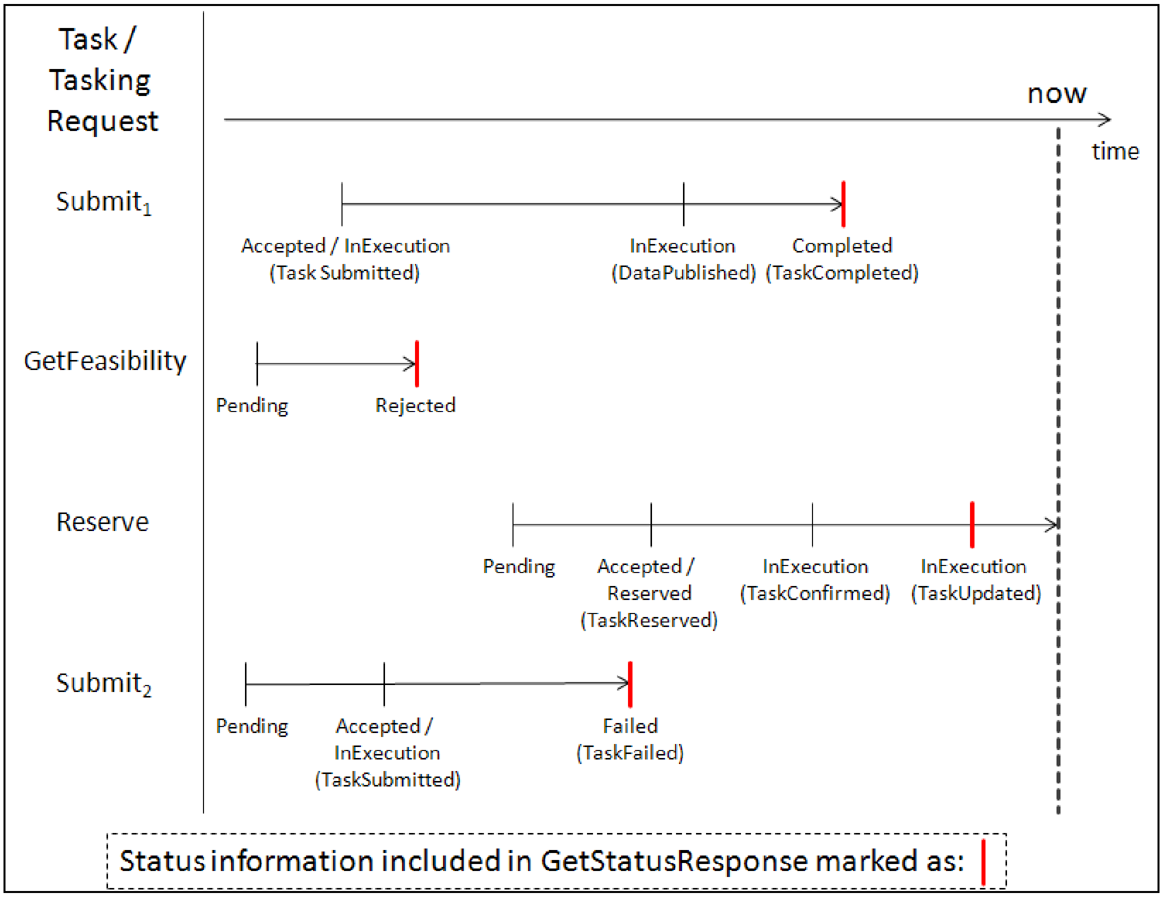

Task / Tasking Request
now
time
Submit$_1$
Accepted / InExecution (Task Submitted)
InExecution (DataPublished)
Completed (TaskCompleted)
GetFeasibility
Pending
Rejected
Reserve
Pending
Accepted / Reserved (TaskReserved)
InExecution (TaskConfirmed)
InExecution (TaskUpdated)
Submit$_2$
Pending
Accepted / InExecution (TaskSubmitted)
Failed (TaskFailed)
Status information included in GetStatusResponse marked as:

**Figure 22 – Status information returned for various exemplary tasks/tasking requests when the "since" parameter was not used in GetStatus request**

In this situation, the latest status is reported each time. This scenario assumes that the SPS server hasn't deleted any information about finalized tasks yet. See section 7.3.3.3 for more information on status hold-back time.

Table 34 provides an overview of the usage of the various *StatusReport* properties in a *GetStatus* response for tasking requests generated in consequence of *GetFeasibility* and *Update* requests. Table 35 provides the same information for tasking requests generated in consequence of *Reserve* and *Submit* requests.

Once the tasks are scheduled, i.e. the tasking request was accepted, the properties shall be used as illustrated in Table 36 and Table 37.

**Table 34 – Providing status information on GetFeasibility and Update requests**

| **property name/cardinality** | **State Transition (From → To)** | | | |
|---|---|---|---|---|
| | **Initial → Pending** | **Initial \| Pending → Accepted** | **Initial \| Pending → Rejected** | **Pending → Rejected (TaskingRequestExpired)** |
| task/1 | identifier provided in *GetStatus* request [1] | | | |
| estimatedToC/0..1 | NA [2] | | | |
| event (code)/0..1 | NA | NA | NA | TaskingRequestExpired |
| percentCompletion /0..1 | NA [3] | | | |
| procedure/1 | identifier of procedure that GetFeasibility request was made for/that belongs to task for which Update request was made | | | |
| requestStatus (code)/1 | Pending | Accepted | Rejected | Rejected |
| statusMessage/0..* | service may provide additional information to client in human readable form | | | |
| taskingParameters/ 0..1 | NA | parameters used in tasking request | parameters used in tasking request | parameters used in tasking request |
| taskStatus (code)/0..1 | NA | | | |
| updateTime/1 | point in time when transition into new state was made | | | |
| alternative/0..* | NA | | | |
| StatusReport is Encoded as ReservationReport | NA | | | |

NA = not applicable, means that property is not used in the response

Notes:
1 has to be an identifier known to the service (otherwise an exception is thrown)
2 only applicable to scheduled tasks that have not been finalized yet
3 only applicable to scheduled tasks that are being or have been executed

**Table 35 – Providing status information on Reserve and Submit requests**

| **property name/cardinality** | **State Transition (From → To)** | | | | |
|---|---|---|---|---|---|
| | **Initial → Pending** | **Initial \| Pending → Accepted / Reserved (tasking request was Reserve)** | **Initial \| Pending → Accepted / InExecution (tasking request was Submit)** | **Initial \| Pending → Rejected** | **Pending → Rejected (TaskingRequestExpired)** |
| task/1 | identifier provided in request[1] | | | | |
| estimatedToC/0..1 | NA | optional | optional | NA | NA |
| event (code)/0..1 | NA | TaskReserved | TaskSubmitted | NA | TaskingRequestExpired |
| percentCompletion /0..1 | NA | NA | 0 | NA | NA |
| procedure/1 | identifier of procedure that Reserve/Submit request was made for | | | | |
| requestStatus (code)/1 | Pending | Accepted | Accepted | Rejected | Rejected |
| statusMessage/0..* | service may provide additional information to client in human readable form | | | | |
| taskingParameters/ 0..1 | NA | parameters used in tasking request | parameters used in tasking request | parameters used in tasking request | parameters used in tasking request |
| taskStatus (code)/0..1 | NA | Reserved | InExecution | NA | NA |
| updateTime/1 | point in time when transition was made | point in time when transition into Accepted/Reserved state was made | point in time when transition into Accepted/InExecution state was made | point in time when transition was made | point in time when transition was made |
| alternative/0..* | NA | | | | |
| StatusReport is Encoded as ReservationReport | no | yes | no | no | no |

NA = not applicable, means that the element is not used in the response
Notes:
1 has to be an identifier known to the service (otherwise an exception is thrown)

**Table 36 – Providing status information on scheduled tasks (part 1)**

| property name/cardinality | State Transition (From → To) | | | |
|---|---|---|---|---|
| | Reserved → Reserved (task updated) | Reserved → InExecution (task confirmed) | InExecution → InExecution (task updated) | InExecution → InExecution (data published) |
| task/1 | identifier provided in request [1] | | | |
| estimatedToC/0..1 | optional | | | |
| event (code)/0..1 | TaskUpdated | TaskConfirmed | TaskUpdated | DataPublished |
| percentCompletion /0..1 | optional | 0 | optional | optional |
| procedure/1 | identifier of procedure associated to the task | | | |
| requestStatus (code)/1 | Accepted | | | |
| statusMessage/0..* | service may provide additional information to client in human readable form | | | |
| taskingParameters/ 0..1 | parameters used in update request | NA | parameters used in update request | NA |
| taskStatus (code)/0..1 | Reserved | InExecution | InExecution | InExecution |
| updateTime/1 | point in time when transition was made | | | |
| alternative/0..* | NA | | | |
| StatusReport is Encoded as ReservationReport | yes | no | no | no |
| NA = not applicable<br><br>Notes:<br>1 has to be an identifier known to the service (otherwise an exception is thrown) | | | | |

**Table 37 – Providing status information on scheduled tasks (part 2)**

| property name/cardinality | State Transition (From → To) | | | |
|---|---|---|---|---|
| | **Reserved → Final (ReservationExpired)** | **InExecution → Final (TaskCompleted)** | **Scheduled (Reserved or InExecution) → Final (TaskCancelled)** | **Scheduled (Reserved or InExecution) → Final (TaskFailed)** |
| task/1 | identifier provided in request [1] | | | |
| estimatedToC/0..1 | NA [2] | | | |
| event (code)/0..1 | ReservationExpired | TaskCompleted | TaskCancelled | TaskFailed |
| percentCompletion/0..1 | NA | 100 | optional | optional |
| procedure/1 | identifier of procedure associated to the task | | | |
| requestStatus (code)/1 | Accepted | | | |
| statusMessage/0..* | service may provide additional information to client in human readable form | | | |
| taskingParameters/0..1 | NA | | | |
| taskStatus (code)/0..1 | Expired | Completed | Cancelled | Failed |
| updateTime/1 | point in time when transition into new state was made | | | |
| alternative/0..* | NA | | | |
| StatusReport is Encoded as ReservationReport | yes | no | no | no |

NA = not applicable

Notes:

1 has to be an identifier known to the service (otherwise an exception is thrown)

2 only applicable to scheduled tasks that have not been finalized yet

#### 7.3.6.5 Exceptions

| Requirement | |
|---|---|
| http://www.opengis.net/spec/SPS/2.0/req/GetStatusResponse/exceptions | |
| REQ 61. | When an SPS server encounters an error while performing a *GetStatus* operation, it shall return an exception message as specified in clause 7.2. In addition:<br>• If a *GetStatus* request contains a "since" property but the server does not support state logger functionality (i.e., it only keeps track of the current/last status of a task/tasking request - see clause 7.3.2.4.3 for further information), an exception with code *OptionNotSupported* and locator value *since* shall be thrown.<br>• If an SPS has removed status information for a requested task/tasking request after the required provision time has passed, it shall throw an *StatusInformationExpired* exception. |

#### 7.3.6.6 Examples

Clause 9.6 provides example XML instances for the GetStatus operation request and response.

### 7.3.7 GetTask Operation

#### 7.3.7.1 Introduction

The *GetTask* operation allows SPS clients to retrieve complete information about a given task or tasking request. Currently, this operation is only marginal different from *GetStatus*. The main reason for this operation is to serve as an extension point for future extensions to this standard.

This includes status information about the task. Per default only the latest status is provided by an SPS. If *state logger* functionality is supported by the service (see clause 7.3.2.4.3) then the complete state history shall be returned. If the GetTaskResponse contains multiple status reports then it is recommended that the service lists them in ascending temporal order regarding the update time of each report.

However, a service may discard such information after a certain point in time. This point in time is defined by the *minStatusTime* value provided in the *Contents* section (see clause 7.3.3.3) of the service's Capabilities document. In that case the service throws an according exception.

### 7.3.7.2 Data Types

The conceptual model of the *GetTask* operation is shown in the following UML diagram.

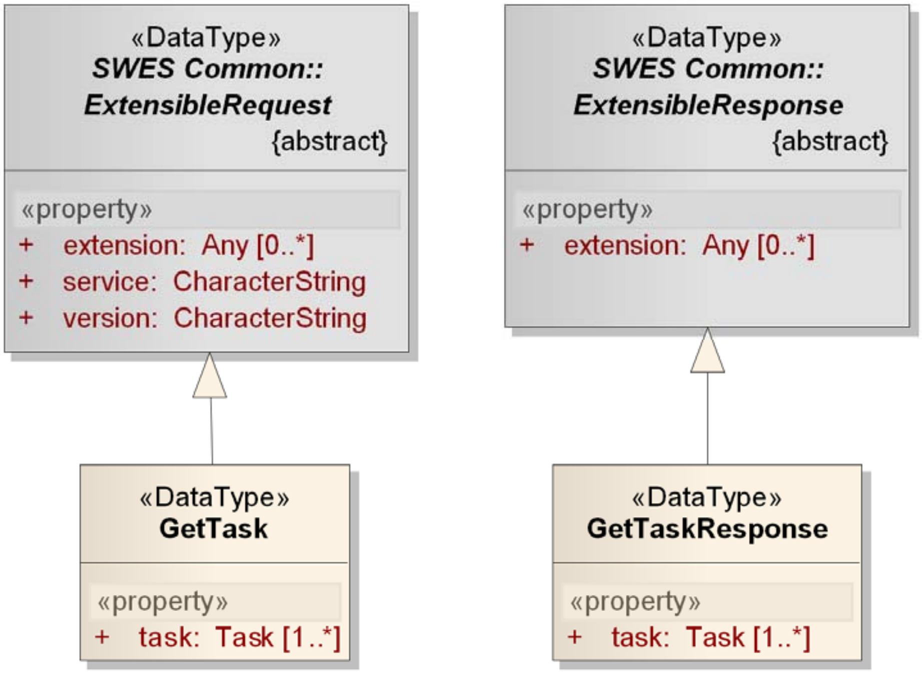


**Figure 23 — Data types of the GetTask operation**

The details of the operation request and response are explained in the following subclauses.

### 7.3.7.3 Operation Request – GetTask

Sending an instance of the *GetTask* data type to the service performs an SPS *GetTask* operation request.

The *GetTask* data type is derived from the SWES *ExtensibleRequest* data type specified in clause 9 of [OGC 09-001] and therefore inherits all the properties contained in that data type. *GetTask* does not restrict the content model of *ExtensibleRequest*.

| Requirement | |
|---|---|
| http://www.opengis.net/spec/SPS/2.0/req/GetTaskRequest/dataType | |
| REQ 62. | The *GetTask* data type shall contain the properties defined for SWES *ExtensibleRequest*. In addition, it shall contain the property according to Table 38. |

**Table 38 — Properties in the GetTask data type**

| Name | Definition | Data type and values | Multiplicity and use |
|---|---|---|---|
| task | Pointer to the task on which information is requested. | Task [id]<br>see clause 7.3.1.6 | One to many (mandatory) |
| id) Note: the primary use of this property is to provide a pointer/identifier – see OGC 09-001 clause 16.3.1 for further details. | | | |

#### 7.3.7.4 Operation Response – GetTaskResponse

The *GetTaskResponse* data type represents the response to an SPS *GetTask* operation request.

The *GetTaskResponse* data type is derived from the SWES *ExtensibleResponse* data type (see clause 9 of [OGC 09-001] and therefore inherits all the properties contained in that data type. *GetTaskResponse* does not restrict the content model of *ExtensibleResponse*.

| Requirement | |
|---|---|
| http://www.opengis.net/spec/SPS/2.0/req/GetTaskResponse/dataType | |
| REQ 63. | The *GetTaskResponse* data type shall contain the properties defined for SWES *ExtensibleResponse*. In addition, it shall contain the property according to Table 39. |

**Table 39 — Properties in the GetTaskResponse data type**

| Name | Definition | Data type and values | Multiplicity and use |
|---|---|---|---|
| task | the task that was requested | Task<br>see clause 7.3.1.6 | One to many (mandatory) |

| Requirement | |
|---|---|
| http://www.opengis.net/spec/SPS/2.0/req/GetTaskResponse/properties | |
| REQ 64. | A task (or tasking request) contained in the GetTaskResponse shall provide status information for the state transition(s) it made according to Table 34 to Table 37. |

To clarify which status information is provided for a task in the *GetTaskResponse*, we consider the possible cases. If the *state logger* conformance class is not supported by the service then per default the service only stores the latest state of a task/tasking request. As a result, only information about the latest status would be included in the *GetTaskResponse* for a task – this is similar to the situation for the *GetStatus* operation depicted in Figure 22. If, on the other hand, the *state logger* conformance class is supported by the service then the complete status information of a task/tasking request shall be provided in the *GetTaskResponse*. Figure 24 shows exemplary cases.

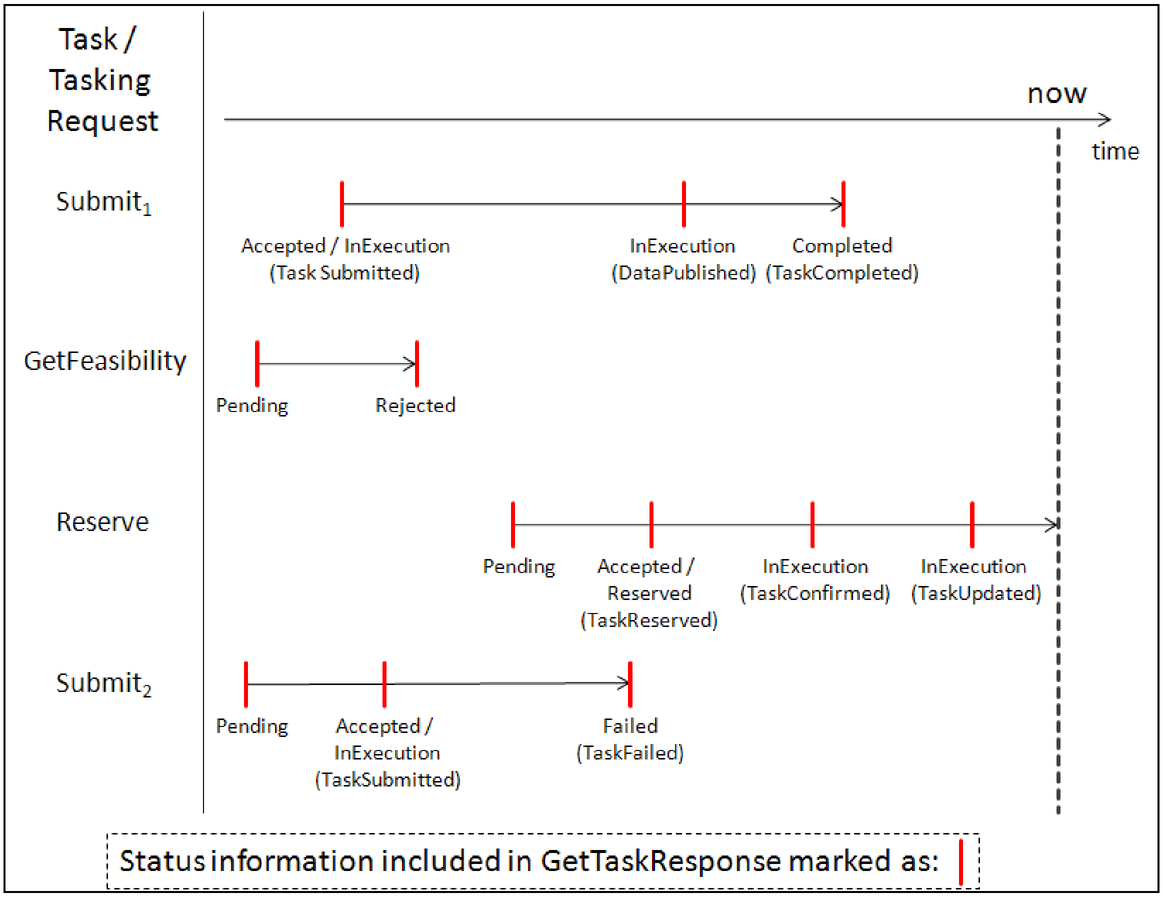

Task / Tasking Request
now
time
Submit$_1$
Accepted / InExecution (Task Submitted)
InExecution (DataPublished)
Completed (TaskCompleted)
GetFeasibility
Pending
Rejected
Reserve
Pending
Accepted / Reserved (TaskReserved)
InExecution (TaskConfirmed)
InExecution (TaskUpdated)
Submit$_2$
Pending
Accepted / InExecution (TaskSubmitted)
Failed (TaskFailed)
Status information included in GetTaskResponse marked as:

**Figure 24 – Status information returned in the GetTaskResponse for various exemplary tasks/tasking requests when the state logger conformance class is supported by the service**

### 7.3.7.5 Exceptions

| Requirement | |
|---|---|
| http://www.opengis.net/spec/SPS/2.0/req/GetTaskResponse/exceptions | |
| REQ 65. | When an SPS server encounters an error while performing a *GetTask* operation, it shall return an exception message as specified in clause 7.2. In addition, if the minimum storage time of status information for finalized tasks has passed and the service already removed that information, it shall throw a *StatusInformationExpired* exception. |

#### 7.3.7.6 Examples

Clause 9.6 provides example XML instances for the GetTask operation request and response.

### 7.3.8 DescribeResultAccess Operation

#### 7.3.8.1 Introduction

The *DescribeResultAccess* operation allows SPS clients to retrieve information on how to access data that was produced by a specific task, or how to retrieve data for a given sensor that is tasked by this SPS in general. The response can point to:

- a SOS, WMS, WFS
- any other OGC Web Service that provides data
- any data file or folder on an ftp server
- any data file or file container that is accessible over the Internet

Clients provide the identifier of either a sensor or task to identify the information they are interested in. Table 40 defines the semantics that of both variations.

**Table 40 — Semantics of DescribeResultAccess operation request using task or procedure identifier**

| DescribeResultAccess request | DescribeResultAccess response | Applicable Reference Usage Options (see Table 41) |
|---|---|---|
| including a procedure identifier | Reference(s) that points to the service(s) providing data for that procedure. The response contains the base URL to the service. It is then the client's task to explore all available data.<br>Primarily useful to learn in advance at which service types/instances or via which protocols data is going to be made available. | 3, 5 |
| including task identifier | Reference(s) to the concrete data of the specified task (concrete file/folder on a server, full [OGC] service request that delivers all data etc.) | 1, 2, 3, 4, 5<br>option 3 should be avoided if possible [1] |
| Notes:<br>1) When providing information about a task, option 3 is the easiest solution for SPS providers but the hardest for clients. However, in some domains the link to the service might already suffice as additional information and constraints enable clients to create the request for retrieving their data themselves. Security issues may also require this option to be used. | | |

The result contains one or more reference group elements, which are defined by [OGC 06-121r3], to describe where data is or will be stored.

#### 7.3.8.1.1 Reference group usage

| Requirement | |
|---|---|
| http://www.opengis.net/spec/SPS/2.0/req/DescribeResultAccess/referenceGroup/procedure | |
| REQ 66. | If the *DescribeResultAccess* request contains a *procedure* identifier (see Table 40), then the response shall contain one or many *ReferenceGroup(s)* with references to the possible data storage locations/services for that *procedure*.<br>In most cases a single group will be used, but there are situations that require the usage of multiple groups, as described further below. |

| Requirement | |
|---|---|
| http://www.opengis.net/spec/SPS/2.0/req/DescribeResultAccess/referenceGroup/task | |
| REQ 67. | If the *DescribeResultAccess* contains a task identifier then each *ReferenceGroup* shall describe the complete data set that was gathered for the requested task. |

| Requirement | |
|---|---|
| http://www.opengis.net/spec/SPS/2.0/req/DescribeResultAccess/referenceGroup/incremental | |
| REQ 68. | If an SPS server publishes data gathered for a task incrementally, then new references shall be added to the according reference group(s). Clients can differentiate new references from those they already received by the identifier provided with each reference. |

The reference group(s) contains all references required to access the data gathered for a specific task once the task is complete. Before, the set of references in a group can be incomplete, as new references are added whenever new data was published and thus made accessible to clients.

It may happen that all data for a given task will be published towards the end of the lifetime of a task. In this case, a *DescribeResultAccess* would yield no results before the status of the task is set to *Final*. In some cases published data might also be temporarily unavailable due to failures of the data services. Both situations are recognized by this standard and can be communicated via the *DescribeResultAccessResponse* to clients.

The main purpose of allowing more than one reference group in a *DescribeResultAccessResponse* is to support the provision of data

- in various forms of data representations – data can be published as raw binary data, O&M encoded observations, NetCDF files, image files, video streams etc.
- in various processing stages – data can be published as received by the sensor, after level-1 quality checks were performed, level-2 quality checks etc.
- in various storage stages – in some domains data is made accessible in some kind of immediately accessible cache but soon after a task/mission was completed the data is moved from there to a long-term data archive
- via different protocols – access to data may be performed via FTP, HTTP etc.

As an example, an SPS server may use some transient storage to make intermediate results available to the client in various representations using diverse access protocols. Once the next processing step is reached, this data becomes obsolete and some references in the reference groups might be removed, updated, or simply superseded by new reference groups that now point to the next level data. The SPS server would continuously update the reference groups, and clients can differentiate them from those they already know via the identifier assigned to each group by the service.

| Requirement | |
|---|---|
| http://www.opengis.net/spec/SPS/2.0/req/DescribeResultAccess/referenceGroup/Aggregation | |
| REQ 69. | Any SPS shall aggregate references in reference groups according to Table 41. |

How references are used in reference groups is defined in the following clause.

#### 7.3.8.1.2 Reference usage

SPS does not provide a direct data access operation. Instead, it provides references to the data or services hosting the data. The various options in describing the data access are outlined in Table 41. Further on, it describes the concrete semantics and usage of the *ReferenceGroup* properties in detail.

**Table 41 – Service Reference Mapping**

| Option | SPS provides | Used elements of DescribeResultAccessResponse | Cardinality |
|---|---|---|---|
| 1 | a URL that contains the full request string to be sent against the data service<br>information encoded as OWS Common Reference | reference (URI): Full request as sent to data service using HTTP GET | 1 |
| | | role (URI): describes the role of this reference, in other words what the SPS provides with this reference<br>value shall be identifier for this reference option:<br>http://www.opengis.net/spec/SPS/2.0/referenceType/FullURLAccess | 1 |
| | | title (String): Human readable title for this reference | 0..1 |
| | | identifier (URI): unique identifier for the reference | 1 |
| | | abstract (String with optional language code): Brief narrative description of this reference (for example what it references), normally available for display to a human | 0..* |
| | | format (mime type): defines the response format as provided by data service<br>(Table 42 provides guidance) | 1 |
| | | metadata (AbstractMetadata, at least one SPSMetadata shall be included): provides specific metadata for (service) references given in a DescribeResultAccessResponse – in case of SPSMetadata it identifies the specification that defines the type of service/method used for accessing data (see Table 50). | 1..* |

| Option | SPS provides | Used elements of DescribeResultAccessResponse | Cardinality |
|---|---|---|---|
| 2 | XML encoded query to be sent against service using HTTP POST<br>information encoded as OWS Common ServiceReference | reference (URI): Service URL | 1 |
| | | role (URI): describes the role of this reference, in other words what the SPS provides with this reference<br>value shall be identifier for this reference option:<br>http://www.opengis.net/spec/SPS/2.0/referenceType/FullServiceAccess | 1 |
| | | title (String): Human readable title for this reference | 0..1 |
| | | identifier (URI): unique identifier for the reference | 1 |
| | | abstract (String with optional language code): Brief narrative description of this reference (for example what it references), normally available for display to a human | 0..* |
| | | format (mime type): defines the response format as provided by data service<br>(Table 42 provides guidance) | 1 |
| | | requestMessage (String): The XML-encoded operation request message to be sent to the URI provided in xlink:href<br>OR<br>requestMessageReference (URI): Reference to the XML-encoded operation request message to be sent to the URI provided in xlink:href | 1 |
| | | metadata (AbstractMetadata, at least one SPSMetadata shall be included): provides specific metadata for (service) references given in a DescribeResultAccessResponse – in case of SPSMetadata it identifies the specification that defines the type of service/method used for accessing data (see Table 50). | 1..* |

| Option | SPS provides | Used elements of DescribeResultAccessResponse | Cardinality |
|---|---|---|---|
| 3 | Link to service. Client needs to explore the service itself<br>information encoded as OWS Common Reference | reference (URI): service URL | 1 |
| | | role (URI): describes the role of this reference, in other words what the SPS provides with this reference<br>value shall be identifier for this reference option:<br>http://www.opengis.net/spec/SPS/2.0/referenceType/ServiceURL | 1 |
| | | title (String): Human readable title for this reference | 0..1 |
| | | identifier (URI): unique identifier for the reference | 1 |
| | | abstract (String with optional language code): Brief narrative description of this reference (for example what it references), normally available for display to a human | 0..1 |
| | | format (mime type): defines the response format as provided by data service if it can be identified<br>(Table 42 provides guidance) | 0..1 |
| | | metadata (AbstractMetadata, at least one SPSMetadata shall be included): provides specific metadata for (service) references given in a DescribeResultAccessResponse – in case of SPSMetadata it identifies the specification that defines the type of service/method used for accessing data (see Table 50). | 1..* |
| 4 | Resource on a server (e.g. file or dynamically created resource like video stream)<br>information encoded as OWS Common Reference | reference (URI): Link to the file on a server, transport protocol is implied by URI | 1 |
| | | role (URI): describes the role of this reference, in other words what the SPS provides with this reference<br>value shall be identifier for this reference option:<br>http://www.opengis.net/spec/SPS/2.0/referenceType/Resource | 1 |
| | | title (String): Human readable title for this reference | 0..1 |
| | | identifier (URI): unique identifier for the reference | 1 |
| | | abstract (String with optional language code): Brief narrative description of this reference (for example what it references), normally available for display to a human | 0..* |
| | | format (mime type): defines the mimeType of the resource<br>(Table 42 provides guidance) | 1 |

| Option | SPS provides | Used elements of DescribeResultAccessResponse | Cardinality |
|---|---|---|---|
| 5 | Folder on a server<br>information encoded in:<br>ows:Reference | reference (URI): Link to the folder on a server | 1 |
| | | role (URI): describes the role of this reference, in other words what the SPS provides with this reference<br>value shall be identifier for this reference option:<br>http://www.opengis.net/spec/SPS/2.0/referenceType/Folder | 1 |
| | | title (String): Human readable title for this reference | 0..1 |
| | | identifier (URI): unique identifier for the reference | 1 |
| | | abstract (String with optional language code): Brief narrative description of this reference (for example what it references), normally available for display to a human | 0..* |
| | | format (ows:MimeType): defines the mimeType of the files in that folder (in case they are homogeneous) | 0..1<br>provide if files in folder (will) have homogeneous format/mime type |

Options 1 and 2 provide full service access, either via a GET request or via an XML encoded POST request to be sent to the service in order to retrieve task data. However, a client may still need to modify a given request before it can actually execute it, for example if security conditions apply that require request enrichment with security elements such as tokens, signatures, etc.

Operation requests wrapped in a SOAP envelope are another example. If for example WS-Addressing is used, then the SPS may provide a value for the *wsa:Action* element. However, for the *wsa:ReplyTo* element it can only state the anonymous endpoint (and even that may not be permitted by the referenced service so the SPS would have to omit the *wsa:ReplyTo* element in the given request) while it cannot provide a meaningful value for the *wsa:MessageID* element. See example provided in clause 9.6.6for further information.

**Table 42 – Examples of applicable mime types when referencing data**

| Reference resolves to | applicable mime type [1] |
|---|---|
| SOS GetObservation response | application/xml |
| O&M Observation 2.0 XML instance | application/gml+xml |
| SensorML 1.0.1 XML instance | application/xml |
| PNG image | image/png |
| JPEG 2000 image | image/jp2 |
| MP-4 video file | video/mp4 |
| Motion JPEG 2000 video file | video/mj2 |

Notes:

[1] The provision of a mime type does not mean that this is the only one applicable. For example in case of a SOS GetObservation response the according request could ask for a different response format other than the default one. The returned XML could also be binarized, resulting in another mime type (e.g. application/exi).

Figure 25 illustrates the mapping of OWS Common (Service) Reference properties to the elements and attributes of the according XML type.

OwsCommon *References* are used as follows:

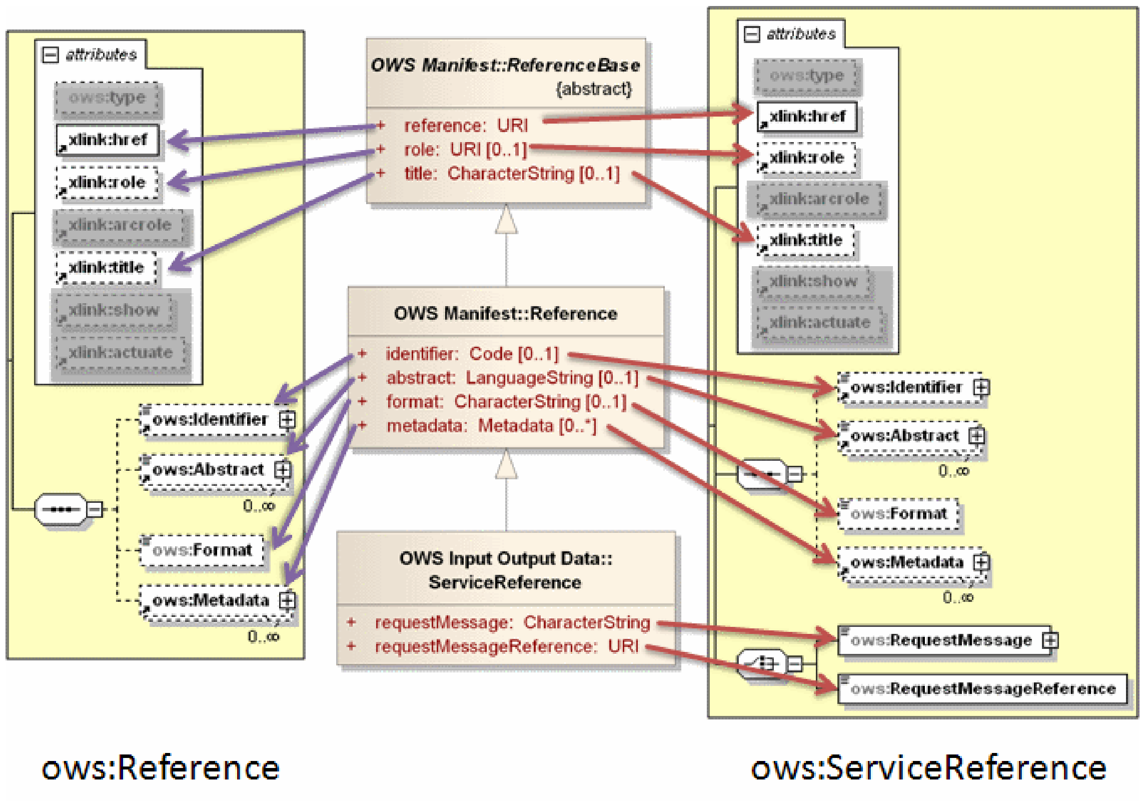


**Figure 25 — Mapping of UML Reference elements to XML Schema elements. Rarely used elements are grayed out**

Note: Data production as a result of tasking a sensor and subsequent data access are two decoupled processes. Theoretically, the SPS might use a number of data storage services to store the data (e.g. for performance reasons). This leads to the situation that the SPS returns a list of data references for a given sensor identifier. Then, it is up to the client to explore those services to discover the relevant data.

Note: in XML Schema, an *ows:Reference* element allows a number of metadata information to be provided to clients. This can be used to provide further information about or for accessing the referenced data (for example metadata about the processing applied to the referenced data). At the moment no model for such specific metadata is defined. However, according models could be defined in other documents – like it is done in clause 7.3.8.10 – and used by an SPS. Clients that do not understand such metadata can simply ignore it.

If the service supports publish/subscribe functionality, it sends notifications to the client to indicate that new data is available. It is up to the SPS instance to decide when it publishes data. However, sending a *task completed* notification to clients implies that all data generated for the task has been published and that there will be no more *data published* notification.

In case that no data is available for a given task, the service returns a *DataNotAvailable* data type giving the reason why the data is not available.

| Requirement | |
|---|---|
| http://www.opengis.net/spec/SPS/2.0/req/DescribeResultAccess/duration | |
| REQ 70. | An SPS shall provide result access information for a task at least as long as it provides status information for that task (clause 7.3.3.3 defines how long that information has to be stored at minimum). |

A service should provide this information longer. How much longer is not defined by this standard. Specific application domains and/or profiles and extensions can define this in more detail.

#### 7.3.8.2 Data Types

The conceptual model of the DescribeResultAccess operation is shown in the following UML diagram.

