Open GeoSpatial Consortium Inc.

Date: 2006-07-26

Reference number of this OpenGIS© Project Document: **OGC 06-035r1**

Version: 0.0.3

Category: Candiate OpenGIS© Implementation Specification

Editor: Peter Baumann

OGC Best Practices: Web Coverage Processing Service (WCPS)

Warning

This document is not an OGC Implementation Specification. This is an OGC Best Practices Recipients Document. Recipients of this document are invited to submit, with their comments, notification of any relevant patent rights of which they are aware and to provide supporting documentation.

Document type:	Best Practices Document
Document subtype:	(none)
Document stage:	Draft
Document language:	English

Contents

i. Preface

The OGC Web Coverage Processing Service (WCPS) is the result of work from both a theoretical perspective as well as the implementation and testing of a reference implementation. Nevertheless, there is still considerable room for improvement. Hints and comments are therefore most welcome by the document editor.

WCPS is based on the conceptual model of WCS. As the revision of WCS proceeds, WCPS will have to be adapted to maintain coherence.

This is an OGC Best Practices Document. This is a document containing discussion of best practices related to either the use and/or implementation of an adopted OGC document or an OpenGIS candidate specification that has not be submitted using the RFC process but is considered mature enough for implementation and for release to the public. Best Practices Documents are an official position of the OGC and thus represent an official endorsement of the content of the paper.

ii. Submitting organizations

The following organizations have submitted this Candidate Implementation Specification to the Open Geospatial Consortium, Inc.

- International University Bremen
- rasdaman GmbH

iii. Document Contributors

Name	**Email**	**Organization**	**Phone**
Baumann, Peter	p.baumann@iu-bremen.de	International University Bremen, rasdaman GmbH	+49-421-200-3178

iv. Revision history

Date	Release	Author	Paragraph modified	Description
2005-06-06	0.0.1	Peter Baumann, Trimita Chakma		Initial draft
2005-12-15	0.0.2	Peter Baumann, Georgi Chulkov		Reworked based on WCS progress
2005-04-20	0.0.3	Sean Forde		Minor editorial changes for publication as a discussion paper
2006-7-026	0.0.3	Carl Reed		Mincor changes to get document ready for posting as an OGC Best Practices paper.

v. Changes to the OpenGIS® Abstract Specification

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

vi. Future Work

This framework is planned to be enhanced and extended, among others, with the following features:

1) Streamlining with the OpenGIS Web Processing Service (WPS) (presumably WCPS will be described as an application profile of WPS).

2) WCPS is based on the conceptual model of the OpenGIS Web Coverage Service (WCS) Implementation Specification. As the revision of WCS proceeds, WCPS will have to be adapted to maintain coherence.

3) Geo coordinate handling currently is not harmonized with GML, ISO 19123, WCS, and relevant OGC standards (this in particular relates to the relevant XML Schema definitions). As soon as WCS 1.1 is stable and officially issued, WCPS definition must be adapted accordingly.

4) Once the schema is fixed, corresponding UML diagrams have to be added.

5) The Conformance Section has to be provided.

Foreword

Some of the elements of this document OGC 06-035r1 may be the subject of patent rights. The Open GeoSpatial Consortium Inc. shall not be held responsible for identifying any such patent rights.

Introduction

The Web Coverage Processing Service (WCPS) supports retrieval and processing of geospatial coverage data. WCPS uses the coverage model of the OGC Web Coverage Service (WCS) Implementation Specification [4] in which coverages are defined as “digital geospatial information representing space-varying phenomena”, currently constrained to equally spaced grids.

WCPS provides access to original or derived sets of geospatial coverage content, in forms that are useful for client-side rendering, input into scientific models, and other client applications. As such, WCPS includes WCS functionality and extends it with an expression language to form requests of arbitrary complexity allowing, e.g., multi-valued coverage results.

It is recommended this document be read in conjunction with the WCS Implementation Specification document [4].

Comment [b1]: Replace with then current WCS spec

OpenGIS Interface:
Web Coverage Processing Service (WCPS)

1 Scope

This document specifies how a Web Coverage Processing Service (WCPS) can describe, request, and deliver multi-dimensional grid coverage data over the World Wide Web.

Grid coverages have a domain comprised of regularly spaced locations along an arbitrary number of axes. Specific semantics is associated with spatio-temporal axes; A coverage can optionally have an x axis, a y axis (which, if both present, **shall** bear a common coordinate reference system), a time axis, and an elevation axis. A coverage's grid point (i.e., cell) data types define, at each location in the domain, either a single (scalar) value (such as elevation), or an ordered series of values (such as brightness values in different parts of the electromagnetic spectrum).

Result coverages can be transmitted directly or made available for download by URLs communicated to the client.

