Open Geospatial Consortium Inc.

Date: 2007-06-08

Reference number of this document: OGC 05-007r7

Version: 1.0.0

Category: OpenGIS® Standard

Editor: Peter Schut

OpenGIS® Web Processing Service

Document type: OGC Publicly Available Standard
Document Subtype: OGC Standard
Document stage: Approved
Document language: English

License Agreement

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

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

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

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

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

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

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

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

Warning

OGC official documents use a triple decimal-dot notation (i.e. MM.xx.ss). This document may be identified as MM.xx (Major.minor) and may include increments to the third dot series (schema changes) without any modification to this document, or the version displayed on the document. This means, for example, that a document labelled with versions 1.1.0 and 1.1.1 or even 1.1.9 are exactly the same except for modifications to the official schemas that are maintained and perpetually located at: http://schemas.opengis.net/. Note that corrections to the document are registered via corrigendums. A corrigendum will change the base document and notice will be given by appending a c# to the version (where # specifies the corrigendum number). In corrigendums that correct both the schemas and the base document, the third triplet of the document version will increment and the 'c1' or subsequent identifier will be appended, however the schemas will only increase the third triplet of the version.

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

Contents

Page

Figures

Page

Tables

Page

i. Preface

This document specifies the interface to a Web Processing Service (WPS). This document is the result of work undertaken to support the Canadian Geospatial Data Infrastructure (CGDI), and in particular the National Land and Water Information Service (NLWIS), and the National Forest Information Service (NFIS). The specification was first implemented as a prototype in 2004 by Agriculture and Agri-Food Canada (AAFC). In the first half of 2005, it was the subject of a successful OGC Interoperability Experiment.

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

ii. Document terms and definitions

This document uses the specification 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 specification

iii. Submitting organizations

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

GeoConnections / Natural Resources Canada

PCI Geomatics

iv. Document contributor contact points

All questions regarding this document should be directed to the editor or the WPS Revision Working Group contributors:

Name	Organization
Theodor Förster	ITC
Christian Heier	Wupperverband
Steven Keens	PCI Geomatics
Christian Kiehle	lat/lon GmbH
Rachel ONeil	ESRI Canada
Nicole Ostlaender	Joint Research Centre (JRC)
Joan Maso Pau	Universitat Autònoma de Barcelona (CREAF)
Peter Schut	GeoConnections
Arliss Whiteside	BAE Systems - National Security Solutions

The Revision Working Group acknowledges the formative input to this specification from the following contributors:

Name	Organization
Mike Adair	GeoConnections
Harald Borsutzky	University of Muenster - Institute for Geoinformatics
Stephane Fellah	PCI Geomatics
Xiaoyuan Geng	GeoConnections
Martin Kyle	Galdos Systems
Weisheng Li	PCI Geomatics
Maru Newby	GeoConnections

v. Revision history

Date	Release	Editor	Primary clauses modified	Description
05 May 2004	0.1.0	Peter. Schut	All	Initial document, formatted for OGC template
22 May 2004	0.1.0	Peter Schut	All	Cleaned up some problems, added informative examples in Annex B
21 Oct. 2004	0.2.0	Stephane Fellah	Content	Rewrite the schema and the Table of Contents
22 Nov. 2004	0.2.0	Xiaoyuan Geng	All	Created document using the latest OGC template, the initial draft, and schema

Date	Release	Editor	Primary clauses modified	Description
24 Dec. 2004	0.2.1	Peter Schut	All	Minor corrections and revisions throughout, additions of human readable explanations of schemas
11 April 2005	0.2.3	Peter Schut	All	Upgrade based on results to date of WPSie.
05 April 2005	0.3.0	Peter Schut	All	Upgrade based on results to date of WPSie and alignment with OWS Common
13 July 2005	0.4.0	Peter Schut	6 & 7	Complete documentation of each element and renaming of elements to eliminate confusion caused by abstractions
1 Sept 2005	0.4.0	Arliss Whiteside	All	Added UML diagrams, aligned with new schemas.
16 Sept 2005	0.4.0	Peter Schut and Arliss Whiteside	All	Final editing and cleanup for the WPS RFC
08 June 2007	1.0.0	Peter Schut	All	Complete rewrite based on comments received in the WPS RFC and additional change requests handled by the WPS RWG.

vi. Changes to the OGC Abstract Specification

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

vii. Future work

Foreword

The Web Processing Service (WPS) was originally named Geoprocessing Service (OGC document number 04-043). The name was changed to "Web Processing Service" early in its development to avoid the acronym GPS, since this would have caused confusion with the conventional use of this acronym for Global Positioning System. Since WPS is an OGC specification, the term geospatial would have been redundant. A version of WPS was released in September 2005 as document number 05-007r4, and was the subject of an OGC RFC. This document replaces those earlier draft documents.

The WPS Interoperability Experiment (see OGC document 05-051r1) demonstrated that clients developed by different organizations could readily access and bind to services that are set up in accordance with the WPS Implementation Specification. Version 1.0.0 incorporates the recommendations that were made during that Interoperability Experiment and subsequent comments received through OGC's RFC process and the subsequent OWS-4 test bed.

This document includes five annexes; Annexes A, B, and D are normative, while Annex C and F are informative.

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

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

Introduction

This document specifies the interface to a Web Processing Service (WPS). WPS defines a standardized interface that facilitates the publishing of geospatial processes, and the discovery of and binding to those processes by clients. "Processes" include any algorithm, calculation or model that operates on spatially referenced data. "Publishing" means making available machine-readable binding information as well as human-readable metadata that allows service discovery and use.

A WPS can be configured to offer any sort of GIS functionality to clients across a network, including access to pre-programmed calculations and/or computation models that operate on spatially referenced data. A WPS may offer calculations as simple as subtracting one set of spatially referenced numbers from another (e.g., determining the difference in influenza cases between two different seasons), or as complicated as a global climate change model. The data required by the WPS can be delivered across a network, or available at the server.

This interface specification provides mechanisms to identify the spatially referenced data required by the calculation, initiate the calculation, and manage the output from the calculation so that the client can access it. This Web Processing Service is targeted at processing both vector and raster data.

The WPS specification is designed to allow a service provider to expose a web accessible process, such as polygon intersection, in a way that allows clients to input data and execute the process with no specialized knowledge of the underlying physical process interface or API. The WPS interface standardizes the way processes and their inputs/outputs are described, how a client can request the execution of a process, and how the output from a process is handled.

Because WPS offers a generic interface, it can be used to wrap other existing and planned OGC services that focus on providing geospatial processing services.

OpenGIS® Web Processing Service

1 Scope

This document specifies the interface to a general purpose Web Processing Service (WPS). A WPS provides client access across a network to pre-programmed calculations and/or computation models that operate on spatially referenced data. The calculation can be extremely simple or highly complex, with any number of data inputs and outputs.

This document does not specify the specific processes that could be implemented by a WPS. Instead, it specifies a generic mechanism that can be used to describe and web-enable any sort of geospatial process. To achieve interoperability, each process must be specified in a separate document, which might be called an Application Profile of this specification.

This document does not specify any specific data required or output by the WPS. Instead, it identifies a generic mechanism to describe the data inputs required and produced by a process. This data can be delivered across the network, or available at the server. This data can include image data formats such as GeoTIFF, or data exchange standards such as Geography Markup Language (GML). Data inputs can be legitimate calls to OGC web services. For example, a data input for an intersection operation could be a polygon delivered in response to a WFS request, in which case the WPS data input would be the WFS query string.

This document does not address the archival, cataloguing, discovery, or retrieval of information that has been created by a WPS.

2 Conformance

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

3 Normative references

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

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

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

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

In addition to this document, this specification includes several normative XML Schema Document files as listed in Annex B and referenced throughout the text. These XML Schema Documents include complete documentation of the meaning of each element, attribute, and type. These XML Schema Documents and the documentation contained therein shall be considered normative as specified in Subclause 11.6.3 of [OGC 06-121r3].

4 Terms and definitions

For the purposes of this specification, the definitions specified in Clause 4 of the OWS Common Implementation Specification [OGC 06-121r3] shall apply. In addition, the following terms and definitions apply.

4.1
input
data provided to a **process**

4.2
literal
any process input or output whose value can be represented in a character string, supplemented by metadata as needed

4.3
literal (XML encoding)
any process input or output whose value can be represented in a xsd:string supplemented by XML attributes as needed

NOTE A literal process input or output can be a character string, integer, general number, URI, measure, etc.

4.4
map
pictorial representation of geographic data

4.5
process
model or calculation that is made available at a **service instance**

4.6
output
result returned by a **process**

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.

5.2 UML notation

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

5.3 Used parts of other documents

This document uses significant parts of document [OGC 06-121r3]. To reduce the need to refer to that document, this document copies some of those parts with small modifications. To indicate those parts to readers of this document, the copied portions are shown with a light grey background (15%).

5.4 Platform-neutral and platform-specific specifications

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

EXAMPLES 1 Platform-neutral specifications are contained in Subclauses 8.3.1, 8.3.3, 9.2.1, 9.3.1, 10.2.1, 10.3.1, and 10.3.2.

The specified platform-neutral data could be encoded in many alternative ways, each appropriate to one or more specific DCPs. This document now specifies only encoding appropriate for use of HTTP GET transfer of operations requests (using KVP encoding), and for use of HTTP POST transfer of operations requests (using XML or KVP encoding). However, the same operation requests and responses (and other data) could be encoded for other specific computing platforms, including SOAP/WSDL.

EXAMPLES 2 Platform-specific specifications for KVP encoding are contained in Subclauses 9.2.1 and 10.2.2.

EXAMPLES 3 Platform-specific specifications for XML encoding are contained in Subclauses 8.3.2, 8.3.4, 9.2.3, 9.3.2, 10.2.3, and 10.3.3.

6 WPS overview

The specified Web Processing Service (WPS) provides client access to pre-programmed calculations and/or computation models that operate on spatially referenced data. The data required by the service can be delivered across a network, or available at the server. This data can use image data formats or data exchange standards such as Geography Markup Language (GML). The calculation can be as simple as subtracting one set of spatially referenced numbers from another (e.g. determining the difference in influenza cases between two different seasons), or as complicated as a global climate change model.

Enabling geospatial processing on the Internet requires the development of a wide variety web services to support atomic geospatial operations as well as sophisticated modelling capabilities. It is important to standardize the way that these processes are called, in order to reduce amount of programming required, and to facilitate the implementation and adoption of new services. WPS is intended to help OGC members to achieve these goals.

6.1 WPS Operations

The WPS interface specifies three operations that can be requested by a client and performed by a WPS server, all mandatory implementation by all servers. Those operations are:

a) GetCapabilities – This operation allows a client to request and receive back service metadata (or Capabilities) documents that describe the abilities of the specific server implementation. The GetCapabilities operation provides the names and general descriptions of each of the processes offered by a WPS instance. This operation also supports negotiation of the specification version being used for client-server interactions.

b) DescribeProcess – This operation allows a client to request and receive back detailed information about the processes that can be run on the service instance, including the inputs required, their allowable formats, and the outputs that can be produced.

c) Execute – This operation allows a client to run a specified process implemented by the WPS, using provided input parameter values and returning the outputs produced.

These operations have many similarities to other OGC Web Services, including the WMS, WFS, and WCS. The interface aspects that are common with these other OWSs are specified in the OpenGIS® Web Services Common Implementation Specification [OGC 06-121r3]. Some of these common aspects are normatively referenced herein, instead of being repeated in this specification.

Figure 1 is a simple UML diagram summarizing the WPS interface. This class diagram shows that the WPS interface class inherits the getCapabilities operation from the OGCWebService interface class, and adds the DescribeProcess and Execute operations. (This capitalization of names uses the OGC/ISO profile of UML.) A more complete UML model of the WPS interface is provided in Annex C (informative).

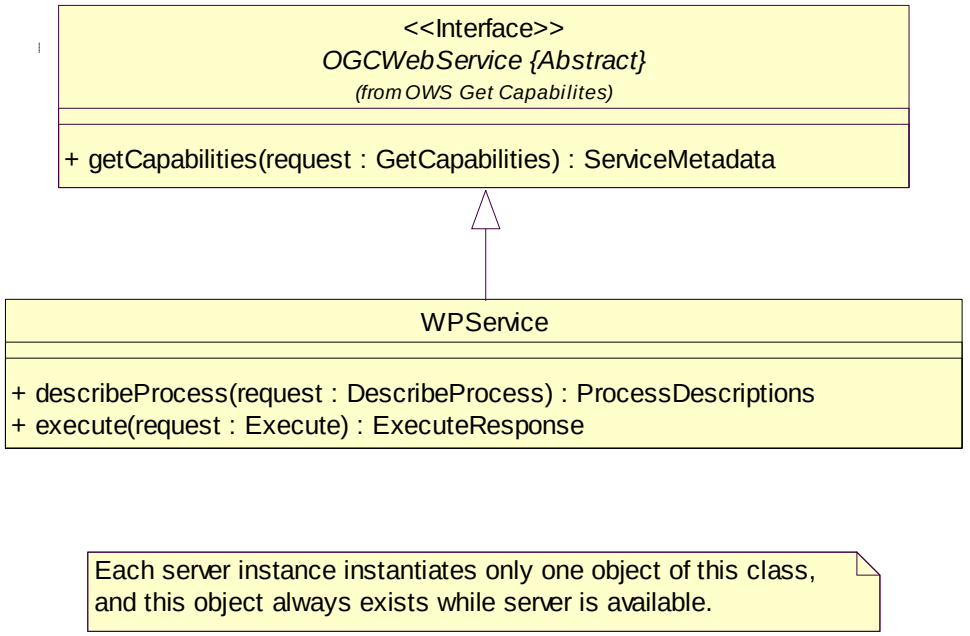

Figure 1 — WPS interface UML diagram

NOTE In this UML diagram, the request and response for each operation is shown as a single parameter that is a data structure containing multiple lower-level parameters, which are discussed in subsequent clauses. The UML classes modelling these data structures are included in the complete UML model in Annex C.

For example, consider the simple case of a process that can intersect two polygons. The response to a GetCapabilities request might indicate that the WPS supports an operation called "intersect", and that this operation is limited to intersecting one polygon with a second polygon. The response to a DescribeProcess request for the "intersect" process might indicate that it requires two inputs, namely: "FirstPolygon" and "SecondPolygon", and that these inputs must be provided in GML 2.2. Furthermore, the process will produce one output, in either GML 2.2, or GML 3.1, and it can be delivered as a web-accessible resource.

The client would run the process by calling the Execute operation, and might choose to provide the two input polygons embedded directly within the request, and identify that the output should be stored as a web-accessible resource. After completion, the process would return an ExecuteResponse XML document that identifies the inputs and outputs, indicates whether or not the process executed successfully, and if successful, contains a reference to the web-accessible resource.

Each of the three WPS operations is described in more detail in Clause 7 and subsequent clauses.

6.2 Generic nature of WPS

WPS is a generic interface in that it does not identify any specific processes that are supported. Instead, each implementation of WPS defines the processes that it supports,

as well as their associated inputs and outputs. WPS can be thought of as an abstract model of a web service, for which profiles need to be developed to support use, and standardized to support interoperability. As with the other OGC specifications GML and CAT, it is the development, publication, and adoption of profiles which define the specific uses of this specification.

WPS discovery and binding mechanisms follow the OGC model set by WMS and WFS, in that WPS defines a GetCapabilities operation, and requests are based on HTTP Get and Post. WPS does more than just describe the service interface, in that it specifies a request/response interface that defines how to:

- encode requests for process execution
- encode responses from process execution
- embed data and metadata in process execution inputs/outputs
- reference web-accessible data inputs/outputs
- support long-running processes
- return process status information
- return processing errors
- request storage of process outputs

6.3 Middleware nature of WPS

WPS allows for the provision of input data in two different methods. Data can either be embedded in the Execute request, or referenced as a web accessible resource. In the former approach, WPS acts as a stand-alone service. In the latter fashion, WPS acts as middleware service for data, by obtaining data from an external resource in order to run a process on the local implementation.

WPS allows existing software interfaces to be wrapped up and presented to the network as web services. Implementations of WPS can thus be considered middleware for software.

6.4 WPS Profiles

The WPS specification by itself allows service developers to reuse significant amounts of code in the development of web interfaces, while at the same time facilitating ease of understanding among web application developers. However, fully-automated interoperability can be achieved only through using standardized profiles. While it is possible to write a generic client for WPS, the use of a profile enables optimization of interoperable client user interface behaviour, as well as the publish/find/bind paradigm. To achieve high interoperability, each process shall be specified in an Application Profile of this specification.

A WPS Application Profile describes how WPS shall be configured to serve a process that is recognized by OGC. An Application Profile consists of

1. An OGC URN that uniquely identifies the process (mandatory)
2. A reference response to a DescribeProcess request for that process (mandatory).
3. A human-readable document that describes the process and its implementation (optional, but recommended).
4. A WSDL description for that process (optional).

WPS Application Profiles are intended for consumption by web service registries that maintain searchable metadata for multiple service instances.