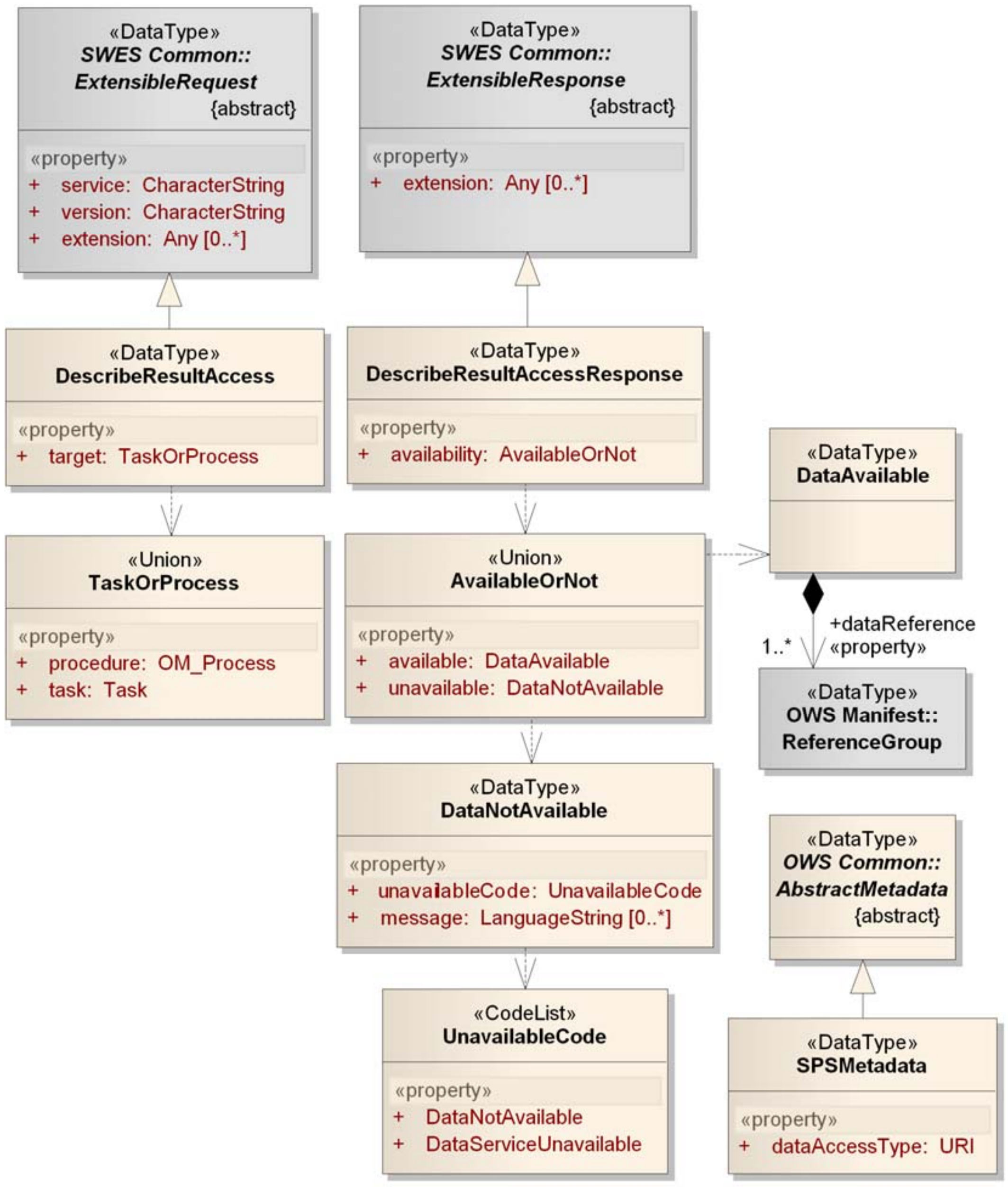


**Figure 26 — Data types of the DescribeResultAccess operation**

The details of the operation request and response are explained in the following subclauses.

### 7.3.8.3 Operation Request – DescribeResultAccess

Sending an instance of the *DescribeResultAccess* data type to the service performs an SPS *DescribeResultAccess* operation request.

The *DescribeResultAccess* data type is derived from the SWES *ExtensibleRequest* data type specified in clause 9 of [OGC 09-001] and therefore inherits all the properties contained in that data type. DescribeResultAccess does not restrict the content model of *ExtensibleRequest*.

| Requirement | |
|---|---|
| http://www.opengis.net/spec/SPS/2.0/req/DescribeResultAccessRequest/dataType | |
| REQ 71. | The *DescribeResultAccess* data type shall contain the properties defined for SWES *ExtensibleRequest*. In addition, it shall contain the property according to Table 43. |

**Table 43 — Property in the *DescribeResultAccess* data type**

| Name | Definition | Data type and values | Multiplicity and use |
|---|---|---|---|
| target | Pointer to either a task or procedure. | TaskOrProcess, see clause 7.3.8.4 further below | One (mandatory) |

### 7.3.8.4 TaskOrProcess

In a *DescribeResultAccess* request, the ID of either a task or procedure is used to define the semantics of the request.

| Requirement | |
|---|---|
| http://www.opengis.net/spec/SPS/2.0/req/TaskOrProcess/dataType | |
| REQ 72. | The *TaskOrProcess* union shall contain one of the properties/choices according to Table 44. |

**Table 44 — Properties in the *TaskOrProcess* union**

| Name | Definition | Data type and values | Multiplicity and use |
|---|---|---|---|
| task | Pointer to a task. | Task [id], see clause 7.3.1.6<br>value shall point to a task that is or was executed by the service | One (mandatory)<br>Because TaskOrProcess is a union, either a task or procedure shall be used (i.e. there is a choice between the properties) |
| procedure | Pointer to a procedure tasked by the service. | OM_Process [id], see ISO DIS 19156<br>value shall point to one of the procedures listed in the service's contents section | |
| id) Note: the primary use of this property is to provide a pointer/identifier – see OGC 09-001 clause 16.3.1 for further details. | | | |

### 7.3.8.5 Operation Response - DescribeResultAccessResponse

The *DescribeResultAccessResponse* data type represents the response to an SPS *DescribeResultAccess* operation request.

The *DescribeResultAccessResponse* data type is derived from the SWES *ExtensibleResponse* data type specified in clause 9 of [OGC 09-001] and therefore inherits all the properties contained in that data type. *DescribeResultAccessResponse* does not restrict the content model of *ExtensibleResponse*.

| Requirement | |
|---|---|
| http://www.opengis.net/spec/SPS/2.0/req/DescribeResultAccessResponse/dataType | |
| REQ 73. | The *DescribeResultAccessResponse* data type shall contain the properties defined for SWES *ExtensibleResponse*. In addition, it shall contain the properties according to Table 45. |

**Table 45 — Property in the DescribeResultAccessResponse data type**

| Name | Definition | Data type and values | Multiplicity and use |
|---|---|---|---|
| availability | indicates that data is available or not | AvailableOrNot, see clause 7.3.8.6 | One (mandatory) |

### 7.3.8.6 AvailableOrNot

A *DescribeResultAccessResponse* either indicates that data gathered in a task is available at given service references or that it is not available for some reason.

| Requirement | |
|---|---|
| http://www.opengis.net/spec/SPS/2.0/req/AvailableOrNot/dataType | |
| REQ 74. | The *AvailableOrNot* union shall contain the properties/choices according to Table 46. |

**Table 46 — Properties in the AvailableOrNot union**

| Name | Definition | Data type and values | Multiplicity and use |
|---|---|---|---|
| available | indicates that task data is available and contains (service) reference(s) to retrieve the data | DataAvailable, see clause 7.3.8.7 | One (mandatory)<br>Because AvailableOrNot is a union, either available or unavailable shall be used (i.e. there is a choice between the properties) |
| unavailable | indicates that data is not available and explains why it is not available | DataNotAvailable, see clause 7.3.8.8 | |

### 7.3.8.7 DataAvailable

This data type contains a list of one or more groups of service references. They point to services that generally store data from the requested procedure. They can also point to the data directly. Distributing data across several service instances can be performed inside a reference group. The response can contain more than one *ReferenceGroup*, if the application design requires it (see clause 7.3.8.1).

| Requirement | |
|---|---|
| http://www.opengis.net/spec/SPS/2.0/req/DataAvailable/dataType | |
| REQ 75. | The *DataAvailable* data type shall contain the property according to Table 47. |

**Table 47 — Property in the DataAvailable data type**

| Name | Definition | Data type and values | Multiplicity and use |
|---|---|---|---|
| dataReference | group of (service) references with which the complete set of data gathered for a task can be retrieved | ReferenceGroup, see table 46 in [06-121r3] | One or more (mandatory)<br>use of ReferenceGroup and contained (Service)References as defined in clause 7.3.8.1 |

| Requirement | |
|---|---|
| http://www.opengis.net/spec/SPS/2.0/req/DescribeResultAccessResponse/identification | |
| REQ 76. | The service shall assign a unique identifier for each ReferenceGroup and the references contained in that group. Those identifiers shall not be changed while the according object (reference group or reference) exists. |

Note: thus for example a change to the set of references contained in a reference group does not change the identifier of this group.

#### 7.3.8.8 DataNotAvailable

This data type expresses that no data is available.

| Requirement | |
|---|---|
| http://www.opengis.net/spec/SPS/2.0/req/DataNotAvailable/dataType | |
| REQ 77. | The *DataNotAvailable* data type shall contain the properties according to Table 48. |

**Table 48 — Properties in the DataNotAvailable data type**

| Name | Definition | Data type and values | Multiplicity and use |
|---|---|---|---|
| message | human readable message that provides further information or reason why no data is available | LanguageString, see clause 10.7 in [OGC 06-121r3] | Zero or more (optional)<br>Include one for each language represented |
| unavailableCode | identifies the reason why data is unavailable | UnavailableCode, see clause 7.3.8.9 | One (mandatory) |

#### 7.3.8.9 UnavailableCode

This type is a list of codes signifying the reason why result access information for a given task or sensor is not available.

| Requirement | |
|---|---|
| http://www.opengis.net/spec/SPS/2.0/req/DescribeResultAccessResponse/UnavailableCode | |
| REQ 78. | The *UnavailableCode* code list shall contain the properties/choices according to Table 49. |

**Table 49 — Properties in the UnavailableCode code list**

| Name [a] | Definition | Value |
|---|---|---|
| DataNotAvailable | Result access information is not available because no data has been published yet. | “DataNotAvailable” |
| DataServiceUnavailable | Result access information is not available because one or more of the services that are assigned to store the data gathered in a task is currently unavailable. | “DataServiceUnavailable” |
| a Although some values listed in the column appear to contain spaces, they shall not contain spaces. | | |

The code list is extensible. The pattern for new codes is the regular expression:

```
other: [A-Za-z0-9_]{2,}
```

However, for interoperability reasons a service should not use an arbitrary code that is not defined by an official SPS extension.

#### 7.3.8.10 SPSMetadata

This data type provides SPS specific metadata for (service) references given in a DescribeResultAccessResponse.

The *SPSMetadata* data type is derived from the OWS Common *AbstractMetadata* data type [OGC 06-121r3]. That data type does not define any property and thus SPSMetadata does not inherit any property from it. However, it is a child of AbstractMetadata and thus is a valid substitute whenever such metadata is provided, for example in references (see clause 7.3.8.1.2).

| Requirement | |
|---|---|
| http://www.opengis.net/spec/SPS/2.0/req/DescribeResultAccessResponse/SPSMetadata | |
| REQ 79. | The *SPSMetadata* data type shall contain the properties according to Table 50. |

**Table 50 — Properties in the SPSMetadata data type**

| Name | Definition | Data type and values | Multiplicity and use |
|---|---|---|---|
| dataAccessType | Identifies the specification that defines the type of service/method used for accessing data. | URI | one (mandatory)<br>Use the following URIs with given priority:<br>1) URI that uniquely identifies the operation with which data is accessed [1] – example: *http://www.opengis.net/sos/2.0/GetObservation* ,<br>2) target namespace of the XML Schema definition for the operation/service via which data is accessed [2]– example: *http://www.opengis.net/wcs/1.1* or *http://nonogc.org/operation/x*,<br>3) the OGC name of the specification where the operation is defined in [2]– example: *http://www.opengis.net/doc/IS/WMS/1.3*,<br>4) *http://www.opengis.net/def/nil/OGC/0/unknown* if no specific URI is known that uniquely identifies the service/method |
| Notes:<br>1) Whenever the SOAP binding of a service specification defines action URIs for its operations, the action URI for the operation request is a suitable value to use. This URI can also be used to identify the operation even if its KVP binding is actually used in a reference.<br>2) References that provide the service URL only but not a full GET or POST request with which the data can be retrieved may not be able to state via which specific operation data is retrieved. In that case the provision of the target namespace assigned to the XML Schema of the service suffices – if it uniquely identifies one specific version of that specification. Otherwise another identifier for the specification (version) – for OGC specifications this could be the OGC name of the document – should be used. | | | |

The SPSMetadata data type can be subclassed, for example by extensions to this specification, to provide additional metadata if required. Such extensions can also define their own data type that derives from OWS Common AbstractMetadata and properly specify when it needs to be included in (service) references.

### 7.3.8.11 Exceptions

| Requirement | |
|---|---|
| http://www.opengis.net/spec/SPS/2.0/req/DescribeResultAccesResponse/exceptions | |
| REQ 80. | When an SPS server encounters an error while performing a *DescribeResultAccess* operation, it shall return an exception message as specified in clause 7.2. |

### 7.3.8.12 Examples

Clause 9.6 provides example XML instances for the DescribeResultAccess operation request and response.

### 7.3.9 Reserve Operation

#### 7.3.9.1 Introduction

The *Reserve* operation allows SPS clients to reserve a task. The client encodes the tasking parameters according to the parameter description do a *DescribeTasking* response. Thus, reserving a task is practically similar to submitting a task, except that the task is not performed until the client sends a *Confirm* request.

The *Reserve* operation is part of the *ReservationManager* interface (see clause 7.1). A reservation can be cancelled sending a *Cancel* request at any time.

Clients can set expiration time for reserved tasks. If the service does not accept the reservation time, it rejects the request and provides an appropriate message explaining the reason. Otherwise the service reserves the task until the reservation expires or the client confirmed or cancelled the reservation. SPS servers provide reservation time in case the client provides no timing in the request. An expired reservation cannot be revitalized.

#### 7.3.9.2 Data Types

The conceptual model of the Reserve operation is shown in the following UML diagram.

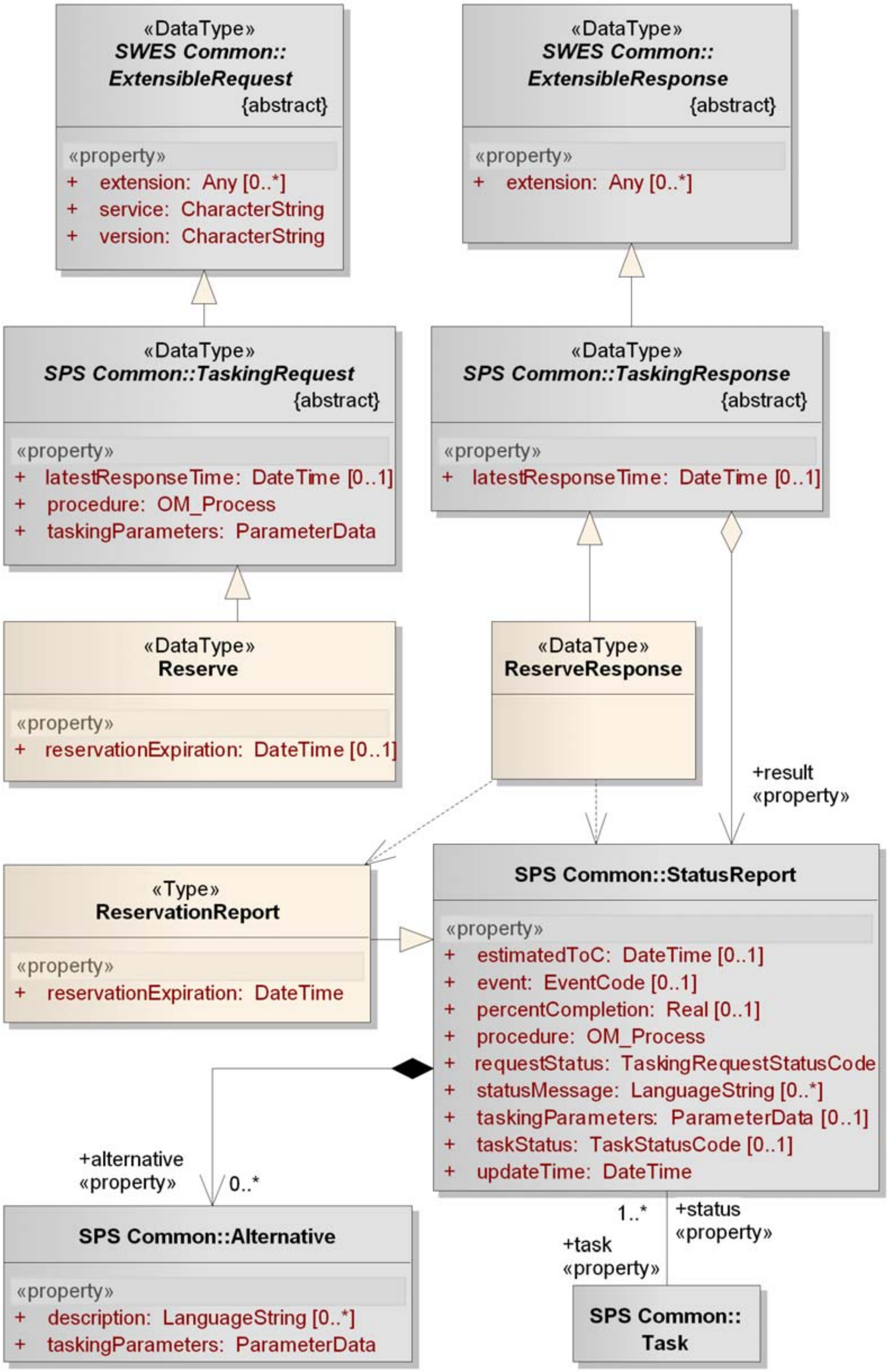

«DataType»
SWES Common::
ExtensibleRequest
{abstract}
«property»
+ extension: Any [0..*]
+ service: CharacterString
+ version: CharacterString
«DataType»
SWES Common::
ExtensibleResponse
{abstract}
«property»
+ extension: Any [0..*]
«DataType»
SPS Common::TaskingRequest
{abstract}
«property»
+ latestResponseTime: DateTime [0..1]
+ procedure: OM_Process
+ taskingParameters: ParameterData
«DataType»
SPS Common::TaskingResponse
{abstract}
«property»
+ latestResponseTime: DateTime [0..1]
«DataType»
Reserve
«property»
+ reservationExpiration: DateTime [0..1]
«DataType»
ReserveResponse
+result
«property»
«Type»
ReservationReport
«property»
+ reservationExpiration: DateTime
SPS Common::StatusReport
«property»
+ estimatedToC: DateTime [0..1]
+ event: EventCode [0..1]
+ percentCompletion: Real [0..1]
+ procedure: OM_Process
+ requestStatus: TaskingRequestStatusCode
+ statusMessage: LanguageString [0..*]
+ taskingParameters: ParameterData [0..1]
+ taskStatus: TaskStatusCode [0..1]
+ updateTime: DateTime
+alternative
«property»
0..*
1..*
+status
«property»
+task
«property»
SPS Common::Alternative
«property»
+ description: LanguageString [0..*]
+ taskingParameters: ParameterData
SPS Common::
Task

**Figure 27 — Data types of the Reserve operation**

The details of the operation request and response are explained in the following subclauses.

#### 7.3.9.3 Operation Request - Reserve

Sending an instance of the *Reserve* data type to the service performs an SPS *Reserve* operation request.

The *Reserve* data type is derived from the abstract *TaskingRequest* data type (see clause 7.3.1.3) and therefore inherits all the properties contained in that data type. Reserve does not restrict the content model of *TaskingRequest*.

| Requirement | |
|---|---|
| http://www.opengis.net/spec/SPS/2.0/req/ReserveRequest/dataType | |
| REQ 81. | The *Reserve* data type shall contain the properties defined for *TaskingRequest*. In addition, it shall contain the properties according to Table 51. |

**Table 51 — Property in the Reserve data type**

| Name | Definition | Data type and values | Multiplicity and use |
|---|---|---|---|
| reservation Expiration | point in time when the reservation shall expire | DateTime<br>(see ISO 19103 and OGC 07-036 Table D.2)<br>value shall be a point in time in the future | Zero or one (optional) |

| Requirement | |
|---|---|
| http://www.opengis.net/spec/SPS/2.0/req/ReserveRequest/timing | |
| REQ 82. | The *Reserve* data type is a *TaskingRequest*. If the client defines a *reservationExpiration* time and this time has already passed when the SPS receives the request, then the SPS shall reject the request. |

#### 7.3.9.4 Operation Response - ReservationReport

The *ReserveResponse* data type represents the response to an SPS *Reserve* operation request. It is derived from the *TaskingResponse* data type (see clause 7.3.1.4) and therefore inherits all the properties contained in that data type. *ReserveResponse* neither restricts the content model of *TaskingResponse* nor adds additional properties.

| Requirement | |
|---|---|
| http://www.opengis.net/spec/SPS/2.0/req/ReserveResponse/dataType | |
| REQ 83. | The *ReserveResponse* data type shall contain the properties defined for *TaskingResponse*. |

A *ReserveResponse* contains either a *ReservationReport* or a *StatusReport* to indicate the result of the requested operation. The concrete report type depends on the status of the request. The only difference between both report types is the additional expirationTime property of the *ReservationReport*.

| Requirement | |
|---|---|
| http://www.opengis.net/spec/SPS/2.0/req/ReserveResponse/ReportType | |
| REQ 84. | A *ReservationReport* shall be returned if the request gets accepted. In all other cases, a *StatusReport* shall be returned. |

As a *Reserve* request is a tasking request, the final result of that request might not be directly available and would then be pending.

| Requirement | |
|---|---|
| http://www.opengis.net/spec/SPS/2.0/req/ReserveResponse/ReportProperties | |
| REQ 85. | The properties of the *ReservationReport* or *StatusReport* shall be used as defined in the Table 52. |

The transitions starting from the *Pending* state are not applicable for reporting in the *ReserveResponse,* as the *ReserveResponse* provides only information about the first state transition, i.e. from initial to accepted, pending, or rejected. Clients retrieve further state transitions via notifications or *GetStatus* operation calls.

**Table 52 – StatusReport usage for different state transitions of a Reserve request**

| **property name/cardinality** | **Reserve Request State Transitions (From → To)** | | | | | |
|---|---|---|---|---|---|---|
| | **Initial → Pending** | **Initial → Accepted** | **Initial → Rejected** | **Pending → Accepted** | **Pending → Rejected** | **Pending → Rejected (request expired)** |
| task/1 | new identifier provided by service | | | identifier previously provided by service | | |
| estimatedToC/0..1 | NA | optional | NA | optional | NA | NA |
| event (code)/0..1 | NA | TaskReserved | NA | TaskReserved | NA | TaskingRequestExpired |
| percentCompletion/0..1 | NA [1] | | | | | |
| procedure/1 | identifier of procedure for which Reserve request was made | | | | | |
| requestStatus (code)/1 | Pending | Accepted | Rejected | Accepted | Rejected | Rejected |
| statusMessage/0..* | service may provide additional information to client in human readable form | | | | | |
| taskingParameters/0..1 | NA | | | | | |
| taskStatus (code)/0..1 | NA | Reserved | NA | Reserved | NA | NA |
| updateTime/1 | point in time when transition was made | | | | | |
| alternative/0..* | may be provided by service | | | | | |
| StatusReport encoded as Reservation Report | no | yes | no | yes | no | no |
| Applicable in Reserve Response | yes | yes | yes | no | no | no |
| NA = not applicable<br>Notes<br>1 only applicable to tasks that are being or have been executed | | | | | | |

### 7.3.9.5 ReservationReport

The *ReservationReport* type is derived from the *StatusReport* type (see clause 7.3.1.5) and therefore inherits all the properties contained in that type. *ReservationReport* does not restrict the content model of *StatusReport*.

| Requirement | |
|---|---|
| http://www.opengis.net/spec/SPS/2.0/req/ReserveResponse/ReservationReport/dataType | |
| REQ 86. | The *ReservationReport* type shall contain the properties defined for *StatusReport*. In addition, it shall contain the property according to Table 53. |

**Table 53 — Property in the ReservationReport type**

| Name | Definition | Data type and values | Multiplicity and use |
|---|---|---|---|
| reservation Expiration | point in time when the (task) reservation will expire | DateTime<br>(see ISO 19103 and OGC 07-036 Table D.2)<br>value shall be a point in time in the future | One (mandatory) |

#### 7.3.9.6 Exceptions

| Requirement | |
|---|---|
| http://www.opengis.net/spec/SPS/2.0/req/ReserveResponse/exceptions | |
| REQ 87. | When an SPS server encounters an error while performing a *Reserve* operation, it shall return an exception message as specified in clause 7.2. |

#### 7.3.9.7 Examples

Clause 9.6 provides example XML instances for the Reserve operation request and response.

### 7.3.10 Confirm Operation

#### 7.3.10.1 Introduction

The *Confirm* operation allows SPS clients to confirm a reserved task. If accepted, the task transits from state *Reserved* to *InExecution* (see clause 10).

#### 7.3.10.2 Data Types

The conceptual model of the *Confirm* operation is shown in the following UML diagram.

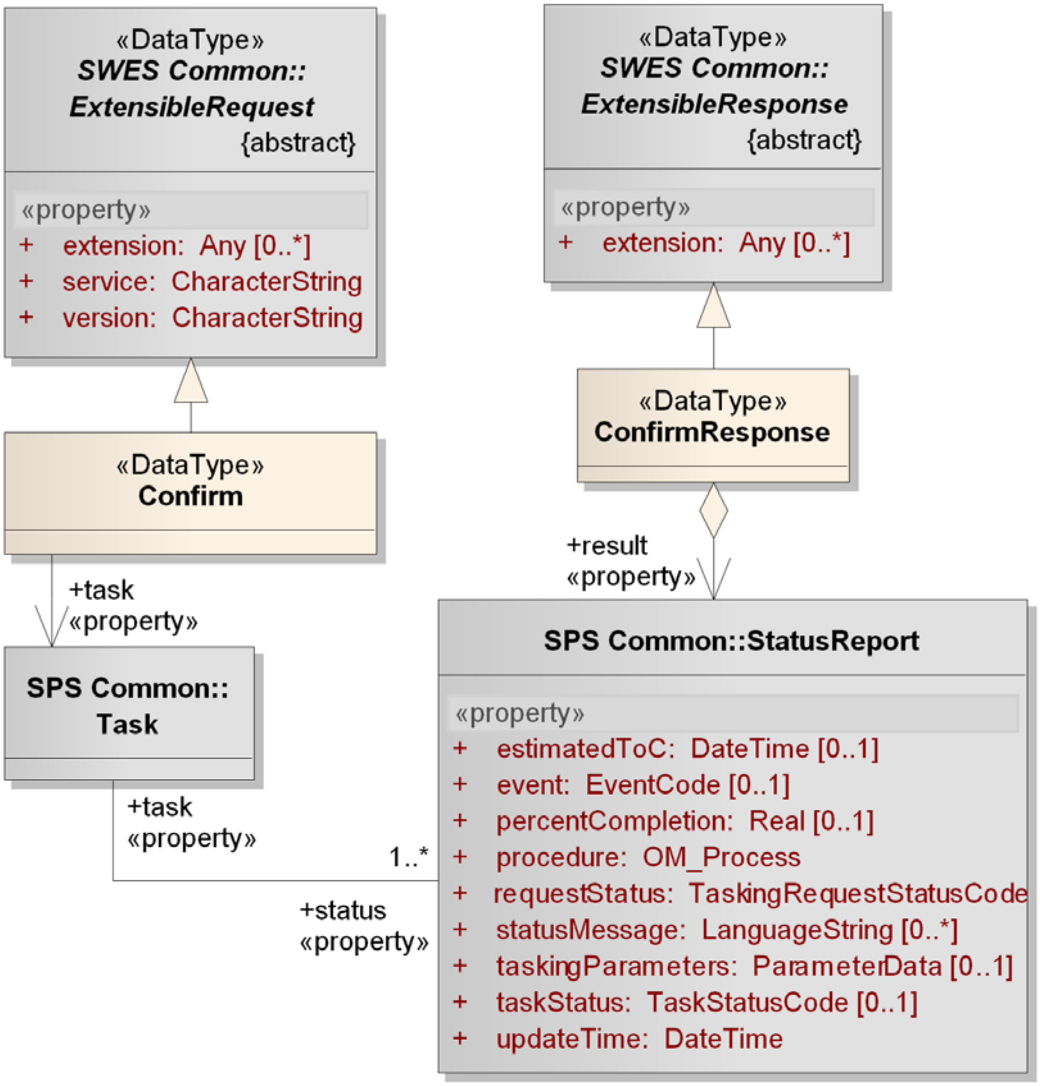


**Figure 28 — Data types of the Confirm operation**

The details of the operation request and response are explained in the following subclauses.

### 7.3.10.3 Operation Request - Confirm

Sending an instance of the *Confirm* data type to the service performs an SPS *Confirm* operation request.

The *Confirm* data type is derived from the abstract SWES *ExtensibleRequest* data type specified in clause 9 of [OGC 09-001] and therefore inherits all the properties contained in that data type. *Confirm* does not restrict the content model of *ExtensibleRequest*.

| Requirement | |
|---|---|
| http://www.opengis.net/spec/SPS/2.0/req/ConfirmRequest/dataType | |
| REQ 88. | The *Confirm* data type shall contain the properties defined for SWES *ExtensibleRequest*. In addition, it shall contain the property according to Table 54. |

**Table 54 — Property in the Confirm data type**

| Name | Definition | Data type and values | Multiplicity and use |
|---|---|---|---|
| task | Pointer to the reserved task that is requested to be confirmed. | Task [id], see clause 7.3.1.6 | One (mandatory) |
| id) Note: the primary use of this property is to provide a pointer/identifier – see OGC 09-001 clause 16.3.1 for further details. | | | |

### 7.3.10.4 Operation Response - ConfirmResponse

The *ConfirmResponse* data type represents the response to an SPS *Confirm* operation request.

The *ConfirmResponse* data type is derived from the SWES *ExtensibleResponse* data type specified in clause 9 of [OGC 09-001] and therefore inherits all the properties contained in that data type. *ConfirmResponse* does not restrict the content model of *ExtensibleResponse*.

| Requirement | |
|---|---|
| http://www.opengis.net/spec/SPS/2.0/req/ConfirmResponse/dataType | |
| REQ 89. | The *ConfirmResponse* data type shall contain the properties defined for SWES *ExtensibleResponse*. In addition, it shall contain the property according to Table 55. |

**Table 55 — Property in the ConfirmResponse data type**

| Name | Definition | Data type and values | Multiplicity and use |
|---|---|---|---|
| result | report with the outcome of the confirmation request | StatusReport, see 7.3.1.5<br>Properties of StatusReport shall be used as defined in Table 56 | One (mandatory) |

Table 56 illustrate the usage of the *StatusReport* properties in *Confirm* responses.

**Table 56 – StatusReport property usage in Confirm operation response**

| property name/cardinality | Operation outcome | |
|---|---|---|
| | Confirmation was accepted | Confirmation was rejected [7] |
| task/1 | task identifier used in request | |
| estimatedToC/0..1 | NA [1] | |
| event (code)/0..1 | NA [2] | |
| percentCompletion/0..1 | NA [3] | |
| procedure/1 | identifier of procedure associated with reserved task | |
| requestStatus (code)/1 | Accepted | Rejected |
| statusMessage/0..* | usage optional | service should indicate why the confirmation was rejected |
| taskingParameters/0..1 | NA [4] | |
| taskStatus (code)/0..1 | NA [5] | |
| updateTime/1 | point in time when confirmation was accepted | point in time when confirmation was rejected |
| alternative/0..* | NA [6] | |

NA = not applicable

NOTES:
1 only applicable to scheduled tasks that have not been finalized yet
2,4,5 only applicable to tasking requests and tasks
3 only applicable to tasks that are being or have been executed
6 only applicable to tasking requests
7 in this case the reserved task fails

#### 7.3.10.5 Exceptions

| Requirement | |
|---|---|
| http://www.opengis.net/spec/SPS/2.0/req/ConfirmResponse/exceptions | |
| REQ 90. | When an SPS server encounters an error while performing a *Confirm* operation, it shall return an exception message as specified in clause 7.2. |

#### 7.3.10.6 Examples

Clause 9.6 provides example XML instances for the Confirm operation request and response.

### 7.3.11 GetFeasibility Operation

#### 7.3.11.1 Introduction

The *GetFeasibility* operation allows SPS clients to obtain information about the feasibility of a tasking request. See section 6.3.4 for further details on *GetFeasibility* checks. The client encodes the tasking parameters according to the parameter description given in the *DescribeTasking* response.

An SPS may be capable of computing alternatives for requested parameter settings in a tasking request. These alternatives may slightly modify the tasking parameters contained in the request (e.g. to change the time frame of an intended task by a few minutes) or suggest completely new sets of tasking parameters that lead to similar results. Once the feasibility study is completed, the alternatives would be provided as part of the *GetFeasibility* response. Each alternative should represent a feasible task at the time when the alternative was computed. Clients should be aware that the feasibility might change at any time afterwards.

#### 7.3.11.2 Data Types

The conceptual model of the *GetFeasibility* operation is shown in the following UML diagram.

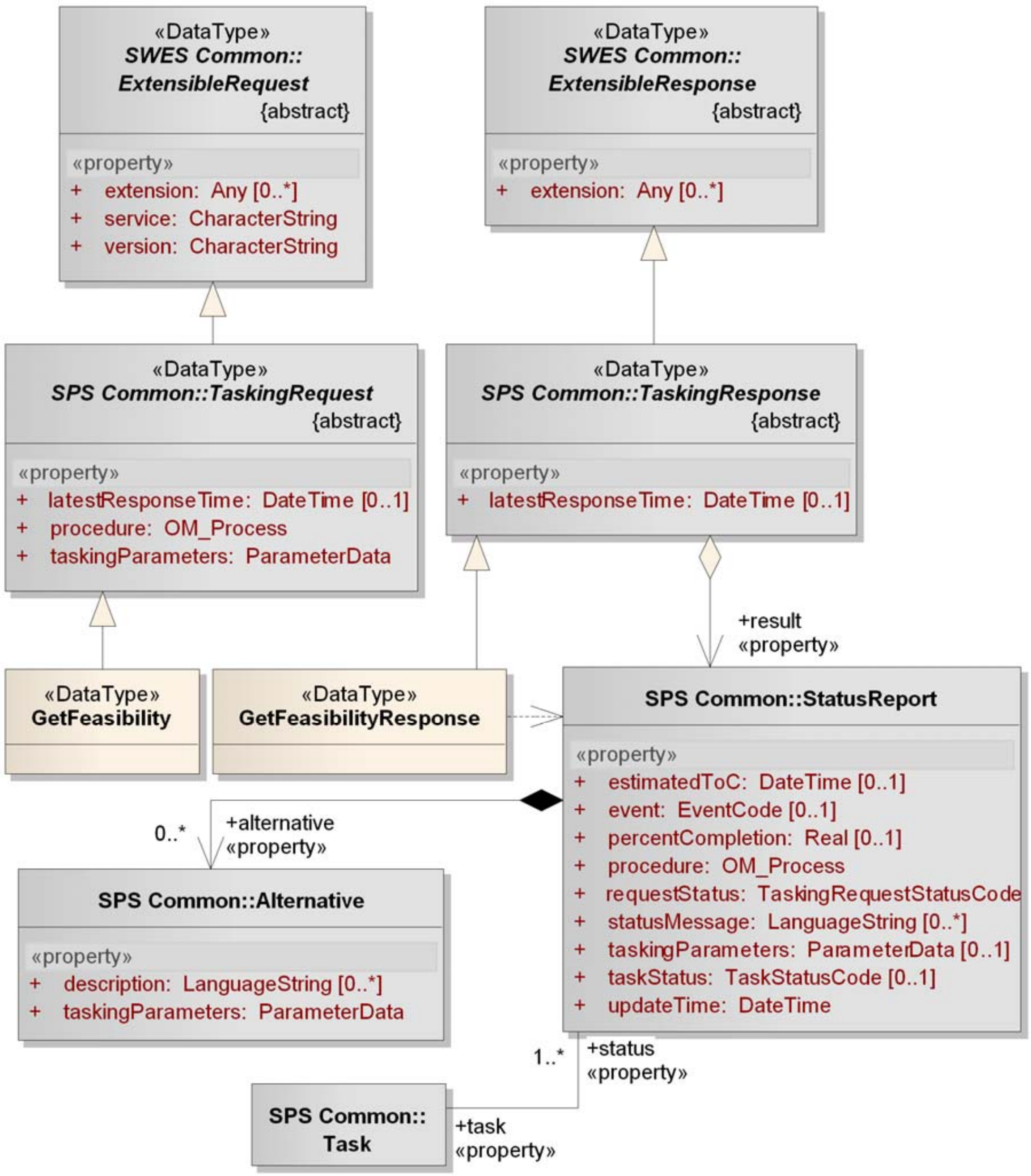

«DataType»
SWES Common::
ExtensibleRequest
{abstract}
«property»
+ extension: Any [0..*]
+ service: CharacterString
+ version: CharacterString
«DataType»
SWES Common::
ExtensibleResponse
{abstract}
«property»
+ extension: Any [0..*]
«DataType»
SPS Common::TaskingRequest
{abstract}
«property»
+ latestResponseTime: DateTime [0..1]
+ procedure: OM_Process
+ taskingParameters: ParameterData
«DataType»
SPS Common::TaskingResponse
{abstract}
«property»
+ latestResponseTime: DateTime [0..1]
+result
«property»
«DataType»
GetFeasibility
«DataType»
GetFeasibilityResponse
SPS Common::StatusReport
«property»
+ estimatedToC: DateTime [0..1]
+ event: EventCode [0..1]
+ percentCompletion: Real [0..1]
+ procedure: OM_Process
+ requestStatus: TaskingRequestStatusCode
+ statusMessage: LanguageString [0..*]
+ taskingParameters: ParameterData [0..1]
+ taskStatus: TaskStatusCode [0..1]
+ updateTime: DateTime
0..*
+alternative
«property»
SPS Common::Alternative
«property»
+ description: LanguageString [0..*]
+ taskingParameters: ParameterData
1..*
+status
«property»
SPS Common::
Task
+task
«property»

**Figure 29 — Data types of the GetFeasibility operation**

The details of the operation request and response are explained in the following subclauses.