The Web Coverage Processing Service provides two operations: *GetCapabilities* and *ProcessCoverage*. The *GetCapabilities* operation, like in WCS, returns an XML document describing the service and brief descriptions of the data collections from which clients may request coverages; additionally WCPS specific processing service capabilities are delivered. Clients would generally run the *GetCapabilities* operation when opening a session with some particular server and cache its result for use throughout the session.

The *ProcessCoverage* operation describes how to process and analyse coverages and coverage sets stored on the server as well as to extract information – both grid data and metadata – from coverages. To this end, requests are phrased in a formally defined processing language that supports coverage expressions of unlimited complexity. Result coverages can be transmitted directly back to the client or made available for download by URLs communicated to the client.

Coverages advertised by a service can be stored on the corresponding server, but the service may well itself rely on external data sources to substantiate the portfolio. In any case, the appearance towards the service clients always is one homogeneously accessible coverage offering.

For future versions it is intended, to extend WCPS to incorporate further coverage types defined in the OpenGIS Abstract Specification (Topic 6, "The Coverage Type", OGC document 00-106), in synchronization with WCS.

2 Conformance

Conformance with this OGC Implementation Specification may be checked using all the relevant tests specified in Annex D (normative).

3 Normative references

The following normative documents contain provisions that, through reference in this text, constitute provisions of this specification. 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.

IETF RFC 2045 (November 1996), Multipurpose Internet Mail Extensions (MIME) Part One: Format of Internet Message Bodies, Freed, N. and Borenstein N., eds., <http://www.ietf.org/rfc/rfc2045.txt>

IETF RFC 2616 (June 1999), Hypertext Transfer Protocol – HTTP/1.1, Gettys, J., Mogul, J., Frystyk, H., Masinter, L., Leach, P., and Berners-Lee, T., eds., <http://www.ietf.org/rfc/rfc2616.txt>

IETF RFC 2396 (August 1998), Uniform Resource Identifiers (URI): Generic Syntax, Berners-Lee, T., Fielding, N., and Masinter, L., eds., <http://www.ietf.org/rfc/rfc2396.txt>

ISO 19105: Geographic information — Conformance and Testing

ISO 19123, Geographic Information — Coverage Geometry and Functions

OGC 02-023r4, OpenGIS Geography Markup Language (GML) Implementation Specification, v3.00 <http://www.opengis.org/techno/documents/02-023r4.pdf>

OGC 03-065r6, OpenGIS Web Coverage Service Implementqation Specification (WCS) Implementation Specification, v1.0.0 <http://www.opengis.org/techno/documents/03-065r6.pdf>

OGC AS 0, The OpenGIS Abstract Specification Topic 0: Overview, OGC document 99-100r1 <http://www.opengis.org/techno/abstract/99-100r1.pdf>

OGC AS 12, The OpenGIS Abstract Specification Topic 12: OpenGIS Service Architecture (Version 4.2), Kottman, C. (ed.), <http://www.opengis.org/techno/specs.htm>

XML 1.0, W3C Recommendation 6 October 2000, Extensible Markup Language (XML) 1.0 (2nd edition), World Wide Web Consortium Recommendation, Bray, T., Paoli, J., Sperberg-McQueen, C.M., and Maler, E., eds., <http://www.w3.org/TR/2000/REC-xml>

W3C Recommendation 2 May 2001: XML Schema Part 0: Primer, <http://www.w3.org/TR/2001/REC-xmlschema-0-20010502/>

W3C Recommendation 2 May 2001: XML Schema Part 1: Structures, <http://www.w3.org/TR/2001/REC-xmlschema-1-20010502/>

W3C Recommendation 2 May 2001: XML Schema Part 2: Datatypes, <http://www.w3.org/TR/2001/REC-xmlschema-2-20010502/>

4 Terms and definitions

For the purposes of this document, the terms and definitions given in the above references (in particular: WCS) apply, as do the following terms.

4.1
dimension
an independent axis of the coverage, usually representing a spatial or the temporal dimension.

4.2
cell
an element of a coverage which is uniquely identified by its spatio-temporal position; a cell contains a value of the type specified in the coverage definition.

4.3
cell type
the data type of a cell; it can be an atomic type (such as `char` for 8-bit greyscale images) or a composite "struct" type (such as `struct { char red, green, blue;}` for RGB images, using Java syntax); this relates to the terms "range" and "range type" in WCS.

4.4
cell domain
the extent of a coverage, whereby, along each axis, cells are numbered consecutively in ascending order using (not necessarily nonnegative) integers. A cell domain can be characterized completely by its lower and upper corner point coordinates.

4.5
domain
the extent of a coverage, whereby axes with a geo semantics have geographic coordinates, a time axis has time coordinates. Geo and time axes are optional.