Geospatial infrastructures can establish a geospatial processing web by specifying a repository that contains a semantically defined hierarchy of processes, each identified by a URN. A WPS Application Profile can define each unique process within the repository, and each WPS instance can refer to that URN. The current specification fully supports this approach to standardized semantically-driven service discovery.

6.5 Service chaining with WPS

A WPS process is normally an atomic function that performs a specific geospatial calculation. Chaining of WPS processes facilitates the creation of repeatable workflows. WPS processes can be incorporated into service chains in a number of ways:

1. A BPEL engine can be used to orchestrate a service chain that includes one or more WPS processes.
2. A WPS process can be designed to call a sequence of web services including other WPS processes, thus acting as the service chaining engine.
3. Simple service chains can be encoded as part of the execute query. Such cascading service chains can be executed even via the GET interface.

6.6 WPS and SOAP/WSDL

WPS is compatible with both WSDL and SOAP, and definitions for how to use WPS with these standards have been defined in this specification.

SOAP can be used to package WPS requests and responses. SOAP describes a message exchange mechanism which contains an env:body element, but it does not describe the contents of that body, i.e. the payload. WPS describes a message exchange mechanism that can be used if SOAP is not required (for security such as encryption or authentication), but it goes beyond SOAP by specifying what the payload should look like. Elements that are common to all payloads have been generalized in the WPS specification, and this standardization dramatically simplifies the amount of custom coding required to implement an interface for any new service. WPS enables the development of both software frameworks and generic clients. The use of SOAP to wrap

WPS requests offers the ability to add security certificates as well as encryption to web-based geoprocessing transactions.

WPS supports WSDL. WSDL identifies how a service should be described, but not what the service interface should look like. WPS describes a significant portion - the common portion - of what any service should look like. WSDL offers a less comprehensive but more widely adopted alternative to the publishing mechanism built in to the WPS interface specification. (WPS offers more documentation than can be published via WSDL, and more sophisticated service chaining capabilities.)

WPS supports the use of WSDL for an individual WPS process, as well as for the entire WPS instance that may include several processes. It is not possible to generate a single generic WSDL document that describes all WPS implementations, since WSDL requires specific binding information that is only found in WPS profiles. It is possible to use WPS without WSDL if dynamic binding to well known service instances (e.g. WPS profiles) is required. WSDL is required in order to facilitate dynamic binding to dynamic services (i.e. WPS instances with unknown profiles).

WPS offers the following advantages to an approach restricted to the current SOAP/WSDL specifications.

1. It supports the OGC GetCapabilities construct, which simplifies its adoption within the geospatial community that has already adopted OGC specs,

2. For a single output, it supports the direct return of that output without any XML wrapper, which allows for REST-like architecture while still enabling publish and find

3. It specifies how to determine the status of a process, which enables the support long-running processes.

4. It specifies exactly how to package and describe the inputs and outputs, which facilitates the development of reusable software frameworks and clients.

5. It specifies how to request storage of process outputs, which facilitates service chaining and subsequent retrieval.

6. It specifies how to reference web resources as inputs/outputs, which facilitates service chaining.

7. It specifies how to describe and embed complex inputs, which facilitates the development of reusable components to store and extract these inputs from a processing request.

8. It offers a service discovery mechanism that can be used without the overhead and complexity of WSDL, while at the same time supporting the option to use WSDL when required to facilitate discovery and binding.

9. It facilitates service chaining, since a WPS service can be constructed to call other services, including other WPS services.

10. It defines standard error messages, which simplifies the coding of error response handling for multiple processes.

11. It enables the client to choose whether to use a REST or SOAP architectural approach, since it specifies how to support both architectures from one service implementation.

7 Shared aspects

7.1 Introduction

This clause specifies aspects of WPS behavior that are shared by multiple operations.

The "Multiplicity and use" columns in the tables in this document specify whether a parameter or data structure must be present and populated in an operation request or response. All WPS servers shall implement each "mandatory" and "optional" parameter and data structure, checking that each request parameter and data structure is received with any allowed value(s). Similarly, all WPS clients shall implement each "mandatory" and "optional" parameter and data structure, using specified values.

7.2 Shared data structures

This clause specifies some of the data structures and parameters used by multiple operation requests and responses specified in the following clauses. The data structure names, parameter names, meanings, data types, and multiplicity shall be as specified in Table 1 and Table 2.

NOTE 1 The 3 parameters listed below (with partial grey backgrounds) are partially copied from Table 32 in Subclause 10.6.1 of [OGC 06-121r3].

Table 1 — Parameters in Description data structure

Name	Definition	Data type and value	Multiplicity and use
Identifier	Unambiguous identifier or name of a process, input, or output, unique for this server	ows:CodeType, as adaptation of MD_Identifier class in ISO 19115	One (mandatory)
Title	Title of a process, input, or output, normally available for display to a human	Character string type, not empty Includes xml:lang attribute	One (mandatory) [a]
Abstract	Brief narrative description of a process, input, or output, normally available for display to a human	Character string type, not empty Includes xml:lang attribute	Zero or one (optional) Include when available and useful
a When this element is not mandatory it is noted in the referring table.			

Table 2 — Parts of ProcessBrief data structure

Name	Definition	Data type and value	Multiplicity and use
Identifier	Inherited from Description data structure, see Table 1, applied to a process	ows:CodeType	One (mandatory)
Title		Character string type	One (mandatory)
Abstract		Character string type	Zero or one (optional)
Metadata	Reference to more metadata about this process	ows:Metadata, see Table 32 of OGC 06-121r3	Zero or more (optional) Include when useful
Profile	Profile to which the WPS process complies	URN type. E.g. OGC:WPS:somename	Zero or more (optional) [a]
WSDL	Location of a WSDL document which describes this process.	WSDLtype See Table 3	Zero or one (optional)
process Version	Release version of process (not of WPS specification)	ows:VersionType, see OGC 06-121r3	Zero or one (optional) Include when needed to identify process version [b]

a OGC will normally define only one profile URN to which a process corresponds. The ability to support multiple profile URNs is designed to support evolution in URNs and multiple URN authorities.

b The processVersion is informative only. Version negotiation for processVersion is not available. Requests to Execute a process do not include a processVersion identifier.

Table 3 — Parts of WSDL data structure

Name	Definition	Data type and value	Multiplicity and use
xlink:href	URL from which the WSDL document can be retrieved.	xlink:href type	One (mandatory)[a]

a The processVersion is informative only. Version negotiation for processVersion is not available. Requests to Execute a process do not include a processVersion identifier.

Table 4 — Parts of Format data structure

Name	Definition	Data type	Multiplicity
mimeType	Identification of mime type of this input or requested for this output parameter's value	Character String type, not empty ows:MimeType	Zero or one (optional) Include when format not in http header
encoding	Reference to encoding of this input or requested for this output	URI type	Zero or one (optional) Include when not default encoding
schema	Reference to XML Schema Document that specifies content model of input or output parameter's value	URL type	Zero or one (optional) Include when XML encoded resource

7.3 Operation request encoding

The encoding of operation requests shall use HTTP GET with KVP encoding and HTTP POST with XML encoding as specified in Clause 11 of [OGC 06-121r3]. Table 5 summarizes the three Service operations and their encoding methods defined in this specification.

Table 5 — Operation request encoding

Operation name	Request encoding
GetCapabilities (mandatory)	KVP and optional XML
DescribeProcess (mandatory)	KVP and optional XML
Execute (mandatory)	XML and optional KVP

8 GetCapabilities operation (mandatory)

8.1 Introduction

The mandatory GetCapabilities operation allows clients to retrieve service metadata from a server. The response to a GetCapabilities request shall be a XML document containing service metadata about the server, including brief metadata describing all the processes implemented. This clause specifies the XML document that a WPS server must return to describe its capabilities.

8.2 GetCapabilities operation request

The GetCapabilities operation request shall be as specified in Subclauses 7.2 and 7.3 of OWS Common [OGC 06-121r3]. The value of the "service" parameter shall be "WPS".

The "Multiplicity and use" column in Table 1 of [OGC 06-121r3] specifies the optionality of each listed parameter in the GetCapabilities operation request. Table 6 specifies the implementation of those parameters by WPS clients and servers.

Table 6 — Implementation of parameters in GetCapabilities operation request

Name	Multiplicity	Client implementation	Server implementation
service	One (mandatory)	Each parameter shall be implemented by all clients, using specified value.	Each parameter shall be implemented by all servers, checking that each parameter is received with specified value.
Request	One (mandatory)		
AcceptVersions	Zero or one (optional)	Should be implemented by all clients, using specified values.	Shall be implemented by all servers, checking if parameter is received with specified value(s).
language	Zero or one (optional)	Should be implemented by all clients	Should be implemented by servers offering multilingual capabilities

8.2.1 HTTP GET request using KVP encoding (mandatory)

All WPS servers shall implement HTTP GET transfer of the GetCapabilities operation request, using KVP encoding. WPS servers shall NOT implement KVP encoding using HTTP POST transfer.

EXAMPLE To request a WPS capabilities document, a client could issue the following KVP encoded GetCapabilities operation request:

```
http://foo.bar/foo?
    service=WPS&
    Request=GetCapabilities&
    AcceptVersions=1.0.0&
    language=en-CA
```

8.2.2 GetCapabilities HTTP POST request using XML encoding (optional)

WPS servers may also implement HTTP POST transfer of the GetCapabilities operation request, using XML encoding only. This capability is provided to support SOAP. The following schema specifies the contents and structure of a GetCapabilities operation request encoded in XML:

wpsGetCapabilities_request.xsd

POST：requestXML格式

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

examples\10_wpsGetCapabilities_request.xml

8.3 GetCapabilities operation response

8.3.1 Normal response

The service metadata document shall be an XML Capabilities document that contains the parameters and sections specified in Table 7.

NOTE The shaded areas in the following table are largely copied from section 7.4.2 of [OGC 06-121r3].

Table 7 —Parts of Capabilities document

Name	Definition	Data type and value	Multiplicity and use
service	Service Identifier	Character String type, not empty Shall contain "WPS"	One (mandatory)
version	Specification version for operation	Character String type, not empty Shall contain "1.0.0"	One (mandatory)
update Sequence	Service metadata document version, having values that are "increased" whenever any change is made in service metadata document.	Character String type, not empty. Values are selected by each server implementation.	Zero or One (optional)
lang	Language Identifier	Character string type, not empty RFC4646 language code of the human readable text	One (mandatory)
Service Identification	Metadata about this specific server.	The schema of this section shall be the same as for all OWSs, as specified in Subclause 7.4.4 and owsServiceIdentification.xsd of [OGC 06-121r3].	One (mandatory)
Service Provider	Metadata about the organization operating this server.	The schema of this section shall be the same for all OWSs, as specified in Subclause 7.4.5 and owsServiceProvider.xsd of [OGC 06-121r3].	One (mandatory)
Operations Metadata	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 almost the same as for all OWSs, as specified in Subclause 7.4.6 and owsOperationsMetadata.xsd of [OGC 06-121r3], modified as specified in subclause 8.3.2 below.	One (mandatory)
Process Offerings	Unordered list of brief descriptions of the processes offered by the server	ProcessOfferings data structure, see subclause 8.3.3 below.	One (mandatory)
Languages	Languages supported by the server	Languages data structure, see subclause 8.3.4 below.	One (mandatory)
WSDL	Location of a WSDL document describing all operations and processes offered by the server	WSDL data structure, see subclause 8.3.5 below.	Zero or One (optional)

8.3.2 OperationsMetadata section contents

For the WPS, the OperationsMetadata section shall be similar to all other OGC Web Services, as specified in Subclause 7.4.6 and owsOperationsMetadata.xsd of [OGC 06-121r3]. The operations that shall exist in all WPS servers and therefore shall be described in the OperationsMetadata section are shown in Table 8.

Table 8 — Operations described in the OperationsMetadata section

Operation name	Meaning
GetCapabilities	The GetCapabilities operation is implemented by this server.
DescribeProcess	The DescribeProcess operation is implemented by this server.
Execute	The Execute operation is implemented by this server.

8.3.3 ProcessOfferings section

The ProcessOfferings section of a WPS service metadata document shall contain a brief description of each of the processes offered by the service. The ProcessOfferings section shall include the subsections specified in Table 9.

Table 9 — Parts of ProcessOfferings section

Name	Definition	Data type	Multiplicity and use
Process	Brief description of process, not including input and output parameters	ProcessBrief data structure, see Table 2	One or more (mandatory) One for each process implemented by server

NOTE The UML class diagram contained in Subclause C.4 provides a graphical view of the contents of the ProcessOfferings section listed in Table 9.

8.3.4 Languages section

The Languages section of a WPS service metadata document shall contain a list of the default and optional languages offered by the service. The Languages section shall include the subsections specified in Table 10.

Table 10 — Parts of Languages section

Name	Definition	Data type	Multiplicity and use
Default	Identifies the default language that will be used unless the operation request specifies another supported language.	Includes one Language data structure, as specified in Table 11	One (mandatory)

Name	Definition	Data type	Multiplicity and use
Supported	Unordered list of references to all of the languages supported by this service. The default language shall be included in this list.	Includes one or more Language data structures, as specified in Table 11	One (mandatory)

Table 11 —Language data structure

Name	Definition	Data type	Multiplicity and use
Language	Identifier of a language supported by the server.	Character String type, not empty. This language identifier shall be as specified in IETF RFC 4646.	One (mandatory)

8.3.5 WSDL section

The WSDL section of a WPS service metadata document shall identify the location of a WSDL document which describes the entire service. The ProcessOfferings section shall include the subsections specified in Table 9.

Table 12 — Parts of WSDL section

Name	Definition	Data type	Multiplicity and use
href	The URL from which the WSDL document can be retrieved.	xlink:href type	One (mandatory)

NOTE It is also possible to describe each individual process supported by the service using separate WSDL documents that can be identified in the DescribeProcess response.

8.3.6 Capabilities document XML encoding

The XML schema for a WPS service metadata document is at:

wpsGetCapabilities_response.xsd

This XML schema extends ows:CapabilitiesBaseType in owsCommon.xsd of [OGC 06-121r3].

An example of a ProcessOfferings and a Languages section are included at the end of the following Capabilities document example. In order to obtain detailed information about a process, the DescribeProcess operation can be used.

EXAMPLE: A GetCapabilities operation response for WPS can look like this:

examples\20_wpsGetCapabilities_response.xml

比\10的复杂，？？

8.3.7 GetCapabilities exceptions

When a WPS server encounters an error while performing a GetCapabilities operation, it shall return an exception report message as specified in Clause 8 of [OGC 06-121r3]. The allowed exception codes shall include those listed in Table 5 of Subclause 7.4.1 of [OGC 06-121r3].

9 DescribeProcess operation (mandatory)

9.1 Introduction

The mandatory DescribeProcess operation allows WPS clients to request a full description of one or more processes that can be executed by the Execute operation. This description includes the input and output parameters and formats. This description can be used to automatically build a user interface to capture the parameter values to be used to execute a process instance.

9.2 DescribeProcess operation request

9.2.1 DescribeProcess request parameters

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

NOTE 1 The first three parameters listed below are largely copied from Table 26 in Subclause 9.2.1 of [OGC 06-121r3]. The Identifier parameter is partially copied from Table 31 in Subclause 10.6.1 of that document.

Table 13 — Parameters in DescribeProcess operation request

Name [a]	Definition	Data type and value	Multiplicity and use
service	Service type identifier	Character String type Value is "WPS"	One (mandatory)
request	Operation name	Character String type Value is "DescribeProcess"	One (mandatory)
version	Specification version for operation	Character String type, not empty Value is specified by each WPS Implementation Specification and Schemas version	One (mandatory)
language	Language identifier	Character string type, RFC4646 language code of the human readable text. Must be a language listed in the Capabilities Languages element.	Zero or one (optional)
Identifier	Identifier Process identifier	Character String type, not empty Value is process Identifier defined in ProcessOfferings section of service metadata (Capabilities) document	One or more (mandatory) One for each desired Process, unordered list

a The name capitalization rules being used here are specified in Subclause 11.6.2 of [OGC 06-121r3].

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

NOTE 3 The UML class diagrams contained in Subclause C.5 provides a graphical view of the contents of the DescribeProcess operation request listed in Table 13.

9.2.2 DescribeProcess HTTP GET request KVP encoding (mandatory)

All WPS servers shall implement HTTP GET transfer of the DescribeProcess operation request, using KVP encoding. The KVP encoding of the DescribeProcess operation request shall use the parameters specified in Table 14. The parameters listed in Table 14 shall be as specified in Table 13 above.

Table 14 — DescribeProcess operation request URL parameters

Name and example [a]	Optionality	Definition and format
service=WPS	Mandatory	Service type identifier
request=DescribeProcess	Mandatory	Operation name
version=1.0.0	Mandatory	WPS specification and schema version for this operation
Language=en-CA	Optional	Language of the human readable text in the response.
Identifier=intersection,union[b]	Mandatory	List of one or more process identifiers as listed in the Capabilities document, separated by commas

a All parameter names are here listed using mostly lower case letters. However, any parameter name capitalization shall be allowed in KVP encoding, see Subclause 11.5.2 of [OGC 06-121r3].

b The use of the Identifier ALL is restricted as an option for an Identifier in the DescribeProcess operation. When Identifier=ALL, the DescribeProcess operation shall return process descriptions for all processes served up by the WPS instance.