### 7.3.11.3 Operation Request - GetFeasibility

Sending an instance of the GetFeasibility data type to the service performs an SPS GetFeasibility operation request.

The *GetFeasibility* data type is derived from the *TaskingRequest* data type (see clause 7.3.1.3) and therefore inherits all the properties contained in that data type. *GetFeasibility* neither restricts the content model of *TaskingRequest* nor adds additional properties.

| Requirement | |
|---|---|
| http://www.opengis.net/spec/SPS/2.0/req/GetFeasibilityRequest/dataType | |
| REQ 91. | The *GetFeasibility* data type shall contain the properties defined for *TaskingRequest*. |

### 7.3.11.4 Operation Response - GetFeasibilityResponse

The *GetFeasibilityResponse* data type represents the response to an SPS *GetFeasibility* operation request.

The *GetFeasibilityResponse* data type is derived from the *TaskingResponse* data type (see clause 7.3.1.4) and therefore inherits all the properties contained in that data type. *GetFeasibilityResponse* neither restricts the content model of *TaskingResponse* nor adds additional properties.

| Requirement | |
|---|---|
| http://www.opengis.net/spec/SPS/2.0/req/GetFeasibilityResponse/dataType | |
| REQ 92. | The *GetFeasibilityResponse* data type shall contain the properties defined for *TaskingResponse*. |

A *GetFeasibilityResponse* contains a *StatusReport* (see clause 7.3.1.5) to indicate the result of the requested operation. As a *GetFeasibility* request is a tasking request, the final result of that request might not be directly available and would then be pending. The properties of a *StatusReport* and the possible transitions (see clause 6.3.6) shall be used as defined in the following table.

All transitions starting from the *Pending* state are not applicable for reporting in the response, as the response provides only information about the first state transition, i.e. from initial to accepted, pending, or rejected. Clients retrieve further state transitions via notifications or *GetStatus* operation calls.

**Table 57 – StatusReport usage for different state transitions of a GetFeasibility request**

| property name/cardinality | GetFeasibility Request State Transitions (From → To) | | | | | |
|---|---|---|---|---|---|---|
| | **Initial → Pending** | **Initial → Accepted** | **Initial → Rejected** | **Pending → Accepted** | **Pending → Rejected** | **Pending → Rejected (request expired)** |
| task/1 | new identifier provided by service | | | identifier previously provided by service | | |
| estimatedToC/0..1 | NA | | | | | |
| event (code)/0..1 | NA | | | | | TaskingRequestExpired |
| percentCompletion/0..1 | NA [1] | | | | | |
| procedure/1 | identifier of procedure for which GetFeasibility request was made | | | | | |
| requestStatus (code)/1 | Pending | Accepted | Rejected | Accepted | Rejected | Rejected |
| statusMessage/0..* | service may provide additional information to client in human readable form | | | | | |
| taskingParameters/0..1 | NA [2] | | | | | |
| taskStatus (code)/0..1 | NA [3] | | | | | |
| updateTime/1 | point in time when transition was made | | | | | |
| alternative/0..* | may be provided by service | | | | | |
| StatusReport encoded as ReservationReport | no | | | | | |
| Applicable in GetFeasibility Response | yes | yes | yes | no | no | no |

NA = not applicable, means that property is not provided

Notes:

1 only applicable to tasks that are being or have been executed

2 not applicable in direct response to tasking request

3 GetFeasibility does not lead to a scheduled task

### 7.3.11.5 Exceptions

| Requirement | |
|---|---|
| http://www.opengis.net/spec/SPS/2.0/req/GetFeasibilityResponse/exceptions | |
| REQ 93. | When an SPS server encounters an error while performing a *GetFeasibility* operation, it shall return an exception message as specified in clause 7.2. |

#### 7.3.11.6 Examples

Clause 9.6 provides example XML instances for the GetFeasibility operation request and response.

### 7.3.12 Update Operation

#### 7.3.12.1 Introduction

The Update operation allows SPS clients to update a successfully submitted/reserved task that has not been finalized yet.

| Requirement | |
|---|---|
| http://www.opengis.net/spec/SPS/2.0/req/Update/Rules | |
| REQ 94. | The client encodes the tasking parameters according to the parameter description of the DescribeTasking response. This description indicates which parameters can be updated (see clause 7.4.3). The following rules apply:<br>1. The default value of the *updatable* property on a SWE Common *AbstractDataComponent* used for describing the syntax and semantics of tasking parameters at SPS shall have the default value *true*. Thus, whenever a client encounters a tasking parameter component where the updatable property is omitted, that component is considered to be updatable. If the parameter is not set, the parameter is considered as non-updateable.<br>2. The structure of the tasking parameters in an *Update* request reflect the description in the *DescribeTasking* response with non-updatable parameters being removed. Thus fields/items of *DataRecords*/*DataChoices* in the parameter description shall be removed entirely if the contained data component is not updatable.<br>3. Any SPS server shall reject *Update* requests if clients try to update non-updateable parameters. The server shall |

| | |
|---|---|
| | identify the critical paramter. *Update* requests containing a single non-updateable parameter shall be rejected completely.<br>4. If an SPS flags a *DataRecord*/*DataChoice* that is (part of) of a tasking parameter description as updatable then at least one of the fields/items in that *DataRecord*/*DataChoice* shall be updatable as well.<br>5. An SPS shall not set the updatable flag on a data component that is contained in the *elementType* property of a *DataArray*/*Matrix*. To indicate updateability of a *DataArray*/*Matrix*, the updatable property of the *DataArray*/*Matrix* itself shall be used. A service may, however, flag fields of a *DataRecord* that represent the *elementType* of a *DataArray*/*Matrix* as updatable. |

Note: as a result of the above rules, updating tasks of a given procedure might not be allowed. Also, each updatable *DataRecord*/*DataChoice* has at least one *field*/*item* that can be used in an update.

Performing an update of a task is a distinct action. As such, the update request is treated separate from the task itself.

| Requirement | |
|---|---|
| http://www.opengis.net/spec/SPS/2.0/req/UpdateRequest/identifier | |
| REQ 95. | Any SPS shall assign a unique identifier to an *UpdateRequest*. This is especially needed for a pending *UpdateRequest* that enters the *Pending* state.<br>Clients use this identifier to query the status of the update request. If the request gets accepted, then the update was successful and the service shall keep track of this event in the status log of the updated task (as it triggers a transition, see clause 10). Another client querying the status of the task itself will then know that the task has just been updated. If the update was rejected, the task is unchanged (no transition was made). |

#### 7.3.12.2 Data Types

The conceptual model of the Update operation is shown in the following UML diagram.

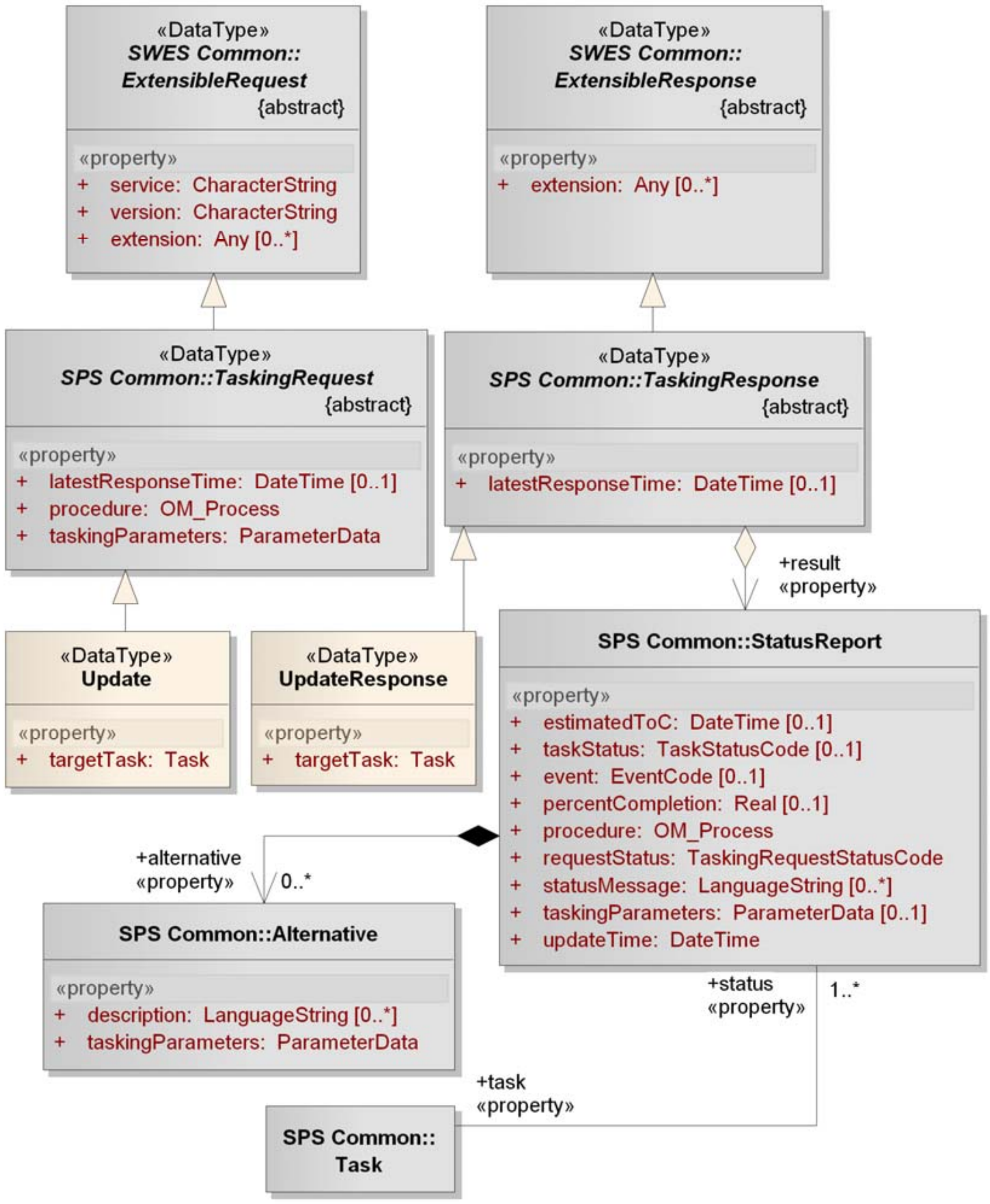


**Figure 30 — Data types of the Update operation**

The details of the operation request and response are explained in the following subclauses.

### 7.3.12.3 Operation Request - Update

Sending an instance of the Update data type to the service performs an SPS Update operation request.

The *Update* data type is derived from the *TaskingRequest* data type (see clause 7.3.1.3) and therefore inherits all the properties contained in that data type. *Update* does not restrict the content model of *TaskingRequest*.

| Requirement | |
|---|---|
| http://www.opengis.net/spec/SPS/2.0/req/UpdateRequest/dataType | |
| REQ 96. | The *Update* data type shall contain the properties defined for *TaskingRequest*. In addition, it shall contain the properties according to Table 58. |

**Table 58 — Property in the Update data type**

| Name | Definition | Data type and values | Multiplicity and use |
|---|---|---|---|
| targetTask | Pointer to the (scheduled) task to update. | Task [id], see clause 7.3.1.6<br>value shall be a pointer to a task that is scheduled by the service | One (mandatory) |
| id) Note: the primary use of this property is to provide a pointer/identifier – see OGC 09-001 clause 16.3.1 for further details. | | | |

### 7.3.12.4 Operation Response - UpdateResponse

The *UpdateResponse* data type represents the response to an SPS *Update* operation request.

The *UpdateResponse* data type is derived from the *TaskingResponse* data type (see clause 7.3.1.4) and therefore inherits all the properties contained in that data type.

| Requirement | |
|---|---|
| http://www.opengis.net/spec/SPS/2.0/req/UpdateResponse/dataType | |
| REQ 97. | The *UpdateResponse* data type shall contain the properties defined for *TaskingResponse*. In addition, it shall contain the properties according to Table 59. |

**Table 59 — Property in the UpdateResponse data type**

| Name | Definition | Data type and values | Multiplicity and use |
|---|---|---|---|
| targetTask | Pointer to the (scheduled) task to update. | Task [id], see clause 7.3.1.6<br>value shall be a pointer to a task that is scheduled by the service | One (mandatory) |
| id) Note: the primary use of this property is to provide a pointer/identifier – see OGC 09-001 clause 16.3.1 for further details. | | | |

An *UpdateResponse* contains a *StatusReport* (see clause 7.3.1.5) to indicate the result of the requested operation. As an *Update* request is a tasking request, the final result of that request might not be directly available and would then be pending.

| Requirement | |
|---|---|
| http://www.opengis.net/spec/SPS/2.0/req/UpdateResponse/statusReportUsage | |
| REQ 98. | The properties of a *StatusReport* for the possible transitions of an Update (tasking) request shall be used as defined in Table 60. |

All transitions starting from the *Pending* state are not applicable for reporting in the response, as the response provides only information about the first state transition, i.e. from initial to accepted, pending, or rejected. Clients retrieve further state transitions via notifications or *GetStatus* operation calls.

### Table 60 – StatusReport usage for different state transitions of an Update request

| **property name/cardinality** | **Update Request State Transitions (From → To)** | | | | | |
|---|---|---|---|---|---|---|
| | **Initial → Pending** | **Initial → Accepted** | **Initial → Rejected** | **Pending → Accepted** | **Pending → Rejected** | **Pending → Rejected (request expired)** |
| task/1 | new identifier provided by service | | | identifier previously provided by service | | |
| estimatedToC/0..1 | NA | | | | | |
| event (code)/0..1 | NA | | | | | TaskingRequestExpired |
| percentCompletion/0..1 | NA | | | | | |
| procedure/1 | identifier of procedure that belongs to task for which Update request was made | | | | | |
| requestStatus (code)/1 | Pending | Accepted | Rejected | Accepted | Rejected | Rejected |
| statusMessage/0..* | service may provide additional information to client in human readable form | | | | | |
| taskingParameters/0..1 | NA | | | | | |
| taskStatus (code)/0..1 | NA | | | | | |
| updateTime/1 | point in time when transition was made | | | | | |
| alternative/0..* | may be provided by service | | | | | |
| StatusReport encoded as ReservationReport | NA | | | | | |
| Applicable in Update Response | yes | yes | yes | no | no | no |
| NA = not applicable | | | | | | |

#### 7.3.12.5 Exceptions

| Requirement | |
|---|---|
| http://www.opengis.net/spec/SPS/2.0/req/UpdateResponse/exceptions | |
| REQ 99. | When an SPS server encounters an error while performing a *Update* operation, it shall return an exception message as specified in clause 7.2. |

#### 7.3.12.6 Examples

Clause 9.6 provides example XML instances for the Update operation request and response.

### 7.3.13 Cancel Operation

#### 7.3.13.1 Introduction

The *Cancel* operation allows SPS clients to cancel a scheduled task (see clause 6.3.6). The service may reject the cancellation. The response should indicate why the cancellation did not succeed. If the cancellation was rejected, the task remains in its current state.

#### 7.3.13.2 Data Types

The conceptual model of the *Cancel* operation is shown in the following UML diagram.

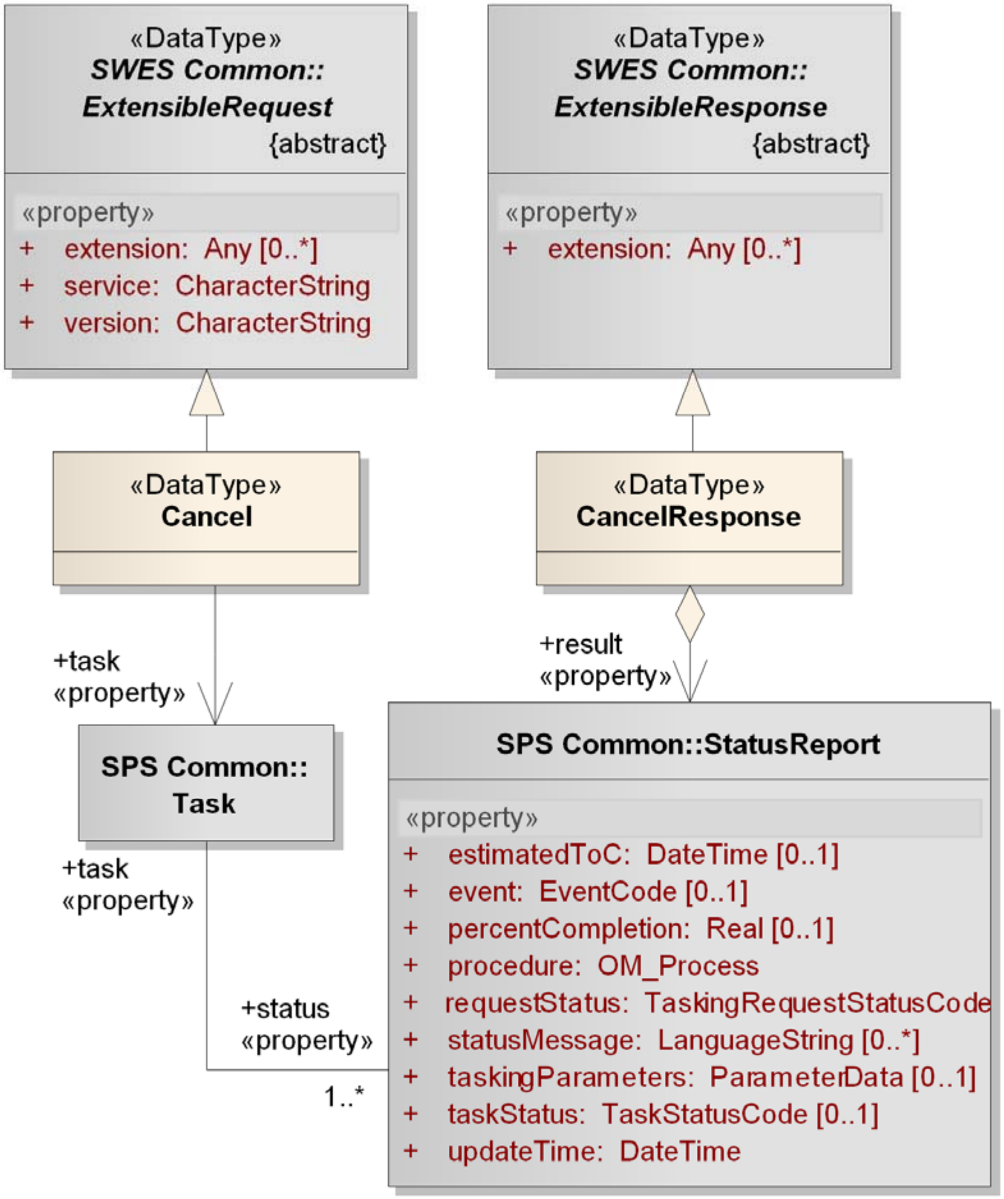


**Figure 31 — Data types of the Cancel operation**

The details of the operation request and response are explained in the following subclauses.

### 7.3.13.3 Operation Request - Cancel

Sending an instance of the *Cancel* data type to the service performs an SPS *Cancel* operation request.

The *Cancel* data type is derived from the SWES *ExtensibleRequest* data type specified in clause 9 of [OGC 09-001] and therefore inherits all the properties contained in that data type. *Cancel* does not restrict the content model of *ExtensibleRequest*.

| Requirement | |
|---|---|
| http://www.opengis.net/spec/SPS/2.0/req/CancelRequest/dataType | |
| REQ 100. | The *Cancel* data type shall contain the properties defined for SWES *ExtensibleRequest*. In addition, it shall contain the property according to Table 61. |

**Table 61 — Property in the Cancel data type**

| Name | Definition | Data type and values | Multiplicity and use |
|---|---|---|---|
| task | Pointer to the (scheduled) task to cancel. | Task [id], see clause 7.3.1.6<br>value shall be a pointer to a task that is scheduled by the service | One (mandatory) |
| id) Note: the primary use of this property is to provide a pointer/identifier – see OGC 09-001 clause 16.3.1 for further details. | | | |

#### 7.3.13.4 Operation Response - CancelResponse

The *CancelResponse* data type represents the response to an SPS *Cancel* operation request.

The *CancelResponse* data type is derived from the SWES *ExtensibleResponse* data type specified in clause 9 of [OGC 09-001] and therefore inherits all the properties contained in that data type. *CancelResponse* does not restrict the content model of *ExtensibleResponse*.

| Requirement | |
|---|---|
| http://www.opengis.net/spec/SPS/2.0/req/CancelResponse/dataType | |
| REQ 101. | The *CancelResponse* data type shall contain the properties defined for SWES *ExtensibleResponse*. In addition, it shall contain the property according to Table 62. |

**Table 62 — Properties in the CancelResponse data type**

| Name | Definition | Data type and values | Multiplicity and use |
|---|---|---|---|
| result | report with the outcome of the cancellation request | StatusReport, see 7.3.1.5<br>Properties of StatusReport shall be used as defined in Table 63 | One (mandatory) |

Table 63 illustrates the usage of the *StatusReport* properties in a *CancelResponse*.

**Table 63 – StatusReport property usage in Cancel operation response**

| **property name/cardinality** | **Operation outcome** | |
|---|---|---|
| | **Cancellation was accepted** | **Cancellation was rejected** |
| task/1 | task identifier used in request | |
| estimatedToC/0..1 | NA [1] | |
| event (code)/0..1 | NA [2] | |
| percentCompletion/0..1 | NA [3] | |
| procedure/1 | identifier of procedure associated with scheduled task | |
| requestStatus (code)/1 | Accepted | Rejected |
| statusMessage/0..* | usage optional | service should indicate why the cancellation was rejected |
| taskingParameters/0..1 | NA [4] | |
| taskStatus (code)/0..1 | NA [5] | |
| updateTime/1 | point in time when cancellation was accepted | point in time when cancellation was rejected |
| alternative/0..* | NA [6] | |
| NA = not applicable, means the property is not used in the StatusReport<br>Notes:<br>1 only applicable to scheduled tasks that have not been finalized yet<br>2,4,5 only applicable to tasking requests and tasks<br>3 only applicable to tasks that are being or have been executed<br>6 only applicable to tasking requests | | |

#### 7.3.13.5 Exceptions

| Requirement | |
|---|---|
| http://www.opengis.net/spec/SPS/2.0/req/CancelResponse/exceptions | |
| REQ 102. | When an SPS server encounters an error while performing a *Cancel* operation, it shall return an exception message as specified in clause 7.2. |

#### 7.3.13.6 Examples

Clause 9.6 provides example XML instances for the Cancel operation request and response.

## 7.4 SPS tasking parameters representation

SPS servers describe optional and mandatory tasking parameters. Clients use the definition to provide corresponding tasking parameter values. To ensure common understanding between client and server, a common exchange protocol is used to express both descriptions and tasking parameter values.

SPS uses the types defined in the SweCommon Data Model (OGC 08-094) to define tasking parameters. The tasking parameters of a given procedure are defined in the *DescribeTaskingResponse*. Clients have to use one of the encodings provided in the contents section of the capabilities (e.g. *TextEncoding*, *XMLEncoding*, etc.) to encode the tasking parameters in the various tasking requests.

**Listing 1 – example of an SPS tasking parameter description**

```
<swe:DataRecord …>
  <swe:field name="taskTimeFrame">
    <swe:TimeRange definition="http://www.opengis.net/def/property/OGC-
SPS/0/TaskTimeFrame" referenceFrame="http://www.opengis.net/def/trs/BIPM/0/UTC"
optional="false" updatable="false">
      <swe:uom xlink:href="http://www.opengis.net/def/uom/ISO-8601/0/Gregorian"/>
    </swe:TimeRange>
  </swe:field>
  <swe:field name="positioningChoice">
    <swe:DataChoice optional="true">
      <swe:item name="pointToLookAt">
        <swe:Vector definition="http://www.opengis.net/def/property/OGC-SPS-X-
CAM/0/PointToLookAt" referenceFrame="http://www.opengis.net/def/crs/EPSG/0/4979">
          <swe:coordinate name="lat">
            <swe:Quantity
definition="http://sweet.jpl.nasa.gov/2.0/spaceCoordinates.owl#Latitude" axisID="Lat">
              <swe:uom xlink:href="deg"/>
            </swe:Quantity>
          </swe:coordinate>
          <swe:coordinate name="long">
            <swe:Quantity
definition="http://sweet.jpl.nasa.gov/2.0/spaceCoordinates.owl#Longitude" axisID="Long">
              <swe:uom code="deg"/>
            </swe:Quantity>
          </swe:coordinate>
          <swe:coordinate name="h">
            <swe:Quantity
definition="http://sweet.jpl.nasa.gov/2.0/spaceCoordinates.owl#Vertical" axisID="h">
              <swe:uom code="m"/>
              <swe:value>0</swe:value>
            </swe:Quantity>
          </swe:coordinate>
        </swe:Vector>
      </swe:item>
      <swe:item name="relativePositioning">
        <swe:DataRecord definition="http://www.opengis.net/def/property/OGC-SPS-X-
CAM/0/RelativePan">
          <swe:field name="relativeHorizontalPan">
            <swe:Quantity definition="http://www.opengis.net/def/property/OGC-SPS-X-
CAM/0/RelativeHorizontalPan" optional="true">
              <swe:uom code="deg"/>
              <swe:constraint>
                <swe:AllowedValues>
                  <swe:interval>-180 180</swe:interval>
                </swe:AllowedValues>
              </swe:constraint>
            </swe:Quantity>
          </swe:field>
          <swe:field name="relativeVerticalPan">
```

```
            <swe:Quantity definition="http://www.opengis.net/def/property/OGC-SPS-X-
CAM/0/RelativeVerticalPan" optional="true">
              <swe:uom code="deg"/>
              <swe:constraint>
                <swe:AllowedValues>
                  <swe:interval>-90 90</swe:interval>
                </swe:AllowedValues>
              </swe:constraint>
            </swe:Quantity>
          </swe:field>
        </swe:DataRecord>
      </swe:item>
    </swe:DataChoice>
  </swe:field>
  <swe:field name="focalLength">
    <swe:Quantity definition="http://www.opengis.net/def/property/OGC-SPS-X-
CAM/0/FocalLength" optional="true">
      <swe:uom code="mm"/>
      <swe:constraint>
        <swe:AllowedValues>
          <swe:interval>3.5 10</swe:interval>
        </swe:AllowedValues>
      </swe:constraint>
    </swe:Quantity>
  </swe:field>
</swe:DataRecord>
```

**Listing 2 – example of tasking parameters corresponding to description provided by client in given encoding**

```
<sps:ParameterData …>
  <sps:encoding>
    <swe:TextEncoding tokenSeparator="," blockSeparator="@@"/>
  </sps:encoding>
<sps:values>2010-08-20T12:37:00+02:00,2010-08-20T14:30:00+02:00,Y,pointToLookAt,51.902112
,8.192728,0,Y,3.5</sps:values>
</sps:ParameterData>
```

### 7.4.1 Optional Parameters

As defined in clause 7.3.4 (DescribeTasking), the SPS provides any number of derivates from *AbstractDataComponent* that have to be used by clients in order to task the service. All *AbstractDataComponents* have an *optional* attribute.

Following OGC 08-094, only components that are listed inside the fields of a SWE Common *DataRecord* shall have the *optional* attribute with value *true*.

| Requirement | |
|---|---|
| http://www.opengis.net/spec/SPS/2.0/req/TaskingParameters/optionalParameters | |
| REQ 103. | Components that are not contained in a *DataRecord* field shall not use the *optional* attribute or, if they do use it, set its value to *false* (which is the default value for this attribute). |

Clients can omit values for components that are marked as optional in a tasking request.

Items of a SWE Common *DataChoice* cannot be flagged as being optional. Thus, either the whole choice is optional (in that case it is a field in a *DataRecord*) or mandatory.

**Listing 3 – example for optional and required tasking parameters**

```
<swe:DataRecord …>
  <!-- Mandatory Parameter-->
  <swe:field name="taskTimeFrame">
    <swe:TimeRange optional="false" …>
      <!-- … -->
    </swe:TimeRange>
  </swe:field>
  <!-- Optional Parameter-->
  <swe:field name="positioningChoice">
    <swe:DataChoice optional="true" …>
      <swe:item name="pointToLookAt">
        <swe:Vector …>
          <!-- … -->
        </swe:Vector>
      </swe:item>
      <swe:item name="relativePositioning">
        <swe:DataRecord …>
          <!-- Optional Parameter-->
          <swe:field name="relativeHorizontalPan">
            <swe:Quantity optional="true" …>
              <!-- … -->
            </swe:Quantity>
          </swe:field>
          <!-- Optional Parameter-->
          <swe:field name="relativeVerticalPan">
            <swe:Quantity optional="true" …>
              <!-- …-->
            </swe:Quantity>
          </swe:field>
        </swe:DataRecord>
      </swe:item>
    </swe:DataChoice>
  </swe:field>
  <!-- Optional Parameter-->
  <swe:field name="focalLength">
    <swe:Quantity optional="true" …>
      <!-- …-->
    </swe:Quantity>
  </swe:field>
</swe:DataRecord>
```

### 7.4.2 Default Values

All data components defined in [OGC 08-094] can be either used as data descriptors or data containers. Data containers set the attribute values, data descriptors don’t. SPS uses both descriptors and containers to describe tasking parameters! Given values indicate default values. The SPS can set default values for each tasking parameter. An SPS may but is not required to provide default values. The client can either accept this default value and use it as-is or overwrite it in a tasking request.

**Listing 4 – example of a tasking parameter description without default values**

```
<swe:DataRecord …>
  <!-- -->
  <swe:field name="positioningChoice">
    <swe:DataChoice …>
      <swe:item name="pointToLookAt">
        <swe:Vector …>
          <!-- -->
          <swe:coordinate name="h">
            <swe:Quantity
definition="http://sweet.jpl.nasa.gov/2.0/spaceCoordinates.owl#Vertical" axisID="h">
              <swe:uom code="m"/>
            </swe:Quantity>
          </swe:coordinate>
        </swe:Vector>
      </swe:item>
      <!-- -->
    </swe:DataChoice>
  </swe:field>
  <!-- -->
</swe:DataRecord>
```

**Listing 5 – example of a tasking parameter description including default values**

```
<swe:DataRecord …>
  <!-- -->
  <swe:field name="positioningChoice">
    <swe:DataChoice …>
      <swe:item name="pointToLookAt">
        <swe:Vector …>
          <!-- -->
          <swe:coordinate name="h">
            <swe:Quantity …>
              <swe:uom code="m"/>
              <swe:value>0</swe:value>
            </swe:Quantity>
          </swe:coordinate>
        </swe:Vector>
      </swe:item>
      <!-- -->
    </swe:DataChoice>
  </swe:field>
  <!-- -->
</swe:DataRecord>
```

If an SPS uses a *DataArray* or *Matrix* in its tasking parameters description then it may provide default values that are encoded according to a description that the service also provides in that array/matrix. As clients may safely ignore given default values, they can also ignore unknown/unsupported encodings that they might encounter in a DataArray/Matrix provided by an SPS.

### 7.4.3 Updatable parameters

*AbstractDataComponents* as defined in OGC 08-094 clause 7.2 provide an optional *updatable* attribute. This attribute is set to *true* if the corresponding tasking parameter can be included in an *Update* request. If the attribute is not set or set to false, the tasking parameter cannot be updated and thus no value for it is included in an *Update* request.

Clients can simply strip any component from the tasking parameter description retrieved via DescribeTasking that are not updatable. The resulting description defines the structure of the parameters to be included in an update request. If the data component that represents the whole tasking parameter descriptor for tasking a given procedure is not updatable then the *Update* operation is not realized for that procedure.

**Listing 6 – example of tasking parameter description with updatable and non-updatable parameters**

```
<!-- Update operation is implemented for procedure -->
<swe:DataRecord …>
  <!-- Parameter not updatable -->
  <swe:field name="taskTimeFrame">
    <swe:TimeRange … updatable="false">
      <!-- -->
    </swe:TimeRange>
  </swe:field>
  <!-- Parameter is updatable -->
  <swe:field name="positioningChoice">
    <swe:DataChoice …>
      <!-- Choice item is available for Update -->
      <swe:item name="pointToLookAt">
        <swe:Vector …>
          <!-- -->
        </swe:Vector>
      </swe:item>
      <!-- Choice item is available for Update -->
      <swe:item name="relativePositioning">
        <swe:DataRecord …>
          <!-- Record field is available for Update -->
          <swe:field name="relativeHorizontalPan">
            <swe:Quantity …>
              <!-- -->
            </swe:Quantity>
          </swe:field>
          <!-- Record field is available for Update -->
          <swe:field name="relativeVerticalPan">
            <swe:Quantity …>
              <!-- -->
            </swe:Quantity>
          </swe:field>
        </swe:DataRecord>
      </swe:item>
    </swe:DataChoice>
  </swe:field>
  <!-- Parameter is updatable -->
  <swe:field name="focalLength">
    <swe:Quantity …>
      <!-- -->
    </swe:Quantity>
  </swe:field>
</swe:DataRecord>
```

Removing all components that are not updatable would result in the following parameter description. A client would use this description in an *Update* request.

**Listing 7 – example of tasking parameter description for update request where all non-updatable parameters have been removed**

```
<swe:DataRecord …>
  <swe:field name="positioningChoice">
    <swe:DataChoice …>
      <swe:item name="pointToLookAt">
        <swe:Vector …>
          <!-- -->
        </swe:Vector>
      </swe:item>
      <swe:item name="relativePositioning">
        <swe:DataRecord …>
          <swe:field name="relativeHorizontalPan">
            <swe:Quantity …>
              <!-- -->
            </swe:Quantity>
          </swe:field>
          <swe:field name="relativeVerticalPan">
            <swe:Quantity …>
              <!-- -->
            </swe:Quantity>
          </swe:field>
        </swe:DataRecord>
      </swe:item>
    </swe:DataChoice>
  </swe:field>
  <swe:field name="focalLength">
    <swe:Quantity …>
      <!-- -->
    </swe:Quantity>
  </swe:field>
</swe:DataRecord>
```

### 7.4.4 Constraints/restrictions

Most of the simple components defined in SWE Common allow provision of *constraint* attributes. Those can be set by SPS to constrain allowed values for tasking parameters.

**Listing 8 – example of constraints/restrictions on tasking parameter values**

```
<swe:DataRecord …>
  <swe:field name="taskTimeFrame">
    <!-- -->
  </swe:field>
  <swe:field name="positioningChoice">
    <swe:DataChoice optional="true">
      <swe:item name="pointToLookAt">
        <!-- -->
      </swe:item>
      <swe:item name="relativePositioning">
        <swe:DataRecord …>
          <swe:field name="relativeHorizontalPan">
            <swe:Quantity …>
              <swe:uom code="deg"/>
               <swe:constraint>
                 <swe:AllowedValues>
                   <swe:interval>-180 180</swe:interval>
                 </swe:AllowedValues>
               </swe:constraint>
            </swe:Quantity>
          </swe:field>
          <swe:field name="relativeVerticalPan">
            <swe:Quantity …>
              <swe:uom code="deg"/>
               <swe:constraint>
                 <swe:AllowedValues>
                   <swe:interval>-90 90</swe:interval>
```

```
                </swe:AllowedValues>
              </swe:constraint>
            </swe:Quantity>
          </swe:field>
        </swe:DataRecord>
      </swe:item>
    </swe:DataChoice>
  </swe:field>
  <swe:field name="focalLength">
    <swe:Quantity …>
      <swe:uom code="mm"/>
      <swe:constraint>
        <swe:AllowedValues>
          <swe:interval>3.5 10</swe:interval>
        </swe:AllowedValues>
      </swe:constraint>
    </swe:Quantity>
  </swe:field>
</swe:DataRecord>
```

### 7.4.5 Definition (observedProperty)/Semantics

Each tasking parameter sets the value(s) for a single property. The property is defined using the *definition* attribute of the data component. Resolving the URN can retrieve the semantics.

### 7.4.6 Uoms

The unit of measure (UOM) is defined in data component using the mechanisms described in [OGC08-094] clause 6.2.3.