4.6
range list
the component(s) of a cell type. This relates to the WCS term "range set", but emphasises that the types are sorted, allowing to address them not only by name, but also by position. Sort order is defined by the sequence indicated in a *GetCapabilities* response.

5 Conventions

5.1 Symbols (and abbreviated terms)

The following symbols and abbreviated terms are used in this document.

API	Application Program Interface
CRS	Coordinate Reference System
DCP	Distributed Computing Platform
GML	OGC Geography Markup Language, v3.00 (OGC 03-023r4)
ISO	International Organization for Standardization
OGC	Open GeoSpatial Consortium
OWS	OGC Web Service
UML	Unified Modeling Language
XML	Extensible Markup Language
1D	One Dimensional
2D	Two Dimensional
3D	Three Dimensional
4D	Four Dimensional

5.2 UML notation

Certain diagrams that appear in this document are presented using static structure diagrams in the Unified Modeling Language (UML) [OMG]. The UML notations used in this document are described in the diagram below.

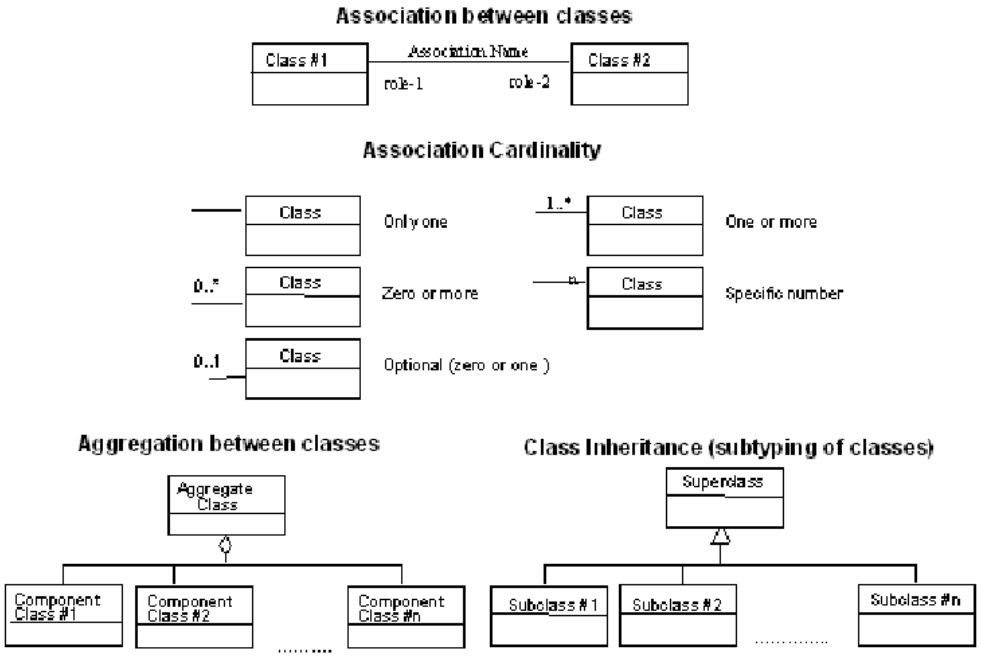

Figure 1 - UML Notation

In these UML class diagrams, the class boxes with three compartments and a light background are the primary classes being shown in this diagram, usually the classes from one UML package. The class boxes with a grey background are other classes used by these primary classes, usually classes from other packages. The class boxes without compartments do not show the class attributes, which are usually shown on another class diagram.

In these class diagrams, the following five stereotypes of UML classes are used:

a) <<Interface>> A definition of a set of operations that is supported by objects having this interface. An Interface class cannot contain any attributes.

b) <<DataType>> A descriptor of a set of values that lack identity (independent existence and the possibility of side effects). A DataType is a class with no operations whose primary purpose is to hold the information.

c) <<Enumeration>> A data type whose instances form a list of alternative literal values. Enumeration means a short list of well-understood potential values within a class.

d) <<CodeList>> is a flexible enumeration that uses string values for expressing a long list of potential alternative values. If the list alternatives are completely known, an enumeration shall be used; if the only likely alternatives are known, a code list shall be used. Code lists are more likely to have their values exposed to the user.

e) <<Type>> A stereotyped class used for specification of a domain of instances (objects), together with the operations applicable to the objects. A Type class may have attributes and associations.

NOTE All the stereotypes listed above are adapted from Subclause 6.8 of ISO 19103.

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

a) string – A sequence of characters

b) Boolean – A value specifying TRUE or FALSE

c) URI – An identifier of a resource that provides more information about data

d) URL – An identifier of an on-line resource that can be electronically accessed

5.3 XML schema notation

Several diagrams in this document represent XML Schema constructs using the graphical symbols provided by the XML Spy software suite (Altova, Inc. / www.xmlspy.com). These are depicted in Figure 2 below.