EXAMPLE An example DescribeProcess operation request KVP encoded for HTTP GET is:

http://foo.bar/foo?
Service=WPS&
Request=DescribeProcess&
Version=1.0.0&
Language=en-CA
Identifier=intersection,union

9.2.3 DescribeProcess HTTP POST request XML encoding (optional)

WPS servers may also implement HTTP POST transfer of the DescribeProcess operation request, using XML encoding only. The following schema specifies the contents and structure of a DescribeProcess operation request encoded in XML:

wpsDescribeProcess_request.xsd

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

examples\30_wpsDescribeProcess_request.xml

又一个XML

9.3 DescribeProcess operation response

9.3.1 DescribeProcess response parameters

The normal response to a valid DescribeProcess operation request shall be a ProcessDescriptions data structure, which contains one or more Process Descriptions for the requested process identifiers. Each Process Description includes the brief information returned in the ProcessOfferings section of the service metadata (Capabilities) document, plus descriptions of the input and output parameters. Each process can have any number of input and output parameters.

Each parameter is described by a data structure that specifies the allowable formats, encodings, and units of measure (when applicable). For each input parameter, the process can indicate that it needs one of the following:

a) "ComplexData" (such as XML or imagery), in one of the following allowable combinations of format (mimetype, encoding, and schema). The value of this complex data structure can be (either) directly encoded in the Execute operation request or made available through a web accessible URL.

b) "LiteralData", with a specified data type, allowable values, default value, and allowable unit of measure indicated.

c) BoundingBoxData, using one of the supported coordinate reference systems.

For each output parameter, the process can indicate similar information about the corresponding forms of output parameters. Again, there are three types of process outputs: ComplexOutput, LiteralOutput, and BoundingBoxOutput.

More precisely, a response from the DescribeProcess operation shall be a ProcessDescriptions data structure that includes one or more ProcessDescription data structures, as listed in

Table 15. The ProcessDescription data structure shall include the parts specified in Table 16 through Table 29. All these tables specify the UML model data type plus the multiplicity and use of each listed part in the DescribeProcess operation response.

The “Multiplicity and use” columns in the following tables specify the optionality of each listed parameter and data structure in the DescribeProcess operation response. Each “mandatory” parameter and data structure shall be implemented by all OWS servers, using a specified value(s). Each “optional” parameter and data structure shall also be implemented by all OWS servers, using specified values, for each implemented process for which that metadata is relevant and available.

Table 15 — Parts of ProcessDescriptions data structure

Name	Definition	Data type	Multiplicity and use
Process Description	Full description of process, including all input and output parameters	ProcessDescription data structure, see Table 16	One or more (mandatory) One for each Process identified in operation request
service	Service Identifier	Character String type, not empty Shall contain "WPS"	One (mandatory)
version	Specification version for operation	Character String type, not empty Value is specified by each Implementation Specification and Schemas version	One (mandatory)
lang	Language Identifier	Character string type, not empty RFC4646 language code of the human readable text	One (mandatory)

NOTE 1 The UML class diagrams contained in Subclause C.5 provide a useful graphical view of the contents of the ProcessDescriptions contents listed in

Table 15 through Table 30.

Table 16 — Parts of ProcessDescription data structure

Name	Definition	Data type and values	Multiplicity and use
Identifier	Inherited from ProcessBrief data structure, see Table 2	ows:CodeType	One (mandatory)
Title		Character string type	One (mandatory)
Abstract		Character string type	Zero or one (optional)
Metadata		ows:Metadata	Zero or more (optional)
Profile		URN type. E.g. OGC:WPS:somename	Zero or more (optional)
processVersion		ows:VersionType	One (mandatory)
WSDL	Location of a WSDL document that describes this process. An example is shown in Annex F section D5.	WSDL data structure, see Table 17	Zero or one (optional)
DataInputs	List of the required and optional inputs to this process	DataInputs data structure, see Table 18	Zero or one (optional) Include if any inputs [a]
ProcessOutputs	List of the required and optional outputs from executing this process	ProcessOutputs data structure, see Table 34	One (mandatory)
storeSupported	Indicates if complex data output(s) from this process can be requested to be stored by the WPS server as web-accessible resources	Boolean type Values are: true and false Default is false (return directly in response)	Zero or one (optional) Include when storage of outputs including the Execute response document is supported [b]
statusSupported	Indicates if Execute operation response can be returned quickly with status information	Boolean type Values are: true and false Default is false [c]	Zero or one (optional) Include when updating of status data is supported [d]

a In almost all cases, at least one process input is required. However, the process inputs list may be empty when all the inputs are predetermined fixed resources. In this case, those resources shall be identified in the Abstract parameter describing the process.

b By default, storage is not supported, and all outputs are returned encoded in the Execute response. If "storeSupported" is "true", the Execute operation request may request that the execute response and specific outputs of the process shall be stored as a web-accessible resources so that they can be retrieved as required. Support for storage of outputs is recommended to facilitate service chaining when outputs are large.

c By default, the server shall not update the Status element, and the Execute operation response shall not be returned until process execution is complete.

d The “statusSupported” parameter is used to support asynchronous requests via a type of polling. If “statusSupported” is "true", the server shall keep the Status element of the stored Execute response document up to date while the request is being processed. The client can poll the updated Execute operation response via the URL identified for this purpose in the Execute response document. If "false" the Status element shall not be updated by the server until processing has completed or failed. The ability to update the Status element is recommended for processes that are not instantaneous.

NOTE 2 The first 6 parameters listed above (with partially grey background) are copied from Table 2 in Subclause 7.2 of this document.

Table 17 — Parts of WSDL data structure

Name	Definition	Data type and values	Multiplicity and use
href	The URL from which the WSDL document can be retrieved.	URI type	One (mandatory)

Table 18 — Parts of DataInputs data structure

Name	Definition	Data type and values	Multiplicity and use
Input	Description of mandatory or optional input to this process	InputDescription data structure, see Table 19	One or more (mandatory) Include one for each possible process input, unordered

Table 19 — Parts of InputDescription data structure

Name	Definition	Data type and values	Multiplicity and use
Identifier	Inherited from Description data structure, see Table 1, applied to an input	ows:CodeType	One (mandatory)
Title		Character string type	One (mandatory)
Abstract		Character string type	Zero or one (optional)
minOccurs [a]	Minimum number of times that values for this parameter are required	nonNegativeInteger type "0" means the parameter is optional	One (mandatory)
maxOccurs [a]	Maximum number of times that this parameter may be present	positiveInteger type	One (mandatory)
Metadata	Reference to more metadata about this input	ows:Metadata, see Table 32 of OGC 06-121r3	Zero or more (optional) Include when useful
InputForm Choice	Identifies the type of this input, and provides supporting information	InputFormChoice data structure, see Table 20	One (mandatory)

a The minOccurs and maxOccurs parameters have similar semantics to the like-named XML Schema occurrence constraints.

NOTE 3 The first 3 parameters listed above (with partially grey background) are copied from Table 1 in Subclause 7.2 of this document.

Table 20 — Parts of InputFormChoice data structure

Name	Definition	Data type	Multiplicity
ComplexData	Indicates that this input shall be a complex data structure (such as a GML fragment), and provides lists of formats, encodings, and schemas supported [b]	**Supported-ComplexData data structure, see** Table 21	Zero or one (conditional) [a]
LiteralData	Indicates that this input shall be a simple literal value (such as an integer) that is embedded in the execute request, and describes the possible values	LiteralInput data structure, see Table 25	Zero or one (conditional) [a]
BoundingBox Data	Indicates that this input shall be a BoundingBox data structure that is embedded in execute request, and provides a list of the CRSs supported in these Bounding Boxes	SupportedCRSs data structure, see Table 31	Zero or one (conditional) [a]

a One and only one of these three items shall be included.

b The value of this complex data structure can be input either embedded in the Execute request or remotely accessible to the server.

Table 21 — Parts of ComplexData data structure

Name	Definition	Data type and values	Multiplicity and use
Default	Identifies the default Format, Encoding, and Schema supported for this input or output. The process shall expect input in or produce output in this combination of Format/Encoding/Schema unless the Execute request specifies otherwise. This element is mandatory.	Default Format data structure, see Table 22	One (mandatory)
Supported	Combination of format, encoding, and/or schema supported by process input or output.	Supported Format data structure, see Table 24	One (mandatory)[a]
maximum Megabytes	The maximum file size, in megabytes, of this input. If the input exceeds this size, the server will return an error instead of processing the inputs.	Integer	Zero or one (optional)

a This data structure shall be repeated for each combination of Format / Encoding / Schema that is supported for this Input / Output, including the default Format / Encoding / Schema combination.

Table 22 — Parts of Default Format data structure

Name	Definition	Data type and values	Multiplicity and use
Format	Identification of default Format for process input or output [a]	Format data structure, see Table 23	One (mandatory)
a The process shall expect input or produce output in this Format unless the Execute request specifies another supported Format.			

Table 23 — Parts of Format data structure

Name	Definition	Data type and values	Multiplicity and use
MimeType	Identification of default Format for process input or output [a]	Character String type, not empty ows:MimeType	One (mandatory)
Encoding	Reference to default encoding for process input or output [b]	URI type	Zero or one (optional)[c]
Schema	Reference to default XML Schema Document for process input or output [d]	URI type	Zero or one (optional) Include when encoded using XML schema [e]
a The process shall expect input or produce output in this MimeType unless the Execute request specifies another supported MimeType. b The process shall expect input or produce output using this encoding unless the Execute request specifies another supported encoding. c This element shall be included when the default Encoding is other than the encoding of the XML response document (e.g. UTF-8). This parameter shall be omitted when there is no Encoding required for this input/output. d The process shall expect input or produce output conformant with this XML element or type unless the Execute request specifies another supported XML element or type. e This element shall be omitted when there is no XML Schema associated with this input/output (e.g., a GIF file). This parameter shall be included when this input/output is XML encoded using an XML schema. When included, the input/output shall validate against the referenced XML Schema. Note: If the input/output uses a profile of a larger schema, the server administrator should provide that schema profile for validation purposes.			

Table 24 — Parts of Supported Format data structure

Name	Definition	Data type and values	Multiplicity and use
Format	Identification of Formats supported by process input or output [a]	Format data structure, see Table 23	One or more (mandatory)
a The process shall expect input or produce output in this Format unless the Execute request specifies another supported Format.			

Table 25 — Parts of LiteralInput data structure

Name	Definition	Data type	Multiplicity and use
DataType	Data Type of this output (or input)	ows:DataType type	Zero or one (optional)
UOMs	List of units of measure supported for this numerical output (or input)	UOMs data structure, see Table 26	Zero or one (optional) Include if the literal value has units of measure
LiteralValues Choice	Identifies type of literal input and provides supporting information	LiteralValuesChoice data structure, see Table 29	One (mandatory)
DefaultValue	Default value of this input, encoded in character string [a]	CharacterString, not empty	Zero or one (optional) Include when default exists
a The DefaultValue shall be understood to be consistent with the unit of measure selected in the Execute request. If the Execute request does not indicate a unit of measure, DefaultValue shall apply to the default unit of measure for this input.			

Table 26 — Parts of UOMs data structure

Name	Definition	Data type	Multiplicity and use
Default	Identifies default unit of measure of this or input [a]	Default UOM data structure, see Table 27	One (mandatory)
Supported	Units of measure supported for this input	Supported UOM data structure, see Table 28	One (mandatory)
a A specific input or output for a WPS instance will always have just one measure type (length, area, speed, weight, etc.).			

Table 27 — Parts of Default UOM data structure

Name	Definition	Data type	Multiplicity and use
UOM	Default unit of measure of this input	ows:UoM data structure	One (mandatory)

Table 28 — Parts of Supported UOM data structure

Name	Definition	Data type	Multiplicity and use
UOM	Unit of measure supported for this input	ows:UoM data structure	One or more (mandatory) Include for all of the UoMs supported for this input, including the default UoM.

Table 29 — Parts of LiteralValuesChoice data structure

Name	Definition	Data type	Multiplicity
AllowedValues	Indicates that are finite set of values and ranges allowed for this input, and contains ordered list of all valid values and/or ranges	ows:AllowedValues data structure	Zero or one (conditional) [a]
AnyValue	Indicates that any value is allowed for this input	ows:AnyValue data structure	Zero or one (conditional) [a]
Values Reference	References an externally defined finite set of values and ranges for this input	ValuesReference data structure, see Table 30	Zero or one (conditional) [a]
a One and only one of these three items shall be included.			

Table 30 — Parts of ValuesReference data structure

Name	Definition	Data type	Multiplicity and use
reference	URL from which this set of ranges and values can be retrieved	URI type	One (mandatory)
valuesForm	Reference to a description of the mimetype, encoding, and schema used for this set of values and ranges.	URI type	One (mandatory)

Table 31 — Parts of SupportedCRSs data structure

Name	Definition	Data type	Multiplicity and use
Default	Reference to the default coordinate reference system (CRS)	Default CRS data type, see Table 32	One (mandatory)
Supported	Reference to one coordinate reference system (CRS)	Supported CRS data type, see Table 33	One or more (optional) Include for each additional CRS supported

Table 32 — Parts of Default CRS data structure

Name	Definition	Data type	Multiplicity and use
CRS	Reference to one coordinate reference system (CRS)	URI type	One (mandatory)

Table 33 — Parts of Supported CRS data structure

Name	Definition	Data type	Multiplicity and use
CRS	Reference to one coordinate reference system (CRS)	URI type	One or more (mandatory) Include for all of the CRSs supported for this input, including the default CRS.

Table 34 — Parts of ProcessOutputs data structure

Name	Definition	Data type and values	Multiplicity and use
Output	Description of mandatory or optional output from executing this process	OutputDescription data structure, see Table 35	One or more (mandatory) Include one for each possible process output, unordered

Table 35 — Parts of OutputDescription data structure

Name	Definition	Data type	Multiplicity and use
Identifier	Inherited from Description data structure, see Table 1, applied to an output	ows:CodeType	One (mandatory)
Title		Character string type	One (mandatory)
Abstract		Character string type	Zero or one (optional)
Metadata	Reference to more metadata about this output	ows:Metadata, see Table 32 of OGC 06-121r3	Zero or more (optional) Include when useful
OutputFormChoice	Identifies the type of this output and provides supporting information	OutputFormChoice data structure, see Table 36	One (mandatory)

Table 36 — Parts of OutputFormChoice data structure

Name	Definition	Data type	Multiplicity
ComplexOutput	Indicates that this output shall be a complex data set (such as a GML fragment), and provides lists of formats, encodings, and schemas supported for this output [b, c]	**ComplexData data structure, see** Table 21	Zero or one (conditional) [a]
LiteralOutput	Indicates that this output shall be a simple literal value (such as an integer) that is embedded in execute response, and describes the possible values	LiteralOutput data structure, see Table 37	Zero or one (conditional) [a]
BoundingBox Output	Indicates that this output shall be a BoundingBox data structure that is embedded in execute response, and provides a list of the CRSs supported in these Bounding Boxes	SupportedCRSs data structure, see Table 31	Zero or one (conditional) [a]

a One and only one of these three items shall be included.

b The client can select from among the identified mime type, encodings, and schemas to specify the form of the output. This allows for complete specification of particular versions of GML, or image formats.

c The Execute request can indicate how the value of a complex data structure shall be output. The value may be either 1) embedded in the Execute operation response, 2) made available via a separate web-accessible resource that is referenced in the Execute operation response, or 3) returned in its raw form directly to the client instead of being embedded or referenced in an Execute operation response document. The intent of this behavior is to ensure flexibility to support a variety of client processing requirements. Support for direct response is intended for data types that are not coded in XML, or where recoding to the normal WPS execute response structure would complicate processing to no benefit (i.e. both client and server understand another encoding and the WPS execute response XML would simply act as a transport envelope.)

Table 37 — Parts of LiteralOutput data structure

Name	Definition	Data type	Multiplicity and use
DataType	Data type of this output (or input)	ows:DataType data structure	Zero or one (optional) Include when data type not character string
UOMs	List of units of measure supported of this numerical output (or input)	UOMs data structure, see Table 26	Zero or one (optional) Include when value(s) have a unit of measure

9.3.2 DescribeProcess response XML encoding

The wpsDescribeProcess_response.xsd schema specifies the contents and structure of a DescribeProcess operation response, always encoded in XML. The schema contains annotations that specify the meaning and use of each element and attribute.

wpsDescribeProcess_response.xsd

EXAMPLE: A DescribeProcess operation response for WPS can look like this encoded in XML:

examples\40_wpsDescribeProcess_response.xml

9.3.3 DescribeProcess exceptions

When a WPS server encounters an error while performing a DescribeProcess operation, it shall return an exception report message as specified in Subclause 8 of [OGC 06-121r3]. The allowed standard exception codes shall include those listed in Table 38. For each listed exceptionCode, the contents of the "locator" parameter value shall be as specified in the right column of Table 38.