### 7.4.7 Encoding (XML, text, binary)

SPS defines the supported encodings as described in clause 7.3.3.3. It is recommended to use either *TextEncoding* or *XMLEncoding* (defined in OGC08-094 clause 7.6), though the advanced encoding package, which supports raw and base 64 binary blocks (defined in OGC08-094 clause 7.7), can be supported as well.

# 8 Publish/Subscribe

## 8.1 Introduction

The publish/subscribe functionality is an optional feature of SPS.

The SPS model defines events, which can be published to interested consumers via a publish/subscribe interface (see clause 10, more specifically see Figure 32 and Figure 34). The events represent state changes of a task or tasking request. However, further events can also be recognized by an SPS, for example the events defined in [OGC 09-001].

NOTE: As discussed in chapter 17.2 of [OGC 09-001], the realization of the publish/subscribe functionality defined in this chapter shall be documented in a specific binding for this standard (e.g., the realization in the SOAP binding is documented in clause 9).

A publish/subscribe interface may support various subscription models as explained in [OGC 09-001] and [OGC 09-032]. If supported, content based filtering using XPath 1.0 and FES 1.1 [OGC 04-095] shall be implemented as specified in clause 17.2.3 of [OGC 09-001].

### 8.2 SPS Events

The publishable events recognized by this standard are defined in Table 64.

| Requirement | |
|---|---|
| http://www.opengis.net/spec/SPS/2.0/req/Events/eventTypes | |
| REQ 104. | Any SPS implementing publish/subscribe functionality shall implement the events according to Table 64. |

An SPS may also recognize and publish the events defined by the SWE Service Model (see OGC 09-001 clause 17.2) or any other event.

| Requirement | |
|---|---|
| http://www.opengis.net/spec/SPS/2.0/req/Events/channelBasedSubscription | |
| REQ 105. | If an SPS supports channel based subscriptions (see clause 8.3), it shall state the topics and thus supported events in the topic set contained in its notification metadata (see clause 7.3.2.4). |

**Table 64 — SPS Events and their encoding**

| **Event name** [a] | **Event definition** | **State transition from → to** | **Encoding of the event** | **Use at SPS that implements Publish/Subscribe** |
|---|---|---|---|---|
| TaskingRequestAccepted | Tasking request was accepted. | Initial \| Pending → Accepted | see Table 65 | mandatory |
| TaskingRequestRejected | Tasking request was rejected. | Initial \| Pending → Rejected | see Table 65 | optional |
| TaskingRequestPending | Tasking request is pending. | Initial → Pending | see Table 65 | optional |
| TaskingReque | see Table 14 | Pending → Rejected (TaskingRequestExpi | see Table | mandatory |

| | | | | |
|---|---|---|---|---|
| stExpired | | red) | 65 | |
| DataPublished | | InExecution → InExecution (DataPublished) | see Table 66 | mandatory |
| ReservationExpired | | Reserved → Final | see Table 66 | conditional<br>implement if Reserve operation is realized |
| TaskCancelled | | Scheduled (Reserved or InExecution) → Final (TaskCancelled) | see Table 66 | conditional<br>implement if Cancel operation is realized |
| TaskCompleted | | InExecution → Final (TaskCompleted) | see Table 66 | mandatory |
| TaskConfirmed | | Reserved → InExecution | see Table 66 | conditional<br>implement if Confirm operation is realized |
| TaskUpdated | | InExecution → InExecution (TaskUpdated) \| Reserved → Reserved (TaskUpdated) | see Table 66 | conditional<br>implement if Update operation is realized |
| TaskFailed | | Scheduled (Reserved or InExecution) → Final (TaskFailed) | see Table 66 | mandatory |

[a] Although some values listed in the column appear to contain spaces, they shall not contain spaces.

**Table 65 – StatusReport encoding for notification of tasking request state transition**

| | **Transition from → to** | **StatusReport encoding and property usage for notification of state transition for** | | | |
|---|---|---|---|---|---|
| | | GetFeasibility request | Reserve request | Submit request | Update request |
| **State Transitions** | Initial → Pending | as defined for according state transition in Table 57 | as defined for according state transition in Table 52 | as defined for according state transition in Table 31 | as defined for according state transition in Table 60 |
| | Initial \| Pending → Rejected | | | | |
| | Pending → Rejected (TaskingRequestExpired) | | | | |
| | Initial \| Pending → Accepted | | | | |
| NA = not applicable | | | | | |

**Table 66 – StatusReport encoding for notification of scheduled task state transition**

| | **Transition from → to** | **StatusReport encoding and property usage for notification of state transition** |
|---|---|---|
| **State Transitions** | Reserved → Reserved (TaskUpdated) | as defined for according state transition in Table 36 and Table 37 but without provision of taskingParameters |
| | InExecution → InExecution (TaskUpdated) | |
| | Reserved → InExecution (TaskConfirmed) | |
| | Reserved → Final (ReservationExpired) | |
| | InExecution → InExecution (DataPublished) | |
| | InExecution → Final (TaskCompleted) | |
| | Scheduled (Reserved or InExecution) → Final (TaskFailed) | |
| | Scheduled (Reserved or InExecution) → Final (TaskCancelled) | |
| NA = not applicable | | |

Clause 9.6 provides example XML instances for notificatios of some of the SPS events.

### 8.3 Channel based filtering/SPS notification topics

When using channel based filtering, it is imperative to define which channels can be used and which notifications are sent on each channel. The definitions of events recognized by this standard are listed in Table 64. Each event is given by its name and definition.

The OASIS WS-Topics standard defines the *TopicNamespace* type as a mean to group and describe channels/topics that belong to a specific (target) namespace. The topic namespace of this standard is defined through Listing 9 and Table 67.

**Listing 9 – SPS Topic Namespace**

```
<wstop:TopicNamespace xmlns:wstop="http://docs.oasis-
open.org/wsn/t-1" xmlns:sps="http://www.opengis.net/sps/2.0"
name="SPS-Topic-Namespace"
targetNamespace="http://www.opengis.net/sps/2.0" final="true">
  <wstop:Topic name="TaskEvent">
    <wstop:Topic name="TaskFailure"
messageTypes="sps:StatusReport"/>
    <wstop:Topic name="TaskCancellation"
messageTypes="sps:StatusReport"/>
    <wstop:Topic name="TaskCompletion"
messageTypes="sps:StatusReport"/>
    <wstop:Topic name="TaskConfirmation"
messageTypes="sps:StatusReport"/>
    <wstop:Topic name="TaskUpdate"
messageTypes="sps:StatusReport"/>
    <wstop:Topic name="DataPublication"
messageTypes="sps:StatusReport"/>
    <wstop:Topic name="TaskReservation"
messageTypes="sps:ReservationReport"/>
    <wstop:Topic name="TaskSubmission"
messageTypes="sps:StatusReport"/>
    <wstop:Topic name=”ReservationExpiration”
messageTypes=”sps:ReservationReport”/>
  </wstop:Topic>
  <wstop:Topic name=”TaskingRequestEvent”>
    <wstop:Topic name=”TaskingRequestExpiration”
messageTypes=”sps:StatusReport”/>
    <wstop:Topic name=”TaskingRequestRejection”
messageTypes=”sps:StatusReport”/>
    <wstop:Topic name=”TaskingRequestAcceptance”
messageTypes=”sps:StatusReport”/>
    <wstop:Topic name=”TaskingRequestPending”
messageTypes=”sps:StatusReport”/>
  </wstop:Topic>
</wstop:TopicNamespace>
```

The following table defines which events are published on which topics. The events and their encoding are defined in Table 64.

| Requirement | |
|---|---|
| http://www.opengis.net/spec/SPS/2.0/req/Events/topics | |
| REQ 106. | An SPS that supports channel based filtering/notification shall publish events on topics according to Table 67. Such a service shall publish only those SPS events that belong to topics listed in the topic set of the service (the topic set is part of the notification metadata contained in the Capabilities of the service, see clause 7.3.2.4). Events from a different topic namespace may be published by the service. |

Note: this is to ensure that an SPS instance is publishing SPS events according to what the service advertised via its topic set.

| Requirement | |
|---|---|
| http://www.opengis.net/spec/SPS/2.0/req/Events/topics/conditions | |
| REQ 107. | Table 67 also defines which topics shall be implemented by an SPS that supports channel based filtering/notification topics in general and which shall be implemented under certain conditions. The required topics shall be listed in the topic set of the service. |

Each SPS may implement additional topics defined in other standards (like [OGC 09-001]).

**Table 67 — Topics and the events posted on them**

| Topic name | Parent topic name | Name of event(s) posted on topic | Use at SPS that realizes channel based filtering/notification topics |
|---|---|---|---|
| TaskEvent | - | *no events are posted on this topic- it is only used for grouping of topics in the SPS topic namespace* | |
| TaskSubmission | TaskEvent | TaskingRequestAccepted (the StatusReport encoding the event shall have an event property with value TaskSubmitted) | mandatory |
| DataPublication | TaskEvent | DataPublished | |
| TaskCompletion | TaskEvent | TaskCompleted | |
| TaskFailure | TaskEvent | TaskFailed | |
| TaskReservation | TaskEvent | TaskingRequestAccepted (the ReservationReport encoding the event shall have an event property with value TaskReserved) | conditional<br>implement if Reserve&Confirm operations are realized |
| ReservationExpiration | TaskEvent | ReservationExpired | |
| TaskConfirmation | TaskEvent | TaskConfirmed | |
| TaskUpdate | TaskEvent | TaskUpdated | conditional<br>implement if Update operation is realized |
| TaskCancellation | TaskEvent | TaskCancelled | conditional<br>implement if Cancel operation is realized |
| TaskingRequestEvent | - | *no events are posted on this topic- it is only used for grouping of topics in the SPS topic namespace* | |
| TaskingRequestAcceptance | TaskingRequestEvent | TaskingRequestAccepted | optional [1] |
| TaskingRequestRejection | TaskingRequestEvent | TaskingRequestRejected | optional [1] |
| TaskingRequestExpiration | TaskingRequestEvent | TaskingRequestExpired | mandatory |
| TaskingRequestPending | TaskingRequestEvent | TaskingRequestPending | optional [1] |

Notes:

[1] If topic is implemented then publication of according event is required.

Clause 9.6 describes a tasking scenario with example XML instances, one of which is an SPS Capabilities document. It contains an exemplary topic set.

# 9 SOAP binding

## 9.1 Introduction

This section defines the realization of functionality defined in this standard for a service using SOAP. This standard does not prescribe usage of either SOAP 1.1 or SOAP 1.2. It also does not prescribe WSDL 1.1 or WSDL 2.0.

This standard does not define any specific policy statements to be included in a WSDL document or in service requests and responses for defining certain established, available or desired behavior. If the need for such policies arises in the future, necessary policy statements can be included in the standard and/or its extensions.

## 9.2 Exceptions

| Requirement | |
|---|---|
| http://www.opengis.net/spec/SPS/2.0/req/SOAP/exceptions | |
| REQ 108. | The operations defined in this standard use exception codes defined by OWS Common [OGC 06-121r3] chapter 8, SWES [OGC 09-001] chapter 15 as well as Table 6 in this standard.<br>The encoding of these exceptions for the operations used by this standard (in a SOAP binding) shall be as defined in clause 19.2 of [OGC 09-001]. |

NOTE Each operation defined in this standard can have additional requirements with respect to the implementation of the ows:Exception element to be used in the [Details] property (see [OGC 09-001] clause 19.2.1) of faults generated while performing that operation. These requirements are stated in the according clauses of each operation.

Clause 9.6 provides example XML instances for SOAP faults that inform about service exceptions.

The following subclauses define the SOAP fault encoding of the SPS exceptions that are introduced in chapter 7.2. The definitions are provided using abstract (SOAP) fault properties as described in OGC 09-001 chapter 19.2.1. These abstract fault properties are mapped to the properties of SOAP 1.1/1.2 faults as defined in sections 19.2.2 and 19.2.3 of OGC 09-001.

### 9.2.1 StatusInformationExpired exception

The meaning of this exception (code) is defined in clause 7.2 of this standard.

| Requirement | |
|---|---|
| http://www.opengis.net/spec/SPS/2.0/req/SOAP/Fault/StatusInformationExpired | |
| REQ 109. | The abstract fault properties for this exception shall be as follows:<br>• [Code] The *QName* soap11:Service (SOAP 1.1) or soap12:Receiver (SOAP 1.2)<br>• [Subcode] The *QName sps:StatusInformationExpired*<br>• [Reason] the string: “The status information for the requested task / tasking request has already expired.”<br>• [Details] An *ows:Exception* element as defined in clause 8.2 of [OGC 06-121r3] |

### 9.2.2 ModificationOfFinalizedTask exception

The meaning of this exception (code) is defined in clause 7.2 of this standard.

| Requirement | |
|---|---|
| http://www.opengis.net/spec/SPS/2.0/req/SOAP/Fault/ModificationOfFinalizedTask | |
| REQ 110. | The abstract fault properties for this exception shall be as follows:<br>• [Code] The *QName* soap11:Client (SOAP 1.1) or soap12:Sender (SOAP 1.2)<br>• [Subcode] The *QName sps:ModificationOfFinalizedTask*<br>• [Reason] the string: “The requested task has already been finalized.”<br>• [Details] An *ows:Exception* element as defined in clause 8.2 of [OGC 06-121r3] |

## 9.3 Action URIs

For the SOAP binding, a standard needs to define action URIs for the following features:

- as SOAPAction HTTP header field of a SOAP 1.1 request
- as action parameter in a SOAP 1.2 request (SOAP 1.2 feature: “http://www.w3.org/2003/05/soap/features/action/”)
- as WS-Addressing [action] message addressing property

NOTE If and how a service instance makes use of one or more of these features depends on the chosen SOAP and WSDL version as well as on the requirements of the service instance.

| Requirement | |
|---|---|
| http://www.opengis.net/spec/SPS/2.0/req/SOAP/ActionURIs | |
| REQ 111. | The action URIs shall be used for the various message facets (requests and responses of operations) according to Table 68.<br>The action URI for SPS specific exceptions shall be as defined in Table 69. |

The actions URIs for the operations specified by the SWE Service Model (DescribeSensor and UpdateSensorDescription) are defined in [OGC 09-001] clause 19.3.

The actions URIs for exceptions/fault message types that SPS operations use are also defined in [OGC 09-001] clause 19.3.

**Table 68 — Action URIs for SPS message facets**

| Message Facet a | Action URI a | Applicable in feature (Y=yes, N=no) | | |
|---|---|---|---|---|
| | | SOAP 1.1 SOAPAction | SOAP 1.2 action | WS-Addressing [action] |
| GetCapabilities request | http://www.opengis.net/sps/2.0/GetCapabilities | Y | Y | Y |
| GetCapabilities response | http://www.opengis.net/sps/2.0/GetCapabilitiesResponse | N | N | Y |
| DescribeTasking request | http://www.opengis.net/sps/2.0/DescribeTasking | Y | Y | Y |
| DescribeTasking response | http://www.opengis.net/sps/2.0/DescribeTaskingResponse | N | N | Y |
| Cancel request | http://www.opengis.net/sps/2.0/Cancel | Y | Y | Y |
| Cancel response | http://www.opengis.net/sps/2.0/CancelResponse | N | N | Y |
| Confirm request | http://www.opengis.net/sps/2.0/Confirm | Y | Y | Y |
| Confirm response | http://www.opengis.net/sps/2.0/ConfirmResponse | N | N | Y |
| DescribeResultAccess request | http://www.opengis.net/sps/2.0/DescribeResultAccess | Y | Y | Y |
| DescribeResultAccess response | http://www.opengis.net/sps/2.0/DescribeResultAccessResponse | N | N | Y |
| GetFeasibility request | http://www.opengis.net/sps/2.0/GetFeasibility | Y | Y | Y |
| GetFeasibility response | http://www.opengis.net/sps/2.0/GetFeasibilityResponse | N | N | Y |
| GetStatus request | http://www.opengis.net/sps/2.0/GetStatus | Y | Y | Y |
| GetStatus response | http://www.opengis.net/sps/2.0/GetStatusResponse | N | N | Y |
| GetTask request | http://www.opengis.net/sps/2.0/GetTask | Y | Y | Y |

| Message Facet a | Action URI a | Applicable in feature (Y=yes, N=no) | | |
|---|---|---|---|---|
| | | SOAP 1.1 SOAPAction | SOAP 1.2 action | WS-Addressing [action] |
| GetTask response | http://www.opengis.net/sps/2.0/ GetTaskResponse | N | N | Y |
| Reserve request | http://www.opengis.net/sps/2.0/ Reserve | Y | Y | Y |
| Reserve response | http://www.opengis.net/sps/2.0/ ReserveResponse | N | N | Y |
| Submit request | http://www.opengis.net/sps/2.0/ Submit | Y | Y | Y |
| Submit response | http://www.opengis.net/sps/2.0/ SubmitResponse | N | N | Y |
| Update request | http://www.opengis.net/sps/2.0/ Update | Y | Y | Y |
| Update response | http://www.opengis.net/sps/2.0/ UpdateResponse | N | N | Y |
| a Although some values listed in the column appear to contain spaces, they shall not contain spaces.<br>NOTE The action URIs for the messages defined by the SWE Service Model and WS-Notification are not listed here – they can be found in table 35 of [OGC 09-001] and the according paragraphs of WS-Notification. | | | | |

Clause 9.6 provides example XML instances for operation requests and responses, some of which are wrapped by a SOAP envelope. These examples make use of the action URIs defined in this section.

**Table 69 — Action URI for SPS exceptions/fault types**

| Exception/Fault type | WS-Addressing [action] message addressing property value |
|---|---|
| Exception defined by SPS | http://www.opengis.net/sps/2.0/Exception |

### 9.4 Realization of Publish/Subscribe

| Requirement | |
|---|---|
| http://www.opengis.net/spec/SPS/2.0/req/SOAP/PubSub | |
| REQ 112. | In the SOAP binding of this service, Publish/Subscribe functionality shall be implemented as defined in clause 19.4 of [OGC 09-001]. |

Clause 9.6 provides example XML instances for subscribing to and being notified of SPS events.

### 9.5 Realization of Asynchronous Request/Response

| Requirement | |
|---|---|
| http://www.opengis.net/spec/SPS/2.0/req/SOAP/WSAdressing | |
| REQ 113. | As defined in clause 19.4 of [OGC 09-001], an implementation of this standard shall use WS-Addressing to enable asynchronous request-response in the SOAP binding of the service. The behavior for handling asynchronous tasking responses shall be compliant to section 7.3.1.3. |

| Requirement | |
|---|---|
| http://www.opengis.net/spec/SPS/2.0/req/SOAP/AnonymousURI | |
| REQ 114. | If a client used the *anonymous-URI* (see 09-032 section 10.2) as value of the *wsa:ReplyTo* property in the SOAP header of a tasking request – and the service supports the anonymous-URI feature – then the tasking response shall be sent in the synchronous backchannel of the transport protocol (e.g. the HTTP response message). |

If a client used the *none-URI* (see 09-032 section 10.2) as value of the *wsa:ReplyTo* property in the SOAP header of a tasking request then the service can discard any operation response that would normally be generated, as the client is not receiving it anyway. Only the response requirements of the underlying communication protocol need to be satisfied, e.g. in case of HTTP an HTTP response message has to be returned.

### 9.6 SPS Examples Scenario

In the following, a scenario of tasking a pan, tilt, zoom camera is elaborated with XML examples.

Note: this scenario only covers parts of the overall functionality that can be realized via an SPS. It does not cover all possible cases, situations and client/service interactions.

Note: some but not all of the following examples are wrapped in a SOAP envelope. Unwrapped examples can easily be augmented with the missing information.

### 9.6.1 Retrieving the Capabilities Document

2010-08-20T11:00:00+02:00 – The client sends a GetCapabilities request to the service.

**Listing 10 - GetCapabilities request example**

```
<soap12:Envelope xmlns:soap12="http://www.w3.org/2003/05/soap-envelope"
xmlns:sps="http://www.opengis.net/sps/2.0"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
  <soap12:Body>
    <sps:GetCapabilities/>
  </soap12:Body>
</soap12:Envelope>
```

2010-08-20T11:00:01+02:00 – The service sends a response with the Capabilities document.

**Listing 11 - SPS Capabilities document example**

```
<soap12:Envelope xmlns:soap12="http://www.w3.org/2003/05/soap-envelope"
xmlns:sps="http://www.opengis.net/sps/2.0" xmlns:xlink="http://www.w3.org/1999/xlink"
xmlns:wsa="http://www.w3.org/2005/08/addressing"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns:ows="http://www.opengis.net/ows/1.1" xmlns:gml="http://www.opengis.net/gml/3.2"
xmlns:swes="http://www.opengis.net/swes/2.0" xmlns:wstop="http://docs.oasis-
open.org/wsn/t-1">
  <soap12:Body>
    <sps:Capabilities version="2.0.0">
      <ows:ServiceIdentification>
        <ows:Title xml:lang="en-us">SPS Specification Service</ows:Title>
        <ows:ServiceType>SPS</ows:ServiceType>
        <ows:ServiceTypeVersion>2.0.0</ows:ServiceTypeVersion>
        <ows:Profile>http://www.opengis.net/spec/SPS/2.0/conf/BasicPubSub</ows:Profile>

<ows:Profile>http://www.opengis.net/spec/SPS/2.0/conf/ChannelBasedPubSub</ows:Profile>
        <ows:Profile>http://www.opengis.net/spec/SPS/2.0/conf/Core</ows:Profile>

<ows:Profile>http://www.opengis.net/spec/SPS/2.0/conf/FeasibilityController</ows:Profile>

<ows:Profile>http://www.opengis.net/spec/SPS/2.0/conf/ReservationManager</ows:Profile>
        <ows:Profile>http://www.opengis.net/spec/SPS/2.0/conf/SOAP</ows:Profile>
        <ows:Profile>http://www.opengis.net/spec/SPS/2.0/conf/StateLogger</ows:Profile>
        <ows:Profile>http://www.opengis.net/spec/SPS/2.0/conf/TaskCanceller</ows:Profile>
        <ows:Profile>http://www.opengis.net/spec/SPS/2.0/conf/TaskUpdater</ows:Profile>
        <ows:Profile>http://www.opengis.net/spec/SPS/2.0/conf/XMLEncoding</ows:Profile>

<ows:Profile>http://www.opengis.net/spec/SWES/2.0/conf/BasicSWEServiceMetadata</ows:Profi
le>

<ows:Profile>http://www.opengis.net/spec/SWES/2.0/conf/SensorProvider</ows:Profile>

<ows:Profile>http://www.opengis.net/spec/SWES/2.0/conf/SensorHistoryProvider</ows:Profile
>
        <ows:Profile>http://www.opengis.net/spec/SWES/2.0/conf/XMLEncoding</ows:Profile>
        <ows:Profile>http://www.opengis.net/spec/SWES/2.0/conf/SOAPBinding</ows:Profile>

<ows:Profile>http://www.opengis.net/spec/SWES/2.0/conf/PublishSubscribe</ows:Profile>
        <ows:Profile>http://www.opengis.net/spec/SWE/2.0/conf/core</ows:Profile>
        <ows:Profile>http://www.opengis.net/spec/SWE/2.0/conf/uml-simple-
components</ows:Profile>
        <ows:Profile>http://www.opengis.net/spec/SWE/2.0/conf/uml-record-
components</ows:Profile>
```

```
        <ows:Profile>http://www.opengis.net/spec/SWE/2.0/conf/uml-choice-
components</ows:Profile>
        <ows:Profile>http://www.opengis.net/spec/SWE/2.0/conf/uml-simple-
encodings</ows:Profile>
        <ows:Profile>http://www.opengis.net/spec/SWE/2.0/conf/xsd-simple-
components</ows:Profile>
        <ows:Profile>http://www.opengis.net/spec/SWE/2.0/conf/xsd-record-
components</ows:Profile>
        <ows:Profile>http://www.opengis.net/spec/SWE/2.0/conf/xsd-choice-
components</ows:Profile>
        <ows:Profile>http://www.opengis.net/spec/SWE/2.0/conf/xsd-simple-
encodings</ows:Profile>
        <ows:Profile>http://www.opengis.net/spec/SWE/2.0/conf/general-encoding-
rules</ows:Profile>
        <ows:Profile>http://www.opengis.net/spec/SWE/2.0/conf/text-encoding-
rules</ows:Profile>
      </ows:ServiceIdentification>
      <ows:ServiceProvider>
        <ows:ProviderName>SWE SPS 2.0 SWG</ows:ProviderName>
        <ows:ProviderSite xlink:href="http://www.opengeospatial.org/swe/sps"/>
        <ows:ServiceContact>
          <ows:IndividualName>Johannes Echterhoff</ows:IndividualName>
          <ows:ContactInfo>
            <ows:Phone>
              <ows:Voice>0049...</ows:Voice>
            </ows:Phone>
          </ows:ContactInfo>
        </ows:ServiceContact>
      </ows:ServiceProvider>
      <ows:OperationsMetadata>
        <ows:Operation name="GetCapabilities">
          <ows:DCP>
            <ows:HTTP>
              <ows:Post xlink:href="http://www.ogc.org/SPS"/>
            </ows:HTTP>
          </ows:DCP>
          <ows:Parameter name="Sections">
            <ows:AllowedValues>
              <ows:Value>All</ows:Value>
              <ows:Value>ServiceIdentification</ows:Value>
              <ows:Value>ServiceProvider</ows:Value>
              <ows:Value>OperationsMetadata</ows:Value>
              <ows:Value>Contents</ows:Value>
              <ows:Value>Notifications</ows:Value>
            </ows:AllowedValues>
          </ows:Parameter>
        </ows:Operation>
        <ows:Operation name="DescribeTasking">
          <ows:DCP>
            <ows:HTTP>
              <ows:Post xlink:href="http://www.ogc.org/SPS"/>
            </ows:HTTP>
          </ows:DCP>
        </ows:Operation>
        <ows:Operation name="Submit">
          <ows:DCP>
            <ows:HTTP>
              <ows:Post xlink:href="http://www.ogc.org/SPS"/>
            </ows:HTTP>
          </ows:DCP>
        </ows:Operation>
        <ows:Operation name="DescribeResultAccess">
          <ows:DCP>
            <ows:HTTP>
              <ows:Post xlink:href="http://www.ogc.org/SPS"/>
            </ows:HTTP>
          </ows:DCP>
        </ows:Operation>
        <ows:Operation name="GetFeasibility">
          <ows:DCP>
            <ows:HTTP>
              <ows:Post xlink:href="http://www.ogc.org/SPS"/>
            </ows:HTTP>
          </ows:DCP>
        </ows:Operation>
        <ows:Operation name="Update">
          <ows:DCP>
            <ows:HTTP>
              <ows:Post xlink:href="http://www.ogc.org/SPS"/>
            </ows:HTTP>
          </ows:DCP>
        </ows:Operation>
        <ows:Operation name="GetStatus">
          <ows:DCP>
            <ows:HTTP>
              <ows:Post xlink:href="http://www.ogc.org/SPS"/>
            </ows:HTTP>
          </ows:DCP>
          <ows:Parameter name="since">
            <ows:AnyValue/>
          </ows:Parameter>
        </ows:Operation>
        <ows:Operation name="GetTask">
          <ows:DCP>
            <ows:HTTP>
              <ows:Post xlink:href="http://www.ogc.org/SPS"/>
            </ows:HTTP>
          </ows:DCP>
        </ows:Operation>
        <ows:Operation name="Cancel">
          <ows:DCP>
            <ows:HTTP>
              <ows:Post xlink:href="http://www.ogc.org/SPS"/>
            </ows:HTTP>
          </ows:DCP>
        </ows:Operation>
        <ows:Operation name="Reserve">
          <ows:DCP>
            <ows:HTTP>
              <ows:Post xlink:href="http://www.ogc.org/SPS"/>
            </ows:HTTP>
          </ows:DCP>
        </ows:Operation>
        <ows:Operation name="Confirm">
          <ows:DCP>
            <ows:HTTP>
              <ows:Post xlink:href="http://www.ogc.org/SPS"/>
            </ows:HTTP>
          </ows:DCP>
        </ows:Operation>
        <ows:Constraint name="PostEncoding">
          <ows:AllowedValues>
            <ows:Value>SOAP</ows:Value>
          </ows:AllowedValues>
        </ows:Constraint>
      </ows:OperationsMetadata>
      <sps:notifications>
        <swes:NotificationProducerMetadata>
          <swes:producerEndpoint>
            <wsa:EndpointReference>
              <wsa:Address>http://www.ogc.org/SPS/Producer</wsa:Address>
            </wsa:EndpointReference>
          </swes:producerEndpoint>
          <swes:supportedDialects>
            <swes:FilterDialectMetadata>
              <swes:topicExpressionDialect>http://docs.oasis-open.org/wsn/t-
1/TopicExpression/Simple</swes:topicExpressionDialect>
              <swes:topicExpressionDialect>http://docs.oasis-open.org/wsn/t-
1/TopicExpression/Concrete</swes:topicExpressionDialect>
              <swes:topicExpressionDialect>http://docs.oasis-open.org/wsn/t-
1/TopicExpression/Full</swes:topicExpressionDialect>
              <swes:topicExpressionDialect>http://www.w3.org/TR/1999/REC-xpath-
19991116</swes:topicExpressionDialect>
```

```
              <swes:messageContentDialect>http://www.w3.org/TR/1999/REC-xpath-
19991116</swes:messageContentDialect>
            </swes:FilterDialectMetadata>
          </swes:supportedDialects>
          <swes:fixedTopicSet>false</swes:fixedTopicSet>
          <swes:servedTopics>
            <wstop:TopicSet>
              <sps:TaskEvent>
                <sps:TaskFailure wstop:topic="true"/>
                <sps:TaskCancellation wstop:topic="true"/>
                <sps:TaskCompletion wstop:topic="true"/>
                <sps:TaskConfirmation wstop:topic="true"/>
                <sps:TaskUpdate wstop:topic="true"/>
                <sps:DataPublication wstop:topic="true"/>
                <sps:TaskReservation wstop:topic="true"/>
                <sps:TaskSubmission wstop:topic="true"/>
                <sps:ReservationExpiration wstop:topic="true"/>
              </sps:TaskEvent>
              <sps:TaskingRequestEvent>
                <sps:TaskingRequestExpiration wstop:topic="true"/>
              </sps:TaskingRequestEvent>
              <swes:CapabilitiesChange>
                <swes:OfferingAddition wstop:topic="true"/>
                <swes:OfferingDeletion wstop:topic="true"/>
              </swes:CapabilitiesChange>
            </wstop:TopicSet>
          </swes:servedTopics>
          <swes:usedTopicNamespace targetNamespace="http://www.opengis.net/sps/2.0"
final="true">
            <wstop:Topic name="TaskEvent">
              <wstop:Topic name="TaskFailure" messageTypes="sps:StatusReport"/>
              <wstop:Topic name="TaskCancellation" messageTypes="sps:StatusReport"/>
              <wstop:Topic name="TaskCompletion" messageTypes="sps:StatusReport"/>
              <wstop:Topic name="TaskConfirmation" messageTypes="sps:StatusReport"/>
              <wstop:Topic name="TaskUpdate" messageTypes="sps:StatusReport"/>
              <wstop:Topic name="DataPublication" messageTypes="sps:StatusReport"/>
              <wstop:Topic name="TaskReservation" messageTypes="sps:ReservationReport"/>
              <wstop:Topic name="TaskSubmission" messageTypes="sps:StatusReport"/>
              <wstop:Topic name="ReservationExpiration"
messageTypes="sps:ReservationReport"/>
            </wstop:Topic>
            <wstop:Topic name="TaskingRequestEvent">
              <wstop:Topic name="TaskingRequestExpiration"
messageTypes="sps:StatusReport"/>
              <wstop:Topic name="TaskingRequestRejection"
messageTypes="sps:StatusReport"/>
              <wstop:Topic name="TaskingRequestAcceptance"
messageTypes="sps:StatusReport"/>
              <wstop:Topic name="TaskingRequestPending" messageTypes="sps:StatusReport"/>
            </wstop:Topic>
          </swes:usedTopicNamespace>
          <swes:usedTopicNamespace targetNamespace="http://www.opengis.net/swes/2.0"
final="true">
            <wstop:Topic name="CapabilitiesChange" messageTypes="swes:SWESEvent">
              <wstop:Topic name="OfferingAddition" messageTypes="swes:OfferingChanged"/>
              <wstop:Topic name="OfferingDeletion" messageTypes="swes:OfferingChanged"/>
            </wstop:Topic>
            <wstop:Topic name="SensorInsertion" messageTypes="swes:SensorChanged"/>
            <wstop:Topic name="SensorDescriptionUpdate"
messageTypes="swes:SensorDescriptionUpdated"/>
          </swes:usedTopicNamespace>
        </swes:NotificationProducerMetadata>
      </sps:notifications>
      <sps:contents>
        <sps:SPSContents>

<swes:procedureDescriptionFormat>http://www.opengis.net/sensorML/1.0.1</swes:procedureDes
criptionFormat>
          <swes:observableProperty>http://www.opengis.net/def/propertyType/x-
radiance</swes:observableProperty>
          <swes:offering>
            <sps:SensorOffering>
```

```
              <swes:identifier>http://www.ogc.org/sps/offering1</swes:identifier>
              <swes:procedure>http://www.ogc.org/procedure/camera/1</swes:procedure>
              <sps:observableArea>
                <sps:byPolygon>
                  <gml:Polygon gml:id="gid01">
                    <gml:exterior>
                      <gml:LinearRing>
                        <gml:pos
srsName="http://www.opengis.net/def/crs/EPSG/0/4326">51.9 8.186</gml:pos>
                        <gml:pos
srsName="http://www.opengis.net/def/crs/EPSG/0/4326">51.9005 8.186</gml:pos>
                        <gml:pos
srsName="http://www.opengis.net/def/crs/EPSG/0/4326">51.9005 8.199</gml:pos>
                        <gml:pos
srsName="http://www.opengis.net/def/crs/EPSG/0/4326">51.9 8.199</gml:pos>
                      </gml:LinearRing>
                    </gml:exterior>
                  </gml:Polygon>
                </sps:byPolygon>
              </sps:observableArea>
            </sps:SensorOffering>
          </swes:offering>
          <sps:minStatusTime>PT12H</sps:minStatusTime>

<sps:supportedEncoding>http://www.opengis.net/swe/2.0/TextEncoding</sps:supportedEncoding
>
        </sps:SPSContents>
      </sps:contents>
    </sps:Capabilities>
  </soap12:Body>
</soap12:Envelope>
```

### 9.6.2 Getting Result Access Information for a Procedure

2010-08-20T11:06:00+02:00 - The client sends a DescribeResultAccess request to the service to learn which data storages the SPS uses to make data gathered by procedure *http://www.ogc.org/procedure/camera/1* accessible.

**Listing 12 - DescribeResultAccess request example targetting a procedure**

```
<sps:DescribeResultAccess service="SPS" version="2.0.0"
xmlns:sps="http://www.opengis.net/sps/2.0" xmlns:swe="http://www.opengis.net/swe/2.0"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
  <sps:target>
    <sps:procedure>http://www.ogc.org/procedure/camera/1</sps:procedure>
  </sps:target>
</sps:DescribeResultAccess>
```

2010-08-20T11:06:01+02:00 – The service sends a response with references to data storages (a Sensor Observation Service and an online folder).

**Listing 13 - DescribeResultAccess response example**

```
<sps:DescribeResultAccessResponse xmlns:sps="http://www.opengis.net/sps/2.0"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns:swe="http://www.opengis.net/swe/2.0" xmlns:ows="http://www.opengis.net/ows/1.1"
xmlns:xlink="http://www.w3.org/1999/xlink">
  <sps:availability>
    <sps:available>
      <sps:DataAvailable>
        <sps:dataReference>
          <ows:ReferenceGroup>

<ows:Identifier>http://www.ogc.org/procedure/camera/1/accessReferenceGroups/1</ows:Identi
fier>
            <ows:Reference xlink:href="http://www.ogc.org/SOS"
xlink:role="http://www.opengis.net/spec/SPS/2.0/referenceType/ServiceURL">

<ows:Identifier>http://www.ogc.org/procedure/camera/1/accessReferenceGroups/1/references/
1</ows:Identifier>
              <ows:Metadata>
                <sps:SPSMetadata>
                  <sps:dataAccessType>http://www.opengis.net/sos/2.0</sps:dataAccessType>
                </sps:SPSMetadata>
              </ows:Metadata>
            </ows:Reference>
          </ows:ReferenceGroup>
        </sps:dataReference>
        <sps:dataReference>
          <ows:ReferenceGroup>

<ows:Identifier>http://www.ogc.org/procedure/camera/1/accessReferenceGroups/2</ows:Identi
fier>
            <ows:Reference xlink:href="http://www.ogc.org/SOS/procedure/camera/1/videos"
xlink:role="http://www.opengis.net/spec/SPS/2.0/referenceType/Folder">

<ows:Identifier>http://www.ogc.org/procedure/camera/1/accessReferenceGroups/2/references/
1</ows:Identifier>
              <ows:Format>video/mj2</ows:Format>
            </ows:Reference>
          </ows:ReferenceGroup>
        </sps:dataReference>
      </sps:DataAvailable>
    </sps:available>
  </sps:availability>
</sps:DescribeResultAccessResponse>
```

### 9.6.3 Getting the Tasking Parameter Description

2010-08-20T11:08:32+02:00 – The client sends a DescribeTasking request for the procedure to the service to find out about the available tasking options.