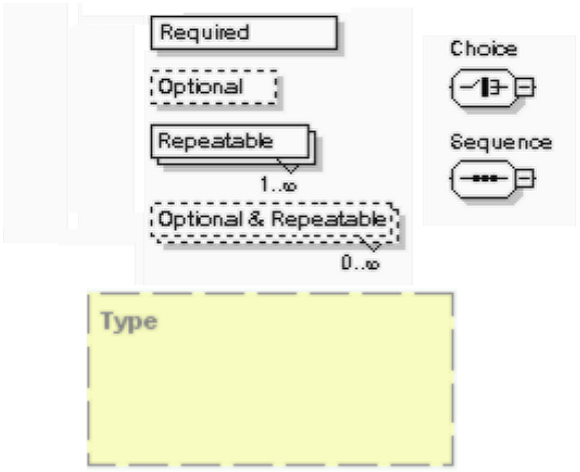

Figure 2. XML Schema graphic symbols

6 Basic service elements

6.1 Introduction

This clause describes aspects of Web Coverage Processing Server behavior (more generally, of OGC Web Service behavior) that are independent of particular operations, or that are common to several operations or interfaces.

6.2 Version numbering and negotiation

6.2.1 Version number form

The published specification version number contains three positive integers, separated by decimal points, in the form "x.y.z". The numbers "y" and "z" will never exceed 99. Each OWS specification is numbered independently.

6.2.2 Version changes

A particular specification's version number **shall** be changed with each revision. The number **shall** increase monotonically and **shall** comprise no more than three integers separated by decimal points, with the first integer being the most significant. There may be gaps in the numerical sequence. Some numbers may denote experimental or interim versions. Service instances and their clients need not support all defined versions, but **shall** obey the negotiation rules below.

6.2.3 Appearance in requests and in service metadata

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

6.2.4 Version number negotiation

A Client may negotiate with a Service Instance to determine a mutually agreeable specification version. Negotiation is performed using the **GetCapabilities** operation [see Clause 8] according to the following rules.

All Capabilities XML **shall** include a protocol version number. In response to a **GetCapabilities** request containing a version number, an OGC Web Service **shall** either respond with output that conforms to that version of the specification, **or** negotiate a mutually agreeable version if the requested version is not implemented on the server. If no version number is specified in the request, the server **shall** respond with the highest version it understands and label the response accordingly.

Version number negotiation occurs as follows:

a) If the server implements the requested version number, the server **shall** send that version.

b) If a version unknown to the server is requested, the server **shall** send the highest version it knows that is less than the requested version.

c) If the client request is for a version lower than any of those known to the server, then the server **shall** send the lowest version it knows.

d) If the client does not understand the new version number sent by the server, it **may** either cease communicating with the server **or** send a new request with a new version number that the client does understand but which is less than that sent by the server (if the server had responded with a lower version).

e) If the server had responded with a higher version (because the request was for a version lower than any known to the server), and the client does not understand the proposed higher version, then the client **may** send a new request with a version number higher than that sent by the server.

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

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

Example 2 - Server understands versions 4, 5 and 8. Client understands version 3. Client requests version 3. Server responds with version 4. Client does not understand that version or any higher version, so negotiation fails and client ceases communication with that server.

The negotiated version parameter **shall** be supplied with ProcessCoverage requests.

6.3 General HTTP request rules

6.3.1 Overview

At present, the only distributed computing platform (DCP) explicitly supported by OGC Web Services is the World Wide Web itself, or more specifically Internet hosts implementing the Hypertext Transfer Protocol (HTTP). Thus the Online Resource of each operation supported by a service instance is an HTTP Uniform Resource Locator (URL). The URL may be different for each operation, or the same, at the discretion of the service provider. Each URL **shall** conform to the description in [HTTP] but is otherwise implementation-dependent; only the parameters comprising the service request itself are mandated by the OGC Web Services specifications.

HTTP supports two request methods: GET and POST. One or both of these methods may be defined for a particular OGC Web Service type and offered by a service instance, and the use of the Online Resource URL differs in each case.

An Online Resource URL intended for HTTP GET requests is in fact only a URL prefix to which additional parameters must be appended in order to construct a valid Operation request. A URL prefix is defined as an opaque string including the protocol, hostname, optional port number, path, a question mark '?', and, **optionally**, one or more server-specific parameters ending in an ampersand '&'. The prefix uniquely identifies the particular service instance. For HTTP GET, the URL prefix **shall** end in either a '?' (in the absence of additional server-specific parameters) or a '&'. In practice, however, Clients **should** be prepared to add a necessary trailing '?' or '&' before appending the Operation parameters defined in this specification in order to construct a valid request URL.