NOTE To reduce the need for readers to refer to other documents, all the values listed below are copied from Table 25 in Subclause 8.3 of [OGC 06-121r3].

Table 38 — Exception codes for DescribeProcess operation

exceptionCode value	Meaning of code	"locator" value
MissingParameterValue	Operation request does not include a parameter value, and this server did not declare a default value for that parameter	Name of missing parameter
InvalidParameterValue	Operation request contains an invalid parameter value	Name of parameter with invalid value
NoApplicableCode	No other exceptionCode specified by this service and server applies to this exception	Omit "locator" parameter or specify the vendor specific exception code

10 Execute operation (mandatory)

10.1 Introduction

The mandatory Execute operation allows WPS clients to run a specified process implemented by a server, using the input parameter values provided and returning the output values produced. Inputs can be included directly in the Execute request, or reference web accessible resources. The outputs can be returned in the form of an XML response document, either embedded within the response document or stored as web accessible resources. If the outputs are stored, the Execute response shall consist of a XML document that includes a URL for each stored output, which the client can use to retrieve those outputs. Alternatively, for a single output, the server can be directed to return that output in its raw form without being wrapped in an XML reponse document.

Normally, the response document is returned only after process execution is completed. However, a client can instruct the server to return the Execute response document immediately following acceptance by the server of the Execute request. In this case, the Execute response includes a URL from which the response document can later be retrieved during and after process execution. The server can be instructed to provide

regular updates to a measure of the amount of processing remaining if the process is not complete. This allows the client to determine the process status by polling this URL. An example of how this works is shown in the UML activity diagram shown in Figure 2.

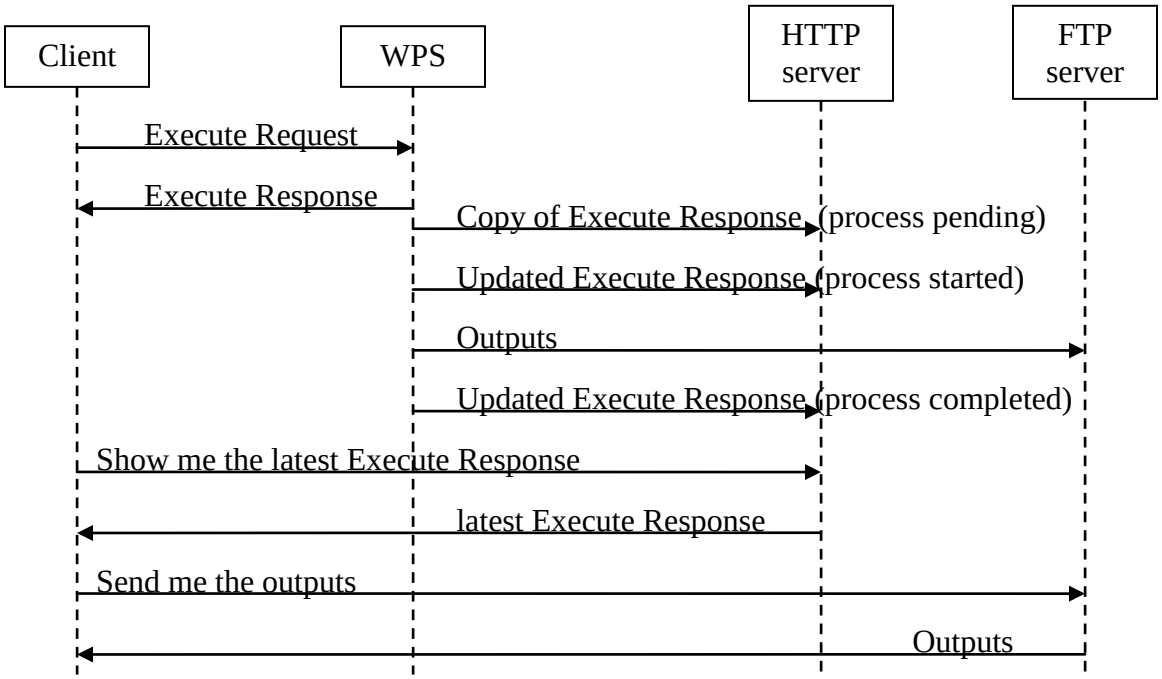

Figure 2 — Activity diagram when client requests storage of results

10.2 Execute operation request

10.2.1 Execute request parameters

A request to perform the Execute operation shall include the parameters listed and defined in Table 39 through Table 48. These tables specify the UML model data type, source of values, and multiplicity of each listed parameter in the operation request, plus the meaning to servers when each optional parameter is included. Although some values listed in the "Name" columns appear to contain spaces, they shall not contain spaces.

Table 39 — Parts of Execute operation request

Name	Definition	Data type and value	Multiplicity and use
service	Service type identifier	Character String type, not empty Value is OWS type abbreviation, namely "WPS"	One (mandatory)
request	Operation name	Character String type, not empty Value is operation name, namely "Execute"	One (mandatory)
version	Specification version for operation	Character String type, not empty Value is specified by each Implementation Specification and Schemas version	One (mandatory)
Identifier	Unambiguous identifier or name of a process	ows:CodeType, as adaptation of MD_Identifier in ISO 19115 Value is process Identifier used in Capabilities document.	One (mandatory)
DataInputs	List of inputs provided to this process execution	DataInputs data structure, see Table 40	Zero or one (optional) Include if any input [a]
Response Form	Defines the response type of the WPS, either raw data or XML document. If absent, the response shall be a response document which includes all outputs encoded in the response.	ResponseForm type data structure, see Table 49	Zero or one (optional) Include when rawDataOutput or non-default outputs are requested
language	Language identifier	Character string type, RFC4646 language code of the human readable text. Must be a language listed in the Capabilities Languages element.	One (optional)

a It is possible to have no inputs provided only when all the inputs are predetermined fixed resources. In all other cases, at least one input is required.

NOTE 1 The first three parameters listed above are largely copied from Table 26 in Subclause 9.2.1 of [OGC 06-121r3]. The Identifier parameter is largely copied from Table 1 in Subclause 7.2 of this document.

NOTE 2 The data type of some parameters is specified as "Character String type, not empty". In the XML Schema Documents specified herein, these parameters are encoded with the xsd:string type, which validates with a null value, contrary to the requirement that the string be "not empty".

NOTE 3 The UML class diagrams contained in Subclause C.6 provides a useful graphical view of the contents of the Execute operation request listed in Table 39 through Table 48.

The operation request provides support for multiple inputs. Each input refers to one of the forms of input that may be required for a single Execute request. The normal way to provide large inputs to a WPS is through providing one or more URIs (usually URLs) of input values, unless the inputs are simple scalar values. This is not intended to be used to facilitate batch processing (e.g., multiple images to be processed through a single

algorithm). If a process is to be run multiple times (probably using different inputs each time), each run shall be submitted as a separate Execute operation request.

Note that the list of process inputs and outputs is unsorted. This means that when large numbers of inputs or outputs exist for a process, an excessive overhead may be placed on server and client portal interface software as it attempts to interpret the incoming data stream and transform it as required. For such cases, the implementer is advised to package and sort the inputs / outputs in an efficient manner, and identify this optimal sorting requirement in the process description

Table 40 — Parts of DataInputs data structure

Name	Definition	Data type	Multiplicity and use
Input	Value of input to this process execution	InputType data structure, see Table 41	One or more (mandatory) Include one for each input, unordered

Table 41 — Parts of InputType data structure

Name	Definition	Data type	Multiplicity and use
Identifier	Description data structure applied to an input, see Table 1	ows:CodeType	One (mandatory)
Title		Character string type	Zero or one (optional) [a]
Abstract		Character string type	Zero or one (optional)
InputData Form Choice	Identifies the type of this input or output value, and provides supporting information	InputDataForm Choice data structure see Table 42	One (mandatory)
a Title for both inputs and outputs is optional in the Execute request. Title should be omitted in cases where a direct response is requested, or the title will not be used by the client			

NOTE 4 The first 3 parameters listed above (with partially grey background) are copied from Table 1 in Subclause 7.2 of this document.

Table 42 — Parts of InputDataFormChoice data structure

Name	Definition	Data type	Multiplicity
Reference	Identifies this input data as a web accessible resource, and references that resource	InputReference data structure, see Table 43	Zero or one (conditional) [a]
Data	Identifies this input data as being encapsulated in the Execute request	DataType data structure, see Table 46	Zero or one (conditional) [a]
a One and only one of these two items shall be included.			

Table 43 — Parts of InputReference data structure

Name	Definition	Data type	Multiplicity and Use
mimeType	Format data structure, see Table 4	Character String type	Zero or one (optional)
encoding		URI type	Zero or one (optional)
schema		URL type	Zero or one (optional)
href	Reference to web-accessible resource to be used as input	URL type	One (mandatory)
method	Identifies the HTTP method. Allows a choice of GET or POST. Default is GET.	Character String type	Zero or one (optional)
Header	Extra HTTP request headers needed by the service identified in ../Reference/@href. For example, an HTTP SOAP request requires a SOAPAction header. This permits the creation of a complete and valid POST request.	Header data structure, see Table 44	Zero or one (optional)
Body	The contents of this element to be used as the body of the HTTP request message to be sent to the service identified in ../Reference/@href. For example, it could be an XML encoded WFS request using HTTP POST, or a SOAP message.	Any type	Zero or one (conditional) [a]
Body Reference	Reference to a remote document to be used as the body of the an HTTP POST request message to the service identified in ../Reference/@href.	BodyReference data structure, see Table 45	Zero or one (conditional) [a]

a One and only one of these two items shall be included.

Table 44 — Parts of Header data structure

Name	Definition	Data type and values	Multiplicity and use
key	Key portion of the Key-Value pair in the HTTP request header.	String type, not empty	One (mandatory)
value	Value portion of the Key-Value pair in the HTTP request header.	String type, not empty	One (mandatory)

Table 45 — Parts of BodyReference data structure

Name	Definition	Data type and values	Multiplicity and use
href	Reference to a remote document to be used as the body of an HTTP POST request message. This attribute shall contain a URL from which this input can be electronically retrieved.	URI type	One (mandatory)

Table 46 — Parts of DataType data structure

Name	Definition	Data type	Multiplicity
ComplexData	Identifies this input or output data as a complex data structure, and provides that value	ComplexData data structure, see Table 47 [a]	Zero or one (conditional) [b]
LiteralData	Identifies this input or output data as a literal data of a simple quantity, and provides that data encoded in a character string	Character String type, not empty. Includes attributes identified in LiteralData data structure, see Table 48	Zero or one (conditional) b
BoundingBox Data	Identifies this input or output value as a BoundingBox data structure, and provides that value	ows:BoundingBox data structure, see Subclause 10.2 of [OGC 06-121r3]	Zero or one (conditional) [b]

a For an input, this element may be used by a client for any process input coded as ComplexData in the ProcessDescription. For an output, this element shall be used by a server whenever ComplexData output is returned in an execute response document.

b One and only one of these three items shall be included.

Table 47 — Parts of ComplexData data structure

Name	Definition	Data type	Multiplicity
mimeType	Uses Format data structure, see Table 4	Character String type	Zero or one (optional)
encoding		URI type	Zero or one (optional)
schema		URL type	Zero or one (optional)
(any) [a]	Complex value to be used as input to process	Any type	One (mandatory)

a The complex value is embedded here as part of the ComplexData element, in the mimeType, encoding, and schema indicated by the first three parameters if they exist, or by the relevant defaults.

Table 48 — Parts of LiteralData data structure

Name	Definition	Data type	Multiplicity
dataType	Identifier of data type of this literal value	URI type	Zero or one (optional) [a]
uom	Identifier of unit of measure of this literal numerical value	URI type	Zero or one (optional) [b]

a This dataType should be included for each quantity whose value is not a simple string.

b Where the literal data has units of measure which are necessary to state, such as when measuring distances, then that UoM shall be identified. When the literal data has no units then the UoM shall be omitted. This UOM shall be one identified in the Process's SupportedUOMs for this input or output parameter.

Table 49 — Parts of ResponseForm data structure

Name	Definition	Data type	Multiplicity
Response Document	Indicates that the outputs shall be returned as part of a WPS response document.	ResponseDocument data structure, see Table 50	Zero or one (conditional) [a]
RawData Output	Indicates that the output shall be returned directly as raw data, without a WPS response document.	RawDataOutput data structure, see Table 52	Zero or one (conditional) [a]
a One and only one of these two items shall be included.			

Table 50 — Parts of ResponseDocument data structure

Name	Definition	Data type	Multiplicity and use
store Execute Response	Indicates if the execute response document shall be stored.	Boolean type	Zero or one (optional) Default is “false” [a]
lineage	Indicates if the Execute operation response shall include the DataInputs and OutputDefinitions elements.	Boolean type	Zero or one (optional) Default is “false” [b]
status	Indicates if the stored execute response document shall be updated to provide ongoing reports on the status of execution.	Boolean type	Zero or one (optional) Default is “false” [c]
Output	Definition of format, encoding, and schema for output to be returned from this process	DocumentOutputDefinition data structure, see Table 51	One or more (mandatory) Include wherever non-default output values are requested [d]

a If "true" then the executeResponseLocation attribute in the execute response becomes mandatory, which will point to the location where the executeResponseDocument is stored. The service shall respond immediately to the request and return an executeResponseDocument as specified in section 10.3.1. The “storeExecuteResponse” parameter value “true” is recommended when a process takes a long time to execute. It is also recommended to make service chaining more efficient when the output(s) are large. However, this parameter should not be included unless the corresponding storeSupported parameter is included and is “true” in the ProcessDescription for this process.

b If lineage is "true" the server shall include in the execute response a complete copy of the DataInputs and OutputDefinition elements as received in the execute request. If lineage is "false" then these elements shall be omitted from the response.

c If status is "true" and storeExecuteResponse is "true", then the Status element of the execute response document stored at executeResponseLocation is kept up to date by the process, as specified in section 10.3.1.

d This OutputDefinition shall be repeated for each Output that offers a choice of format, and the client wishes to use one that is not identified as the default, and/or for each Output that the client wishes to customize the descriptive information about the output.

Table 51 — Parts of DocumentOutputDefinition data structure

Name	Definition	Data type	Multiplicity and use
mimeType	Uses Format data structure, see Table 4	Character String type	Zero or one (optional)[a]
encoding		URI type	Zero or one (optional)[a]
schema		URL type	Zero or one (optional)[a]
uom	Identifier of unit of measure requested for this output	URI type	Zero or one (optional)[b]
asReference	Specifies if this output should be stored by the process as a web-accessible resource.	Boolean type	Zero or one (optional)[c] Default is "false"
Identifier	Inherited from Description data structure, see Table 1, applied to an output	ows:CodeType	One (mandatory)
Title		Character string type	Zero or one (optional) [d]
Abstract		Character string type	Zero or one (optional) [e]

a A Format can be referenced when a client chooses to specify a format other than the default Format supported for a ComplexData output. This Format shall be a Format referenced for this output in the Process full description.

b A uom can be referenced when a client chooses to specify one of the uoms supported for a LiteralData output. This uom shall be a unit of measure referenced for this output in the Process full description.

c If asReference is "true", the server shall store this output so that the client can retrieve it as required. If store is "false", all the output shall be encoded in the Execute operation response document. This parameter shall not be included unless the corresponding "storeSupported" parameter is included and is "true" in the ProcessDescription for this process.

d This element should be used if the client wishes to customize the Title in the execute response. This element should not be used if the Title provided for this output in the ProcessDescription is adequate.

e This element should be used if the client wishes to customize the Abstract in the execute response. This element should not be used if the Abstract provided for this output in the ProcessDescription is adequate.

Table 52 — Parts of RawDataOutput data structure

Name	Definition	Data type	Multiplicity and use
Identifier	Inherited from Description data structure, see Table 1, applied to an output	ows:CodeType	One (mandatory) [a]
mimeType	Uses Format data structure, see Table 4	Character String type	Zero or one (optional)
encoding		URI type	Zero or one (optional)
schema		URL type	Zero or one (optional)
uom	Identifier of unit of measure requested for this output	URI type	Zero or one (optional)[b]

a A Format can be referenced when a client chooses to specify a format other than the default Format supported for a ComplexData output. This Format shall be a Format referenced for this output in the Process full description.

b A uom can be referenced when a client chooses to specify one of the uoms supported for a LiteralData output. This uom shall be a unit of measure referenced for this output in the Process full description.

NOTE 5 The parameters listed in Table 52 above are a subset of those found in Table 51 above.

10.2.2 Execute HTTP GET request KVP encoding (optional)

Servers may implement HTTP GET transfer of the Execute operation request, using KVP encoding. The KVP encoding of the Execute operation request shall use the parameters specified in Table 53. The parameters listed in Table 53 shall be as specified in Table 39 above. KVP encoding is suitable for simple Execute requests. More complex requests such as those which require the inclusion of embedded complex values should use the XML encoding.