**Listing 14 - DescribeTasking request example**

```
<sps:DescribeTasking service="SPS" version="2.0.0"
xmlns:sps="http://www.opengis.net/sps/2.0" xmlns:xsi="http://www.w3.org/2001/XMLSchema-
instance">
  <sps:procedure>http://www.ogc.org/procedure/camera/1</sps:procedure>
</sps:DescribeTasking>
```

2010-08-20T11:08:33+02:00 – The service sends a response with the tasking parameter description for the procedure.

**Listing 15 - DescribeTasking response example**

```
<sps:DescribeTaskingResponse xmlns:gml="http://www.opengis.net/gml/3.2"
xmlns:sps="http://www.opengis.net/sps/2.0" xmlns:swe="http://www.opengis.net/swe/2.0"
```

```
xmlns:swes="http://www.opengis.net/swes/2.0" xmlns:xlink="http://www.w3.org/1999/xlink"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
  <sps:taskingParameters name="CameraTask">
    <swe:DataRecord>
      <swe:field name="taskTimeFrame">
        <swe:TimeRange definition="http://www.opengis.net/def/property/OGC-
SPS/0/TaskTimeFrame" referenceFrame="http://www.opengis.net/def/trs/BIPM/0/UTC"
optional="false" updatable="false">
          <swe:label>Task Timeframe</swe:label>
          <swe:description>Desired start and end time for tasking the
sensor</swe:description>
          <swe:uom xlink:href="http://www.opengis.net/def/uom/ISO-8601/0/Gregorian"/>
        </swe:TimeRange>
      </swe:field>
      <swe:field name="positioningChoice">
        <swe:DataChoice optional="true">
          <swe:item name="pointToLookAt">
            <swe:Vector definition="http://www.opengis.net/def/property/OGC-SPS-X-
CAM/0/PointToLookAt" referenceFrame="http://www.opengis.net/def/crs/EPSG/0/4979">
            <swe:label>Look Pointer</swe:label>
            <swe:description>3D location where the camera should look
at</swe:description>
              <swe:coordinate name="lat">
                <swe:Quantity
definition="http://sweet.jpl.nasa.gov/2.0/spaceCoordinates.owl#Latitude" axisID="Lat">
                  <swe:label>Geodetic latitude</swe:label>
                  <swe:uom xlink:href="deg"/>
                </swe:Quantity>
              </swe:coordinate>
              <swe:coordinate name="long">
                <swe:Quantity
definition="http://sweet.jpl.nasa.gov/2.0/spaceCoordinates.owl#Longitude" axisID="Long">
                  <swe:label>Geodetic longitude</swe:label>
                  <swe:uom code="deg"/>
                </swe:Quantity>
              </swe:coordinate>
              <swe:coordinate name="h">
                <swe:Quantity
definition="http://sweet.jpl.nasa.gov/2.0/spaceCoordinates.owl#Vertical" axisID="h">
                  <swe:label>Ellipsoidal height</swe:label>
                  <swe:uom code="m"/>
                  <swe:value>0</swe:value>
                </swe:Quantity>
              </swe:coordinate>
            </swe:Vector>
          </swe:item>
          <swe:item name="relativePositioning">
            <swe:DataRecord definition="http://www.opengis.net/def/property/OGC-SPS-X-
CAM/0/RelativePan">
            <swe:label>Relative Positioning</swe:label>
            <swe:description>Camera movement relative to the current
position</swe:description>
              <swe:field name="relativeHorizontalPan">
                <swe:Quantity definition="http://www.opengis.net/def/property/OGC-SPS-X-
CAM/0/RelativeHorizontalPan" optional="true">
                  <swe:uom code="deg"/>
                  <swe:constraint>
                    <swe:AllowedValues>
                      <swe:interval>-180 180</swe:interval>
                    </swe:AllowedValues>
                  </swe:constraint>
                </swe:Quantity>
              </swe:field>
              <swe:field name="relativeVerticalPan">
                <swe:Quantity definition="http://www.opengis.net/def/property/OGC-SPS-X-
CAM/0/RelativeVerticalPan" optional="true">
                  <swe:uom code="deg"/>
                  <swe:constraint>
                    <swe:AllowedValues>
                      <swe:interval>-90 90</swe:interval>
                    </swe:AllowedValues>
                  </swe:constraint>
                </swe:Quantity>
              </swe:field>
            </swe:DataRecord>
          </swe:item>
        </swe:DataChoice>
      </swe:field>
      <swe:field name="focalLength">
        <swe:Quantity definition="http://www.opengis.net/def/property/OGC-SPS-X-CAM/0/FocalLength" optional="true">
        <swe:label>Focal length</swe:label>
        <swe:description>Focal length of the camera. Controls the camera's zoom level.</swe:description>
          <swe:uom code="mm"/>
          <swe:constraint>
            <swe:AllowedValues>
              <swe:interval>3.5 10</swe:interval>
            </swe:AllowedValues>
          </swe:constraint>
        </swe:Quantity>
      </swe:field>
    </swe:DataRecord>
  </sps:taskingParameters>
</sps:DescribeTaskingResponse>
```

### 9.6.4 Determining the Feasibility of a Tasking Request

2010-08-20T11:10:00+02:00 - Satisfied with the information the client got about the procedure, the client sends a GetFeasibility request to check if the time frame from 2010-08-20T12:15:00+02:00 to 2010-08-20T14:45:00+02:00 would be a feasible task. The latest response time is set to 2010-08-20T11:15:00+02:00.

**Listing 16 – GetFeasibility request example**

```
<sps:GetFeasibility service="SPS" version="2.0.0"
xmlns:sps="http://www.opengis.net/sps/2.0" xmlns:swe="http://www.opengis.net/swe/2.0"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
  <sps:procedure>http://www.ogc.org/procedure/camera/1</sps:procedure>
  <sps:taskingParameters>
    <sps:ParameterData>
      <sps:encoding>
        <swe:TextEncoding tokenSeparator="," blockSeparator="@@" />
      </sps:encoding>
      <sps:values>2010-08-20T12:15:00+02:00,2010-08-20T14:45:00+02:00,N,N</sps:values>
    </sps:ParameterData>
  </sps:taskingParameters>
  <sps:latestResponseTime>2010-08-20T11:15:00+02:00</sps:latestResponseTime>
</sps:GetFeasibility>
```

2010-08-20T11:10:12+02:00 – The service sends a response indicating that the requested task is not feasible. But it provides two alternatives in the response:

a) First alternative indicates that time frame from 2010-08-20T12:35:00+02:00 to 2010-08-20T14:30:00+02:00 would be feasible.
b) Second alternative indicates that time frame from 2010-08-20T15:10:00+02:00 to 2010-08-20T17:00:00+02:00 would be feasible.

**Listing 17 – GetFeasibility response example**

```
<sps:GetFeasibilityResponse xmlns:sps="http://www.opengis.net/sps/2.0"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns:swe="http://www.opengis.net/swe/2.0">
  <sps:latestResponseTime>2010-08-20T12:00:00+02:00</sps:latestResponseTime>
  <sps:result>
    <sps:StatusReport>
      <sps:task>http://www.ogc.org/procedure/camera/1/tasks/5</sps:task>
      <sps:procedure>http://www.ogc.org/procedure/camera/1</sps:procedure>
      <sps:requestStatus>Rejected</sps:requestStatus>
      <sps:statusMessage xml:lang="en">The task was not feasible because the requested
time frame is not free</sps:statusMessage>
      <sps:updateTime>2010-08-20T11:10:12+02:00</sps:updateTime>
      <sps:alternative>
        <sps:Alternative>
          <sps:taskingParameters>
            <sps:ParameterData>
              <sps:encoding>
                <swe:TextEncoding tokenSeparator="," blockSeparator="@@"/>
              </sps:encoding>
              <sps:values>2010-08-20T12:35:00+02:00,2010-08-
20T14:30:00+02:00,N,N</sps:values>
            </sps:ParameterData>
          </sps:taskingParameters>
        </sps:Alternative>
      </sps:alternative>
            <sps:alternative>
        <sps:Alternative>
          <sps:taskingParameters>
            <sps:ParameterData>
              <sps:encoding>
                <swe:TextEncoding tokenSeparator="," blockSeparator="@@"/>
              </sps:encoding>
              <sps:values>2010-08-20T15:10:00+02:00,2010-08-
20T17:00:00+02:00,N,N</sps:values>
            </sps:ParameterData>
          </sps:taskingParameters>
        </sps:Alternative>
      </sps:alternative>
    </sps:StatusReport>
  </sps:result>
</sps:GetFeasibilityResponse>
```

### 9.6.5 Scheduling a Task (Submit / Reserve)

2010-08-20T11:10:20+02:00 – The client reviews the alternatives and decides to use the first one with slight alteration of the task start time (setting it to 2010-08-20T12:37:00+02:00). The client then adds some more specific parameters to control the camera. It requests that the camera looks at the location [geodetic latitude 51.902112 deg, geodetic longitude 8.192728 deg, ellipsoidal height 0 meter] and sets the focal length to 3.5mm.

2010-08-20T11:12:00+02:00 – The client schedules the task. This can be done either via directly submitting a task or by reserving it first and then confirming it a bit later on (which is useful for scenarios where multiple sensors need to be tasked together).

#### 9.6.5.1 Task Submission

2010-08-20T11:12:00+02:00 – The client sends a Submit request to the service. The latest response time is not set for this request, so the client is willing to wait however long the processing of the response is going to take.

**Listing 18 - Submit request example**

```
<sps:Submit service="SPS" version="2.0.0" xmlns:sps="http://www.opengis.net/sps/2.0"
xmlns:swe="http://www.opengis.net/swe/2.0" xmlns:xsi="http://www.w3.org/2001/XMLSchema-
instance">
  <sps:procedure>http://www.ogc.org/procedure/camera/1</sps:procedure>
  <sps:taskingParameters>
    <sps:ParameterData>
      <sps:encoding>
        <swe:TextEncoding tokenSeparator="," blockSeparator="@@" />
      </sps:encoding>
      <sps:values>2010-08-20T12:37:00+02:00,2010-08-
20T14:30:00+02:00,Y,pointToLookAt,51.902112,8.192728,0,Y,3.5</sps:values>
    </sps:ParameterData>
  </sps:taskingParameters>
</sps:Submit>
```

2010-08-20T11:12:04+02:00 – The service sends a response indicating that the task was accepted and is now in execution, so will be performed as planned.

**Listing 19 - Submit response example**

```
<sps:SubmitResponse xmlns:sps="http://www.opengis.net/sps/2.0"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns:swe="http://www.opengis.net/swe/2.0">
  <sps:result>
    <sps:StatusReport>
      <sps:task>http://www.ogc.org/procedure/camera/1/tasks/6</sps:task>
      <sps:event>TaskSubmitted</sps:event>
      <sps:percentCompletion>0</sps:percentCompletion>
      <sps:procedure>http://www.ogc.org/procedure/camera/1</sps:procedure>
      <sps:requestStatus>Accepted</sps:requestStatus>
      <sps:taskStatus>InExecution</sps:taskStatus>
      <sps:updateTime>2010-08-20T11:12:04+02:00</sps:updateTime>
      <sps:taskingParameters>
        <sps:ParameterData>
          <sps:encoding>
            <swe:TextEncoding tokenSeparator="," blockSeparator="@@"/>
          </sps:encoding>
          <sps:values>2010-08-20T12:37:00+02:00,2010-08-
20T14:30:00+02:00,Y,pointToLookAt,51.902112,8.192728,0,Y,3.5</sps:values>
        </sps:ParameterData>
      </sps:taskingParameters>
    </sps:StatusReport>
  </sps:result>
</sps:SubmitResponse>
```

### 9.6.5.2 Reserving a Task

2010-08-20T11:12:00+02:00 – The client sends a Reserve request to the service. The latest response time is set to 2010-08-20T11:20:00+02:00. The expiration time of the requested reservation is set to 2010-08-20T11:30:00+02:00.

**Listing 20 - Reserve request example**

```
<sps:Reserve service="SPS" version="2.0.0" xmlns:sps="http://www.opengis.net/sps/2.0"
xmlns:swe="http://www.opengis.net/swe/2.0" xmlns:xsi="http://www.w3.org/2001/XMLSchema-
instance">
  <sps:procedure>http://www.ogc.org/procedure/camera/1</sps:procedure>
  <sps:taskingParameters>
    <sps:ParameterData>
      <sps:encoding>
        <swe:TextEncoding tokenSeparator="," blockSeparator="@@" />
      </sps:encoding>
      <sps:values>2010-08-20T12:37:00+02:00,2010-08-
20T14:30:00+02:00,Y,pointToLookAt,51.902112,8.192728,0,Y,3.5</sps:values>
    </sps:ParameterData>
  </sps:taskingParameters>
  <sps:latestResponseTime>2010-08-20T11:20:00+02:00</sps:latestResponseTime>
  <sps:reservationExpiration>2010-08-20T11:30:00+02:00</sps:reservationExpiration>
</sps:Reserve>
```

2010-08-20T11:12:01+02:00 – The service sends a response indicating that the reservation was successful. It will expire at 2010-08-20T11:30:00+02:00.

**Listing 21 - Reserve response example**

```
<sps:ReserveResponse xmlns:sps="http://www.opengis.net/sps/2.0"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
  <sps:result>
    <sps:ReservationReport>
      <sps:task>http://www.ogc.org/procedure/camera/1/tasks/6</sps:task>
      <sps:estimatedToC>2010-08-20T14:30:00+02:00</sps:estimatedToC>
      <sps:event>TaskReserved</sps:event>
      <sps:procedure>http://www.ogc.org/procedure/camera/1</sps:procedure>
      <sps:requestStatus>Accepted</sps:requestStatus>
      <sps:taskStatus>Reserved</sps:taskStatus>
      <sps:updateTime>2010-08-20T11:12:01+02:00</sps:updateTime>
      <sps:reservationExpiration>2010-08-20T11:30:00+02:00</sps:reservationExpiration>
    </sps:ReservationReport>
  </sps:result>
</sps:ReserveResponse>
```

Now there are several options: the task automatically expires at 2010-08-20T11:30:00+02:00, the client confirms, updates or cancels the task beforehand or the task fails for some reason. The option that the reservation was updated is not considere here.

#### 9.6.5.3 Automatic Reservation Expiration

2010-08-20T12:00:00+02:00 – The client sends a GetStatus request to the service. The "since" parameter, although supported by the service, is not used in the request. Thus the current status of the task is requested.

**Listing 22 - GetStatus request example**

```
<sps:GetStatus service="SPS" version="2.0.0" xmlns:sps="http://www.opengis.net/sps/2.0"
xmlns:swe="http://www.opengis.net/swe/2.0" xmlns:xsi="http://www.w3.org/2001/XMLSchema-
instance">
  <sps:task>http://www.ogc.org/procedure/camera/1/tasks/6</sps:task>
</sps:GetStatus>
```

2010-08-20T12:00:01+02:00 – The service sends a response with information about the current status of the task, indicating that the reservation expired (at 2010-08-20T11:30:00+02:00).

**Listing 23 - GetStatus response example for expired reservation**

```
<sps:GetStatusResponse xmlns:sps="http://www.opengis.net/sps/2.0"
xmlns:swe="http://www.opengis.net/swe/2.0" xmlns:swes="http://www.opengis.net/swes/2.0"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
  <sps:status>
    <sps:StatusReport>
      <sps:task>http://www.ogc.org/procedure/camera/1/tasks/6</sps:task>
      <sps:event>ReservationExpired</sps:event>
      <sps:procedure>http://www.ogc.org/procedure/camera/1</sps:procedure>
      <sps:requestStatus>Accepted</sps:requestStatus>
      <sps:statusMessage xml:lang="en">Your reservation expired as it was not confirmed
before the agreed expiration time.</sps:statusMessage>
      <sps:taskStatus>Expired</sps:taskStatus>
      <sps:updateTime>2010-08-20T11:30:00+02:00</sps:updateTime>
    </sps:StatusReport>
  </sps:status>
</sps:GetStatusResponse>
```

### 9.6.5.4 Confirming a Reserved Task

2010-08-20T11:23:00+02:00 – The client sends a Confirm request to the service to confirm the reservation.

**Listing 24 - Confirm request example**

```
<sps:Confirm service="SPS" version="2.0.0" xmlns:sps="http://www.opengis.net/sps/2.0"
xmlns:swe="http://www.opengis.net/swe/2.0" xmlns:xsi="http://www.w3.org/2001/XMLSchema-
instance">
  <sps:task>http://www.ogc.org/procedure/camera/1/tasks/6</sps:task>
</sps:Confirm>
```

2010-08-20T11:23:08+02:00 – The service sends a response indicating that the task was confirmed and is now in execution, so will be performed as planned.

**Listing 25 - Confirm response example**

```
<sps:ConfirmResponse xmlns:sps="http://www.opengis.net/sps/2.0"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
  <sps:result>
    <sps:StatusReport>
      <sps:task>http://www.ogc.org/procedure/camera/1/tasks/6</sps:task>
      <sps:procedure>http://www.ogc.org/procedure/camera/1</sps:procedure>
      <sps:requestStatus>Accepted</sps:requestStatus>
      <sps:updateTime>2010-08-20T11:23:08+02:00</sps:updateTime>
    </sps:StatusReport>
  </sps:result>
</sps:ConfirmResponse>
```

2010-08-20T12:00:00+02:00 – The client sends a GetTask request to the service.

**Listing 26 - GetTask request example**

```
<sps:GetTask service="SPS" version="2.0.0" xmlns:sps="http://www.opengis.net/sps/2.0"
xmlns:swe="http://www.opengis.net/swe/2.0" xmlns:xsi="http://www.w3.org/2001/XMLSchema-
instance">
  <sps:task>http://www.ogc.org/procedure/camera/1/tasks/6</sps:task>
</sps:GetTask>
```

2010-08-20T12:00:01+02:00 – The service sends a response with information about the task, including all state transitions made so far. All transitions are reported because the service supports state logging. Note that here the tasking parameters used in reserving the task are also included.

**Listing 27 - GetTask response example**

```
<sps:GetTaskResponse xmlns:sps="http://www.opengis.net/sps/2.0"
xmlns:swe="http://www.opengis.net/swe/2.0" xmlns:swes="http://www.opengis.net/swes/2.0"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
  <sps:task>
    <sps:Task>
      <swes:identifier>http://www.ogc.org/procedure/camera/1/tasks/6</swes:identifier>
      <sps:status>
        <sps:ReservationReport>
          <sps:task>http://www.ogc.org/procedure/camera/1/tasks/6</sps:task>
          <sps:estimatedToC>2010-08-20T14:30:00+02:00</sps:estimatedToC>
          <sps:event>TaskReserved</sps:event>
          <sps:procedure>http://www.ogc.org/procedure/camera/1</sps:procedure>
          <sps:requestStatus>Accepted</sps:requestStatus>
          <sps:taskStatus>Reserved</sps:taskStatus>
          <sps:updateTime>2010-08-20T11:12:01+02:00</sps:updateTime>
          <sps:taskingParameters>
            <sps:ParameterData>
              <sps:encoding>
                <swe:TextEncoding tokenSeparator="," blockSeparator="@@"/>
              </sps:encoding>
              <sps:values>2010-08-20T12:37:00+02:00,2010-08-
20T14:30:00+02:00,Y,pointToLookAt,51.902112,8.192728,0,Y,3.5</sps:values>
            </sps:ParameterData>
          </sps:taskingParameters>
          <sps:reservationExpiration>2010-08-
20T11:30:00+02:00</sps:reservationExpiration>
        </sps:ReservationReport>
      </sps:status>
      <sps:status>
        <sps:StatusReport>
          <sps:task>http://www.ogc.org/procedure/camera/1/tasks/6</sps:task>
          <sps:event>TaskConfirmed</sps:event>
          <sps:percentCompletion>0</sps:percentCompletion>
          <sps:procedure>http://www.ogc.org/procedure/camera/1</sps:procedure>
          <sps:requestStatus>Accepted</sps:requestStatus>
          <sps:taskStatus>InExecution</sps:taskStatus>
          <sps:updateTime>2010-08-20T11:23:08+02:00</sps:updateTime>
        </sps:StatusReport>
      </sps:status>
    </sps:Task>
  </sps:task>
</sps:GetTaskResponse>
```

### 9.6.5.5 Cancelling a Scheduled Task

2010-08-20T11:23:00+02:00 – The client made up his mind and sends a Cancel request to the service as it does no longer want the task to be executed/reserved.

**Listing 28 - Cancel request example**

```
<sps:Cancel service="SPS" version="2.0.0" xmlns:sps="http://www.opengis.net/sps/2.0"
xmlns:swe="http://www.opengis.net/swe/2.0" xmlns:xsi="http://www.w3.org/2001/XMLSchema-
instance">
  <sps:task>http://www.ogc.org/procedure/camera/1/tasks/6</sps:task>
</sps:Cancel>
```

2010-08-20T11:23:08+02:00 – The service sends a response indicating that the (reserved) task was cancelled.

**Listing 29 - Cancel response example**

```
<sps:CancelResponse xmlns:sps="http://www.opengis.net/sps/2.0"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
  <sps:result>
    <sps:StatusReport>
      <sps:task>http://www.ogc.org/procedure/camera/1/tasks/6</sps:task>
      <sps:procedure>http://www.ogc.org/procedure/camera/1</sps:procedure>
      <sps:requestStatus>Accepted</sps:requestStatus>
      <sps:updateTime>2010-08-20T11:23:08+02:00</sps:updateTime>
    </sps:StatusReport>
  </sps:result>
</sps:CancelResponse>
```

#### 9.6.5.6 Task Failure

2010-08-20T11:29:00+02:00 - Before the task expires, the client sends a GetStatus request to the service. The "since" parameter, although supported by the service, is not used in the request. Thus the current status of the task is requested.

The request is essentially the same as the one shown in Listing 22

2010-08-20T11:29:01+02:00 – The service sends a response with information about the current status of the task, indicating that the reservation failed (at 2010-08-20T11:28:30+02:00).

**Listing 30 - GetStatus response example for failed task**

```
<sps:GetStatusResponse xmlns:sps="http://www.opengis.net/sps/2.0"
xmlns:swe="http://www.opengis.net/swe/2.0" xmlns:swes="http://www.opengis.net/swes/2.0"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
  <sps:status>
    <sps:StatusReport>
      <sps:task>http://www.ogc.org/procedure/camera/1/tasks/6</sps:task>
      <sps:event>TaskFailed</sps:event>
      <sps:procedure>http://www.ogc.org/procedure/camera/1</sps:procedure>
      <sps:requestStatus>Accepted</sps:requestStatus>
      <sps:statusMessage xml:lang="en">Your reservation failed because an emergency
tasking action required use of the resources that were reserved for your
task.</sps:statusMessage>
      <sps:taskStatus>Failed</sps:taskStatus>
      <sps:updateTime>2010-08-20T11:28:30+02:00</sps:updateTime>
    </sps:StatusReport>
  </sps:status>
</sps:GetStatusResponse>
```

#### 9.6.5.7 Updating a Scheduled Task

Assuming that the task is now in execution the client can update it.

2010-08-20T12:40:00+02:00 – The client sends an Update request to the service, requesting that the camera be moved 10 degrees left. The latest response time is set to 2010-08-20T12:41:00+02:00.

**Listing 31 - Update request example**

```
<sps:Update service="SPS" version="2.0.0" xmlns:sps="http://www.opengis.net/sps/2.0"
xmlns:swe="http://www.opengis.net/swe/2.0" xmlns:xsi="http://www.w3.org/2001/XMLSchema-
instance">
  <sps:procedure>http://www.ogc.org/procedure/camera/1</sps:procedure>
  <sps:taskingParameters>
    <sps:ParameterData>
      <sps:encoding>
        <swe:TextEncoding tokenSeparator="," blockSeparator="@@"/>
      </sps:encoding>
      <sps:values>Y,relativePositioning,Y,-10,N,N</sps:values>
    </sps:ParameterData>
  </sps:taskingParameters>
  <sps:latestResponseTime>2010-08-20T12:41:00+02:00</sps:latestResponseTime>
  <sps:targetTask>http://www.ogc.org/procedure/camera/1/tasks/6</sps:targetTask>
</sps:Update>
```

Note: the tasking parameters available for update are a subset of the parameters described in the DescribeTaskingResponse - all parameters that have attribute updatable=false are not used in an update request. The description of the tasking parameter relevant for an Update request is shown in the following listing.

**Listing 32 - DataRecord example with tasking parameter description relevant for Update request**

```
<swe:DataRecord xmlns:gml="http://www.opengis.net/gml/3.2"
xmlns:swe="http://www.opengis.net/swe/2.0" xmlns:xlink="http://www.w3.org/1999/xlink"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
  <swe:field name="positioningChoice">
    <swe:DataChoice optional="true">
      <swe:item name="pointToLookAt">
        <swe:Vector definition="http://www.opengis.net/def/property/OGC-SPS-X-
CAM/0/PointToLookAt" referenceFrame="http://www.opengis.net/def/crs/EPSG/0/4979">
          <swe:label>Look Pointer</swe:label>
          <swe:description>3D location where the camera should look at</swe:description>
          <swe:coordinate name="lat">
            <swe:Quantity
definition="http://sweet.jpl.nasa.gov/2.0/spaceCoordinates.owl#Latitude" axisID="Lat">
              <swe:label>Geodetic latitude</swe:label>
              <swe:uom xlink:href="deg"/>
            </swe:Quantity>
          </swe:coordinate>
          <swe:coordinate name="long">
            <swe:Quantity
definition="http://sweet.jpl.nasa.gov/2.0/spaceCoordinates.owl#Longitude" axisID="Long">
              <swe:label>Geodetic longitude</swe:label>
              <swe:uom code="deg"/>
            </swe:Quantity>
          </swe:coordinate>
          <swe:coordinate name="h">
            <swe:Quantity
definition="http://sweet.jpl.nasa.gov/2.0/spaceCoordinates.owl#Vertical" axisID="h">
              <swe:label>Ellipsoidal height</swe:label>
              <swe:uom code="m"/>
              <swe:value>0</swe:value>
            </swe:Quantity>
          </swe:coordinate>
        </swe:Vector>
      </swe:item>
      <swe:item name="relativePositioning">
        <swe:DataRecord definition="http://www.opengis.net/def/property/OGC-SPS-X-
CAM/0/RelativePan">
          <swe:label>Relative Positioning</swe:label>
          <swe:description>Camera movement relative to the current
position</swe:description>
          <swe:field name="relativeHorizontalPan">
            <swe:Quantity definition="http://www.opengis.net/def/property/OGC-SPS-X-
CAM/0/RelativeHorizontalPan" optional="true">
              <swe:uom code="deg"/>
              <swe:constraint>
```

```
                <swe:AllowedValues>
                  <swe:interval>-180 180</swe:interval>
                </swe:AllowedValues>
              </swe:constraint>
            </swe:Quantity>
          </swe:field>
          <swe:field name="relativeVerticalPan">
            <swe:Quantity definition="http://www.opengis.net/def/property/OGC-SPS-X-
CAM/0/RelativeVerticalPan" optional="true">
              <swe:uom code="deg"/>
              <swe:constraint>
                <swe:AllowedValues>
                  <swe:interval>-90 90</swe:interval>
                </swe:AllowedValues>
              </swe:constraint>
            </swe:Quantity>
          </swe:field>
        </swe:DataRecord>
      </swe:item>
    </swe:DataChoice>
  </swe:field>
  <swe:field name="focalLength">
    <swe:Quantity definition="http://www.opengis.net/def/property/OGC-SPS-X-
CAM/0/FocalLength" optional="true">
      <swe:label>Focal length</swe:label>
      <swe:description>Focal length of the camera. Controls the camera's zoom
level.</swe:description>
      <swe:uom code="mm"/>
      <swe:constraint>
        <swe:AllowedValues>
          <swe:interval>3.5 10</swe:interval>
        </swe:AllowedValues>
      </swe:constraint>
    </swe:Quantity>
  </swe:field>
</swe:DataRecord>
```

2010-08-20T12:40:01+02:00 – The service sends a response indicating that the final decision on the update is pending. The service confirms that the latest response time is 2010-08-20T12:41:00+02:00. Note that the response has a different task identifier than the one used in the request as the response informs about the status of the update request itself, not of the task that was the target of the request. This is necessary to get information about pending update requests via the GetStatus operation as we will see in the following.

**Listing 33 - Update response example indicating request is pending**

```
<sps:UpdateResponse xmlns:sps="http://www.opengis.net/sps/2.0"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns:swe="http://www.opengis.net/swe/2.0">
  <sps:latestResponseTime>2010-08-20T12:41:00+02:00</sps:latestResponseTime>
  <sps:result>
    <sps:StatusReport>
      <sps:task>http://www.ogc.org/procedure/camera/1/tasks/6/updates/1</sps:task>
      <sps:procedure>http://www.ogc.org/procedure/camera/1</sps:procedure>
      <sps:requestStatus>Pending</sps:requestStatus>
      <sps:updateTime>2010-08-20T12:40:01+02:00</sps:updateTime>
    </sps:StatusReport>
  </sps:result>
  <sps:targetTask>http://www.ogc.org/procedure/camera/1/tasks/6</sps:targetTask>
</sps:UpdateResponse>
```

2010-08-20T12:41:01+02:00 – The client sends a GetStatus request to learn what the final decision for the update request was (this request can of course also be sent before the latest response time).

**Listing 34 - GetStatus response example targetting update request**

```
<sps:GetStatus service="SPS" version="2.0.0" xmlns:sps="http://www.opengis.net/sps/2.0"
xmlns:swe="http://www.opengis.net/swe/2.0" xmlns:xsi="http://www.w3.org/2001/XMLSchema-
instance">
  <sps:task>http://www.ogc.org/procedure/camera/1/tasks/6/updates/1</sps:task>
</sps:GetStatus>
```

### 9.6.5.8 Usage of LatestResponseTime

Now there are two options: either the service did or did not provide the final response on the update request before the latest response time.

### 9.6.5.8.1 Final Response Not Provided Before Latest Response Time

2010-08-20T12:41:02+02:00 – The service sends a response indicating that the update request automatically expired and therefore was (automatically) rejected.

**Listing 35 - GetStatus response indicating pending update request expired and was rejected**

```
<sps:GetStatusResponse xmlns:sps="http://www.opengis.net/sps/2.0"
xmlns:swe="http://www.opengis.net/swe/2.0" xmlns:swes="http://www.opengis.net/swes/2.0"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
  <sps:status>
    <sps:StatusReport>
      <sps:task>http://www.ogc.org/procedure/camera/1/tasks/6/updates/1</sps:task>
      <sps:event>TaskingRequestExpired</sps:event>
      <sps:procedure>http://www.ogc.org/procedure/camera/1</sps:procedure>
      <sps:requestStatus>Rejected</sps:requestStatus>
      <sps:updateTime>2010-08-20T12:41:00+02:00</sps:updateTime>
      <sps:taskingParameters>
        <sps:ParameterData>
          <sps:encoding>
            <swe:TextEncoding tokenSeparator="," blockSeparator="@@"/>
          </sps:encoding>
          <sps:values>Y,relativePositioning,Y,-10,N,N</sps:values>
        </sps:ParameterData>
      </sps:taskingParameters>
    </sps:StatusReport>
  </sps:status>
</sps:GetStatusResponse>
```

### 9.6.5.8.2 Final Response is Provided Before Latest Response Time

2010-08-20T12:41:02+02:00 – The service sends a response indicating that the update was accepted and performed as planned.

**Listing 36 - GetStatus response indicating pending update request was accepted**

```
<sps:GetStatusResponse xmlns:sps="http://www.opengis.net/sps/2.0"
xmlns:swe="http://www.opengis.net/swe/2.0" xmlns:swes="http://www.opengis.net/swes/2.0"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
  <sps:status>
    <sps:StatusReport>
      <sps:task>http://www.ogc.org/procedure/camera/1/tasks/6/updates/1</sps:task>
      <sps:procedure>http://www.ogc.org/procedure/camera/1</sps:procedure>
      <sps:requestStatus>Accepted</sps:requestStatus>
      <sps:updateTime>2010-08-20T12:40:50+02:00</sps:updateTime>
      <sps:taskingParameters>
        <sps:ParameterData>
          <sps:encoding>
            <swe:TextEncoding tokenSeparator="," blockSeparator="@@"/>
          </sps:encoding>
          <sps:values>Y,relativePositioning,Y,-10,N,N</sps:values>
        </sps:ParameterData>
      </sps:taskingParameters>
    </sps:StatusReport>
  </sps:status>
</sps:GetStatusResponse>
```

### 9.6.5.9 Task Completion

2010-08-20T14:35:00+02:00 – The client sends a GetTask request to get a complete description of the task the service performed for him.

The request is essentially the same as the one shown in Listing 26.

2010-08-20T14:35:01+02:00 – The service sends a response that includes the full state history of the task (as the service supports state logging).

Note: this example assumes that the task was submitted, not reserved first; intermediate data publication is shown as well a task update.

**Listing 37 - GetTask response for completed task**

```
<sps:GetTaskResponse xmlns:sps="http://www.opengis.net/sps/2.0"
xmlns:swe="http://www.opengis.net/swe/2.0" xmlns:swes="http://www.opengis.net/swes/2.0"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
  <sps:task>
    <sps:Task>
      <swes:identifier>http://www.ogc.org/procedure/camera/1/tasks/6</swes:identifier>
      <sps:status>
        <sps:StatusReport>
          <sps:task>http://www.ogc.org/procedure/camera/1/tasks/6</sps:task>
          <sps:event>TaskSubmitted</sps:event>
          <sps:percentCompletion>0</sps:percentCompletion>
          <sps:procedure>http://www.ogc.org/procedure/camera/1</sps:procedure>
          <sps:requestStatus>Accepted</sps:requestStatus>
          <sps:taskStatus>InExecution</sps:taskStatus>
          <sps:updateTime>2010-08-20T11:12:04+02:00</sps:updateTime>
          <sps:taskingParameters>
            <sps:ParameterData>
              <sps:encoding>
                <swe:TextEncoding tokenSeparator="," blockSeparator="@@"/>
              </sps:encoding>
              <sps:values>2010-08-20T12:37:00+02:00,2010-08-
20T14:30:00+02:00,Y,pointToLookAt,51.902112,8.192728,0,Y,3.5</sps:values>
            </sps:ParameterData>
          </sps:taskingParameters>
        </sps:StatusReport>
      </sps:status>
      <sps:status>
        <sps:StatusReport>
```

```
          <sps:task>http://www.ogc.org/procedure/camera/1/tasks/6</sps:task>
          <sps:event>DataPublished</sps:event>
          <sps:procedure>http://www.ogc.org/procedure/camera/1</sps:procedure>
          <sps:requestStatus>Accepted</sps:requestStatus>
          <sps:taskStatus>InExecution</sps:taskStatus>
          <sps:updateTime>2010-08-20T12:37:00.001+02:00</sps:updateTime>
        </sps:StatusReport>
      </sps:status>
      <sps:status>
        <sps:StatusReport>
          <sps:task>http://www.ogc.org/procedure/camera/1/tasks/6</sps:task>
          <sps:event>TaskUpdated</sps:event>
          <sps:procedure>http://www.ogc.org/procedure/camera/1</sps:procedure>
          <sps:requestStatus>Accepted</sps:requestStatus>
          <sps:taskStatus>InExecution</sps:taskStatus>
          <sps:updateTime>2010-08-20T12:40:50+02:00</sps:updateTime>
          <sps:taskingParameters>
            <sps:ParameterData>
              <sps:encoding>
                <swe:TextEncoding tokenSeparator="," blockSeparator="@@"/>
              </sps:encoding>
              <sps:values>Y,relativePositioning,Y,-10,N,N</sps:values>
            </sps:ParameterData>
          </sps:taskingParameters>
        </sps:StatusReport>
      </sps:status>
      <sps:status>
        <sps:StatusReport>
          <sps:task>http://www.ogc.org/procedure/camera/1/tasks/6</sps:task>
          <sps:event>TaskCompleted</sps:event>
          <sps:percentCompletion>100</sps:percentCompletion>
          <sps:procedure>http://www.ogc.org/procedure/camera/1</sps:procedure>
          <sps:requestStatus>Accepted</sps:requestStatus>
          <sps:taskStatus>Completed</sps:taskStatus>
          <sps:updateTime>2010-08-20T14:30:00+02:00</sps:updateTime>
        </sps:StatusReport>
      </sps:status>
    </sps:Task>
  </sps:task>
</sps:GetTaskResponse>
```

A GetStatus request with since parameter - supported by the service in this scenario - can yield a similar result but clients can also retrieve only those parts of the state log for a task that they do not already know. Let us assume that the client already performed a GetStatus request at 2010-08-20T12:37:05+02:00.

2010-08-20T14:00:00.00+02:00 – The client sends a GetStatus request with "since" parameter to the service, set to the value 2010-08-20T12:37:05+02:00.

**Listing 38 - GetStatus request example with since parameter**

```
<sps:GetStatus service="SPS" version="2.0.0" xmlns:sps="http://www.opengis.net/sps/2.0"
xmlns:swe="http://www.opengis.net/swe/2.0" xmlns:xsi="http://www.w3.org/2001/XMLSchema-
instance">
  <sps:task>http://www.ogc.org/procedure/camera/1/tasks/6</sps:task>
  <sps:since>2010-08-20T12:37:05+02:00</sps:since>
</sps:GetStatus>
```

2010-08-20T14:00:00.01+02:00 – The service sends a response providing information about the last two state transitions only.