An Online Resource URL intended for HTTP POST requests is a complete and valid URL to which Clients transmit encoded requests in the body of the POST document. A WCPS server **shall not** require additional parameters to be appended to the URL in order to construct a valid target for the Operation request.

6.3.2 Key-value pair encoding (GET or POST)

6.3.2.1 Overview

Using Key-Value Pair encoding, a client composes the necessary request parameters as keyword/value pairs in the form "keyword=value", separated by ampersands (‘&’), with appropriate encoding [IETF RFC 2396] to protect special characters. The resulting query string may be transmitted to the server via HTTP GET or HTTP POST, as prescribed in the HTTP Common Gateway Interface (CGI) standard [IETF RFC 2616].

Table 1 summarizes the request parameters for HTTP GET and POST.

Table 1 – Parts of a Key-Value Pair OGC Web Service Request

URL Component	Description
http://host[:port]/path	URL of service operation. The URL is entirely at the discretion of the service provider.
{name[=value]&}	The query string, consisting of one or more standard request parameter name/value pairs defined by an OGC Web Service. The actual list of required and optional parameters is mandated for each operation by the appropriate OWS specification.

Notes: [ ] denotes 0 or 1 occurrence of an optional part; {} denotes 0 or more occurrences.

A request encoded using the HTTP GET method interposes a '?' character between the service operation URL and the query string, to form a valid URI which may be saved as a bookmark, embedded as a hyperlink, or referenced via Xlink in an XML document.

6.3.2.2 Parameter ordering and case

Parameter names **shall not** be case sensitive, but parameter values **shall** be case sensitive.

NOTE In this document, parameter names are typically shown in uppercase for typographical clarity, not as a requirement.

Parameters in a request **may** be specified in any order.

An OGC Web Service **shall** be prepared to encounter parameters that are not part of this specification. In terms of producing results per this specification, an OGC Web Service **shall** ignore such parameters.

6.3.2.3 Parameter lists

Parameters consisting of lists shall use the comma (",") as the delimiter between items in the list.

Example `parameter=item1,item2,item3`

Multiple lists can be specified as the value of a parameter by enclosing each list in parentheses ("(", ")")

Example `parameter=(item1a,item1b,item1c),(item2a,item2b)`

If a parameter name or value includes a space or comma, it shall be escaped using the URL encoding rules [6].

6.3.3 XML encoding

Clients **may** also encode requests in XML for transmission to the server using HTTP GET or HTTP POST. The XML request **shall** conform to the schema corresponding to the chosen operation, and the client **shall** send it to the URL listed for that operation in the server's Getabilities response, in accordance with HTTP POST [7]).

NOTE To support SOAP messaging, clients need only enclose the XML document *ogcdoc* in a SOAP envelope as follows:

```
<env:Envelope xmlns:env="http://www.w3.org/2001/09/soap-envelope">
    <env:Body>
        ogcdoc
    </env:Body>
</env:Envelope>
```

6.4 General HTTP response rules

Upon receiving a valid request, the service **shall** send a response corresponding exactly to the request as detailed in the appropriate specification. Only in the case of Version Negotiation (described above) may the server offer a differing result.

Upon receiving an invalid request, the service **shall** issue a Service Exception as described in Subclause 6.5 below.

NOTE As a practical matter, in the WWW environment a client should be prepared to receive either a valid result, or nothing, or any other result. This is because the client may itself have formed a non-conforming request that inadvertently triggered a reply by something other than an OGC Web Service, because the service itself may be non-conforming, etc.

6.5 Service exceptions

Upon receiving an invalid request, the service **shall** issue a Service Exception XML message to describe to the client application or its human user the reason(s) that the request is invalid.

Service Exception XML **shall** be valid according to the Service Exception XML Schema in Subclause A.7. In an HTTP environment, the MIME type of the returned XML **shall** be "`application/vnd.ogc.se_xml`". Specific error messages can be included either as chunks of plain text or as XML-like text containing angle brackets ("<" and ">") if included in a character data (CDATA) section as shown in the example of Service Exception XML in Subclause A.7.

Service Exceptions **may** include exception codes as indicated in Subclause A.7. Servers **shall not** use these codes for meanings other than those specified. Clients **may** use these codes to automate responses to Service Exceptions.

7 The coverage model

The coverage model of WCPS relies on the coverage model of WCS [4]. In the current version the WCPS coverage model is constrained to equally spaced grids, meaning that the distance between any two adjacent grid points in a coverage is constant.

NOTE This restriction is intended to be relaxed in future WCPS versions.

In Section 7.1 the coverage model used is introduced informally, followed by a formal definition in Section -. Section 7.3 describes, based on this model, the constituents a WCPS coverage has. Subsequently (Section 9.2.2) this forms the basis for the WCPS semantics by describing the effect of a WCPS coverage processing request on the result coverage constituents.