Table 53 — Execute operation request URL parameters

Name and example [a]	Optionality and use	Definition and format
service=WPS	Mandatory	Service type identifier
request=Execute	Mandatory	Operation name
version=1.0.0	Mandatory	Specification and schema version for this operation
language=en-CA	Optional	Language identifier
Identifier=Buffer	Mandatory	Process identifier
DataInputs=[Object=@xlink:href=http%3A%2F%2Ffoo.bar%2Ffoo;BufferDistance=100]	Optional, include when one or more inputs provided	List of identifiers, attributes, and values of inputs to this process execution [b]
ResponseDocument=[BufferedPolygon]	Optional, include when a response document is desired	List of identifiers and attributes of outputs from this process execution [b]
RawDataOutput=[BufferedPolygon]	Optional, include when raw data output is required and there is only one output	Identifier and attributes of the output from this process execution [b]
storeExecuteResponse=true	Optional, include when the response document shall be stored	Boolean value that specifies if the Execute Response shall be stored as a web-accessible resource, as per Table 50.
lineage=true	Optional, include when lineage information shall be included in the response	Boolean value that specifies if lineage information shall be included in the response document, as per Table 50.
status=true	Optional, include when status element shall be updated in the response	Boolean value that specifies if status information shall be updated in the response document, as per Table 50.

a All parameter names are here listed using mostly lower case letters. However, any parameter name capitalization shall be allowed in KVP encoding, see Subclause 11.5.2 of [OGC 06-121r3].

b The value for this field shall be encoded as specified in section 10.2.2.1.

EXAMPLE An example Execute operation request using KVP encoding is:

```
http://foo.bar/foo?
   request=Execute&
   service=WPS&
   version=1.0.0&
   language=en-CA&
   Identifier=Buffer&
   DataInputs=Object=@xlink:href=http%3A%2F%2Ffoo.bar%2Ffoo;BufferDistance=10&
   ResponseDocument=BufferedPolygon&
   StoreExecuteResponse=true
```

10.2.2.1 Encoding of DataInput and Output values (mandatory)

Encoding of the DataInputs, ResponseDocument, and RawDataOutput value fields shall be as follows:

1. A semicolon (;) shall be used to separate one input from the next
2. An equal sign (=) shall be used to separate an input name from its value and attributes, and an attribute name from its value
3. An at symbol (@) shall be used to separate an input value from its attributes and one attribute from another.
4. Field names and attribute names are case sensitive. Incorrect field names and attribute names shall raise an InvalidParameterException.
5. Missing mandatory field names shall raise a MissingParameterValue.
6. All field values and attribute values shall be encoded using the standard Internet practice for encoding URLs [IETF RFC 1738].
7. References using HTTP POST shall not be supported in the KVP encoding.

10.2.2.1.1 Execute DataInput parameter KVP syntax

The DataInput parameter's value, in a KVP Execute request, shall conform to the following grammar (using EBNF (Extended Backus-Naur Form) notation [IETF RFC 2396]):

```
DataInputs := Input *( ";" Input )
Input := BoundingBox | Literal | Complex | Reference

BoundingBox := InputId "=" BoundingBoxValue
BoundingBoxValue := <As defined in OGC # 06-121r3 Subclause 10.2.3>

Literal := InputId "=" Value *( "@" LiteralAttribute )
LiteralAttribute := LiteralAttributeName "=" Value
LiteralAttributeName := "datatype" | "uom"

Complex := InputId "=" Value *( "@" ComplexAttribute )
ComplexAttribute:= ComplexAttributeName "=" Value
ComplexAttributeName := "mimetype" | "encoding" | "schema"

Reference := InputId "=" Value *( "@" ReferenceAttribute )
```

```
ReferenceAttribute:= ReferenceAttributeName "=" Value
ReferenceAttributeName := "href" | ComplexAttributeName

InputId := <Identifier of the input from the process description>
Value := <URL Encoded value being sent>
```

These rules result in a KVP encoded request of the following form:

```
http://host:port/path?name=value...
   &Request=Execute
   &Identifier=ProcessName
   &DataInputs=fieldName=value@attributeName=value@...;nextFieldName=value...
```

Examples (for clarity the values are not URL encoded):

Literal Data example

```
width=35@datatype=xs:integer@uom=meter
```

BoundingBox Data example

```
bboxInput=46,102,47,103,urn:ogc:def:crs:EPSG:6.6:4326,2
```

Reference example

```
fieldName=xml@Format=text/xml@Encoding=utf-8@Schema=xsd@asReference=true
```

10.2.2.2 Chaining of requests using KVP (mandatory)

HTTP requests using KVP encoding shall support the chaining of requests whereby a call to another web service is encoded within the value for a ComplexData href input. Encoding of the entire request to the ComplexData href shall ensure that the request is received and processed unambiguously.

WPS instances shall decode ComplexValueReference values and submit the decoded value to the network as an HTTP KVP GET request.

An example of service chaining via KVP follows. Consider the following WPS Execute request that provides a ComplexValueReference called complexFieldName to a process called Buffer:

```
http://foo.bar.1/wps?version=1.0.0&request=Execute&service=WPS&Identifier=Buffe
r&DataInputs=BufferDistance=100@datatype=integer@uom=meter;Object=http%3A%2F%2
Ffoo%2Ebar%2E2%2Fwps%3Frequest%3DExecute%26service%3Dwps%26version%3D1%
2E0%2E0%26Identifier%3DShpConvertToGML%26DataInputs%3DcomplexFieldName%
3Dhttp%253A%252F%252Ffoo%252Ebar%252Fshapefile%40Format%3Dtext%2Fxml%40
Encoding%3Dutf%2D8%40Schema%3Dgml@Format=text/xml@Encoding=utf-8@Schema=xsd
```

```
Since complexFieldName is defined for ShpConvertToGML as a
ComplexValueReference, the WPS at foo.bar.1 shall decode the input
```

```
value (shown in grey), which results in the following URL, which it
shall submit to the Internet as an HTTP GET request:

http://foo.bar.2/wps?request=Execute&service=wps&version=1.0.0&Identifier=ShpCo
nvertToGML&DataInputs=complexFieldName=http%3A%2F%2Ffoo%2Ebar%2Fshapefile@Fo
rmat=text/xml@Encoding=utf-8@Schema=gml
```

Likewise, this ShpConvertToGML process contains a ComplexValueReference for complexFieldName, so the WPS at foo.bar.2 shall decode the input value http%3A%2F%2Ffoo%2Ebar%2Fshapefile, resulting in the following URL, which it shall submit to the Internet as an HTTP GET request:

```
http://foo.bar/shapefile
```

As indicated in this example, chaining requires increased re-encoding with each level of the chain. The deeper the chain, the more complex the encoding in the request will be. Service chaining via KVP is most suitable for chaining two or three simple processes together. In deeper and more complex processes, the use of a workflow engine can facilitate error handling and permit better control of the processing of the chain.

10.2.3 Execute HTTP POST request XML encoding (mandatory)

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

wpsExecute_request.xsd

EXAMPLES: Some examples of an Execute operation request using XML encoding are:

examples\50_wpsExecute_request_RawDataOutput.xml

examples\51_wpsExecute_request_ResponseDocument.xml

examples\52_wpsExecute_request_ResponseDocument.xml

三个XML

10.3 Execute operation response

10.3.1 Execute response parameters

The form of the response to an Execute operation request depends on the value of the "ResponseForm" parameter in the execute request.

In the most primitive case, when a response form of "RawDataOutput" is requested, process execution is successful, and only one complex output is produced, then the Execute operation response will consist simply of that one complex output in its raw form returned directly to the client. For example, if in the case where a WPS process creates one GIF image as its output, that GIF image would be returned to the client as a direct response to the Execute request.

In all other cases, the response to a valid Execute operation request is an ExecuteResponse XML document. The contents of this ExecuteResponse document depend upon the value of the Execute/ResponseForm/ResponseDocument element, as follows.

If storeExecuteResponse is “true” (see Table 50), then the execute response document shall be stored at a web accessible URL. The executeResponseLocation attribute in the execute response becomes mandatory, which will point to the location where the Response Document is stored. The service shall respond immediately to the request by

- storing the ResponseDocument at a web accessible URL, as well as
- returning the ResponseDocument containing the executeResponseLocation to the client.

The status element which has five possible subelements (choice):ProcessAccepted, ProcessStarted, ProcessPaused, ProcessFailed and ProcessSucceeded, which are chosen and populated as follows:

1. If the process is completed when the initial executeResponseDocument is returned, the element ProcessSucceeded is populated with the process results.
2. If the process already failed when the initial executeResponseDocument is returned, the element ProcessFailed is populated with the Exception.
3. If the process has been paused when the initial executeResponseDocument is returned, the element ProcessPaused is populated.
4. If the process has been accepted when the initial executeResponseDocument is returned, the element ProcessAccepted is populated, including percentage information.
5. If the process execution is ongoing when the initial executeResponseDocument is returned, the element ProcessStarted is populated.

In case 3, 4, and 5, if status updating is requested, updates are made to the executeResponseDocument at the executeResponseLocation until either the process completes successfully or it fails. Regardless, once the process completes successfully, the ProcessSucceeded element is populated, and if it fails, the ProcessFailed element is populated.

If lineage is “true” (see Table 50), the Execute operation response shall include the DataInputs and OutputDefinitions elements. The server shall include in the execute response a complete copy of the DataInputs and OutputDefinition elements as received in the execute request. If lineage is "false" then these elements shall be omitted from the response.

If status is “true” (see Table 50), the stored execute response document shall be updated to provide ongoing reports on the status of execution. If status is "true" and storeExecuteResponse is "true" (and the server has indicated that both storeSupported and statusSupported are "true") then the Status element of the execute response document stored at executeResponseLocation is kept up to date by the process. While the execute response contains ProcessAccepted, ProcessStarted, or ProcessPaused, updates shall be made to the executeResponse document until either the process completes successfully (in which case ProcessSucceeded is populated), or the process fails (in which case ProcessFailed is populated). If status is "false" then the Status element shall not be updated until the process either completes successfully or fails. If status="true" and storeExecuteResponse is "false" then the service shall raise an exception.

If storage of a ComplexData output has not been requested via the “asReference” attribute (see Table 51), the ExecuteResponse document will contain that output. For ComplexData such as images included in the ExecuteResponse document will be encoded. If storage of a ComplexData output is requested, the ExecuteResponse will reference the web-accessible resource where the output can be retrieved.

An ExecuteResponse document returned by the Execute operation shall include the parts listed in Table 54 through Table 61. These tables also specify the UML model data type plus the multiplicity and use of each listed part in the Execute operation response.

NOTE 1 The service parameter listed below is largely copied from Table 26 in Subclause 9.2.1 of [OGC 06-121r3]. The Identifier parameter is largely copied from Table 1 in Subclause 7.2 of this document. The DataInpust and OutputDefinitions data structures are copied from Table 39 of this document.

Table 54 — Parts of ExecuteResponse data structure

Name	Definition	Data type and values	Multiplicity and use
service	Service Identifier	Character String type, not empty Shall contain "WPS"	One (mandatory)
version	Specification version for operation	Character String type, not empty Value is specified by each Implementation Specification and Schemas version	One (mandatory)
lang	Language Identifier	Character string type, not empty RFC4646 language code of the human readable text	One (mandatory)
status Location	Reference to location where current ExecuteResponse document is stored	URL type	Zero or one (optional) Include when storeExecuteResponse= TRUE
service Instance	GetCapabilities URL of the WPS service which was invoked	URL type	One (mandatory)
Process	Process description	ProcessBrief data structure, see Table 2	One (mandatory)
Status	Execution status of this process	Status data structure, see Table 55	One (mandatory) Provide ongoing updates when status=TRUE
DataInputs	List of inputs provided to this process execution	DataInputs data structure, see Table 40	Zero or one (optional) Include if lineage=TRUE [a]
Output Definitions	List of definitions of outputs desired from executing this process	OutputDefinitions data structure, see Table 58	Zero or one (optional) Include if lineage=TRUE [b]
Process Outputs	List of values of outputs from process execution	ProcessOutputs data structure, see Table 59	Zero or one (optional) Include when process execution succeeded

a This DataInputs data structure can be requested to be included in the Execute response, so the client can confirm that the request was received correctly, and to provide a source of metadata if the client wishes to store the result for future reference.

b This OutputDefinitions data structure can be requested to be included in the Execute response, so the client can confirm that the request was received correctly, and to provide a source of metadata if the client wishes to store the result for future reference.

NOTE The UML class diagrams contained in Subclause C.6 provide a useful graphical view of the contents of the ExecuteResponse listed in Table 54 throughTable 61.

Table 55 — Parts of Status data structure

Name	Definition	Data type and values	Multiplicity
creationTime	The time (UTC) that the process finished. If the process has not completed executing, this attribute shall contain the creation time of this document.	dateTime type	One (mandatory)
Process Accepted	Indicates that process has been accepted by server, but is in a queue and has not yet started to execute	Character string type, not empty [b]	Zero or one (conditional) [a]
Process Started	Indicates that process has been accepted by server, and processing has begun	Character string type, not empty [b] Uses attributes identified in the ProcessStarted data structure, see Table 56	Zero or one (conditional) [a]
Process Paused	Indicates that the server has paused the process.	Character string type, not empty [b] Uses attributes identified in the ProcessStarted data structure, see Table 56	Zero or one (conditional) [a]
Process Succeeded	Indicates that process has successfully completed execution	Character string type, not empty [c]	Zero or one (conditional) [a]
ProcessFailed	Indicates that execution of this process has failed, and includes error information [d]	ProcessFailed data structure, see Table 57	Zero or one (conditional) [a]

a One and only one of these four elements can be present

b The contents of this human-readable text string is left open to definition by each server, but is expected to include any messages the server wishes to let the clients know. Such information could include how long the queue is, or any warning conditions that may have been encountered. The client may display this text to a human user.

c The contents of this human-readable text string is left open to definition by each server, but is expected to include any messages the server wished to let the clients know, such as how long the process took to execute, or any warning conditions that may have been encountered. The client may display this text string to a human user. The client should use the presence of this parameter to trigger automated or manual access to the results of process execution. If manual access is intended, the client should use the presence of this parameter to present the results as downloadable links to the user.

d If a process fails for some reason, the implementation raises an error and places it in the exception report included in this ProcessFailed structure. The reason(s) for failure is given in the exception report.

Table 56 — Parts of ProcessStarted data structure

Name	Definition	Data type and values	Multiplicity and use
percent Completed	Percentage of process that has been completed, where 0 means the process has just started, and 99 means the process is almost complete. This value is expected to be accurate to within ten percent	Integer type Values from 0 to 99, inclusive	Zero or one (optional) Include for processes expected to take a long time [a]
a Recommended for use when more than a minute will elapse between request and response. Note that when percentCompleted reaches 100, the ProcessSucceeded element shall be used.			

Table 57 — Parts of ProcessFailed data structure

Name	Definition	Data type	Multiplicity
Exception Report	Exception report containing one or more Exception data structures, each signalling detection of an independent error	ExceptionReport data structure, see Table 23 in OGC 06-121r3	One (mandatory)

Table 58 — Parts of OutputDefinitions data structure

Name	Definition	Data type	Multiplicity and use
Output	Definition of an output from Execute request	DocumentOutputDefinition data structure, see Table 51	One or more (mandatory) Include one for each output, unordered.

Table 59 — Parts of ProcessOutputs data structure

Name	Definition	Data type	Multiplicity and use
Output	Value of output from process execution	OutputData data structure, see Table 60	One or more (mandatory) Include one for each output, unordered. [a]
a It is only necessary to include outputs when the status is ProcessSucceeded.			

Table 60 — Parts of OutputData data structure

Name	Definition	Data type	Multiplicity and use
Identifier	Inherited from Description data structure, see Table 1, applied to an output	ows:CodeType	One (mandatory)
Title		Character string type	One (mandatory)
Abstract		Character string type	Zero or one (optional)
Reference	Indicates that the output is available as a web-accessible reference	OutputReference data structure, see Table 61	Zero or one (conditional) [a]
Data	Indicates that the output is directly embedded in the execute response document.	DataType data structure, see Table 46	Zero or one (conditional) [a]
a One and only one of these two elements shall be present.			

Table 61 — Parts of OutputReference data structure

Name	Definition	Data type	Multiplicity and Use
format	Format data structure, see Table 4	Character String type	Zero or one (optional)
encoding		URI type	Zero or one (optional)
schema		URL type	Zero or one (optional)
href	Reference to a web-accessible resource that is provided by the process as output. This attribute shall contain a URL from which this output can be electronically retrieved.	URL type	One (mandatory)

Once a process has completed successfully, the Status data structure shall include the ProcessSucceeded parameter, and the ProcessOutputs data structure shall be fully populated. If the Execute request indicates asReference="true" for an output, that output shall be stored as web-accessible resource which is indicated using /ExecuteResponse/ProcessOutputs/Output/Reference@href, as shown in this example XML fragment:

```
<Reference href="http://foo.bar/foo.xml"/>
```

If the Execute request indicates status="true", the status portion of the Execute Response document shall be updated, specifically the information described in Table 56.