**Listing 39 - GetStatus response example for request with since parameter**

```
<sps:GetStatusResponse xmlns:sps="http://www.opengis.net/sps/2.0"
xmlns:swe="http://www.opengis.net/swe/2.0" xmlns:swes="http://www.opengis.net/swes/2.0"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
      <sps:status>
        <sps:StatusReport>
          <sps:task>http://www.ogc.org/procedure/camera/1/tasks/6</sps:task>
          <sps:event>TaskUpdated</sps:event>
          <sps:procedure>http://www.ogc.org/procedure/camera/1</sps:procedure>
          <sps:requestStatus>Accepted</sps:requestStatus>
          <sps:taskStatus>InExecution</sps:taskStatus>
          <sps:updateTime>2010-08-20T12:40:50+02:00</sps:updateTime>
          <sps:taskingParameters>
            <sps:ParameterData>
              <sps:encoding>
                <swe:TextEncoding tokenSeparator="," blockSeparator="@@"/>
              </sps:encoding>
              <sps:values>Y,relativePositioning,Y,-10,N,N</sps:values>
            </sps:ParameterData>
          </sps:taskingParameters>
        </sps:StatusReport>
      </sps:status>
      <sps:status>
        <sps:StatusReport>
          <sps:task>http://www.ogc.org/procedure/camera/1/tasks/6</sps:task>
          <sps:event>TaskCompleted</sps:event>
          <sps:percentCompletion>100</sps:percentCompletion>
          <sps:procedure>http://www.ogc.org/procedure/camera/1</sps:procedure>
          <sps:requestStatus>Accepted</sps:requestStatus>
          <sps:taskStatus>Completed</sps:taskStatus>
          <sps:updateTime>2010-08-20T14:30:00+02:00</sps:updateTime>
        </sps:StatusReport>
      </sps:status>
</sps:GetStatusResponse>
```

### 9.6.6 Getting Result Access Information for a Task

2010-08-20T14:36:00+02:00 – The client sends a DescribeResultAccess request to get references to data (services) for the task.

**Listing 40 - DescribeResultAccess request example targetting a task**

```
<soap12:Envelope xmlns:soap12="http://www.w3.org/2003/05/soap-envelope"
xmlns:sps="http://www.opengis.net/sps/2.0"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns:ows="http://www.opengis.net/ows/1.1">
  <soap12:Body>
    <sps:DescribeResultAccess service="SPS" version="2.0.0">
      <sps:target>
        <sps:task>http://www.ogc.org/procedure/camera/1/tasks/6</sps:task>
      </sps:target>
    </sps:DescribeResultAccess>
  </soap12:Body>
</soap12:Envelope>
```

2010-08-20T14:36:01+02:00 - The service sends a response providing the requested information for accessing the data gathered for the task.

**Listing 41 - DescribeResultAccess response example with access information for a task**

```
<soap12:Envelope xmlns:soap12="http://www.w3.org/2003/05/soap-envelope"
xmlns:sps="http://www.opengis.net/sps/2.0"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns:swe="http://www.opengis.net/swe/2.0" xmlns:ows="http://www.opengis.net/ows/1.1"
xmlns:sos="http://www.opengis.net/sos/2.0" xmlns:fes="http://www.opengis.net/fes/2.0"
xmlns:gml="http://www.opengis.net/gml/3.2" xmlns:xlink="http://www.w3.org/1999/xlink">
  <soap12:Body>
```

```
    <sps:DescribeResultAccessResponse>
      <sps:availability>
        <sps:available>
          <sps:DataAvailable>
            <sps:dataReference>
              <ows:ReferenceGroup>

<ows:Identifier>http://www.ogc.org/procedure/camera/1/tasks/6/accessReferenceGroups/1</ow
s:Identifier>
                <ows:ServiceReference xlink:href="http://www.ogc.org/SOS"
xlink:role="http://www.opengis.net/spec/SPS/2.0/referenceType/FullServiceAccess">

<ows:Identifier>http://www.ogc.org/procedure/camera/1/accessReferenceGroups/1/references/
1</ows:Identifier>
                  <ows:Format>application/xml</ows:Format>
                  <ows:Metadata>
                    <sps:SPSMetadata>

<sps:dataAccessType>http://www.opengis.net/sos/2.0/GetObservation</sps:dataAccessType>
                    </sps:SPSMetadata>
                  </ows:Metadata>
                  <ows:RequestMessage>
                    <soap12:Envelope>
                      <soap12:Body>
                        <sos:GetObservation service="SOS" version="2.0.0">

<sos:observedProperty>http://www.opengis.net/def/propertyType/x-
radiance</sos:observedProperty>

<sos:procedure>http://www.ogc.org/procedure/camera/1</sos:procedure>
                          <sos:temporalFilter>
                            <fes:During>
                              <fes:ValueReference>phenomenonTime</fes:ValueReference>
                              <gml:TimePeriod gml:id="gid01">
                                <gml:beginPosition>2010-08-
20T12:37:00+02:00</gml:beginPosition>
                                <gml:endPosition>2010-08-
20T14:30:00+02:00</gml:endPosition>
                              </gml:TimePeriod>
                            </fes:During>
                          </sos:temporalFilter>
                        </sos:GetObservation>
                      </soap12:Body>
                    </soap12:Envelope>
                  </ows:RequestMessage>
                </ows:ServiceReference>
              </ows:ReferenceGroup>
            </sps:dataReference>
            <sps:dataReference>
              <ows:ReferenceGroup>

<ows:Identifier>http://www.ogc.org/procedure/camera/1/tasks/6/accessReferenceGroups/2</ow
s:Identifier>
                <ows:Reference
xlink:href="http://www.ogc.org/procedure/camera/1/videos/task_6.mj2"
xlink:role="http://www.opengis.net/spec/SPS/2.0/referenceType/Resource">

<ows:Identifier>http://www.ogc.org/procedure/camera/1/tasks/6/accessReferenceGroups/2/ref
erences/1</ows:Identifier>
                  <ows:Format>video/mj2</ows:Format>
                </ows:Reference>
              </ows:ReferenceGroup>
            </sps:dataReference>
          </sps:DataAvailable>
        </sps:available>
      </sps:availability>
    </sps:DescribeResultAccessResponse>
  </soap12:Body>
</soap12:Envelope>
```

The example response shows that the client may (need to) modify the given request; for example credentials or WS-Addressing header-information may need to be added.

### 9.6.7 Service Exceptions

At some point in time after the required provision time for status information of a task / tasking request a client might request status information for it via the GetStatus / GetTask operation. If the service then already removed this information, it will return a *StatusInformationExpired* exception.

**Listing 42 – StatusInformationExpired exception example**

```
<soap12:Envelope xmlns:soap12="http://www.w3.org/2003/05/soap-envelope"
xmlns:sps="http://www.opengis.net/sps/2.0"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns:ows="http://www.opengis.net/ows/1.1">
  <soap12:Body>
    <soap12:Fault>
      <soap12:Code>
        <soap12:Value>soap12:Receiver</soap12:Value>
        <soap12:Subcode>
          <soap12:Value>sps:StatusInformationExpired</soap12:Value>
        </soap12:Subcode>
      </soap12:Code>
      <soap12:Reason>
        <soap12:Text xml:lang="en">The status information for the requested task has
already expired.</soap12:Text>
      </soap12:Reason>
      <soap12:Detail>
        <ows:Exception exceptionCode="StatusInformationExpired">
          <ows:ExceptionText>The service has removed all status information for the given
task / tasking request (the required provision time has already
passed).</ows:ExceptionText>
        </ows:Exception>
      </soap12:Detail>
    </soap12:Fault>
  </soap12:Body>
</soap12:Envelope>
```

In case that the client sent a request to the service that is not valid according to its XML Schema definition, the service returns an InvalidRequest exception.

**Listing 43 - InvalidRequest exception example**

```
<soap12:Envelope xmlns:soap12="http://www.w3.org/2003/05/soap-envelope"
xmlns:sps="http://www.opengis.net/sps/2.0"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns:ows="http://www.opengis.net/ows/1.1" xmlns:swes="http://www.opengis.net/swes/2.0">
  <soap12:Body>
    <soap12:Fault>
      <soap12:Code>
        <soap12:Value>soap12:Sender</soap12:Value>
        <soap12:Subcode>
          <soap12:Value>swes:InvalidRequest</soap12:Value>
        </soap12:Subcode>
      </soap12:Code>
      <soap12:Reason>
        <soap12:Text xml:lang="en">The request did not conform to its XML Schema
definition.</soap12:Text>
      </soap12:Reason>
      <soap12:Detail>
        <ows:Exception exceptionCode="InvalidRequest" locator="element sps:extension is
not expected after element sps:GetStatus/sps:task"/>
      </soap12:Detail>
    </soap12:Fault>
  </soap12:Body>
</soap12:Envelope>
```

If the client sent a GetStatus request with a task identifier that is unknown to the service then the service returns an *InvalidParameterValue* exception like the following:

**Listing 44 - InvalidParameterValue exception example**

```
<soap12:Envelope xmlns:soap12="http://www.w3.org/2003/05/soap-envelope"
 xmlns:sps="http://www.opengis.net/sps/2.0" xmlns:xsi="http://www.w3.org/2001/XMLSchema-
instance" xmlns:ows="http://www.opengis.net/ows/1.1">
  <soap12:Body>
    <soap12:Fault>
      <soap12:Code>
        <soap12:Value>soap12:Sender</soap12:Value>
        <soap12:Subcode>
          <soap12:Value>ows:InvalidParameterValue</soap12:Value>
        </soap12:Subcode>
      </soap12:Code>
      <soap12:Reason>
        <soap12:Text xml:lang="en">The request contained an invalid parameter
value.</soap12:Text>
      </soap12:Reason>
      <soap12:Detail>
        <ows:Exception exceptionCode="InvalidParameterValue" locator="task">
          <ows:ExceptionText>The requested task / tasking request is unknown to the
service.</ows:ExceptionText>
        </ows:Exception>
      </soap12:Detail>
    </soap12:Fault>
  </soap12:Body>
</soap12:Envelope>
```

### 9.6.8 Notifications

As the service realizes publish / subscribe functionality, the client may subscribe for notifications published by the service. The following examples are about notifications published for the submitted task and an according subscription.

2010-08-20T11:12:05+02:00 - Right after it received the SubmitResponse telling him that the service accepted its tasking request, the client subscribes to notifications for the task that is in execution. It sends the according Subscribe request to the producer endpoint stated by the SPS in the notifications section of its Capabilities document.

**Listing 45 - Subscribe request example**

```
<soap12:Envelope xmlns:soap12="http://www.w3.org/2003/05/soap-envelope"
xmlns:sps="http://www.opengis.net/sps/2.0"
xmlns:wsa="http://www.w3.org/2005/08/addressing"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns:wsn-b="http://docs.oasis-open.org/wsn/b-2">
  <soap12:Body>
    <wsn-b:Subscribe>
      <wsn-b:ConsumerReference>
        <wsa:Address>http://my.client.com/client/myNotificationConsumer</wsa:Address>
      </wsn-b:ConsumerReference>
      <wsn-b:Filter>
        <wsn-b:TopicExpression Dialect="http://www.w3.org/TR/1999/REC-xpath-
19991116">//sps:TaskEvent/*[@wstop:topic='true']</wsn-b:TopicExpression>
        <wsn-b:MessageContent Dialect="http://www.w3.org/TR/1999/REC-xpath-
19991116">boolean(//*[sps:task = 'http://www.ogc.org/procedure/camera/1/tasks/6'])</wsn-
b:MessageContent>
      </wsn-b:Filter>
    </wsn-b:Subscribe>
  </soap12:Body>
</soap12:Envelope>
```

2010-08-20T11:12:06+02:00 - The service sends a response indicating that the subscription will last until 2010-08-20T14:31:00+02:00.

This time is shortly after the requested task end time. Note, however, that in WS-Notification the choice of the actual termination time depends upon the actual service implementation if no specific time was requested by the client. Although the way the default choice for termination time of a task as shown in this example is a useful pattern, the SPS specification does not state requirements concerning the duration of a task or the termination time of subscriptions that may target notifications published for it. Such requirements could be defined in an SPS extension.

**Listing 46 - Subscribe response example**

```
<soap12:Envelope xmlns:soap12="http://www.w3.org/2003/05/soap-envelope"
xmlns:wsa="http://www.w3.org/2005/08/addressing"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns:wsn-b="http://docs.oasis-open.org/wsn/b-2">
  <soap12:Body>
    <wsn-b:SubscribeResponse>
      <wsn-b:SubscriptionReference>
        <wsa:Address>http://www.ogc.org/SPS/Producer/subscriptions/792</wsa:Address>
      </wsn-b:SubscriptionReference>
      <wsn-b:TerminationTime>2010-08-20T14:31:00+02:00</wsn-b:TerminationTime>
    </wsn-b:SubscribeResponse>
  </soap12:Body>
</soap12:Envelope>
```

Following the examples given so far, the service would have published the following notifications for the task the client targeted in its subscription: a notification for a TaskingRequestAccepted (task was submitted), DataPublished and for a TaskCompleted event

**Listing 47 – Example notification of TaskingRequestAccepted event published on TaskSubmission topic**

```
<soap12:Envelope xmlns:soap12="http://www.w3.org/2003/05/soap-envelope"
xsi:schemaLocation="http://www.w3.org/2003/05/soap-envelope
http://www.w3.org/2003/05/soap-envelope/soap-envelope.xsd
http://www.opengis.net/sps/2.0 http://schemas.opengis.net/sps/2.0/sps.xsd
http://docs.oasis-open.org/wsn/b-2 http://docs.oasis-open.org/wsn/b-2.xsd"
xmlns:sps="http://www.opengis.net/sps/2.0"
xmlns:wsa="http://www.w3.org/2005/08/addressing"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xmlns:wsn-b="http://docs.oasis-
open.org/wsn/b-2">
  <soap12:Header>
    <wsa:To>http://my.client.com/client/myNotificationConsumer</wsa:To>
    <wsa:Action>http://docs.oasis-open.org/wsn/bw-
2/NotificationConsumer/Notify</wsa:Action>
  </soap12:Header>
  <soap12:Body>
    <wsn-b:Notify>
      <wsn-b:NotificationMessage>
        <wsn-b:SubscriptionReference>
          <wsa:Address>http://www.ogc.org/SPS/Producer/subscriptions/792</wsa:Address>
        </wsn-b:SubscriptionReference>
        <wsn-b:Topic Dialect="http://docs.oasis-open.org/wsn/t-
1/TopicExpression/Concrete">sps:TaskEvent/TaskSubmission</wsn-b:Topic>
        <wsn-b:Message>
          <sps:StatusReport>
            <sps:task>http://www.ogc.org/procedure/camera/1/tasks/6</sps:task>
            <sps:event>TaskSubmitted</sps:event>
            <sps:percentCompletion>0</sps:percentCompletion>
            <sps:procedure>http://www.ogc.org/procedure/camera/1</sps:procedure>
            <sps:requestStatus>Accepted</sps:requestStatus>
            <sps:taskStatus>InExecution</sps:taskStatus>
            <sps:updateTime>2010-08-20T11:12:04+02:00</sps:updateTime>
          </sps:StatusReport>
        </wsn-b:Message>
      </wsn-b:NotificationMessage>
    </wsn-b:Notify>
  </soap12:Body>
</soap12:Envelope>
```

**Listing 48 – Example notification of DataPublished event**

```
<soap12:Envelope xmlns:soap12="http://www.w3.org/2003/05/soap-envelope"
xmlns:sps="http://www.opengis.net/sps/2.0"
xmlns:wsa="http://www.w3.org/2005/08/addressing"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns:ows="http://www.opengis.net/ows/1.1" xmlns:wsn-b="http://docs.oasis-open.org/wsn/b-
2">
  <soap12:Header>
    <wsa:To>http://my.client.com/client/myNotificationConsumer</wsa:To>
    <wsa:Action>http://docs.oasis-open.org/wsn/bw-
2/NotificationConsumer/Notify</wsa:Action>
  </soap12:Header>
  <soap12:Body>
    <wsn-b:Notify>
      <wsn-b:NotificationMessage>
        <wsn-b:SubscriptionReference>
          <wsa:Address>http://www.ogc.org/SPS/Producer/subscriptions/792</wsa:Address>
        </wsn-b:SubscriptionReference>
        <wsn-b:Topic Dialect="http://docs.oasis-open.org/wsn/t-
1/TopicExpression/Concrete">sps:TaskEvent/DataPublication</wsn-b:Topic>
        <wsn-b:Message>
          <sps:StatusReport>
            <sps:task>http://www.ogc.org/procedure/camera/1/tasks/6</sps:task>
            <sps:event>DataPublished</sps:event>
            <sps:procedure>http://www.ogc.org/procedure/camera/1</sps:procedure>
            <sps:requestStatus>Accepted</sps:requestStatus>
            <sps:taskStatus>InExecution</sps:taskStatus>
            <sps:updateTime>2010-08-20T12:37:00.001+02:00</sps:updateTime>
          </sps:StatusReport>
        </wsn-b:Message>
      </wsn-b:NotificationMessage>
    </wsn-b:Notify>
  </soap12:Body>
</soap12:Envelope>
```

**Listing 49 – Example notification of TaskCompleted event**

```
<soap12:Envelope xmlns:soap12="http://www.w3.org/2003/05/soap-envelope"
xmlns:sps="http://www.opengis.net/sps/2.0"
xmlns:wsa="http://www.w3.org/2005/08/addressing"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns:ows="http://www.opengis.net/ows/1.1" xmlns:wsn-b="http://docs.oasis-open.org/wsn/b-
2">
  <soap12:Header>
    <wsa:To>http://my.client.com/client/myNotificationConsumer</wsa:To>
    <wsa:Action>http://docs.oasis-open.org/wsn/bw-
2/NotificationConsumer/Notify</wsa:Action>
  </soap12:Header>
  <soap12:Body>
    <wsn-b:Notify>
      <wsn-b:NotificationMessage>
        <wsn-b:SubscriptionReference>
          <wsa:Address>http://www.ogc.org/SPS/Producer/subscriptions/792</wsa:Address>
        </wsn-b:SubscriptionReference>
        <wsn-b:Topic Dialect="http://docs.oasis-open.org/wsn/t-
1/TopicExpression/Concrete">sps:TaskEvent/TaskCompletion</wsn-b:Topic>
        <wsn-b:Message>
          <sps:StatusReport>
          <sps:task>http://www.ogc.org/procedure/camera/1/tasks/6</sps:task>
          <sps:event>TaskCompleted</sps:event>
          <sps:percentCompletion>100</sps:percentCompletion>
          <sps:procedure>http://www.ogc.org/procedure/camera/1</sps:procedure>
          <sps:requestStatus>Accepted</sps:requestStatus>
          <sps:taskStatus>Completed</sps:taskStatus>
          <sps:updateTime>2010-08-20T14:30:00+02:00</sps:updateTime>
        </sps:StatusReport>
        </wsn-b:Message>
      </wsn-b:NotificationMessage>
    </wsn-b:Notify>
  </soap12:Body>
</soap12:Envelope>
```

### 9.6.9 Using WS-Addressing

Usually the communication between client and SPS can be performed via SOAP without the addition of WS-Addressing header information. However, in some cases it is useful to leverage the functionality provided by WS-Addressing. This document is not the place to give a tutorial on WS-Addressing. However, the following listings provide some examples of SPS operation requests and responses (including a WS-Notification Subscribe invocation example) where WS-Addressing header information is added to the SOAP messages.

**Listing 50 – GetCapabilities example using WS-Addressing header information**

```
<soap12:Envelope xmlns:soap12="http://www.w3.org/2003/05/soap-envelope"
xmlns:sps="http://www.opengis.net/sps/2.0"
xmlns:wsa="http://www.w3.org/2005/08/addressing"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
  <soap12:Header>
    <wsa:To>http://www.ogc.org/SPS</wsa:To>
    <wsa:Action>http://www.opengis.net/sps/2.0/GetCapabilities</wsa:Action>
    <wsa:ReplyTo>
      <wsa:Address>http://my.client.com/client/myReceiver</wsa:Address>
    </wsa:ReplyTo>
    <wsa:MessageID>http://my.client.com/uid/msg-0010</wsa:MessageID>
  </soap12:Header>
  <soap12:Body>
    <sps:GetCapabilities/>
  </soap12:Body>
</soap12:Envelope>
```

The meaning of the header fields is as follows:

- wsa:To – address of the intended receiver of this message
- wsa:Action – uniquely identifies the semantics implied by this message; in this example it tells the service that the SPS GetCapabilities operation is invoked
- wsa:ReplyTo – the address of the endpoint where the response is expected to be sent to; in this example the response shall be sent asynchronously
- wsa:MessageID – a unique identifier for the message which is also used in the response sent by the service later on so that the client knows to which request an incoming – asynchronously sent – response refers to

The according reply would look like shown in the following listing.

**Listing 51 – Capabilities example using WS-Addressing header information**

```
<soap12:Envelope xmlns:soap12="http://www.w3.org/2003/05/soap-envelope"
xsi:schemaLocation="http://www.w3.org/2003/05/soap-envelope
http://www.w3.org/2003/05/soap-envelope/soap-envelope.xsd
http://www.opengis.net/sps/2.0 http://schemas.opengis.net/sps/2.0/sps.xsd
http://www.w3.org/2005/08/addressing http://www.w3.org/2005/08/addressing/ws-addr.xsd"
xmlns:sps="http://www.opengis.net/sps/2.0" xmlns:xlink="http://www.w3.org/1999/xlink"
xmlns:wsa="http://www.w3.org/2005/08/addressing"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns:ows="http://www.opengis.net/ows/1.1" xmlns:gml="http://www.opengis.net/gml/3.2"
xmlns:swes="http://www.opengis.net/swes/2.0" xmlns:wstop="http://docs.oasis-
open.org/wsn/t-1">
  <soap12:Header>
    <wsa:To>http://my.client.com/client/myReceiver</wsa:To>
    <wsa:Action>http://www.opengis.net/sps/2.0/GetCapabilitiesResponse</wsa:Action>
    <wsa:RelatesTo>http://my.client.com/uid/msg-0010</wsa:RelatesTo>
  </soap12:Header>
  <soap12:Body>
    <!-- like shown in Listing 11-->
  </soap12:Body>
</soap12:Envelope>
```

As we can see, the wsa:To has the value of the wsa:ReplyTo header field from the request shown in Listing 50 – same for the wsa:RelatesTo element which has the value of the wsa:MessageID from the request. The wsa:Action is now used to convey the information that the SOAP message contains the response to an SPS GetCapabilities invocation.

The Subscribe request as shown in Listing 45 can also be augmented with WS-Addressing header information.

**Listing 52 – Subscribe example using WS-Addressing header information**

```
<soap12:Envelope xmlns:soap12="http://www.w3.org/2003/05/soap-envelope"
xmlns:sps="http://www.opengis.net/sps/2.0"
xmlns:wsa="http://www.w3.org/2005/08/addressing"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns:ows="http://www.opengis.net/ows/1.1" xmlns:wsn-b="http://docs.oasis-open.org/wsn/b-
2">
  <soap12:Header>
    <wsa:To>http://www.ogc.org/SPS/Producer</wsa:To>
    <wsa:Action>http://docs.oasis-open.org/wsn/bw-
2/NotificationProducer/SubscribeRequest</wsa:Action>
    <wsa:ReplyTo>
      <wsa:Address>http://my.client.com/client/myReceiver</wsa:Address>
    </wsa:ReplyTo>
    <wsa:MessageID>http://my.client.com/uid/msg-Sub1</wsa:MessageID>
  </soap12:Header>
  <soap12:Body>
    <wsn-b:Subscribe>
      <wsn-b:ConsumerReference>
        <wsa:Address>http://my.client.com/client/myNotificationConsumer</wsa:Address>
      </wsn-b:ConsumerReference>
      <wsn-b:Filter>
        <!-- omitted for brevity -->
      </wsn-b:Filter>
    </wsn-b:Subscribe>
  </soap12:Body>
</soap12:Envelope>
```

Note that the wsa:ReplyTo in the header only defines where the response to the Subscribe request is to be sent to. The value of the wsn-b:ConsumerReference/wsa:Address element (in the soap12:Body) defines where the notifications of events matching the subscription are to be sent to. The following listing shows an example response for this request.

**Listing 53 – SubscribeResponse example using WS-Addressing header information**

```
<soap12:Envelope xmlns:soap12="http://www.w3.org/2003/05/soap-envelope"
xmlns:wsa="http://www.w3.org/2005/08/addressing"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xmlns:wsn-b="http://docs.oasis-
open.org/wsn/b-2">
  <soap12:Header>
    <wsa:To>http://my.client.com/client/myReceiver</wsa:To>
    <wsa:Action>http://docs.oasis-open.org/wsn/bw-
2/NotificationProducer/SubscribeResponse</wsa:Action>
    <wsa:RelatesTo>http://my.client.com/uid/msg-Sub1</wsa:RelatesTo>
  </soap12:Header>
  <soap12:Body>
    <!-- as shown in Listing 46-->
  </soap12:Body>
</soap12:Envelope>
```

The exceptions shown in Listing 42 to Listing 44 would be modified as shown in Listing 54 to Listing 56. Note that the wsa:Action is different in the following listings as the exceptions shown are defined by OWS Common, the SWE Service Model and the this standard.

**Listing 54 – StatusInformationExpired exception with WS-Addressing header information**

```
<soap12:Envelope xmlns:soap12="http://www.w3.org/2003/05/soap-envelope"
xmlns:sps="http://www.opengis.net/sps/2.0"
xmlns:wsa="http://www.w3.org/2005/08/addressing"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns:ows="http://www.opengis.net/ows/1.1">
  <soap12:Header>
    <wsa:To>http://my.client.com/client/myReceiver</wsa:To>
    <wsa:Action>http://www.opengis.net/sps/2.0/Exception</wsa:Action>
    <wsa:RelatesTo>http://my.client.com/uid/msg-0040</wsa:RelatesTo>
  </soap12:Header>
  <soap12:Body>
    <soap12:Fault>
      <soap12:Code>
        <soap12:Value>soap12:Receiver</soap12:Value>
        <soap12:Subcode>
          <soap12:Value>sps:StatusInformationExpired</soap12:Value>
        </soap12:Subcode>
      </soap12:Code>
      <!-- rest as shown in Listing 42-->
    </soap12:Fault>
  </soap12:Body>
</soap12:Envelope>
```

**Listing 55 – InvalidRequest exception with WS-Addressing header information**

```
<soap12:Envelope xmlns:soap12="http://www.w3.org/2003/05/soap-envelope"
xmlns:sps="http://www.opengis.net/sps/2.0"
xmlns:wsa="http://www.w3.org/2005/08/addressing"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns:ows="http://www.opengis.net/ows/1.1" xmlns:swes="http://www.opengis.net/swes/2.0">
  <soap12:Header>
    <wsa:To>http://my.client.com/client/myReceiver</wsa:To>
    <wsa:Action>http://www.opengis.net/swes/2.0/Exception</wsa:Action>
    <wsa:RelatesTo>http://my.client.com/uid/msg-0030</wsa:RelatesTo>
  </soap12:Header>
  <soap12:Body>
    <soap12:Fault>
      <soap12:Code>
        <soap12:Value>soap12:Sender</soap12:Value>
        <soap12:Subcode>
          <soap12:Value>swes:InvalidRequest</soap12:Value>
        </soap12:Subcode>
      </soap12:Code>
      <!-- rest as shown in Listing 43-->
  </soap12:Body>
</soap12:Envelope>
```

**Listing 56 – InvalidParameterValue exception with WS-Addressing header information**

```
<soap12:Envelope xmlns:soap12="http://www.w3.org/2003/05/soap-envelope"
xmlns:sps="http://www.opengis.net/sps/2.0"
xmlns:wsa="http://www.w3.org/2005/08/addressing"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns:ows="http://www.opengis.net/ows/1.1">
  <soap12:Header>
    <wsa:To>http://my.client.com/client/myReceiver</wsa:To>
    <wsa:Action>http://www.opengis.net/ows/1.1/Exception</wsa:Action>
    <wsa:RelatesTo>http://my.client.com/uid/msg-0020</wsa:RelatesTo>
  </soap12:Header>
  <soap12:Body>
    <soap12:Fault>
      <soap12:Code>
        <soap12:Value>soap12:Sender</soap12:Value>
        <soap12:Subcode>
          <soap12:Value>ows:InvalidParameterValue</soap12:Value>
        </soap12:Subcode>
      </soap12:Code>
      <!-- rest as shown in Listing 44-->
  </soap12:Body>
</soap12:Envelope>
```

# 10 SPS Task/Tasking Request State Machine Documentation

| Requirement | |
|---|---|
| http://www.opengis.net/spec/SPS/2.0/req/Behaviour | |
| REQ 115. | Any SPS shall implement a behavior for handling *tasks* and *tasking requests* as defined by the state machines described in this clause 10. |

Each state machine is documented with diagrams representing the state machine, followed by a documentation of the states that are part of the state machine. For each state, the incoming and outgoing connections are documented.

Finally, all triggers and the event that causes the activation of a trigger are documented. A trigger may have a specific effect, which in the case of SPS is to notify interested clients about the event (*/Notify*). A service that implements according notification functionality – for SPS per default via a publish/subscribe interface – can inform clients about these events; see clause 8 for further details.

## 10.1 Task State Machine

### 10.1.1 Diagrams

The following two diagrams define the state machine of an SPS task.

NOTE: Figure 32 is the same as Figure 9 and Figure 33 is only another representation of the state machine shown in the two previous diagrams – so all three diagrams represent the same state machine.

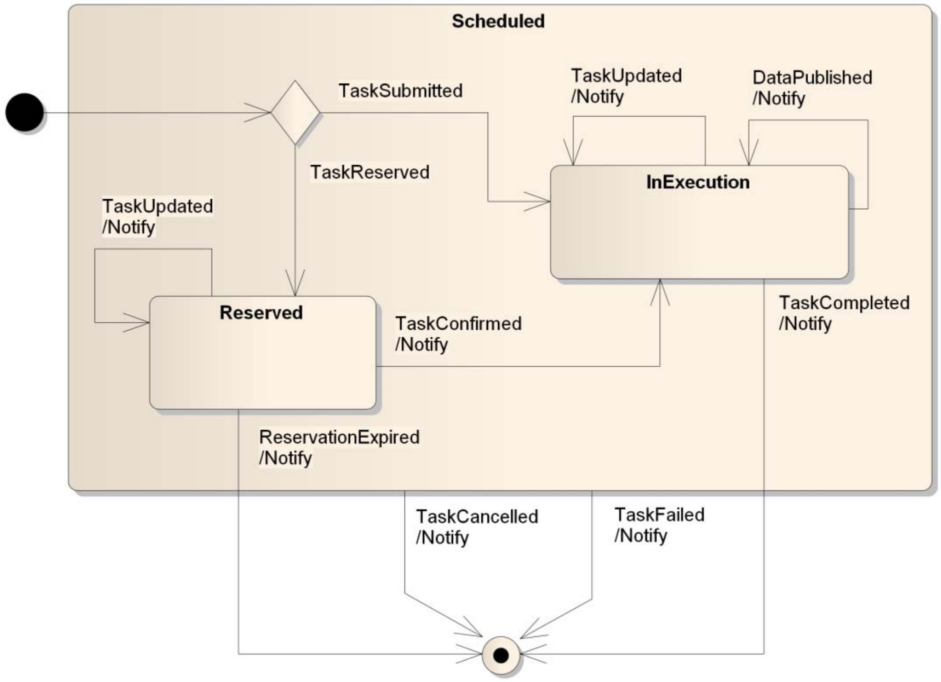
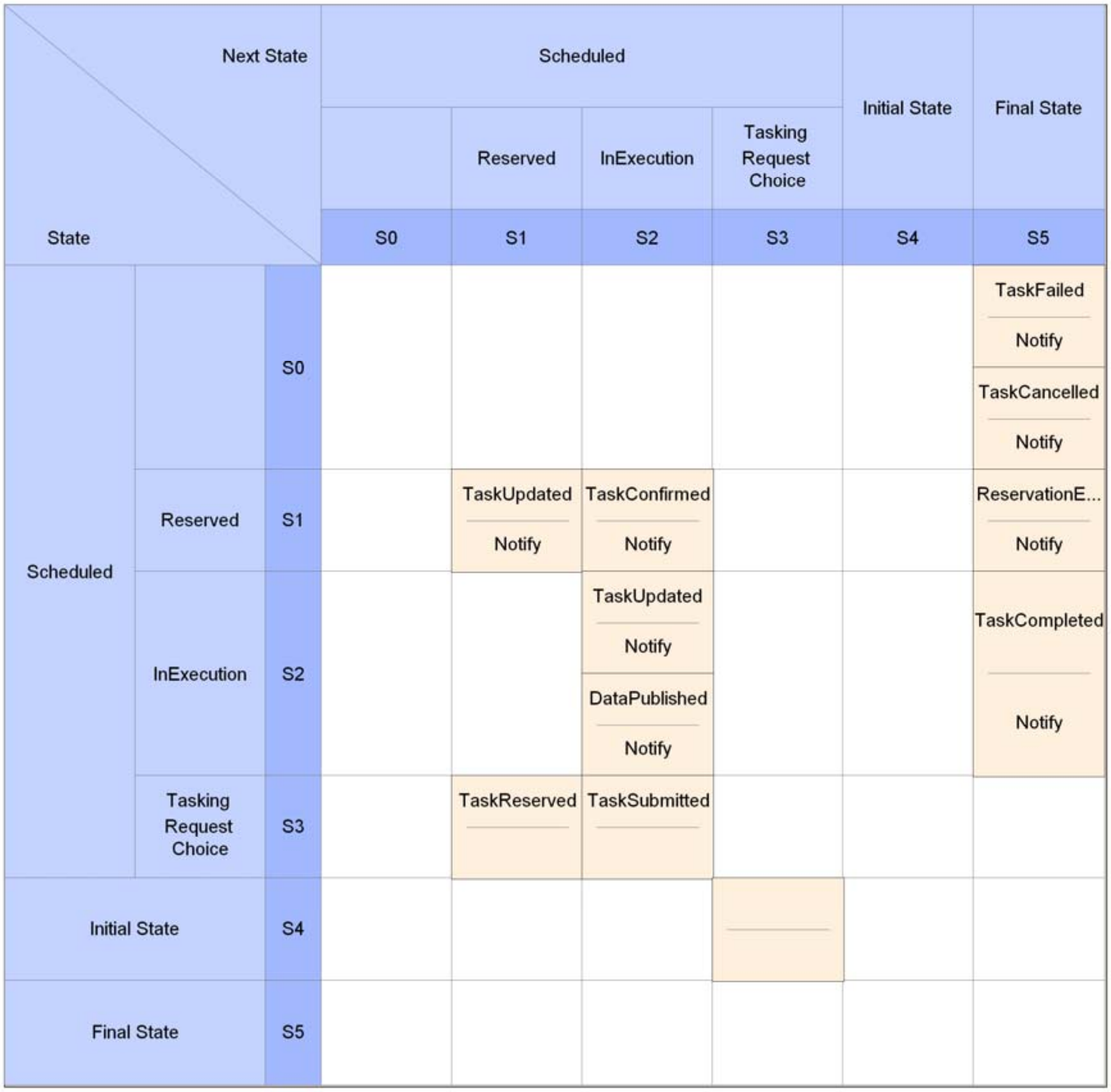


**Figure 32 — task state machine diagram**

An introduction to the state machine depicted in Figure 32 is provided in clause 6.3.6 and thus is not repeated here. The full documentation of the state machine is given in the following clauses.



| State | | Next State | Scheduled | | | | Initial State | Final State |
|---|---|---|---|---|---|---|---|---|
| | | | | Reserved | InExecution | Tasking Request Choice | | |
| | | | S0 | S1 | S2 | S3 | S4 | S5 |
| Scheduled | | S0 | | | | | | TaskFailed<br>Notify<br>TaskCancelled<br>Notify |
| | Reserved | S1 | | TaskUpdated<br>Notify | TaskConfirmed<br>Notify | | | ReservationE...<br>Notify |
| | InExecution | S2 | | | TaskUpdated<br>Notify<br>DataPublished<br>Notify | | | TaskCompleted<br>Notify |
| | Tasking Request Choice | S3 | | TaskReserved | TaskSubmitted | | | |
| Initial State | | S4 | | | | | | |
| Final State | | S5 | | | | | | |

**Figure 33 — task state machine diagram – tabular representation**

### 10.1.2 States/Choices

| Requirement | |
|---|---|
| http://www.opengis.net/spec/SPS/2.0/req/Tasks/StateTransitions | |
| REQ 116. | Any SPS shall implement state transitions as defined in Table 70, Table 71, Table 72, Table 73, Table 74, and Table 75. |

| Requirement | |
|---|---|
| http://www.opengis.net/spec/SPS/2.0/req/Tasks/Notifications | |
| REQ 117. | Any SPS shall send notifications as defined in Table 70, Table 71, Table 72, Table 73, Table 74, and Table 75. |

#### 10.1.2.1 Scheduled State

Any feasible tasking request with the intention to reserve or submit a task gets accepted by the service (otherwise the request would be not feasible) and added to the schedule of the server.