7.1 Informal coverage definition

NOTE This section is non-normative; the normative coverage specification is given in Section 7.2.

Coverage data form n-dimensional cubes, where n>0 is an integer and values are associated with each coordinate point (“cell”) inside the cube, each dimension having a named axis. All values are of the same underylying data type. A set of metadata is associated with the coverage’s cell value set, thus allowing to perform opeations like extraction, analysis, and manipulation of coverages or their components.

A coverage is viewed, for the purpose of this standard, as being subdivided into four layers (see Figure 3):

- **Level 0: grid data** – „The grid data themselves“.

  These data can only be understood (e.g., to extract single cell values) in conjunction with their next-level metadata.

- **Level 1: technical meta data** – Number of axes, extent per axis, cell type, null value, interpolation methods.

  These data describe the coverage sufficiently to access the Level 0 cell values. There is no further semantics (such as space / time) known on this level; axes are numbered (not named), and cells are addressed using integer coordinates in all axes. The list of interpolation methods indicates which methods are available on the given coverage, in case interpolation has to be applied (such as scaling and coordinate system transformation).

- **Level 2: spatio-temporal meta data** – Coordinate reference system, geo / time reference, axis semantics

  Level 2 metadata allow to optionally associate the semantics of space and time with an axis. If an axis represents horizontal space coordinates, then a pertaining

coordinate reference system (which is the same for both x and y axis) describes the axis. A time axis always has the usual time semantics associated.

If Level 2 metadata are associated with a coverage, then it is possible to address coverage cells using spatio-teompral coordinates; to this end, conversion functions from spatio-temporal (Level 2) to integer (Level 1) coordinates are provided.

Additionally, for the purpose of this standard, Level 2 metadata encompass a unique name to allow for referencing a stored coverage.

- **Level 3: general meta data** – „Everything else“

  A coverage usually has additional descriptive data associated which embed it into the overall application model. For example, further description of the meaning of the coverage data, or the time the coverage has been acquired can be associated. These data can be arbitrary, but will always reference the coverage.

This standard considers only Level 0, 1, and 2. No assumptions whatsoever are made about Level 3 in this standard.

Levels do not encapsulate (in the sense of hiding) other layers; applications are free to address coverages on any level, for example on Level 2 using geo coordinates or on Level 1 using cell cordinates.