If the URL indicated in statusLocation is accessed before the process has had time to complete and populate the online resource, the server shall return an HTTP Error 404 (Not Found).

The Execute Response document normally contains the input that was provided by the client, in the DataInputs and OutputDefinitions data structures. This information includes any URIs provided in the execute request. If an input was embedded in the request, then

the server may generate and populate a URL for the payload and reference it, instead of returning a copy of the original payload in the body of the ExecuteResponse. This is recommended in the case of large files.

The WPS is not specifically designed to store outputs for a long time, but it does not preclude such storage either. Clients may wish to download the outputs to some other web-accessible location if long term storage is required.

10.3.2 Execute response XML encoding

The following schema specifies the contents and structure of an Execute operation response, always encoded in XML:

wpsExecute_response.xsd

EXAMPLE: For the Execute operation request example given in the earlier example:

examples\52_wpsExecute_request_ResponseDocument.xml

an example response is:

examples\62_wpsExecute_response.xml

This example response includes the statusLocation as an attribute of the <ExecuteResponse>. This attribute contains a URL that will return an ExecuteResponse document, which contains the latest status information about the Execute request, and, if the process has completed, the URL at which the "BufferedPolygon" output may be retrieved. If the process has not completed by the time the response is sent, the location of the output will not necessarily be identified. In this example, because the execute request also specifies status="true", the URL at which the Response Document is located will be populated when the response document is first returned to the client, and the status portion of the Response Document at this URL shall be repopulated on a regular basis until processing is complete, at which time the location of the output will be identified.

10.3.3 Execute exceptions

When a WPS server encounters an error while performing an Execute operation, it shall return an exception report message as specified in Subclause 8 of [OGC 06-121r3]. The allowed standard exception codes shall include those listed in Table 62. For each listed exceptionCode, the contents of the "locator" parameter value shall be as specified in Table 62.

NOTE To reduce the need for readers to refer to other documents, the first four values listed below are copied from Table 25 in Subclause 8.3 of [OGC 06-121r3].

Table 62 — Exception codes for Execute operation

exceptionCode value	Meaning of code	"locator" value
MissingParameterValue	Operation request does not include a parameter value, and this server did not declare a default value for that parameter	Name of missing parameter
InvalidParameterValue	Operation request contains an invalid parameter value	Name of parameter with invalid value
NoApplicableCode	No other exceptionCode specified by this service and server applies to this exception	Omit "locator" parameter or specify the vendor specific exception code
NotEnoughStorage	The server does not have enough space available to store the inputs and outputs associated with the request.	None, omit "locator" parameter
ServerBusy	The server is too busy to accept and queue the request at this time.	None, omit "locator" parameter
FileSizeExceeded	The file size of one of the input parameters was too large for this process to handle.	Identifer of the parameter which exceeded the maximum file size.
StorageNotSupported	Execute operation request included store="true", bot storage is not offered by this server.	None, omit "locator" parameter
VersionNegotiationFailed	Service version for a ComplexData xlink:href input was not supported by the referenced server, and version negotiation failed.	Identifier of the Input which could not be accessed

If the client requests the use of a format for an input/output that is not supported by the implementation (i.e. identified in the Process Description), the implementation shall raise an InvalidParameterValue error. The exception's locator shall name the parameter and its description shall indicate that the requested format is not supported.

If an input identified by a client does not match the schema identified for that input, the implementation shall raise an InvalidParameterValue error. The exception's locator shall name the parameter and its description shall specify it as an XML schema fault.

EXAMPLE: An example exception response to the Execute operation request is:

examples\90_wpsExceptionReport.xml

Annex A
(normative)

Abstract test suite

EDITOR'S NOTE This annex is currently a first draft largely copied from OWS Common, and needs to be further customized and reviewed.

A.1 Introduction

Each OWS Implementation Specification is required to include an abstract test suite annex before it is submitted to ISO/TC 211. This abstract test suite specifies at a high level how server and client implementations of that specification shall be tested for conformance to the WPS specification. The framework for such abstract test suites is specified in ISO 19105: Geographic information – Conformance and testing, especially Clauses 7 and 9.

An abstract test suite contains multiple abstract tests, grouped into one or more test modules. This abstract test suite consists of three top-level test modules:

a) Client test module – Abstract tests for checking conformance of client implementations with the requirements of this specification that are normatively referenced by an OWS Implementation Specification

b) Server test module – Abstract tests for checking conformance of server implementations with the requirements of this specification that are normatively referenced by an OWS Implementation Specification

Any of these modules could contain lower-level test modules. At this time, only the Server test module contains lower-level test modules, named:

a) All operations implemented test module – Abstract tests for checking server properties that are common to all operations implemented

b) GetCapabilities operation test module – Abstract tests for checking server properties that are specific to the GetCapabilities operation

c) Other operations responses – Abstract tests for checking server properties that apply to all operations except GetCapabilities

In the client and server test modules, all operations specified and implemented shall be tested, including both HTTP GET and HTTP POST transfer of each operation request. In the specification test module, all operations specified shall be checked, including GET and POST transfers of operation requests. And all operation request and response parameters specified or implemented shall be tested. Of course, some operations, transfer methods, and parameters are specified as optional implementation by servers. Any

optional item not implemented by a server shall not be tested. Also, items not implemented by a client shall not be tested.

A.2 Client test module

A.2.1 GetCapabilities operation request

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

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

c) Reference: 8.2

d) Test Type: Basic

A.2.1 DescribeProcess operation request

a) Test Purpose: Verify that a client satisfies all requirements for a DescribeProcess operation request.

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

c) Reference: 9.2

d) Test Type: Basic

A.2.2 Execute operation request

a) Test Purpose: Verify that a client satisfies all requirements on each operation request other than the Execute operation.

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

c) Reference: 10.2

d) Test Type: Basic

A.3 Server test module

A.4.1 All operations implemented test module

A.4.1.1 HTTP protocol usage

a) Test purpose: Verify that the rules and conventions governing the use of HTTP are observed.

b) Test method: Check that the server responds to HTTP requests. Verify that the server accepts and responds to valid and invalid requests according to HTTP standards.

c) Reference: RFC 2616 (*Hypertext Transfer Protocol -- HTTP/1.1*). See <http://www.ietf.org/rfc/rfc2616>.

d) Test type: Capability

A.4.1.2 HTTP response status code

a) Test purpose: Verify that a service request which generates an exception produces a response that contains 1) a service exception report, and 2) a status code indicating an error.

b) Test method: Check the response code in the Status-Line and the message body. Pass if the response code is either 4xx (Client error) or 5xx (Server error) and the body contains a service exception report. Fail otherwise.

c) Reference: RFC 2616, Section 11.

d) Test type: Capability

A.4.2 GetCapabilities operation test module

A.4.2.1 Accept HTTP GET transferred KVP GetCapabilities operation request

a) Test Purpose: Verify that a server accepts at least HTTP GET transferred requests for the GetCapabilities operation.

b) Test Method: Submit KVP-encoded GetCapabilities and other operation requests containing parameter names using various cases and combinations of cases, with a variety of parameter sequences. Verify that the server provides the same response when the same parameter names use different cases and combinations of cases.

c) Reference: 7.3

d) Test Type: Capability

A.4.2.2 Accept HTTP POST transferred XML GetCapabilities operation request

e) Test Purpose: Verify that a server accepts at HTTP POST transferred requests for the GetCapabilities operation if advertised in the GetCapabilities Response.

f) Test Method: Submit XML-encoded GetCapabilities and other operation requests containing parameters using correct and incorrect name capitalizations and parameter sequences and contents. Verify that the server accepts all correct requests, and returns ExceptionReport messages for all incorrect requests.

g) Reference: 8.2.2

h) Test Type: Capability

A.4.2.3 GetCapabilities operation response

a) Test Purpose: Verify that a server satisfies all requirements on the GetCapabilities operation response.

b) Test Method: Make several GetCapabilities requests including a variety of input parameters. Verify that the specified correct response is returned to each request.

c) Reference: 8.3

d) Test Type: Capability

A.4.2.4 Version negotiation

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

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

c) Reference: 8.2

d) Test Type: Capability

A.4.2.5 Handling updateSequence parameter

a) Test Purpose: Verify that a server satisfies the requirements for generating and using the updateSequence parameter, if the server implements the AcceptFormats request parameter.

b) Test Method: Submit GetCapabilities operation requests containing correct and incorrect values of the AcceptFormats parameter. Verify that the server provides the specified correct response to each request.

c) Reference: 8.2

d) Test Type: Capability

A.4.2.6 Language selection

a) Test Purpose: Verify that a server satisfies the requirements for using the Language parameter.

b) Test Method: Submit GetCapabilities operation requests containing various values and combinations of values of the Language parameter. Verify that the server provides the specified correct response to each request

c) Reference: 8.2

d) Test Type: Capability

A.4.3 DescribeProcess operation test module

A.4.3.1 Accept DescribeProcess HTTP GET transferred operation requests

a) Test Purpose: Verify that a server accepts at least HTTP GET transferred requests for the DescribeProcess operation.

b) Test Method: Submit HTTP GET transferred requests for the DescribeProcess operation. Verify that the server accepts and responds to these requests as specified and implemented.

c) Reference: 9.2.2

d) Test Type: Capability

A.4.3.2 Accept DescribeProcess HTTP POST transferred operation requests

a) Test Purpose: Verify that a server accepts at HTTP POST transferred requests for the DescribeProcess operation.

b) Test Method: Submit HTTP POST transferred requests for the DescribeProcess operation. Verify that the server accepts and responds to these requests as specified and implemented.

c) Reference: 9.2.3

d) Test Type: Capability

A.4.3.3 DescribeProcess operation response

a) Test Purpose: Verify that a server satisfies all requirements on the DescribeProcess operation response.

b) Test Method: Make several DescribeProcess requests including a variety of input parameters. Ensure requests include inputs and outputs that consist of ComplexData, LiteralData, and Bounding Box data. Verify that the specified correct response is returned to each request.

c) Reference: 9.3

d) Test Type: Capability

A.4.3.4 Language selection

a) Test Purpose: Verify that a server satisfies the requirements for using the Language parameter for the DescribeProcess operation.

b) Test Method: Submit DescribeProcess operation requests containing various values and combinations of values of the Language parameter. Verify that the server provides the specified correct response to each request

c) Reference:

d) Test Type: Capability

A.4.4 Execute operation test module

A.4.4.1 Accept Execute HTTP GET transferred Execute operation requests

a) Test Purpose: Verify that a server accepts HTTP GET transferred requests for the Execute operation, if advertised in the GetCapabilities Response.

b) Test Method: Submit HTTP GET transferred requests for the Execute operation. Verify that the server accepts and responds to these requests as specified and implemented.

c) Reference: 10.2.2

d) Test Type: Capability

A.4.3.2 Accept Execute HTTP POST transferred operation requests

a) Test Purpose: Verify that a server accepts at HTTP POST transferred requests for the Execute operation.

b) Test Method: Submit HTTP POST transferred requests for the Execute operation. Verify that the server accepts and responds to these requests as specified and implemented.

c) Reference: 10.2.3

d) Test Type: Capability

A.4.3.3 Execute operation response: raw data output

a) Test Purpose: Verify that a server satisfies all requirements on the Execute operation response for raw data output.

b) Test Method: Make several Execute requests including a variety of input parameters. Ensure requests include inputs and outputs that consist of ComplexData, LiteralData, and Bounding Box data. Verify that the specified correct response is returned to each request.

c) Reference: **Error! Reference source not found.**

d) Test Type: Capability

A.4.3.4 Execute operation response: response document

e) Test Purpose: Verify that a server satisfies all requirements on the Execute operation response for response document output.

f) Test Method: Make several Execute requests including a variety of input parameters. Ensure requests include inputs and outputs that consist of ComplexData, LiteralData, and Bounding Box data. Ensure requests include inputs and outputs that are embedded and referenced. Verify that the specified correct response is returned to each request.

g) Reference: 10.3.2

h) Test Type: Capability

A.4.3.5 Execute operation response: updating of response document

i) Test Purpose: Verify that a server updates the Execute operation response document (if supported as indicated by the statusSupported attribute).

j) Test Method: Submit an Execute requests that takes a significant amount of time to process, and ensure that the stored Execute response document is maintained correctly.

k) Reference: 10.3.2

l) Test Type: Capability

A.4.3.6 Language selection

a) Test Purpose: Verify that a server satisfies the requirements for using the Language parameter for the Execute operation.

b) Test Method: Submit Execute operation requests containing various values and combinations of values of the Language parameter. Verify that the server provides the specified correct response to each request

c) Reference:

d) Test Type: Capability

Annex B
(normative)

XML Schema Documents

In addition to this document, this specification includes normative XML Schema Document files. The WPS-specific XML Schema Documents are posted online at the URL http://schemas.opengis.net/wps/1.0.0. These XML Schema Documents are also bundled with the present document. In the event of a discrepancy between the bundled and online versions of the XML Schema files, the online files shall be considered authoritative.

The WPS abilities now specified in this document use a set of XML Schema Documents, all included in the zip file with this document. These XML Schema Documents combine the XML schema fragments listed in various subclauses of this document, eliminating duplications. These XML Schema Documents are named:

wpsAll.xsd (complete WPS package)

wpsDescribeProcess_request.xsd

wpsDescribeProcess_response.xsd

wpsExecute_request.xsd

wpsExecute_response.xsd

wpsGetCapabilities_request.xsd

wpsGetCapabilities_response.xsd

common/DescriptionType.xsd

common/ProcessBriefType.xsd

common/ProcessVersion.xsd

common/RequestBaseType.xsd

common/ResponseBaseType.xsd

common/WSDL.xsd

These XML Schema Documents use and build on the OWS Common XML Schema Documents specified in [OGC 06-121r3] and named:

ows19115subset.xsd

owsCommon.xsd

owsDataIdentification.xsd

owsDomainType.xsd

owsExceptionReport.xsd (slightly modified from owsExceptionReport.xsd)

owsGetCapabilitiesTypes.xsd (slightly modified from owsGetCapabilities.xsd)

owsOperationsMetadata.xsd

owsServiceIdentification.xsd

owsServiceProvider.xsd

In order to validate properly, two OWS Common schema files were modified as indicated above. For this reason, all of the OWS Common schema files are also included in the zip file with this document.

All these XML Schema Documents contain documentation of the meaning of each element and attribute, and this documentation shall be considered normative as specified in Subclause 11.6.3 of [OGC 06-121r3].

Annex C
(informative)

UML model

C.1 Introduction

This annex provides a UML model of the WPS interface, using the OGC/ISO profile of UML summarized in Subclause 5.2 of [OGC 06-121r3].

Figure C.1 is a simple UML diagram summarizing the WPS interface. This class diagram shows that the WPService class inherits the getCapabilities operation from the OGCWebService interface class, and adds the "describeProcess" and "execute" operations. (This capitalization of names uses the OGC/ISO profile of UML.)

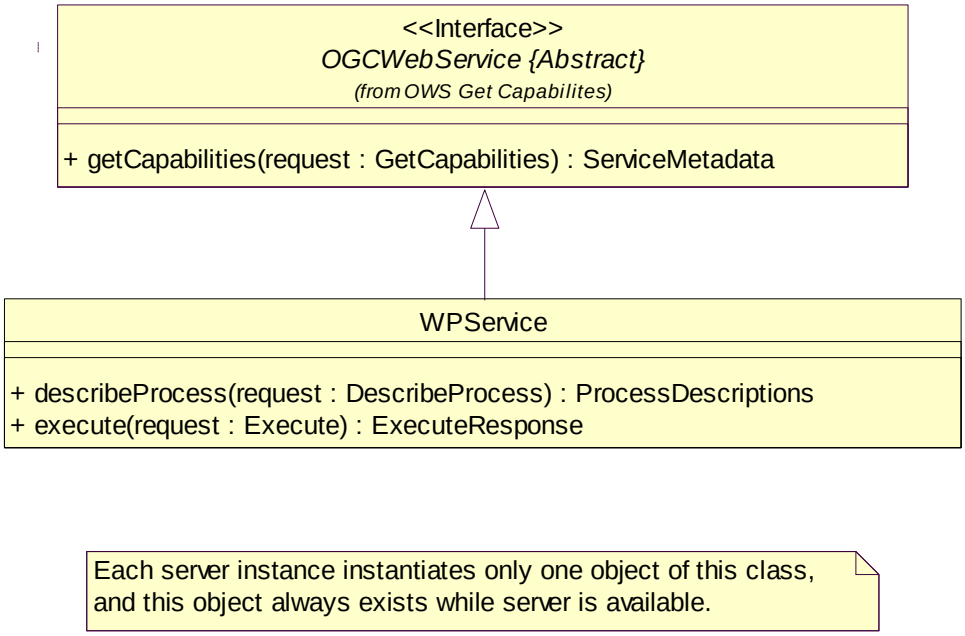

Figure C.1 — WPS interface UML diagram

Each of the three operations uses a request and a response data type, each of which is defined by one or more additional UML classes. The following subclauses provide a more complete UML model of the WPS interface, adding UML classes defining the operation request and response data types.