A task that is scheduled by the service can transition through different substates before it reaches the final state.

A client may cancel a task at any time if the Cancel operation is supported by the service. A task can also fail due to unforeseen circumstances that are in the responsibility of the service provider.

The natural way for a scheduled task to be finalized is that it either expires (in case the task was only reserved) or that it is completed as planned.

**Table 70 — Connections of the *Scheduled* state**

| **Connector (type & name)** | **Source (state)** | **Target (state)** | **Notes** |
|---|---|---|---|
| **Transition**<br>TaskCancellation | Scheduled | Final State | If supported by the service, a client may cancel a scheduled task.<br>A service may reject a cancellation request.<br>Data gathered and published for such a task should not automatically be deleted so that a client can at least retrieve the data that was gathered until the task was cancelled.<br>If supported, the service shall notify interested consumers about this event. |
| **Transition**<br>TaskFailure | Scheduled | Final State | If the service is not able to perform a scheduled task as planned, the task fails.<br>If supported, the service shall notify interested consumers about this event. |

#### 10.1.2.2 InExecution State

A task that enters this state is executed by the service. The service starts the internal processing of the request.

**Table 71 — Connections of the *InExecution* state**

| **Connector (type & name)** | **Source (state)** | **Target (state)** | **Notes** |
|---|---|---|---|
| **Transition**<br>TaskCompletion | InExecution | Final State | If a task is completed as planned, it is finalized.<br>If supported, the service shall notify interested consumers about this event. |
| **Transition**<br>ExecutingTaskUpdate | InExecution | InExecution | If a tasking request to update a task that is in the state *InExecution* is feasible, the update shall be performed and the task shall remain (or transition back) into *InExecution* state.<br>Whether the update results in the transition to the previous substate of *InExecution* or in the transition to a new substate is not further specified here. This behavior can be specified by an SPS extension/profile that defines new substates of *InExecution*.<br>If supported, the service shall notify interested consumers about this event. |
| **Transition**<br>DataPublication | InExecution | InExecution | New data was gathered for the task and published by the service - meaning that a client can now access the new data.<br>If supported, the service shall notify interested consumers about this event. |
| **Transition**<br>TaskConfirmation | Reserved | InExecution | A reserved task that is confirmed by the client shall transition into *InExecution* state.<br>If supported, the service shall notify interested consumers about this event. |
| **Transition**<br>TaskSubmission | Tasking Request Choice | InExecution | A feasible tasking request with the intention to submit a task enters *InExecution* state.<br>Note: a service can support notification that a task was submitted by implementing the TaskingRequestAccepted event (see Table 64). |

#### 10.1.2.3 Reserved State

This state represents a task that has successfully been reserved at the service. The service blocks all resources required to execute the task as long as the reservation has not expired.

The reserved task may be updated if the update is feasible - if it is not feasible the task does not change its state.

If a successful confirmation of the reserved task can no longer be guaranteed, the task fails.

**Table 72 — Connections of the *Reserved* state**

| **Connector (type & name)** | **Source (state)** | **Target (state)** | **Notes** |
|---|---|---|---|
| **Transition** TaskReservation | Tasking Request Choice | Reserved | A feasible tasking request with the intention to reserve a task enters the *Reserved* state.<br>Note: a service can support notification that a task was reserved by implementing the *TaskingRequestAccepted* event (see Table 64). |
| **Transition** ReservedTaskUpdate | Reserved | Reserved | If a tasking request to update a reserved task is feasible, the update shall be performed and the task shall remain (or transition back) in *Reserved* state.<br>If supported, the service shall notify interested consumers about this event. |
| **Transition** ReservationExpiration | Reserved | Final State | If a reserved task expired, it shall be finalized by the service.<br>If supported, the service shall notify interested consumers about this event. |
| **Transition** TaskConfirmation | Reserved | InExecution | A reserved task that is confirmed by the client shall transition into *InExecution* state.<br>If supported, the service shall notify interested consumers about this event. |

#### 10.1.2.4 Tasking Request Choice

Which substate of the *Scheduled* state is entered by a new task depends on the semantics of the tasking request. If the tasking request was sent with the intention to reserve a task then the substate will be *Reserved*. If the intention was to submit a task then the substate will be *InExecution*.

**Table 73 — Connections of the *Tasking Request* choice**

| **Connector (type & name)** | **Source (state)** | **Target (state)** | **Notes** |
|---|---|---|---|
| **Transition**<br>TaskReservation | Tasking Request Choice | Reserved | A feasible tasking request with the intention to reserve a task enters the *Reserved* state.<br>Note: a service can support notification that a task was reserved by implementing the *TaskingRequestAccepted* event (see Table 64). |
| **Transition**<br>TaskSubmission | Tasking Request Choice | InExecution | A feasible tasking request with the intention to submit a task enters the *InExecution* state.<br>Note: a service can support notification that a task was submitted by implementing the *TaskingRequestAccepted* event (see Table 64). |
| **Transition**<br>TaskReservationOrSubmission | Initial State | Tasking Request Choice | A feasible tasking request with the intention to reserve or submit an implied task automatically enters the *Scheduled* state. |

### 10.1.2.5 Final State

A task that was completed, has expired, was cancelled or has failed is in its final state.

The service does not allow any confirmation, update or cancellation of a finalized task. An exception (with *ModificationOfFinalizedTask* code) will be thrown if one of these requests is received for a finalized task. An *InvalidParameterValue* exception is thrown if the task identifier in the request is unknown to the service.

**Table 74 — Connections of the Final state**

| **Connector (type & name)** | **Source (state)** | **Target (state)** | **Notes** |
|---|---|---|---|
| **Transition**<br>TaskCancellation | Scheduled | Final State | If supported by the service, a client may cancel a scheduled task.<br>A service may reject a cancellation request.<br>Data gathered and published for such a task should not automatically be deleted so that a client can at least retrieve the data that was gathered until the task was cancelled.<br>If supported, the service shall notify interested consumers about this event. |
| **Transition**<br>TaskCompletion | InExecution | Final State | If a task is completed as planned, it is finalized.<br>If supported, the service shall notify interested consumers about this event. |
| **Transition**<br>ReservationExpiration | Reserved | Final State | If a reserved task expired, it shall be finalized by the service.<br>If supported, the service shall notify interested consumers about this event. |
| **Transition**<br>TaskFailure | Scheduled | Final State | If the service is not able to perform a scheduled task as planned, the task shall fail.<br>If supported, the service shall notify interested consumers about this event. |

#### 10.1.2.6 Initial State

Once a tasking request with the intention to reserve or submit is received by the service and the implied task is feasible, a task gets scheduled by the service.

**Table 75 — Connections of the Initial state**

| Connector (type & name) | Source (state) | Target (state) | Notes |
|---|---|---|---|
| **Transition** TaskReservationOrSubmission | Initial State | Tasking Request Choice | A feasible tasking request with the intention to reserve or submit an implied task automatically enters the *Scheduled* state. |

### 10.1.3 Events/Trigger

| Requirement | |
|---|---|
| http://www.opengis.net/spec/SPS/2.0/req/Tasks/Events | |
| REQ 118. | If an SPS server supports event notification, events shall be sent as defined in clauses 10.1.3.1 to 10.1.3.9. |

#### 10.1.3.1 DataPublished

New data was published for a task that is *InExecution*.

If supported by the service, this causes a notification of the event.

#### 10.1.3.2 ReservationExpired

A reserved task has expired (the expiration time set by the service is before now - "now" being the time measured by the service).

If supported by the service, this causes a notification of the event.

#### 10.1.3.3 TaskCancelled

A scheduled task has been cancelled.

Data gathered and published for the cancelled task should not automatically be deleted so that a client can retrieve the data that was gathered until the task was cancelled.

If supported by the service, this causes a notification of the event.

#### 10.1.3.4 TaskCompleted

A task that was *InExecution* was completed as planned.

Implies that all data gathered in the task has been published.

If supported by the service, this causes a notification of the event.

#### 10.1.3.5 TaskConfirmed

A reserved task was confirmed.

If supported by the service, this causes a notification of the event.

#### 10.1.3.6 TaskFailed

A scheduled task has failed.

Data gathered and published for the failed task should not automatically be deleted so that a client can at least retrieve the data that was gathered until the task failed.

If supported by the service, this causes a notification of the event.

#### 10.1.3.7 TaskReserved

A task was reserved.

Note: a service can support notification that a task was reserved by implementing the *TaskingRequestAccepted* event (see Table 64).

#### 10.1.3.8 TaskSubmitted

A task was submitted.

Note: a service can support notification that a task was reserved by implementing the *TaskingRequestAccepted* event (see Table 64).

#### 10.1.3.9 TaskUpdated

A task was updated.

If supported by the service, this causes a notification of the event.

## 10.2 Tasking Request State Machine

### 10.2.1 Diagrams

The following diagram defines the state machine of an SPS *tasking request*.

NOTE: Figure 34 is the same as Figure 8

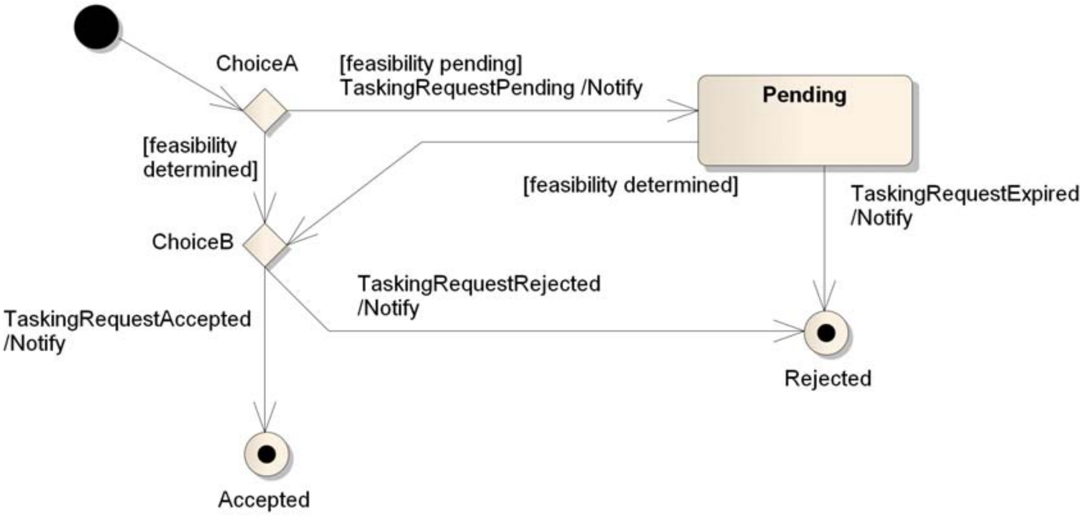


**Figure 34 — tasking request state machine diagram**

An introduction to the state machine depicted in Figure 34 is provided in clause 6.3.6 and thus is not repeated here. The full documentation of the state machine is given in the following clauses.

### 10.2.2 States/Choices

| Requirement | |
|---|---|
| http://www.opengis.net/spec/SPS/2.0/req/TaskingRequests/StateTransitions | |
| REQ 119. | Any SPS shall implement state transitions as defined in Table 76 to Table 81. |

| Requirement | |
|---|---|
| http://www.opengis.net/spec/SPS/2.0/req/TaskingRequests/Notifications | |
| REQ 120. | Any SPS shall send notifications as defined in Table 76 to Table 81. |

#### 10.2.2.1 Pending State

A tasking request of which the feasibility cannot be determined immediately enters *Pending* state.

The tasking request will remain in this state until the feasibility was determined or until the latest response time - if set in the tasking request or initial response to inform the client about the pending state - is reached.

**Table 76 — Connections of the *Pending* state**

| **Connector (type)** | **Source (state)** | **Target (state)** | **Notes** |
|---|---|---|---|
| **Transition** | Pending | Rejected | If the latest response time was set by the client or service for a tasking request and this point in time has been reached, the tasking request automatically transitions into the *Rejected* state.<br>If supported, the service shall notify interested consumers about this event. |
| **Transition** | ChoiceA | Pending | If the service cannot determine the feasibility of a tasking request in a reasonable amount of time, the request transitions into the *Pending* state.<br>If supported, the service shall notify interested consumers about this event. |
| **Transition** | Pending | ChoiceB | If the service can provide a final decision on the feasibility of a pending tasking request, the request transitions on to the final decision point. |

### 10.2.2.2 Accepted State

If the service determines that the tasking request is feasible, the request is in the final state *Accepted*.

If the tasking request had the intention to reserve or submit a task, then a task is scheduled by the service.

If the tasking request had the intention to update a reserved or currently executed task, the update is performed to the task.

**Table 77 — Connections of the *Accepted* state**

| Connector (type) | Source (state) | Target (state) | Notes |
|---|---|---|---|
| **Transition** | ChoiceB | Accepted | If the tasking request is feasible, it transitions on to the final state *Accepted*.<br>If supported, the service shall notify interested consumers about this event. |

### 10.2.2.3 ChoiceA

When receiving a tasking request, the service has to determine the feasibility of a tasking request within a reasonable amount of time. A 'reasonable time' should be a duration that is well below any timeout of the transport protocol used for the communication.

**Table 78 — Connections of the *ChoiceA* choice**

| Connector (type) | Source (state) | Target (state) | Notes |
|---|---|---|---|
| **Transition** | ChoiceA | Pending | If the service cannot determine the feasibility of a tasking request in a reasonable amount of time, the request transitions into the *Pending* state.<br>If supported, the service shall notify interested consumers about this event. |
| **Transition** | ChoiceA | ChoiceB | If the service can determine the feasibility of a tasking request in a reasonable amount of time, the request transitions on to the final decision point. |
| **Transition** | Initial State | ChoiceA | A tasking request automatically reaches the choice where the service decides whether the feasibility of the tasking request can be determined in a reasonable time or not. |

### 10.2.2.4 ChoiceB

Here the service makes his decision whether the request is feasible or not.

**Table 79 — Connections of the *ChoiceB* choice**

| **Connector (type)** | **Source (state)** | **Target (state)** | **Notes** |
|---|---|---|---|
| **Transition** | ChoiceA | ChoiceB | If the service can determine the feasibility of a tasking request in a reasonable amount of time, the request transitions on to the final decision point. |
| **Transition** | Pending | ChoiceB | If the service can provide a final decision on the feasibility of a pending tasking request, the request transitions on to the final decision point. |
| **Transition** | ChoiceB | Accepted | If the tasking request is feasible, it transitions on to the final state *Accepted*.<br>If supported, the service shall notify interested consumers about this event. |
| **Transition** | ChoiceB | Rejected | If the tasking request is not feasible, it transitions on to the final state *Rejected*.<br>If supported, the service shall notify interested consumers about this event. |

### 10.2.2.5 Initial State

A tasking request is sent to the SPS (GetFeasibility, Reserve, Submit, Update).

**Table 80 — Connections of the *Initial* state**

| **Connector (type)** | **Source (state)** | **Target (state)** | **Notes** |
|---|---|---|---|
| **Transition** | Initial State | ChoiceA | A tasking request automatically reaches the choice where the service decides whether the feasibility of the tasking request can be determined in a reasonable time or not. |

### 10.2.2.6 Rejected (Final) State

If the service determines that the tasking request is not feasible, the request is in the final state *Rejected*.

If the tasking request had the intention to reserve or submit a task, then no task is scheduled by the service.

If the tasking request had the intention to update a reserved or currently executed task, the update is not performed.

A service may provide alternative sets of tasking parameters that the client can use to formulate another tasking request.

**Table 81 — Connections of the *Rejected* state**

| **Connector (type)** | **Source (state)** | **Target (state)** | **Notes** |
|---|---|---|---|
| **Transition** | Pending | Rejected | If the latest response time was set by the client or service for a tasking request and this point in time has been reached (the current time being after the latest response time), the tasking request automatically transitions into the *Rejected* state.<br>If supported, the service shall notify interested consumers about this event. |
| **Transition** | ChoiceB | Rejected | If the tasking request is not feasible, it transitions on to the final state *Rejected*.<br>If supported, the service shall notify interested consumers about this event. |

### 10.2.3 Events/Trigger

| Requirement | |
|---|---|
| http://www.opengis.net/spec/SPS/2.0/req/TaskingRequests/EventsTrigger | |
| REQ 121. | If an SPS server supports event notification, events shall be sent as defined in clauses 10.2.3.1 to 10.2.3.3. |

#### 10.2.3.1 TaskingRequestAccepted

A tasking request has been accepted.

If supported by the service, this causes a notification of the event.

#### 10.2.3.2 TaskingRequestExpired

A pending tasking request has expired.

If supported by the service, this causes a notification of the event.

#### 10.2.3.3 TaskingRequestRejected

A tasking request has been rejected.

If supported by the service, this causes a notification of the event.

#### 10.2.3.4 TaskingRequestPending

A tasking request is pending.

If supported by the service, this causes a notification of the event.

# 11 Annex A – Abstract Test Suite and Conformance Testing (normative)

Specific conformance tests for a Sensor Planning Service need to be defined on the concrete service level in order to ensure full interoperability. Thus, the abstract test suite defined herein only ensures general interoperability between client and server.
An SPS implementation shall satisfy the following system characteristics to be minimally conformant with this specification:

## 11.1 Conformance Class – Core

http://www.opengis.net/spec/SPS/2.0/conf/Core

### 11.1.1 Capability Test

http://www.opengis.net/spec/SPS/2.0/conf/Core/Capability

a) Test Purpose: Verify that the server implements the *Core* conformance class.

b) Test Method: Verify that the server implements the following conformance classes: *http://www.opengis.net/spec/SWE/2.0/conf/uml-simple-encodings*, *http://www.opengis.net/spec/SWE/2.0/conf/uml-simple-components*, *http://www.opengis.net/spec/SWES/2.0/conf/BasicSWEServiceMetadata*, *http://www.opengis.net/spec/SWES/2.0/conf/SensorProvider*. Verify the conformance tests listed in section 11.1.2

c) Reference:see references in conformance tests

d) Test Type: Capability

### 11.1.2 Modules with Basic Tests

#### 11.1.2.1 Common Request Response Handling

##### 11.1.2.1.1 Invalid version number

http://www.opengis.net/spec/SPS/2.0/conf/Core/RequestResponse/InvalidVersionNumber

a) Test Purpose: To verify that a request, other than a GetCapabilities request, with the version number set to one that the server does not claim to support in its capabilities document fails.

b) Test Method: Review the response to the GetCapabilites request and determine which request version(s) the server claims to support. Execute one or more SPS requests with a version that is not in the list of supported version and verify that the server generates an InvalidParameterValue exception.

c) Reference: conformance test A.4.2.3 in OGC 06-121r3

d) Test Type: Basic

#### 11.1.2.1.2 Service and version appropriateness

http://www.opengis.net/spec/SPS/2.0/conf/Core/RequestResponse/ServiceAndVersion

a) Test Purpose: To verify that the server recognizes correct values for service and version parameters in operation request other than GetCapabilities.

b) Test Method: Devise and execute a request with correct value ("SPS") for the service (type) request parameter and another request with correct value for the version parameter ("2.0.0"). Verify that the service does not throw an InvalidParameterValue exception with locator version "service" or "version".

c) Reference: Subclause of chapter 7 according to the given operation.

d) Test Type: Basic

### 11.1.2.2 Exception Reporting

#### 11.1.2.2.1 Exception Appropriateness

http://www.opengis.net/spec/SPS/2.0/conf/Core/ExceptionReporting/Appropriateness

a) Test Purpose: Verify that the server generates an appropriate exception by setting the value of the code and locator parameters to an appropriate value.

Test Method: Devise a series of requests that generate an error for each applicable error code used in

b) Figure 12. Verify that server generates an appropriate exception for each case by verifying that the code and locator parameters have been set to the correct value.

c) Reference: 7.2

d) Test Type: Basic

#### 11.1.2.2.2 Exception Model Compliancy

http://www.opengis.net/spec/SPS/2.0/conf/Core/ExceptionReporting/ModelCompliancy

a) Test Purpose: To verify that the exceptions the server generates validate according to the schema defined in Clause 8 of 06-121r3.

b) Test Method: Devise and execute a request that generates an error. Verify that the exception that the server generates is valid.

c) Reference: 7.2

d) Test Type: Basic

#### 11.1.2.3 Service Metadata

##### 11.1.2.3.1 Adherence to property inheritance mechanism

http://www.opengis.net/spec/SPS/2.0/conf/Core/ServiceMetadata/PropertyInheritanceAdherence

a) Test Purpose: To verify that the service adheres to the rules of property inheritance.

b) Test Method: Devise and execute requests that test each of the values for the procedure and procedure description format properties that an offering has when applying the property inheritance mechanism as defined in OGC 09-001 and Table 27.

c) Reference: 7.3.3.4, OGC 09-001, Table 27

d) Test Type: Basic

##### 11.1.2.3.2 Default Service Version

http://www.opengis.net/spec/SPS/2.0/conf/Core/ServiceMetadata/DefaultServiceVersion

a) Test Purpose: To verify that the service supports retrieval of Capabilities in version 2.0.0.

b) Test Method: Devise a GetCapabilities request with acceptVersions parameter set to value “2.0.0” and send it to the service. Verify that the service property in the resulting capabilities document has the value "2.0.0".

c) Reference: 7.3.2.4

d) Test Type: Basic

##### 11.1.2.3.3 GetCapabilities operation facet validity

http://www.opengis.net/spec/SPS/2.0/conf/Core/ServiceMetadata/GetCapabilitiesFacetValidity

a) Test Purpose: To verify that the service provides the correct response when the GetCapabilities operation is invoked.

b) Test Method: Devise and execute a GetCapabilities request. Verify that the service responds with a valid capabilities document or exception.

c) Reference: 7.3.2.3, 7.3.2.4

d) Test Type: Basic

##### 11.1.2.3.4 Indicate support of SWE Common Encodings

http://www.opengis.net/spec/SPS/2.0/conf/Core/ServiceMetadata/SWECommonEncodings

a) Test Purpose: Verify that the service advertises which SWE Common encodings it supports.

b) Test Method: Devise a GetCapabilities request to retrieve the full capabilities document of the service and send it to the service. Get the list of supported SWE Common encodings from the contents section. Verify that the conformance classes for these encodings are listed in the "profile" property of the serviceIdentification section. Ensure that at least the URIs for the "Simple Encodings UML Package" conformance class from the SWE Common Data Model is listed.

c) Reference: 7.3.3.3

d) Test Type: Basic

#### 11.1.2.3.5 Indicate support of SWE Common Structures

http://www.opengis.net/spec/SPS/2.0/conf/Core/ServiceMetadata/SWECommonStructures

a) Test Purpose: Verify that the service advertises which SWE Common structures it supports.

b) Test Method: Devise a GetCapabilities request to retrieve the serviceIdentification section of the service's capabilities document and send it to the service. Get the list of supported conformance classes. Verify that at least the URI for the "Basic Types and Simple Components UML Package" conformance class from the SWE Common Data Model is listed there.

c) Reference: 7.3.3.3

d) Test Type: Basic

#### 11.1.2.3.6 Listing of supported conformance classes

http://www.opengis.net/spec/SPS/2.0/conf/Core/ServiceMetadata/ConformanceClassListing

a) Test Purpose: Verify that a service lists all the conformance classes it supports in its metadata.

b) Test Method: Execute a GetCapabilities request to retrieve the serviceIdentification section. Verify that the service passes all tests associated to the conformance classes that are listed in the profile property of this section.

c) Reference: 7.3.2.4.6

d) Test Type: Basic

#### 11.1.2.3.7 Mandatory Operations

http://www.opengis.net/spec/SPS/2.0/conf/Core/ServiceMetadata/MandatoryOperations

a) Test Purpose: Verify that all mandatory SPS operations are supported by the service.

b) Test Method: Execute a GetCapabilities request to retrieve the operationsMetadata section. Verify that the mandatory operations according to Table 22 are listed there. Execute further GetCapabilities as well as DescribeSensor, DescribeResultAccess, DescribeTasking, GetStatus, GetTask and Submit requests. Verify that the server sends appropriate responses as defined in this specification.

c) Reference: 7.3.2, 7.3.4, 7.3.5, 7.3.6, 7.3.7, 7.3.8 and OGC 09-001

d) Test Type: Basic

#### 11.1.2.3.8 Minimum section set

http://www.opengis.net/spec/SPS/2.0/conf/Core/ServiceMetadata/MinimumSectionSet

a) Test Purpose: Verify that the service supports at least the serviceProvider, serviceIdentification, operationsMetadata and contents sections.

b) Test Method: Create a GetCapabilities request to get the full capabilities document and check that it contains the according sections.

c) Reference: 7.3.2

d) Test Type: Basic

#### 11.1.2.3.9 Number of property values for sensor offering

http://www.opengis.net/spec/SPS/2.0/conf/Core/ServiceMetadata/SensorOfferingPropertyValues

a) Test Purpose: To verify that the server has the correct number of values for the properties contained in the SensorOffering in each of its offerings listed in its contents section.

b) Test Method: Devise and execute a GetCapabilities request that requests the contents section. Verify that the number of values for the procedure, procedure description format, observable property, related feature and observable area properties in each offering after applying the property inheritance mechanism (see OGC 09-001) are as defined in Table 27.

c) Reference: 7.3.3.3, OGC 09-001, Table 27

d) Test Type: Basic

#### 11.1.2.3.10Version negotiation for the GetCapabilities request

http://www.opengis.net/spec/SPS/2.0/conf/Core/ServiceMetadata/VersionNegotiation

a) Test Purpose: To verify that the server correctly handles version negotiation for the GetCapabilities operation.

b) Test Method: Verify that the server conforms to the test described in 06-121r3.

c) Reference: A.4.2.3 of 06-121r3

d) Test Type: Basic

#### 11.1.2.4 DescribeTasking

##### 11.1.2.4.1 DescribeTasking operation facet validity

http://www.opengis.net/spec/SPS/2.0/conf/Core/DescribeTasking/OperationFacetValidity

a) Test Purpose: To verify that the service provides the correct response when the DescribeTasking operation is invoked.

b) Test Method: Devise and execute a DescribeTasking request. Verify that the service responds with a valid DescribeTaskingResponse or exception.

c) Reference: 7.3.4

d) Test Type: Basic

##### 11.1.2.4.2 Provide name for tasking parameter component

http://www.opengis.net/spec/SPS/2.0/conf/Core/DescribeTasking/TaskingParameterNames

a) Test Purpose: To verify that the service provides required name attributes for and in the tasking parameter description.

b) Test Method: Devise and execute a DescribeTasking request for each procedure hosted by the service. Verify that the taskingParameter description in the DescribeTaskingResponse has a properly populated name attribute and also that all SWE Common components eventually contained in the parameter description has such a name attribute.

c) Reference: 7.3.4.4

d) Test Type: Basic

##### 11.1.2.4.3 Tasking Parameter Description Model Validity

http://www.opengis.net/spec/SPS/2.0/conf/Core/DescribeTasking/TaskingParameterModelValidity

a) Test Purpose: To verify that the service uses only those SWE Common Data structures that it indicates support for and that these are valid.

b) Test Method: Devise and execute a DescribeTasking request for each procedure hosted by the service. Verify that the taskingParameter description in the DescribeTaskingResponse uses a SWE Common AbstractDataComponent subtype that is covered by one of the SWE Common conformance classes listed in the service's capabilities document. Ensure that this component passes the tests defined in all conformance classes of the SWE Common Data Model standard listed in the service's capabilities document.

c) Reference: 7.3.4

d) Test Type: Basic

#### 11.1.2.5 Tasking

##### 11.1.2.5.1 Tasking Parameter Usage

http://www.opengis.net/spec/SPS/2.0/conf/Core/Tasking/TaskingParameterUsage

a) Test Purpose: Verify that the service supports the SWE Common encodings as advertised in its capabilities.

b) Test Method: Devise a GetCapabilities request to retrieve the contents section of the service's capabilities document and send it to the service. Get the list of supported SWE Common encodings from the contents section.

   Devise a valid tasking request (Submit and - if implemented - Reserve, Update, GetFeasibility) with tasking parameter values structured according to the tasking parameter description retrieved via the DescribeTasking operation for the tasked procedure and encoded according to an encoding indicated in the tasking request and supported by the service. Send this tasking request to the service. Verify that the service does not return an InvalidParameterValue exception with locator "taskingParameters".

   Similarly, devise a tasking request with invalid tasking parameters (not following the structure defined in the DescribeTasking response, not using an encoding supported by the service or not encoding the values correctly) and send it to the service. Ensure that the service throws an InvalidParameterValue exception with locator "taskingParameters".

c) Reference: 7.2

d) Test Type: Basic

##### 11.1.2.5.2 Tasking request expiration

http://www.opengis.net/spec/SPS/2.0/conf/Core/Tasking/TaskingRequestExpiration

a) Test Purpose: To verify that the service correctly handles tasking request expiration.

b) Test Method: Given a tasking response that has requestStatus "Pending" and defines a latestResponseTime: devise and execute a GetStatus request for the request and send it to the service shortly before the latestResponseTime. If the requestStatus is still "Pending" send another GetStatus request shortly after the latestResponseTime. Verify that the requestStatus in the latest status report is either "Accepted" or "Rejected". If it is "Rejected", check the updateTime of the according status report. If the updateTime is before the latestResponseTime, ensure that the event is not "TaskingRequestExpired". Otherwise ensure that the updateTime value is the same time as the latestResponseTime and that the event is "TaskingRequestExpired".

c) Reference: 7.3.1.3, 7.3.1.4

d) Test Type: Basic

#### 11.1.2.6 State Handling

##### 11.1.2.6.1 GetStatus operation facet validity

http://www.opengis.net/spec/SPS/2.0/conf/Core/StateHandling/GetStatusOperationFacetValidity

a) Test Purpose: To verify that the service provides the correct response when the GetStatus operation is invoked.

b) Test Method: Devise and execute a GetStatus request. Verify that the service responds with a valid GetStatusResponse or exception.

c) Reference: 7.3.6

d) Test Type: Basic

##### 11.1.2.6.2 GetTask operation facet validity

http://www.opengis.net/spec/SPS/2.0/conf/Core/StateHandling/GetTaskOperationFacetValidity

a) Test Purpose: To verify that the service provides the correct response when the GetTask operation is invoked.

b) Test Method: Devise and execute a GetTask request. Verify that the service responds with a valid GetTaskResponse or exception.

c) Reference: 7.3.7

d) Test Type: Basic

##### 11.1.2.6.3 Handling requests for already deleted status information

http://www.opengis.net/spec/SPS/2.0/conf/Core/StateHandling/HandlingRequestsForDeletedStatusInfo

a) Test Purpose: Verify that the service correctly handles GetStatus / GetTask requests that ask for status information of a task / tasking request but that information has already been discarded by the service.

b) Test Method: Devise a GetStatus / GetTask request for a task that was completed by the service and execute it after the minStatusTime has expired. Verify that the service either sends an exception with code StatusInformationExpired (in case it still knows the task but does no longer store status information for it) or an exception with code InvalidParameterValue (in case that the service already removed all information on that task and thus does no longer “know” it) with locator "task".

c) Reference: 7.3.6.5, 7.3.7.5

d) Test Type: Basic

#### 11.1.2.6.4 State handling

http://www.opengis.net/spec/SPS/2.0/conf/Core/StateHandling/ValidStateMachineImplementation

a) Test Purpose: Verify that the service correctly implements the state machines defined for tasking requests / tasks.

b) Test Method: Devise a valid tasking request and send it to the service. Create valid GetTask / GetStatus requests for the according tasks / tasking requests and send them to the service. Inspect the response to verify that no illegal transition for the tasking request / task is made.

c) Reference: 10, 7.3.1.5, 7.3.6, 7.3.7

d) Test Type: Basic

#### 11.1.2.6.5 State information storage

http://www.opengis.net/spec/SPS/2.0/conf/Core/StateHandling/StateInfoStorage

a) Test Purpose: Verify that the service provides the information about the latest state transition of finalized tasks / tasking requests as long as indicated in its service metadata.

b) Test Method: Devise a valid request for all the tasking operations supported by the service (Submit, GetFeasibility, Update, Reserve). Send those to the service.

   Create valid GetTask / GetStatus requests for the according tasks / tasking requests when they were finalized. Send them to the service shortly before the point in time that is defined by the updateTime of the status report that documented the transition into the final state plus the "minStatusTime" duration that is stated in the contents section of the service's capabilities document. Verify that each response contains information on the latest state transition made by the request / task.

c) Reference: 7.3.3.3

d) Test Type: Basic

#### 11.1.2.6.6 State provisioning

http://www.opengis.net/spec/SPS/2.0/conf/Core/StateHandling/StateProvisioning

a) Test Purpose: Verify that the service provides information about the latest state of all tasks / tasking requests.

b) Test Method: Devise a valid request for all the tasking operations supported by the service (Submit, GetFeasibility, Update, Reserve). Send those requests to the service.

   Create valid GetTask / GetStatus requests for the according tasks / tasking requests and send them to the service. Verify that each response contains information on the latest state transitions made by the request / task so far.

c) Reference: 7.3.6, 7.3.7

d) Test Type: Basic

#### 11.1.2.6.7 StatusReport Usage in GetStatusResponse

http://www.opengis.net/spec/SPS/2.0/conf/Core/StateHandling/ReportUsageInGetStatusResponse

a) Test Purpose: To verify that the service correctly provides status information in a GetStatusResponse.

Test Method: Devise and execute a valid GetStatus request for a task / tasking request that was sent to the service previously that has not been finalized too long ago so that the service already discarded status information on that task / tasking request. Verify that the result property of the GetStatusResponse contains a StatusReport as defined in Table 34,

b) Table 35, Table 36 and Table 37 - depending upon the nature of the actual task / tasking request.

c) Reference: 7.3.6.4

d) Test Type: Basic

#### 11.1.2.6.8 StatusReport Usage in GetTaskResponse

http://www.opengis.net/spec/SPS/2.0/conf/Core/StateHandling/ReportUsageInGetTaskResponse

a) Test Purpose: To verify that the service correctly provides status information in a GetTaskResponse.