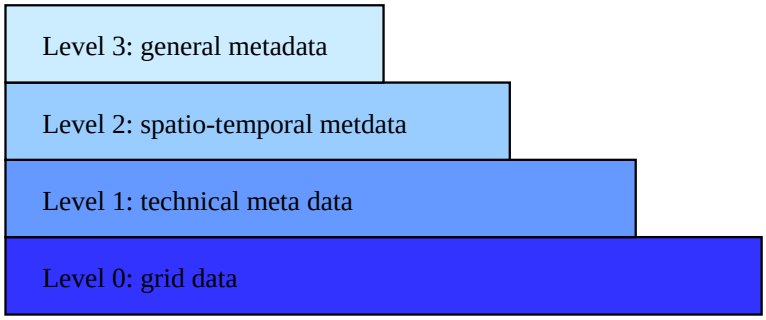

Figure 3. Coverage level stack

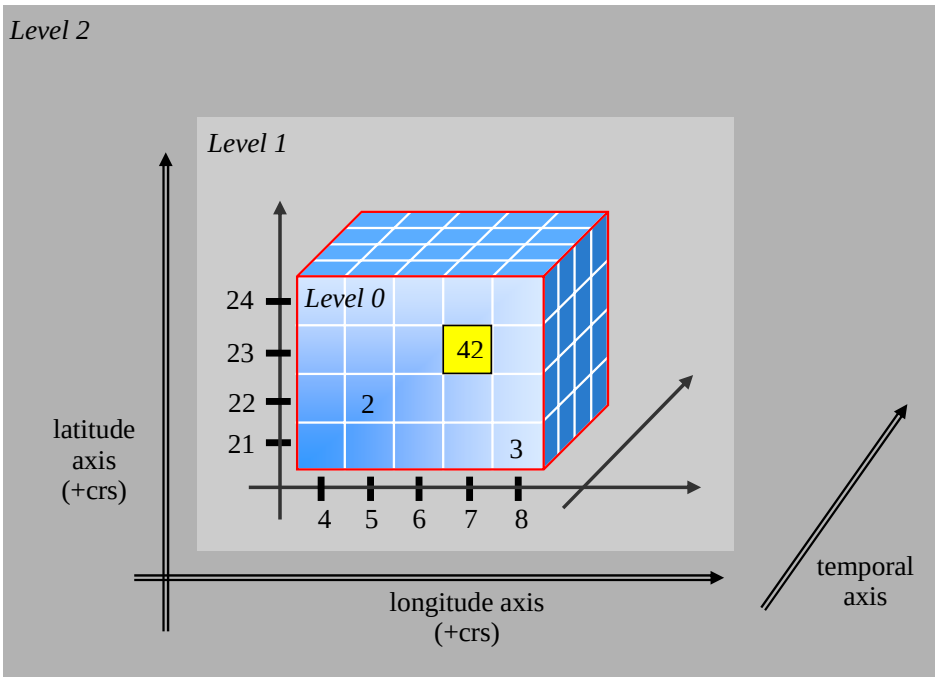

Figure 4. Coverage conceptual model and terminology

7.2 Formal coverage definition

Let d be a non-negative integer, and let $lo_i, hi_i \in \mathbf{Z}$ with $0 \le i < d$ and $lo_i \le hi_i$ be integer numbers. The set D defined as

$$D = \{ c_0 \mid lo_0 \le c_0 \le hi_0 \} \times \ldots \times \{ c_{d-1} \mid lo_{d-1} \le c_{d-1} \le hi_{d-1} \}$$

is called a *cell domain* of *dimension* d with *axis numbers* 0 to d-1; lo_i and hi_i are called the *lower* and *upper bound*, resp. of D in axis i.

The accessor functions *lo* and *hi* extract the lower and upper bound, resp., from a cell domain *D* (Figure 5). In particular, if *D* is given as above, then

$$lo(D) = (lo_0, \ldots, lo_{d-1})$$

$$hi(D) = (hi_0, \ldots, hi_{d-1})$$

and

$$lo_i(D) = lo_i$$

$$hi_i(D) = hi_i$$

Let further some non-empty value set T, called a *cell type*, be given. A coverage C, then, is a mapping from D to T:

$$C: D \rightarrow T$$

Example A coverage which maps CellDomain {0..1023}×{0..767} to cell type {0..255} can represent a greyscale VGA image.

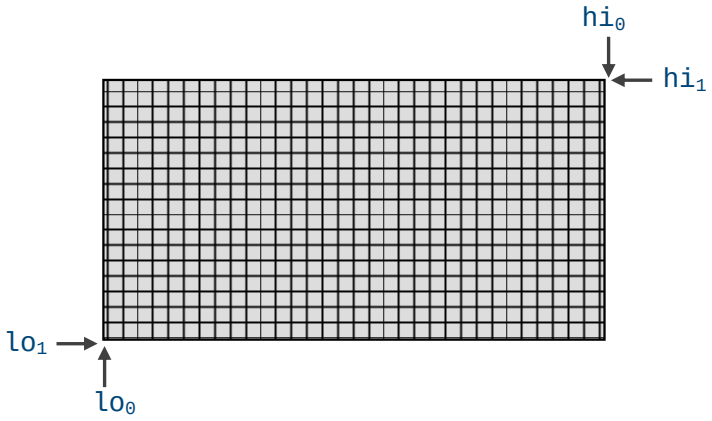

Figure 5. Cell domain with boundary points

NOTE A coverage obviously consists of a multi-dimensional array of cells sharing the same cell type. Coverage cell coordinates together form a simply connected (i.e., without cavities), finite, axis-parallel hypercube in n-dimensional Euclidean space. This geometric shape is also known as *parallel epiped*.

7.3 Coverage constituents

A WCPS coverage consists of the following constituents:

- *Coverage values*:
  The Level 0 coverage cell values (“grid data”).

- *Cell domain*:
  the coverage’s cell domain (coordinate extent), represented by the lower and upper bound coordinate per axis; the data type is an ordered list of (*lo*,*hi*) pairs of integer values with $lo \leq hi$ where *lo* denotes the lower bound and *hi* denotes the upper bound of the coverage in the resp. axis. The list elements, representing the axes, **shall** be numbered started from 0.

  Each cell domain **shall** contain at least one element. The number of components in the cell domain is called the coverage’s **dimension**.

- *Range list*:
  The cell type of this coverage; the data type is an ordered list of (*ComponentName*,*TypeName*) pairs where *ComponentName* identifies the component and

TypeName indicates the data type. Each range list **shall** contain at least one element.

ComponentNames **shall** be of non-zero length and unique within a given range list. Both *ComponentNames* and *TypeNames* **shall** be case sensitive, and they **should** contain only printable ASCII characters.

TypeName **shall** be one of the data types listed in Table 2.

Table 2 – Coverage cell data types.

Cell data type name	meaning
char	8-bit signed integer
unsigned char	8-bit unsigned integer
short	16-bit signed integer
unsigned short	16-bit unsigned integer
int	32-bit signed integer
unsigned int	32-bit unsigned integer
long	64-bit signed integer
unsigned long	64-bit unsigned integer
float	Single precision floating point number
double	Double precision floating point number

NOTE Components of a range list are also known as “bands”.