C.2 UML packages

The WPS interface UML model is organized in four packages:

- WPS Service
- WPS Get Capabilities
- WPS Describe Process, and
- WPS Execute.

These four WPS-specific packages make direct use of four OWS Common packages, named OWS Get Capabilities, ISO 19115 Subset, OWS Common, and OWS Domain.

Each of the four WPS-specific packages identified above is described in the following subclauses. The OWS Get Capabilities, OWS Common, OWS Operations Metadata, OWS Service Identification, OWS Service Provider, and ISO 19115 Subset packages are described in Annex C of [OGC 06-121r3].

C.3 WPS Service package

The WPS Service package is shown in the class diagram in Figure C.3. This diagram does not show the classes used by the WPS operation requests and responses, which are shown (with part of this package) in the WPS Get Capabilities, Describe Process, and Execute packages. This diagram also shows four used classes from other packages. The WPSDescription and ProcessBrief classes introduced by this package are further defined by Table 1 and Table 2 in this document.

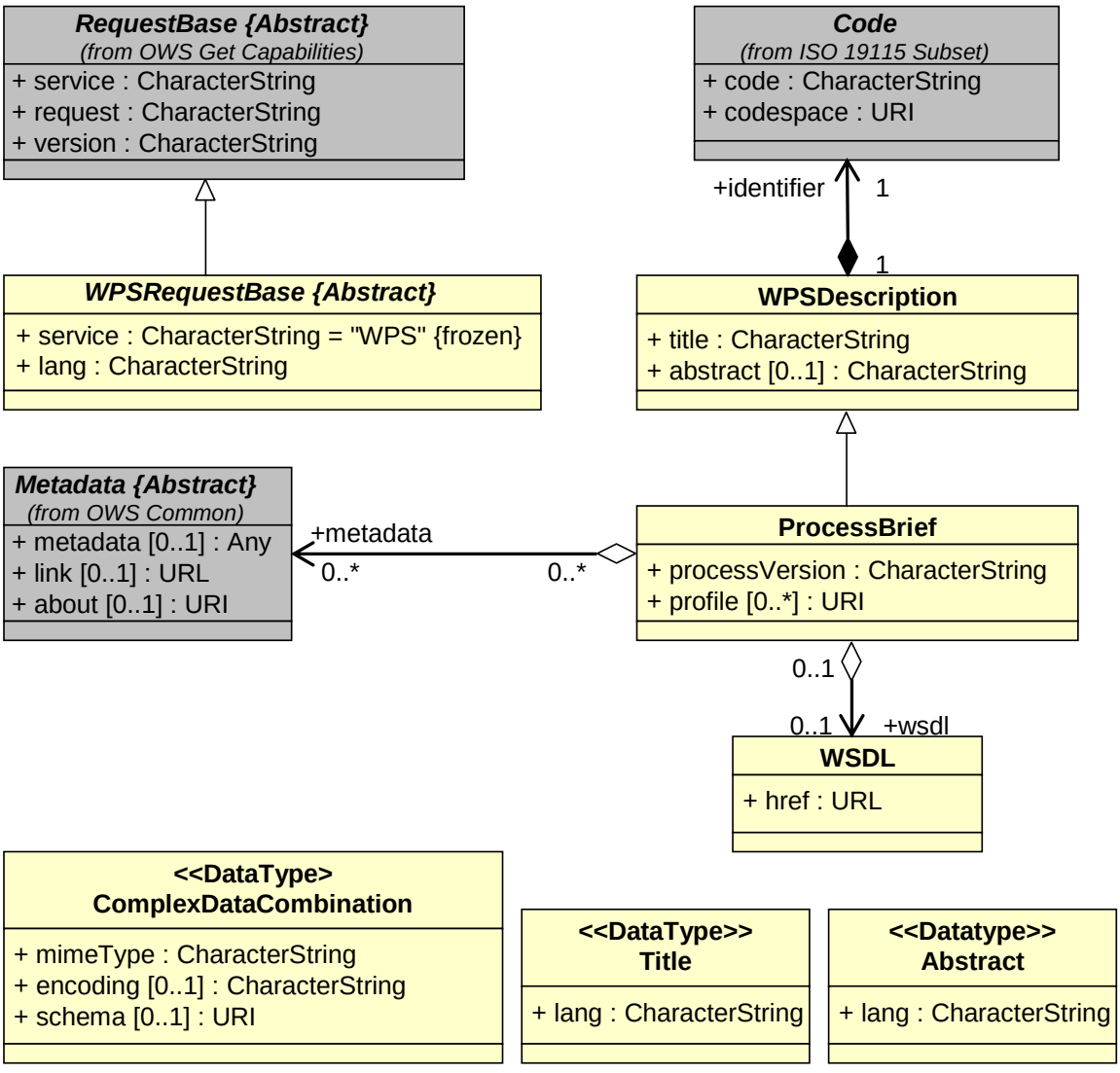

Figure C.3 — WPS Service package class diagram

C.4 WPS Get Capabilities package

The WPS Get Capabilities package is shown in the class diagram in Figure C.4. This diagram also shows many classes from the other packages, with gray fill.

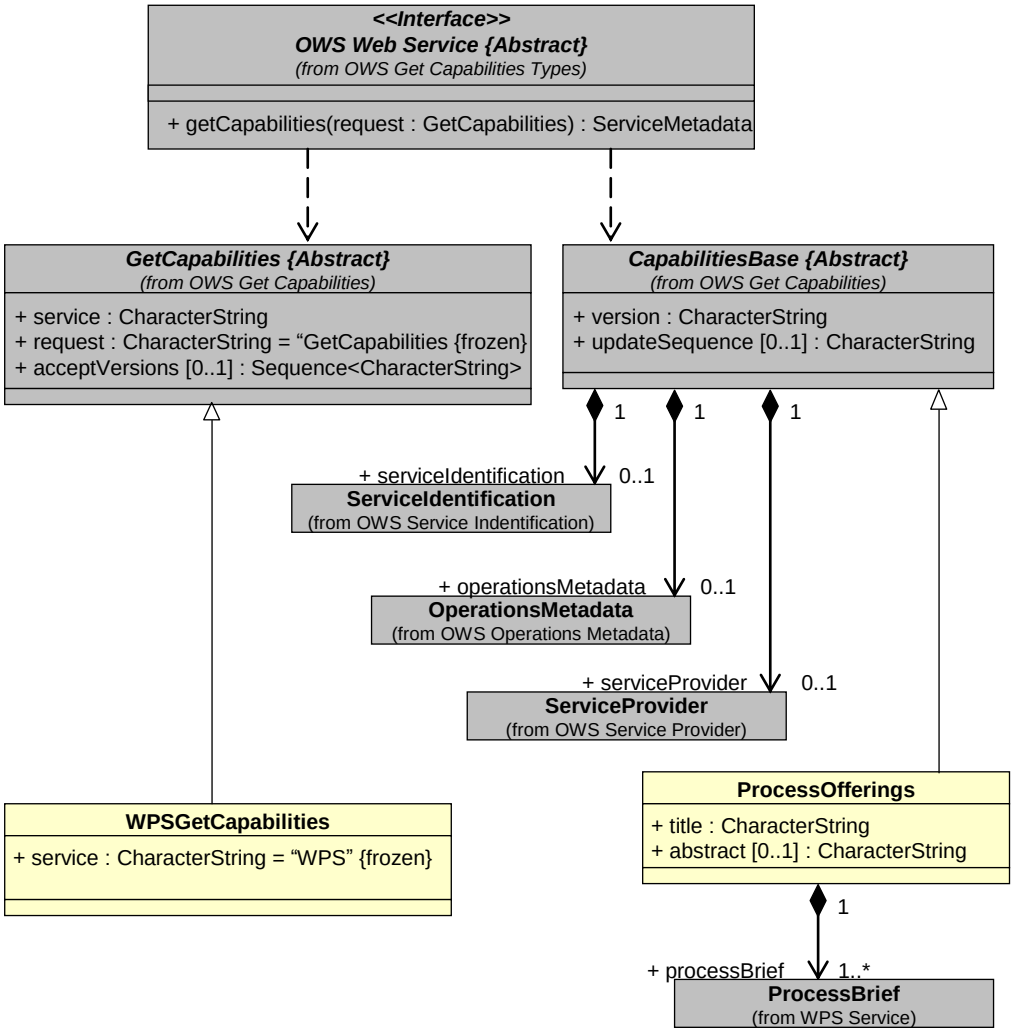

Figure C.4 — WPS Get Capabilities package class diagram

C.5 WPS Describe Process package

The Describe Process package is shown in the class diagrams in Figures C.5 and C.6. These diagrams also show many classes from the other packages, with gray fill. The classes introduced by this package are further defined by Table 13 through Table 37 in this document.

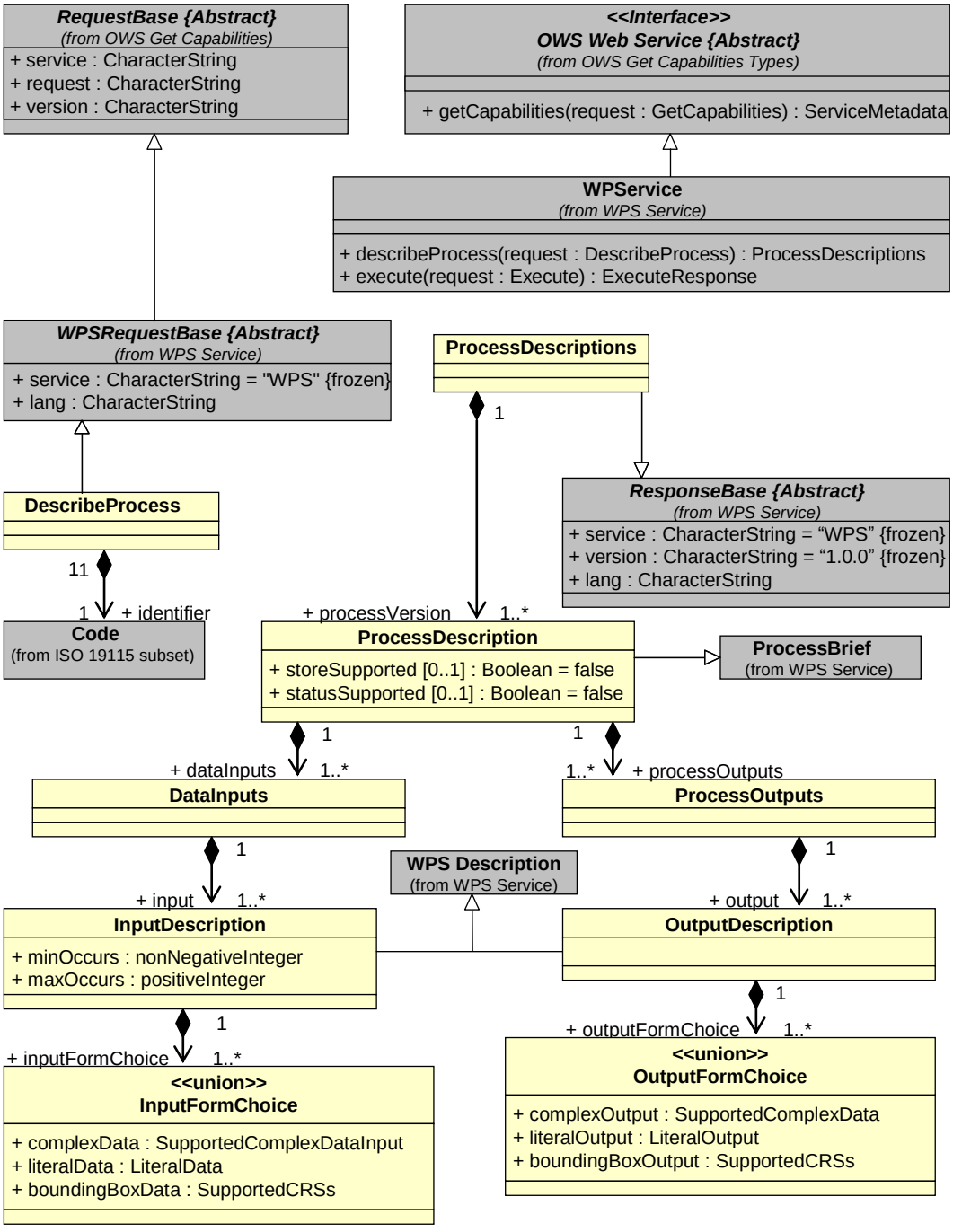

Figure C.5 — Describe Process package class diagram, part 1

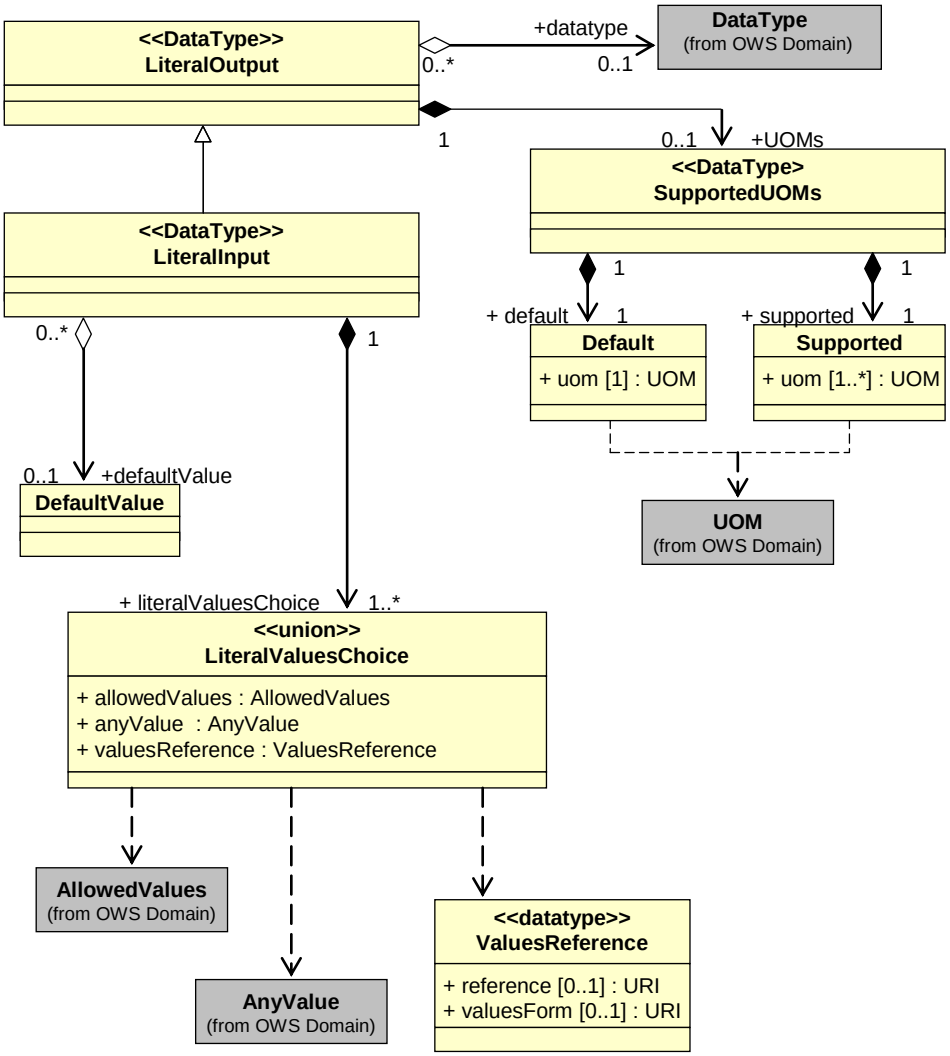

Figure C.6 — Describe Process package class diagram, part 2

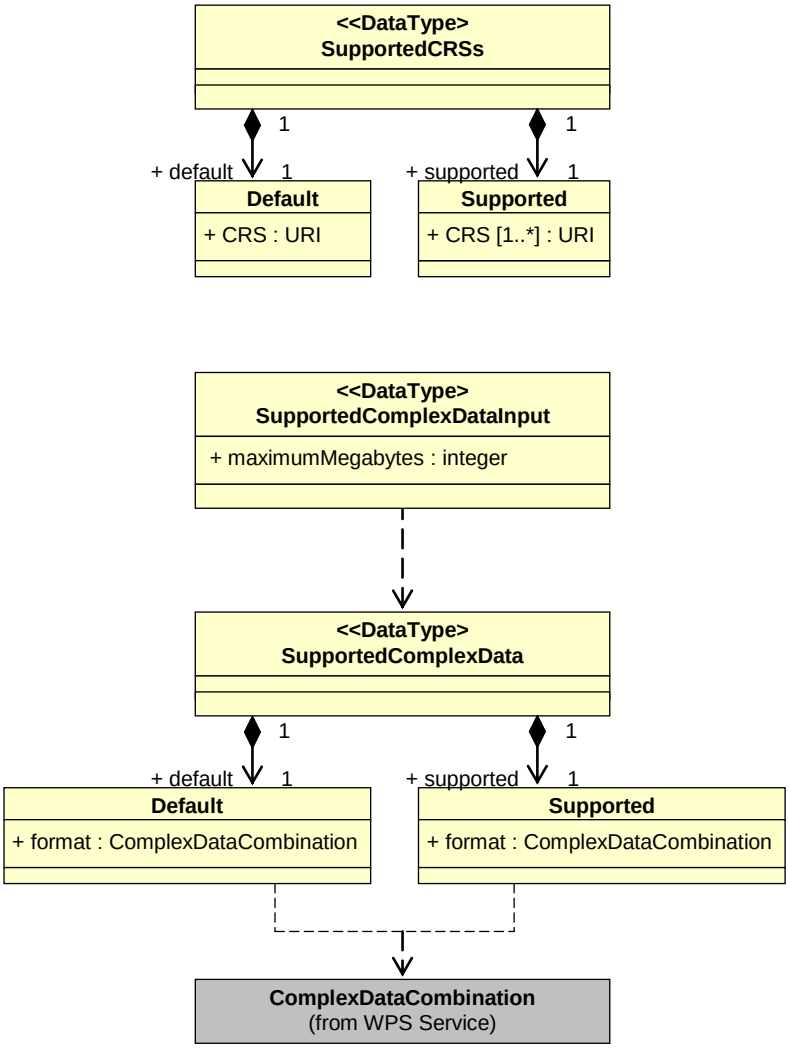

Figure C.7 — Describe Process package class diagram, part 3

C.6 WPS Execute package

The Execute package is shown in the class diagrams in Figures C.8, C.9 and C.10. These diagrams also show many classes from the other packages, with gray fill. The classes introduced by this package are further defined by Table 39 through Table 61 in this document.