Test Method: Devise and execute a valid GetTask request for a task / tasking request that was sent to the service previously that has not been finalized too long ago so that the service already discarded status information on that task / tasking request. Verify that each task in the GetTaskResponse contains StatusReports as defined in Table 34,

b) Table 35, Table 36 and Table 37 - depending upon the nature of the actual task / tasking request.

c) Reference: 7.3.7.4

d) Test Type: Basic

#### 11.1.2.7 Submit

##### 11.1.2.7.1 StatusReport Usage in SubmitResponse

http://www.opengis.net/spec/SPS/2.0/conf/Core/Submit/ReportUsageInSubmitResponse

a) Test Purpose: To verify that the service correctly provides status information in a SubmitResponse.

b) Test Method: Devise and execute a valid Submit request. Verify that the result property of the SubmitResponse contains a StatusReport as defined in Table 31.

c) Reference: 7.3.5.4

d) Test Type: Basic

##### 11.1.2.7.2 Submit operation facet validity

http://www.opengis.net/spec/SPS/2.0/conf/Core/Submit/OperationFacetValidity

a) Test Purpose: To verify that the service provides the correct response when the Submit operation is invoked.

b) Test Method: Devise and execute a Submit request. Verify that the service responds with a valid SubmitResponse or exception.

c) Reference: 7.3.5

d) Test Type: Basic

##### 11.1.2.7.3 Successful task submission

http://www.opengis.net/spec/SPS/2.0/conf/Core/Submit/SuccessfulTaskSubmission

a) Test Purpose: To verify that the service schedules a task if a Submit request is feasible.

b) Test Method: Devise and execute a valid Submit request. Verify via the GetStatus / GetTask operation that a task was scheduled with the same task identifier that was provided in the SubmitResponse. To do this, verify that the latest status of that task makes correct use of the taskStatus property.

c) Reference: 7.3.5, 7.3.1.5

d) Test Type: Basic

#### 11.1.2.8 Result Handling

##### 11.1.2.8.1 DescribeResultAccess operation facet validity

http://www.opengis.net/spec/SPS/2.0/conf/Core/ResultHandling/DescribeResultAccessOperationFacetValidity

a) Test Purpose: To verify that the service provides the correct response when the DescribeResultAccess operation is invoked.

b) Test Method: Devise and execute a DescribeResultAccess request. Verify that the service responds with a valid DescribeResultAccessResponse or exception.

c) Reference: 7.3.8

d) Test Type: Basic

##### 11.1.2.8.2 Handling of data unavailability

http://www.opengis.net/spec/SPS/2.0/conf/Core/ResultHandling/HandlingDataUnavailability

a) Test Purpose: To verify that the service handles data unavailability correctly.

b) Test Method: Devise and execute a DescribeResultAccess request for a tasking request that was just accepted. Ensure that the request is made before the task is completed and before it made a DataPublished transition. Verify that the response contains the unavailableCode "DataNotAvailable".

c) Reference: 7.3.8.1

d) Test Type: Basic

##### 11.1.2.8.3 Identifiers for references and reference groups

http://www.opengis.net/spec/SPS/2.0/conf/Core/ResultHandling/ReferenceAndGroupIdentifiers

a) Test Purpose: To verify that the service assigns identifiers to reference groups and references and that these do not change for as long as the given reference (group) exists.

b) Test Method: Devise and execute DescribeResultAccess requests for an accepted task. When a response indicates that data is available, verify that each reference group and the reference(s) it contains have a unique identifier value. Verify that consecutive responses do not contain a reference (group) that has the exact same property values as a reference (group) in a previous response but which has a different identifier.

c) Reference: 7.3.8.7

d) Test Type: Basic

#### 11.1.2.8.4 Incremental data publication

http://www.opengis.net/spec/SPS/2.0/conf/Core/ResultHandling/IncrementalDataPublication

a) Test Purpose: To verify that the service provides new references when it published new data while a task is in execution.

b) Test Method: If publish / subscribe functionality is supported by the service, subscribe for DataPublished events. Submit a task that is going to be executed by the service. When the submit request was accepted, devise and execute a DescribeResultAccess request for that task. Whenever a DataPublished event for the task was published, execute another DescribeResultAccess request. Compare the references contained in that response with those of the previous response. Verify that new references have been added by checking for references with new identifier values.

c) Reference: 7.3.8.1

d) Test Type: Basic

#### 11.1.2.8.5 Referencing general data services for procedure

http://www.opengis.net/spec/SPS/2.0/conf/Core/ResultHandling/ReferencingDataServicesForProcedure

a) Test Purpose: To verify that the service provides references to possible data storage locations / services when DescribeResultAccess with procedure identifier was made.

b) Test Method: Devise and execute a DescribeResultAccess request. Verify that the references contained in the response are references to folders / services (as defined in Table 40 and Table 41).

c) Reference: 7.3.8.1

d) Test Type: Basic

#### 11.1.2.8.6 Referencing task data

http://www.opengis.net/spec/SPS/2.0/conf/Core/ResultHandling/ReferencingTaskData

a) Test Purpose: To verify that the service provides references to the data gathered for a task when DescribeResultAccess with task identifier was made.

b) Test Method: Devise and execute a DescribeResultAccess request. Verify that the references contained in the response are references as defined in Table 41.

c) Reference: 7.3.8.1

d) Test Type: Basic

#### 11.1.2.8.7 Result access information storage

http://www.opengis.net/spec/SPS/2.0/conf/Core/ResultHandling/ResultAccessInfoStorage

a) Test Purpose: Verify that the service provides result access information for a task that was in execution at least as long as indicated in its service metadata.

b) Test Method: Submit a task that reaches the InExcecution state.

c) Create a valid DescribeResultAccess requests for the according task. When the task was finalized, send the request to the service shortly before the point in time that is defined by the updateTime of the status report that documented the transition into the final state plus the "minStatusTime" duration that is stated in the contents section of the service's capabilities document. Verify that the response contains at least one reference group with references or has the unavailableCode "DataServiceUnavailable".

d) Reference: 7.3.8.1, 7.3.3.3

e) Test Type: Basic

## 11.2 Conformance Class – State Logger

http://www.opengis.net/spec/SPS/2.0/conf/StateLogger

### 11.2.1 Capability Test

http://www.opengis.net/spec/SPS/2.0/conf/StateLogger/Capability

a) Test Purpose: Verify that the server implements the *State Logger* conformance class.

b) Test Method: Verify that the server implements the *Core* conformance class. Verify the conformance tests listed in section 11.2.2

c) Reference: see references in conformance tests

d) Test Type: Capability

### 11.2.2 Modules with Basic Tests

#### 11.2.2.1 Service Metadata

##### 11.2.2.1.1 Advertising support for status history logging

http://www.opengis.net/spec/SPS/2.0/conf/StateLogger/ServiceMetadata/StatusHistorySupportAdvertisement

a) Test Purpose: Verify that the service indicates support for logging of status history in its metadata.

b) Test Method: Execute a GetCapabilities request to retrieve the operationsMetadata section. Verify that the "since" parameter is supported for the GetStatus operation listed there.

c) Reference: 7.3.2.4.3

d) Test Type: Basic

#### 11.2.2.2 Behavior

##### 11.2.2.2.1 GetStatus with since parameter

http://www.opengis.net/spec/SPS/2.0/conf/StateLogger/Behavior/GetStatusSinceParameterHandling

a) Test Purpose: Verify that the service handles GetStatus requests with "since" parameter correctly.

b) Test Method: Devise a valid GetStatus request for a task / tasking request that has already made more than one state transition. Choose a point in time that is between the updateTime of the first state transition and the updateTime of the following state transition. Set the "since" parameter in the GetStatus request to that point in time. Send the request to the service. Verify that the response contains information on all state transitions made by the task / tasking request except the first one.

   Likewise, create and send a GetStatus request with "since" parameter value being a point in time shortly after the updateTime of the last state transition of a finalized task / tasking request (for accepted Submit and Reserve requests, the finalization of the resulting scheduled task matters). Verify that the response does not contain any status information.

c) Reference: 7.3.6

d) Test Type: Basic

##### 11.2.2.2.2 Status history provisioning

http://www.opengis.net/spec/SPS/2.0/conf/StateLogger/Behavior/StatusHistoryProvisioning

a) Test Purpose: Verify that the service provides complete state history for all tasks / tasking requests.

b) Test Method: Devise a valid request for all the tasking operations supported by the service (Submit, GetFeasibility, Update, Reserve). Send those to the service.

   Create valid GetTask requests for the according tasks / tasking requests and send them to the service. Also create valid GetStatus request with since parameter value that is well before the time that the initial tasking request was made. Verify that each response contains information on all the state transitions made by the request / task so far.

   If information on state changes of a task can be retrieved by other means, for example through notifications, verify that this information matches the one retrieved via the GetStatus / GetTask operations.

c) Reference: 7.3.6, 7.3.7

d) Test Type: Basic

##### 11.2.2.2.3 Status history storage

http://www.opengis.net/spec/SPS/2.0/conf/StateLogger/Behavior/StatusHistoryStorage

a) Test Purpose: Verify that the service provides complete state history for all finalized tasks / tasking requests as long as indicated in its service metadata.

b) Test Method: Devise a valid request for all the tasking operations supported by the service (Submit, GetFeasibility, Update, Reserve). Send those to the service.

   Create valid GetTask requests for the according tasks / tasking requests when they were finalized. Also create valid GetStatus requests with since parameter value that is well before the time that the initial tasking request was made. Send them to the service shortly before the point in time that is defined by the updateTime of the status report that documented the transition into the final state plus the "minStatusTime" duration that is stated in the contents section of the service's capabilities document. Verify that each response contains information on all the state transitions made by the request / task.

c) Reference: 7.3.3.3

d) Test Type: Basic

## 11.3 Conformance Class – Reservation Manager

http://www.opengis.net/spec/SPS/2.0/conf/ReservationManager

### 11.3.1 Capability Test

http://www.opengis.net/spec/SPS/2.0/conf/ReservationManager/Capability

a) Test Purpose: Verify that the server implements the *Reservation Manager* conformance class.

b) Test Method: Verify that the server implements the *Core* conformance class. Verify the conformance tests listed in section 11.3.2

c) Reference: see references in conformance tests

d) Test Type: Capability

### 11.3.2 Modules with Basic Tests

#### 11.3.2.1 Structure

##### 11.3.2.1.1 Confirm operation facet validity

http://www.opengis.net/spec/SPS/2.0/conf/ReservationManager/Structure/ConfirmOperationFacetValidity

a) Test Purpose: To verify that the service provides the correct response when the Confirm operation is invoked.

b) Test Method: Devise and execute a Confirm request. Verify that the service responds with a valid ConfirmResponse or exception.

c) Reference: 7.3.10

d) Test Type: Basic

##### 11.3.2.1.2 Reserve operation facet validity

http://www.opengis.net/spec/SPS/2.0/conf/ReservationManager/Structure/ReserveOperationFacetValidity

a) Test Purpose: To verify that the service provides the correct response when the Reserve operation is invoked.

b) Test Method: Devise and execute a Reserve request. Verify that the service responds with a valid ReserveResponse or exception.

c) Reference: 7.3.9

d) Test Type: Basic

#### 11.3.2.2 Service Metadata

##### 11.3.2.2.1 Operations listed in Capabilities

http://www.opengis.net/spec/SPS/2.0/conf/ReservationManager/ServiceMetadata/OperationsListing

a) Test Purpose: Verify that the Reserve and Confirm operations are listed as supported operations in the service's metadata.

b) Test Method: Execute a GetCapabilities request to retrieve the operationsMetadata section. Verify that the Reserve and Confirm operations are listed there as defined in clause 7.3.2.4.2.

c) Reference: 7.3.2.4.2

d) Test Type: Basic

#### 11.3.2.3 Behavior

##### 11.3.2.3.1 Handling of incorrect expiration time

http://www.opengis.net/spec/SPS/2.0/conf/ReservationManager/Behavior/IncorrectExpirationTime

a) Test Purpose: To verify that the service rejects reservation requests with incorrect expiration time.

b) Test Method: Devise a Reserve request with valid tasking parameters and with an expiration time in the past. Execute the request. Verify that the response has requestStatus "Rejected".

c) Reference: 7.3.9.1

d) Test Type: Basic

##### 11.3.2.3.2 Reservation confirmation

http://www.opengis.net/spec/SPS/2.0/conf/ReservationManager/Behavior/ReservationConfirmation

a) Test Purpose: To verify that the service correctly handles the confirmation of a reserved task.

b) Test Method: Given a reserved task that has not expired yet. Confirm the task. If the response has requestStatus rejected, ensure that the task has taskStatus "Failed" (get the status of the task e.g. via GetStatus operation). Otherwise, ensure that task status is either InExecution, Completed, Cancelled or Failed (or substate thereof).

   Note: further checks would be possible if "State Logger" conformance class is implemented.

c) Reference: 7.3.10

d) Test Type: Basic

##### 11.3.2.3.3 Reservation expiration

http://www.opengis.net/spec/SPS/2.0/conf/ReservationManager/Behavior/ReservationExpiration

a) Test Purpose: To verify that the service correctly handles expiration of a reserved task.

b) Test Method: Devise and execute a valid Reserve request that will be accepted. Get the expirationTime of the reservation (e.g. via GetStatus). Do NOT confirm the reservation. After the expirationTime has passed, get the latest status of the task. Verify that the status is encoded as a reservation report. Verify that the updateTime of the reservation report is the same as the expirationTime provided in the report and provided in previous reservation reports for that task. Verify that the event is "ReservationExpired" and that the taskStatus is "Expired".

   Verify that a Confirm of an "Expired" task is rejected by the service.

c) Reference: 7.3.9, 7.3.6.4, 7.3.10

d) Test Type: Basic

## 11.4 Conformance Class – Task Canceller

http://www.opengis.net/spec/SPS/2.0/conf/TaskCanceller

### 11.4.1 Capability Test

http://www.opengis.net/spec/SPS/2.0/conf/TaskCanceller/Capability

a) Test Purpose: Verify that the server implements the *Task Canceller* conformance class.

b) Test Method: Verify that the server implements the *Core* conformance class. Verify the conformance tests listed in section 11.4.2

c) Reference: see references in conformance tests

d) Test Type: Capability

### 11.4.2 Modules with Basic Tests

#### 11.4.2.1 Structure

##### 11.4.2.1.1 Cancel operation facet validity

http://www.opengis.net/spec/SPS/2.0/conf/TaskCanceller/Structure/CancelOperationFacetValidity

a) Test Purpose: To verify that the service provides the correct response when the Cancel operation is invoked.

b) Test Method: Devise and execute a Cancel request. Verify that the service responds with a valid CancelResponse or exception.

c) Reference: 7.3.13

d) Test Type: Basic

#### 11.4.2.2 Behavior

##### 11.4.2.2.1 Cancellation handling

http://www.opengis.net/spec/SPS/2.0/conf/TaskCanceller/Behavior/CancellationHandling

a) Test Purpose: To verify that the service handles task cancellations correctly.

b) Test Method: Given a scheduled task that is not finalized yet. Devise and execute a Cancel request for that task.

   If the request was rejected, verify that the status of the task is not "Cancelled". Otherwise verify that the status is "Cancelled".

c) Reference: 7.3.13.1

d) Test Type: Basic

#### 11.4.2.3 Service Metadata

##### 11.4.2.3.1 Operation listed in Capabilities

http://www.opengis.net/spec/SPS/2.0/conf/TaskCanceller/ServiceMetadata/OperationListing

a) Test Purpose: Verify that the Cancel operation is listed as supported operation in the service's metadata.

b) Test Method: Execute a GetCapabilities request to retrieve the operationsMetadata section. Verify that the Cancel operation is listed there as defined in clause 7.3.2.4.2.

c) Reference: 7.3.2.4.2

d) Test Type: Basic

## 11.5 Conformance Class – Feasibility Controller

http://www.opengis.net/spec/SPS/2.0/conf/FeasibilityController

### 11.5.1 Capability Test

http://www.opengis.net/spec/SPS/2.0/conf/FeasibilityController/Capability

a) Test Purpose: Verify that the server implements the *Feasibility Controller* conformance class.

b) Test Method: Verify that the server implements the *Core* conformance class. Verify the conformance tests listed in section 11.5.2

c) Reference: see references in conformance tests

d) Test Type: Capability

### 11.5.2 Modules with Basic Tests

#### 11.5.2.1 Structure

##### 11.5.2.1.1 GetFeasibility operation facet validity

http://www.opengis.net/spec/SPS/2.0/conf/FeasibilityController/Structure/GetFeasibilityOperationFacetValidity

a) Test Purpose: To verify that the service provides the correct response when the GetFeasibility operation is invoked.

b) Test Method: Devise and execute a GetFeasibility request. Verify that the service responds with a valid GetFeasibilityResponse or exception.

c) Reference: 7.3.11

d) Test Type: Basic

#### 11.5.2.2 Service Metadata

##### 11.5.2.2.1 Operation listed in Capabilities

http://www.opengis.net/spec/SPS/2.0/conf/FeasibilityController/ServiceMetadata/OperationListing

a) Test Purpose: Verify that the GetFeasibility operation is listed as supported operation in the service's metadata.

b) Test Method: Execute a GetCapabilities request to retrieve the operationsMetadata section. Verify that the GetFeasibility operation is listed there as defined in clause 7.3.2.4.2.

c) Reference: 7.3.2.4.2

d) Test Type: Basic

### 11.6Conformance Class – Task Updater

http://www.opengis.net/spec/SPS/2.0/conf/TaskUpdater

### 11.6.1 Capability Test

http://www.opengis.net/spec/SPS/2.0/conf/TaskUpdater/Capability

a) Test Purpose: Verify that the server implements the *Task Updater* conformance class.

b) Test Method: Verify that the server implements the *Core* conformance class. Verify the conformance tests listed in section 11.6.2

c) Reference: see references in conformance tests

d) Test Type: Capability

### 11.6.2 Modules with Basic Tests

#### 11.6.2.1 Structure

##### 11.6.2.1.1 Update operation facet validity

http://www.opengis.net/spec/SPS/2.0/conf/TaskUpdater/Structure/UpdateOperationFacetValidity

a) Test Purpose: To verify that the service provides the correct response when the Update operation is invoked.

b) Test Method: Devise and execute an Update request. Verify that the service responds with a valid UpdateResponse or exception.

c) Reference: 7.3.12

d) Test Type: Basic

#### 11.6.2.2 Behavior

##### 11.6.2.2.1 Handling of updatable DataArray

http://www.opengis.net/spec/SPS/2.0/conf/TaskUpdater/Behavior/UpdatableDataArray

a) Test Purpose: To verify that the service correctly flags the content of an updatable DataArray

b) Test Method: Devise and execute a DescribeTasking request for each procedure. Verify that each tasking parameter description where a DataArray is contained, the elementType description in that array is not flagged as updatable (either the DataArray is updatable in general or it is not; sub components of the component that is the elementType description may be flagged to be updatable).

c) Reference: 7.3.12.1

d) Test Type: Basic

##### 11.6.2.2.2 Handling of updatable DataRecord / DataChoice

http://www.opengis.net/spec/SPS/2.0/conf/TaskUpdater/Behavior/UpdatableDataRecordAndDataChoice

a) Test Purpose: To verify that the service correctly sets the updatable flag on DataRecords and DataChoices

b) Test Method: Devise and execute a DescribeTasking request for each procedure. Verify that each tasking parameter description where a DataRecord/DataChoice is updatable, at least one field/item is updatable as well.

c) Reference: 7.3.12.1

d) Test Type: Basic

### 11.6.2.2.3 Handling update not supported for a given procedure

http://www.opengis.net/spec/SPS/2.0/conf/TaskUpdater/Behavior/UpdateNotSupportedForProcedure

a) Test Purpose: To verify that a service which in general supports the Update operation handles Update requests for tasks of a procedure that has no updatable tasking parameters correctly.

b) Test Method: Devise and execute a Submit request for a procedure where the tasking parameter description (retrieved via DescribeTasking) has no updatable parameters. When the task was accepted, devise and execute an Update request for that task, with the same tasking parameters as those used in the Submit request. Verify that the service returns an UpdateResponse where the requestStatus is set to 'Rejected'.

c) Reference: 7.3.12.1

d) Test Type: Basic

### 11.6.2.2.4 New identifier assignment

http://www.opengis.net/spec/SPS/2.0/conf/TaskUpdater/Behavior/IdentifierAssignment

a) Test Purpose: To verify that the service assigns a new identifier to an incoming Update request and does not mix it up with the task identifier provided in the request.

b) Test Method: Devise and execute an Update request. Verify that the task property in the status report of the UpdateResponse does not have the same value as the task property in the Update request.

c) Reference: 7.3.12.1

d) Test Type: Basic

### 11.6.2.2.5 State transition resulting of task update

http://www.opengis.net/spec/SPS/2.0/conf/TaskUpdater/Behavior/StateTransitions

a) Test Purpose: To verify that the service correctly handles state transitions of a task resulting from an update request to it.

b) Test Method: Devise and execute an Update request targetting a scheduled task. Get the updateTime from the status report that informs about the acceptance / rejection of the update request.

   If the request was accepted, verify that a status report exists for the updated task with the same updateTime and event "TaskUpdated".

   Otherwise (the request was rejected), verify that no such status report exists for the task that was intended to be updated.

c) Reference: 7.3.12.1

d) Test Type: Basic

#### 11.6.2.2.6 Structure of tasking parameters for Update

http://www.opengis.net/spec/SPS/2.0/conf/TaskUpdater/Behavior/HandlingTaskingParametersForUpdate

a) Test Purpose: To verify that tasking parameters used in an Update request are structured correctly.

b) Test Method: Create a task so that it is InExecution (or Reserved, if the Reserve operation is supported). Remove all non-updatable components from the tasking parameter description that was provided by the service in a DescribeTasking request for the procedure associated with the task. Non-updatable components are those components in the description that have the property 'updatable' explicitly set to false. If a non-updatable component is contained in a field/item of a DataRecord/DataChoice then completely remove that field/item. Devise and execute Update requests with tasking parameters structured according to the resulting description. Verify that the service does not throw an InvalidParameterValue exception with locator 'taskingParameters'.

c) Reference: 7.3.12.1

d) Test Type: Basic

### 11.6.2.3 Service Metadata

#### 11.6.2.3.1 Operation listed in Capabilities

http://www.opengis.net/spec/SPS/2.0/conf/TaskUpdater/ServiceMetadata/OperationListing

a) Test Purpose: Verify that the Update operation is listed as supported operation in the service's metadata.

b) Test Method: Execute a GetCapabilities request to retrieve the operationsMetadata section. Verify that the Update operation is listed there as defined in clause 7.3.2.4.2.

c) Reference: 7.3.2.4.2

d) Test Type: Basic

## 11.7Conformance Class – Basic PubSub

http://www.opengis.net/spec/SPS/2.0/conf/BasicPubSub

### 11.7.1 Capability Test

http://www.opengis.net/spec/SPS/2.0/conf/BasicPubSub/Capability

a) Test Purpose: Verify that the server implements the *Basic PubSub* conformance class.

b) Test Method: Verify that the server implements the *Core* conformance class. Verify the conformance tests listed in section 11.7.2

c) Reference: see references in conformance tests

d) Test Type: Capability

### 11.7.2 Modules with Basic Tests

#### 11.7.2.1 Event Publication

##### 11.7.2.1.1 SPS event encoding

http://www.opengis.net/spec/SPS/2.0/conf/BasicPubSub/EventPublication/Encoding

a) Test Purpose: Verify that events are properly encoded.

b) Test Method: Subscribe for all events published by the service. Devise tasking requests that cause publication of according events. For each event received, check that it is encoded as defined in Table 64.

c) Reference: 8.2, Table 64

d) Test Type: Basic

##### 11.7.2.1.2 SPS event publication

http://www.opengis.net/spec/SPS/2.0/conf/BasicPubSub/EventPublication/Publication

a) Test Purpose: Verify that the service publishes the mandatory SPS events.

b) Test Method: Subscribe for all events published by the service. Devise tasking requests that cause publication of according events. At least the SubmissionCompleted / TaskCompleted event should be published by the service for a successfully submitted and completed task. If the service implements the state logger conformance class, i.e. logs all state transitions of a tasking request / task, do the following:

- once a tasking request / task was finalized, get all state information for it via the GetStatus operation
- check that the events published by the service for this tasking request / task are in line with the state transitions documented in the GetStatus response.
- Otherwise check at least that the final state is published correctly.

c) Reference: 8.2, Table 64

d) Test Type: Basic

#### 11.7.2.2 Notification Service Metadata

##### 11.7.2.2.1 Notifications section

http://www.opengis.net/spec/SPS/2.0/conf/BasicPubSub/ServiceMetadata/NotificationsSection

a) Test Purpose: Verify that the service supports the notifications section in the capabilities document.

b) Test Method: Create a GetCapabilities request to get the capabilities document with the notifications section and check that it is implemented correctly.

c) Reference: 7.3.2, OGC 09-001 clause 8

d) Test Type: Basic

## 11.8 Conformance Class – Channel Based PubSub

http://www.opengis.net/spec/SPS/2.0/conf/ChannelBasedPubSub

### 11.8.1 Capability Test

http://www.opengis.net/spec/SPS/2.0/conf/ChannelBasedPubSub/Capability

a) Test Purpose: Verify that the server implements the *Channel Based PubSub* conformance class.

b) Test Method: Verify that the server implements the *Basic PubSub* conformance class. Verify the conformance tests listed in section 11.8.2

c) Reference: see references in conformance tests

d) Test Type: Capability

### 11.8.2 Modules with Basic Tests

#### 11.8.2.1 Channel based Event Publication

##### 11.8.2.1.1 Correct channel assignments

http://www.opengis.net/spec/SPS/2.0/conf/ChannelBasedPubSub/ChannelEventPublication/ChannelAssignments

a) Test Purpose: Verify that published events are assigned to correct channels.

b) Test Method: Create one subscription targetting each of the SPS channels contained in the service's topic set. Devise tasking requests that cause publication of events on each of these channels. For each subscription, check that those and only those events are received that are to be published on the channel associated with that subscription according to Table 67.

c) Reference: 8.3

d) Test Type: Basic

#### 11.8.2.2 Channel based Notification Service Metadata

##### 11.8.2.2.1 Support of Topic Dialect

http://www.opengis.net/spec/SPS/2.0/conf/ChannelBasedPubSub/ServiceMetadata/TopicDialectSupport

a) Test Purpose: Verify that the service supports at least one topic expression dialect.

b) Test Method: Devise and send a GetCapabilities request to retrieve the service's notifications metadata. Inspect which filter dialects are supported. Ensure that at least one topic expression dialect is listed.

c) Reference: 8.3, OGC 09-001 clause 8

d) Test Type: Basic

##### 11.8.2.2.2 Topic Set Contents

http://www.opengis.net/spec/SPS/2.0/conf/ChannelBasedPubSub/ServiceMetadata/TopicSetContents

a) Test Purpose: Verify that the topic set provided by the service contains the required SPS topics.

b) Test Method: Devise and send a GetCapabilities request to retrieve the service's notifications metadata. Retrieve the topic set from that metadata and inspect it. Ensure that all the mandatory topics listed in Table 67 are marked as topics in this topic set.

c) Reference: 8.3

d) Test Type: Basic

### 11.9 Conformance Class – XML Encoding

http://www.opengis.net/spec/SPS/2.0/conf/XMLEncoding

#### 11.9.1 Capability Test

http://www.opengis.net/spec/SPS/2.0/conf/XMLEncoding/Capability

a) Test Purpose: Verify that the server implements the *XML Encoding* conformance class.

b) Test Method: Verify that the server implements the following conformance classes *http://www.opengis.net/spec/SWE/2.0/conf/xsd-simple-components*, *http://www.opengis.net/spec/SWE/2.0/conf/xsd-simple-encodings*, *http://www.opengis.net/spec/SWES/2.0/conf/XMLEncoding*. Verify the conformance tests listed in section 11.9.2

c) Reference: see references in conformance tests

d) Test Type: Capability

#### 11.9.2 Modules with Basic Tests

##### 11.9.2.1 Validation

##### 11.9.2.1.1 XML Encoding Validity

http://www.opengis.net/spec/SPS/2.0/conf/XMLEncoding/Validation/XMLEncoding

a) Test Purpose: Verify that XML implementations of the conceptual types defined in the specification are valid according to their XML Schema implementation.

b) Test Method: For all XML instance documents received from the service that are in the namespace *http://www.opengis.net/sps/2.0*, verify that they are valid according to their XML Schema definition listed in Table 82.

   Note: the *sps.xsd* can be used for validating any such XML instance against its schema definition.

c) Reference: 12

d) Test Type: Basic

##### 11.9.2.1.2 XML Validation Exception Reporting

http://www.opengis.net/spec/SPS/2.0/conf/XMLEncoding/Validation/ExceptionReporting

a) Test Purpose: Verify that the service sends an exception with appropriate code if it received an invalid request.

b) Test Method: For all SPS operations supported by the service, create an XML request instance that is invalid according to its schema definition outlined in Table 82 and send it to the service. Verify that the service returns an exception with code *InvalidRequest*.

c) Reference: 7.2, 12

d) Test Type: Basic

### 11.10 Conformance Class – SOAP

http://www.opengis.net/spec/SPS/2.0/conf/SOAP

#### 11.10.1 Capability Test

http://www.opengis.net/spec/SPS/2.0/conf/SOAP/Capability

a) Test Purpose: Verify that the server implements the *SOAP* conformance class.

b) Test Method: Verify that the server implements the *XML Encoding* conformance class. Do so by checking that the Body element in SOAP messages sent to the service for invoking an SPS operation contains a valid XML representation of the according operation request. Verify that the server implements the *http://www.opengis.net/spec/SWES/2.0/conf/SOAPBinding* conformance classes. Verify the conformance tests listed in section 11.10.2.

c) Reference: see references in conformance tests

d) Test Type: Capability

#### 11.10.2 Modules with Basic Tests

##### 11.10.2.1 Action URIs

###### 11.10.2.1.1Asynchronous request response

http://www.opengis.net/spec/SPS/2.0/conf/SOAP/ActionURIs/AsyncRequestResponse

a) Test Purpose: To verify that WS-Addressing is used to enable asynchronous request / response.

b) Test Method: Get the service metadata (WSDL and / or Capabilities document). Ensure that the service metadata does not indicate support for any asynchronous request response realization technique that could be used by clients other than WS-Addressing.

c) Reference: 9.5

d) Test Type: Basic

#### 11.10.2.1.2Operation Actions

http://www.opengis.net/spec/SPS/2.0/conf/SOAP/ActionURIs/OperationActions

a) Test Purpose: To verify that the service recognizes and uses correct action URIs for operation requests and responses as well as notifications as defined in this standard.

b) Test Method: Depending upon the SOAP binding available at the service, execute a request for each SPS operation supported by the service. Verify that the service uses the correct SOAP action as defined in Table 68 or uses an empty action in its response. If WS-Addressing is used, verify that the service uses the correct WS-Addressing action URIs as defined in Table 68.

c) Reference: 9.3

d) Test Type: Basic

### 11.10.2.2 Exception Handling

#### 11.10.2.2.1Usage of SOAP faults

http://www.opengis.net/spec/SPS/2.0/conf/SOAP/ExceptionHandling/SOAPFaultUsage

a) Test Purpose: Verify that SOAP faults for the SPS operations are encoded correctly.

Test Method: For each SPS operation supported by the service, create one SOAP encoded request that causes an exception with certain code. For each operation, repeat this so that one test request for all applicable exception codes (as listed in

b) Figure 12) is available. Send the requests to the service. Verify that the service returns a SOAP fault as defined in OGC 09-001 clause 19.2 and clause 9.2 in this standard.

c) Reference: 9.2, OGC 09-001 clause 19.2

d) Test Type: Basic

### 11.10.2.3 Service Metadata

#### 11.10.2.3.1SOAP operation encoding advertised

http://www.opengis.net/spec/SPS/2.0/conf/SOAP/ServiceMetadata/OperationEncodingAdvertisement

a) Test Purpose: Verify that the service indicates that it supports the SOAP binding.

b) Test Method: Devise a GetCapabilities request and send it to the service to retrieve the operationsMetadata section of the capabilities document. Verify that a "PostEncoding" constraint for the HTTP POST transfer of all operations exists that has the value "SOAP".

c) Reference: 7.3.2.4.4

d) Test Type: Basic

## 12 Annex B - XML Schema Documents (normative)

In addition to this document, this standard 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 2.0 of this standard, these XML Schema Documents will also be posted online at the URL http://schemas.opengis.net/sps/2.0. In the event of a discrepancy between the bundled and online versions of the XML Schema Documents, the online files shall be considered authoritative.

The data types specified in this standard are contained in thirteen packages which themselves are children of the Sensor Planning Service package (see clause 7.3).

The UML model has been mapped to its XML Schema encoding using the rules described in clause 24 of [OGC 09-001], resulting in the following XML Schema documents:

sps.xsd (includes the other schema through xs:include statements)

spsCancel.xsd

spsCommon.xsd

spsConfirm.xsd

spsContents.xsd

spsDescribeResultAccess.xsd

spsDescribeTasking.xsd

spsGetCapabilities.xsd

spsGetFeasibility.xsd

spsGetStatus.xsd

spsGetTask.xsd

spsReserve.xsd

spsSubmit.xsd

spsUpdate.xsd

| Requirement | |
|---|---|
| http://www.opengis.net/spec/SPS/2.0/req/XML/GeneralEncodingRule | |
| REQ 122. | The XML encoding of the conceptual types defined in this standard shall be as defined by the XML Schema files listed and referenced in clause 12.<br>More specifically, the XML encoding of each conceptual type shall be valid against the XML Schema definition of the according mapping as defined in Table 82. |

The following table provides an overview how each of the conceptual model types defined by this standard has been realized in the XML Schema implementation.

**Table 82 — XML Schema implementation of types defined by the SPS conceptual model**

| UML class | object element | type | property type |
|---|---|---|---|
| *SPS Common Package* | | | |
| StatusReport | sps:StatusReport | sps:StatusReportType | sps:StatusReportPropertyType |
| Task | sps:Task | sps:TaskType | sps:TaskPropertyType |
| Alternative | sps:Alternative | sps:AlternativeType | sps:AlternativePropertyType |
| TaskingResponse | sps:TaskingResponse | sps:TaskingResponseType | sps:TaskingResponsePropertyType |
| TaskingRequest | sps:TaskingRequest | sps:TaskingRequestType | sps:TaskingRequestPropertyType |
| ParameterData | sps:ParameterData | sps:ParameterDataType | sps:ParameterDataPropertyType |
| TaskingRequestStatusCode | - | sps:TaskingRequestStatusCodeType | - |
| TaskStatusCode | - | sps:TaskStatusCodeType | - |
| EventCode | - | sps:EventCodeType | - |
| *SPS Cancel Package* | | | |
| Cancel | sps:Cancel | sps:CancelType | sps:CancelPropertyType |
| CancelResponse | sps:CancelResponse | sps:CancelResponseType | sps:CancelResponsePropertyType |
| *SPS Confirm Package* | | | |
| Confirm | sps:Confirm | sps:ConfirmType | sps:ConfirmPropertyType |
| ConfirmResponse | sps:ConfirmResponse | sps:ConfirmResponseType | sps:ConfirmResponsePropertyType |

| UML class | object element | type | property type |
|---|---|---|---|
| *SPS Contents Package* | | | |
| SensorOffering | sps:SensorOffering | sps:SensorOfferingType | sps:SensorOfferingPropertyType |
| SPSContents | sps:SPSContents | sps:SPSContentsType | sps:SPSContentsPropertyType |
| PointOrPolygon | sps:PointOrPolygon (group) | - | sps:PointOrPolygonPropertyType |
| *SPS DescribeResultAccess Package* | | | |
| DataAvailable | sps:DataAvailable | sps:DataAvailableType | sps:DataAvailablePropertyType |
| DescribeResultAccess | sps:DescribeResultAccess | sps:DescribeResultAccessType | sps:DescribeResultAccessPropertyType |
| DescribeResultAccess Response | sps:DescribeResultAccessResponse | sps:DescribeResultAccessResponseType | sps:DescribeResultAccessResponse PropertyType |
| DataNotAvailable | sps:DataNotAvailable | sps:DataNotAvailableType | sps:DataNotAvailablePropertyType |
| TaskOrProcess | sps:TaskOrProcess (group) | - | sps:TaskOrProcessPropertyType |
| AvailableOrNot | sps:AvailableOrNot (group) | sps:AvailableOrNotType | sps:AvailableOrNotPropertyType |
| UnavailableCode | - | sps:UnavailableCodeType | - |
| *SPS DescribeTasking Package* | | | |
| DescribeTaskingResponse | sps:DescribeTaskingResponse | sps:DescribeTaskingResponseType | sps:DescribeTaskingResponsePropertyType |
| DescribeTasking | sps:DescribeTasking | sps:DescribeTaskingType | sps:DescribeTaskingPropertyType |
| *SPS GetCapabilities Package* | | | |
| GetCapabilities | sps:GetCapabilities | sps:GetCapabilitiesType | sps:GetCapabilitiesPropertyType |
| Capabilities | sps:Capabilities | sps:CapabilitiesType | sps:CapabilitiesPropertyType |
| *SPS GetFeasibility Package* | | | |
| GetFeasibilityResponse | sps:GetFeasibilityResponse | sps:GetFeasibilityResponseType | sps:GetFeasibilityResponseProperty Type |
| GetFeasibility | sps:GetFeasibility | sps:GetFeasibilityType | sps:GetFeasibilityPropertyType |
| *SPS GetStatus Package* | | | |
| GetStatusResponse | sps:GetStatusResponse | sps:GetStatusResponseType | sps:GetStatusResponsePropertyType |
| GetStatus | sps:GetStatus | sps:GetStatusType | sps:GetStatusPropertyType |

| UML class | object element | type | property type |
|---|---|---|---|
| *SPS GetTask Package* | | | |
| GetTask | sps:GetTask | sps:GetTaskType | sps:GetTaskPropertyType |
| GetTaskResponse | sps:GetTaskResponse | sps:GetTaskResponseType | sps:GetTaskResponsePropertyType |
| *SPS Reserve Package* | | | |
| ReservationReport | sps:ReservationReport | sps:ReservationReportType | sps:ReservationReportPropertyType |
| ReserveResponse | sps:ReserveResponse | sps:ReserveResponseType | sps:ReserveResponsePropertyType |
| Reserve | sps:Reserve | sps:ReserveType | sps:ReservePropertyType |
| *SPS Submit Package* | | | |
| SubmitResponse | sps:SubmitResponse | sps:SubmitResponseType | sps:SubmitResponsePropertyType |
| Submit | sps:Submit | sps:SubmitType | sps:SubmitPropertyType |
| *SPS Update Package* | | | |
| UpdateResponse | sps:UpdateResponse | sps:UpdateResponseType | sps:UpdateResponsePropertyType |
| Update | sps:Update | sps:UpdateType | sps:UpdatePropertyType |

# 13 Annex C - Revision history

| Date | Release | Editor | Primary clauses modified | Description |
|---|---|---|---|---|
| 15.08.2009 | 0.0.1 | Ingo Simonis | all | initial version |
| 18.11.2009 | 0.1.0 | Ingo Simonis/Johannes Echterhoff | throughout | changes for RFC draft |
| 04.12.2009 | 0.2.0 | Ingo Simonis/Johannes Echterhoff | all | final RFC draft |
| 17.12.2009 | 0.2.1 | Johannes Echterhoff | all | final changes to RFC document discussed at Dec TC meeting |
| 10.06.2010 | 0.3.0 | Ingo Simonis | all | Integration of RFC comments and discussion |
| 10.08.2010 | 0.4.0 | Johannes Echterhoff | all | Integration of latest discussion and conformance classes |
| 20.08.2010 | 0.5.0 | Ingo Simonis | all | Integration of requirements according to modular spec model |
| 31.08.2010 | 0.6.0 | Johannes Echterhoff | Annex A | revised tests |
| 30.09.2010 | 0.7.0 | Johannes Echterhoff | throughout | included comments received during final SWG review phase |
| 31.01.2011 | 0.8.0 | Ingo Simonis | Future work | added |
| 21.01.2011 | 2.0 | Carl Reed | Various | Prepare for publication |

# Bibliography

[1] OpenGIS® Implementation Specification, *Sensor Planning Service*, OGC document 07-014r3