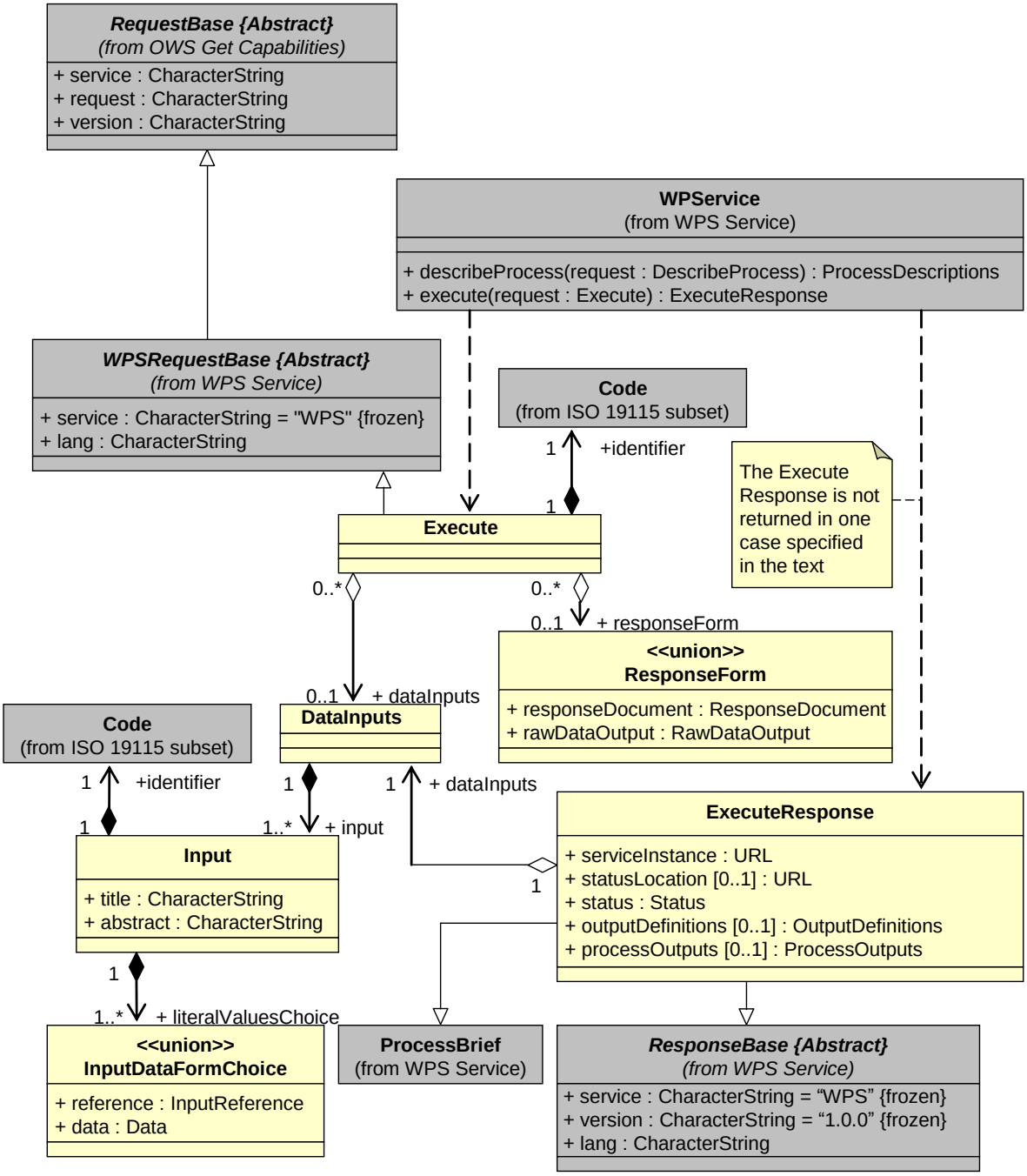

Figure C.8 — Execute package class diagram, part 1

As indicated by the note in Figure C.7, the response to an Execute operation request is not the ExecuteResponse document in one special case. When RawDataOutput is requested, no ExecuteResponse XML document is returned or stored.

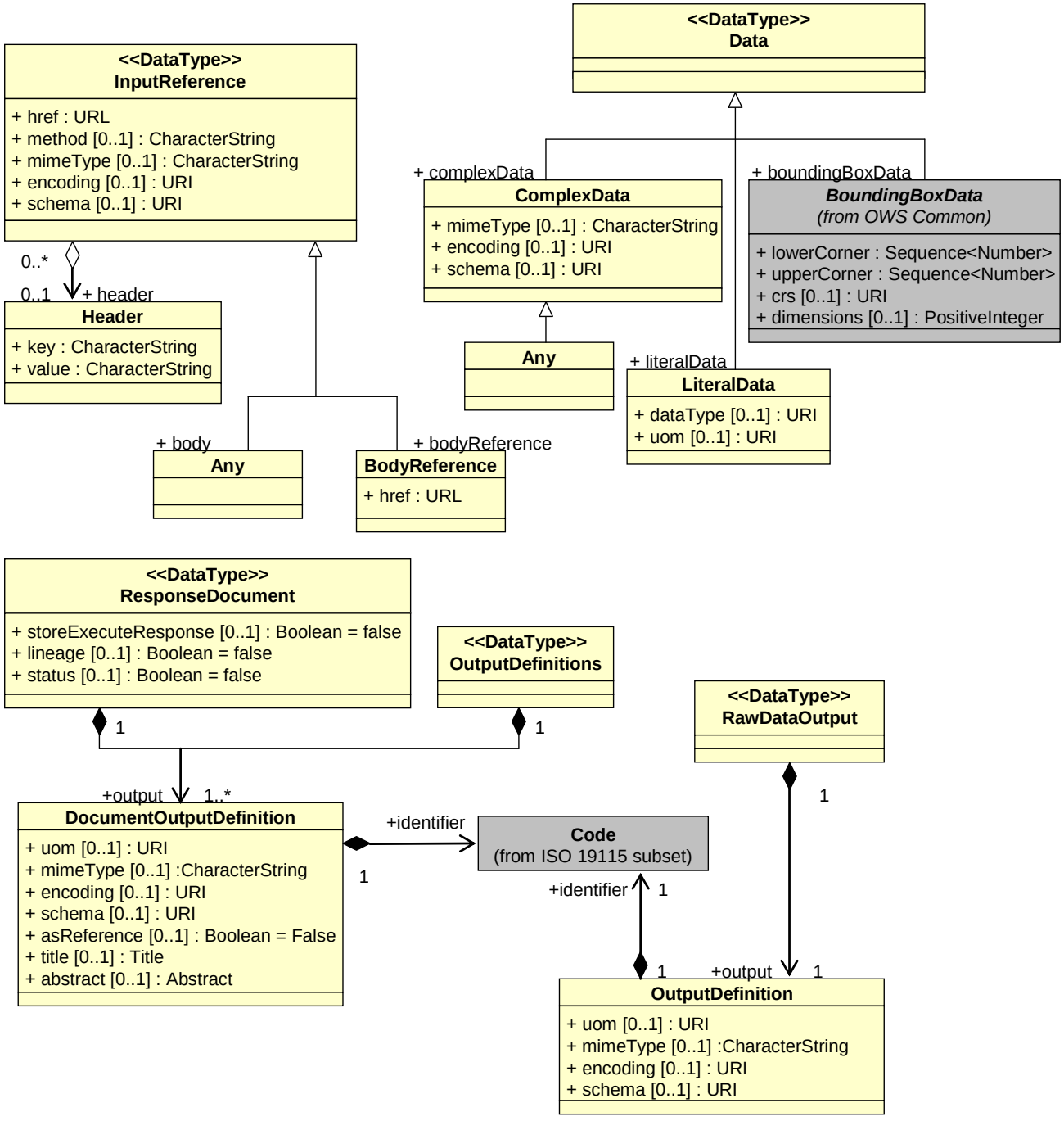

Figure C.9 — Execute package class diagram, part 2

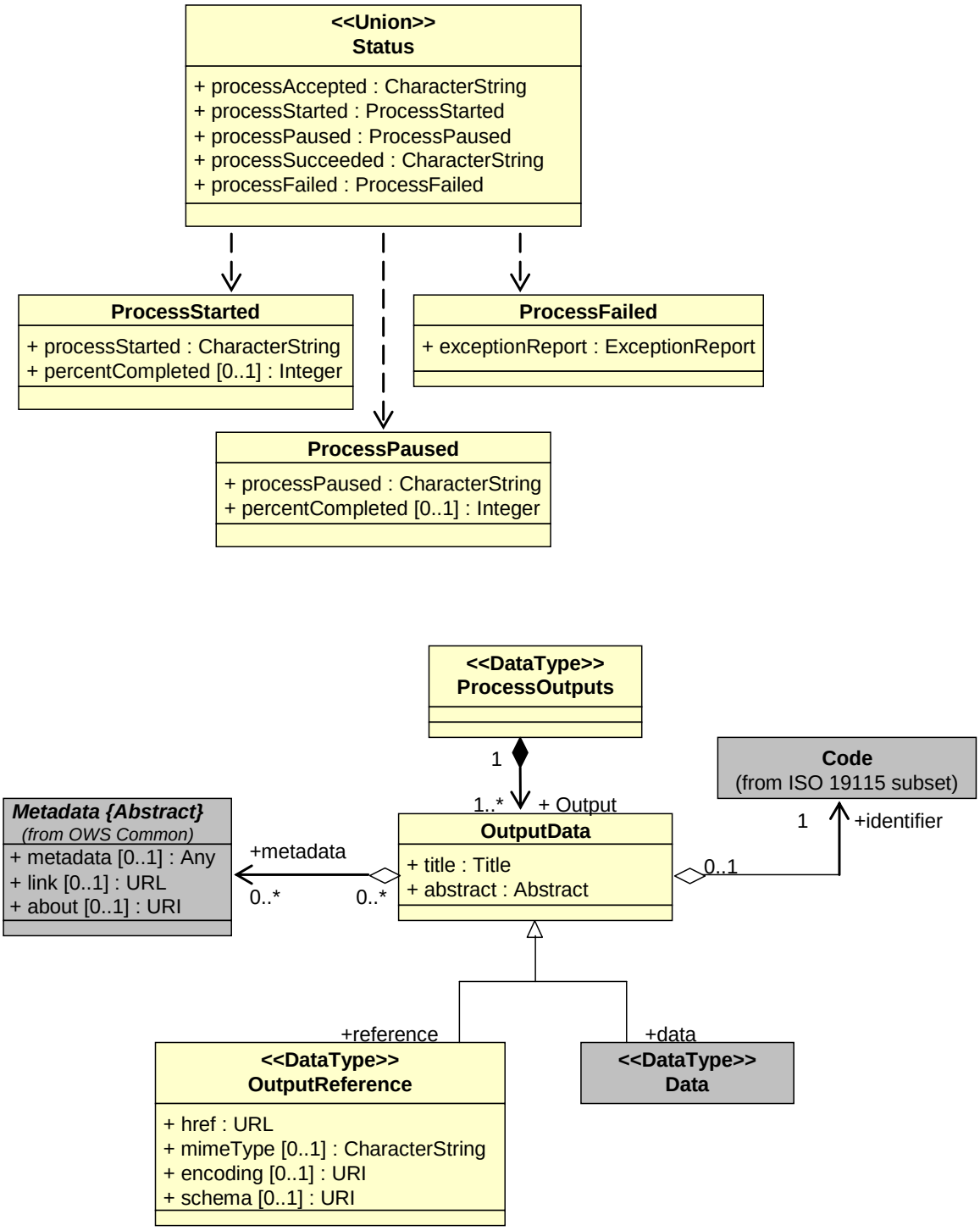

Figure C.10 — Execute package class diagram, part 3

Annex D
(normative)

Use of WPS with SOAP

D.1 Overview

This annex specifies how WPS shall be used with SOAP.

D.2 SOAP encoding of WPS requests and responses

WPS requests and responses encoded in SOAP shall use SOAP document-style encoding (also called message-style or document-literal encoding), as described in [OWS 06-094], for the following operations:

- GetCapabilities request
- GetCapabilities response
- DescribeProcess request
- DescribeProcess response
- Execute response when status=true

An example for a DescribeProcess request follows:

```
<soap:Envelope xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns:xsd="http://www.w3.org/2001/XMLSchema">
   <soap:Body>

<DescribeProcess service="WPS" version="1.0.0"
xmlns="http://www.opengeospatial.net/wps"
xmlns:ows="http://www.opengeospatial.net/ows"
xmlns:xlink="http://www.w3.org/1999/xlink"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://www.opengeospatial.net/wps
..\wpsDescribeProcess.xsd">
   <ows:Identifier>intersection</ows:Identifier>
   <ows:Identifier>union</ows:Identifier>
</DescribeProcess>

   </soap:Body>
</soap:Envelope>
```

WPS execute requests encoded in SOAP shall be encoded as follows:

- the process name shall be turned into an element in the SOAP body by pre-pending the text "ExecuteProcess_"

- each input and output shall be encoded as an element in the SOAP body by using the Identifier as the name of the element.

An example for an Execute request follows.

```
<soap:Envelope xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns:xsd="http://www.w3.org/2001/XMLSchema">
   <soap:Body>
      <ExecuteProcess_GMLBuffer
xmlns="http://wpsint.tigris.org/soap/SpatialAnalysis">

<GmlUrlResource>http://onotta499199/gml/polygon_gml.xml</GmlUrlResource>
         <Distance>10</Distance>
      </ExecuteProcess_GMLBuffer>
   </soap:Body>
</soap:Envelope>
```

SOAP requests to execute a process when RawDataOutput is requested shall generate a SOAP error.

Annex E
(informative)

WSDL best practices

E.1 Overview

This annex indicates best practices for how to implement WSDL support for WPS.

E.2 WSDL document for the entire service

A WSDL document that describes the whole WPS service (i.e. including the GetCapabilities and DescribeProcess operations) should be made available as follows:

```
http://hostname/WPSname?WSDL
```

Where

- *http://hostname/WPSname?* is the root of the HTTP requests to the WPS (e.g. GetCapabilities).

E.3 WSDL document for specific processes

In addition to implementing WSDL for the entire service, it is also useful to create a separate WSDL document for each individual process that can be executed. Such documents should be made available as follows:

```
http://hostname/WPSname/identifier[/service.soap]?WSDL
```

Where

- *identifier* is the Identifier of the WPS process.
- *[/service.soap]* is optional. When present, *http://hostname/WPSname/identifier/[service.soap]?* indicates the root of the HTTP SOAP requests to the WPS. and the service shall return a WSDL that describes the SOAP binding for that WPS process.

Note that the use of "?WSDL" is in keeping with general practice for retrieving WSDL documents. Supporting both an overall WSDL document as well as separate WSDL documents for each process, facilitates discovery of the WPS service including the GetCapabilities and DescribeProcess operations, and yet makes it possible to register/find individual WPS processes separately.

E.4 WSDL example for a complete service

..\examples\example_service.wsdl

Bibliography

[1] OGC 06-121r3, OpenGIS® Web Service Common Implementation Specification version 1.1.0 with Corrigendum 1, at http://portal.opengeospatial.org/files/?artifact_id=20040

[2] OGC 06-094, OWS Common change request: Add SOAP encoding, at http://portal.opengeospatial.org/files/?artifact_id=16086