NOTE It is not required that all components within a coverage are of the same type.

NOTE A future version of this standard will additionally support nested components (nested “structs” in programming languages) and named types (such as “RGBPixel”).

Example The range list
(red,”unsigned char”), (green,”unsigned char”), (blue,”unsigned char”)
describes RGB values.

- *Null value*:
The value which is to be interpreted as null value for this coverage; the data type is the coverage’s cell type.

  WCPS operations **shall** set the null value to 0 for each cell type component of a overage whenever there is no explicit specification and the null value cannot be deduced.

  Example It is impossible to deduce a null value when an operation combines two coverages each having a different null value. In this case, the null value of the resulting coverage will be 0 for each cell type component. This can be changed with a subsequent null value redefinition (see Section 9.2.2.10).

- *Interpolation methods*:
the interpolation methods available on this coverage; the data type is a list of string values taken from Table 3.

NOTE While duplicate values in the list are not forbidden they don't make sense.

Table 3 — Interpolation methods

Interpolation method	Description
nearest neighbor (default)	These are defined in ISO 19123 (Schema for Coverage Geometry and Functions), Annex B.
bilinear	
bicubic	
lost area	
barycentric	

NOTE Table 3 is copied from [4], dropping the *none* element which disallows any interpolation. The effect of the WCS *none* element in WCPS is achieved by an empty interpolation list.

Example The following is a valid list of interpolation methods:
nearest neighbor,bilinear,bicubic

If an interpolation methods list is empty, then this **shall** mean that no interpolation is available; in this situation, requests containing scale (Section 9.2.2.24) and coordinate transformation operations (Section 9.2.2.24) **shall** result in error responses.

Further, a service **shall** respond with an exception if an operation requires interpolation and the interpolation method specified is not contained in the interpolation methods list of the coverage under consideration.

- *Name*:
an identifier for the coverage on hand, or for a set of coverages; the data type is character string.

  Names of stored coverages / coverage sets **shall** be of non-zero length and unique across all coverages / coverage sets offered by a particular WCPS; coverages resulting from a **ProcessCoverage** request **shall** have an empty name ("") associated. Names **shall** contain only printable ASCII characters.

- *Domain*:
the available coverage locations in space and/or time available in this coverage, expressed in geographic, elevation, or time coordinates, resp.; the data type is a list of (*AxisName*,*DomainExtent*,*AxisType*) triples where

  - *AxisName* identifies the axis. AxisNames within a domain **shall** be unique and of non-zero length; they **shall** contain only printable ASCII characters.
  - *DomainExtent* indicates the coverage's extent in the resp. axis. A DomainExtent is specified by a pair of values (*lo*,*hi*) denoting its lower and upper bound. The data type of lower and upper bound **can** be numeric or string, but **shall** be the same for both values.

    NOTE For axes with a spatial semantic associated the values will usually be numeric, for a temporal axis a string notation for points in time is common.

- *AxisType* denotes the type of an axis, which shall be one of the values listed in Table 4.

  NOTE This value determines whether a space or time coordinate transformation (see Section 9.2.2.29) can be applied to the axis on hand.

  NOTE Axis type *elevation* is reserved for future semantic functionality; currently axis type *elevation* is interpreted like *other*.

Each domain **shall** have at most one axis of type *temporal*.

Each domain **shall** have at most one axis of type *x*, and at most one axis of type *y*.

Each domain **shall** have at most one axis of type *elevation*.

The domain is optional for a coverage; if the domain is present, its length **shall** be equal to the length of the coverage's cell domain list, i.e., its dimension.

Table 4 – Coverage axis types.

Axis type	Meaning
x	East-West extent, expressed in the coverage's CRS
y	North-South extent, expressed in the coverage's CRS
temporal	Time; coordinates are expressed as time strings according to [9]
elevation	Geographical elevation, i.e., height
other	None of the above; no spatio-temporal semantics is associated with such an axis

Example Let $x_0,x_1,y_0,y_1 \in \mathbf{R}$ be numbers with $x_0 \le x_1$ and $y_0 \le y_1$. Then, the domain list
(*longitude*, (x_0,x_1), *y*), (*latitude*, (y_0,y_1), *x*)
describes a conventional map of spatial extent between (x_0, y_0) and (y_0,y_1) with axis names *longitude* and *latitude*, resp.

The service does **not need** to publish the mapping between cell and spatio-temporal coordinates.

Let (*a*,(*lo*,*hi*),*t*) be an axis named *a* of some coverage *C* occurring at position *p* in *C*'s domain. Then, the **resolution** of *C* in axis named *a* is given by

$$\mathrm{res}(C,a) = \frac{hi - lo}{\mathrm{hi}_p(\mathrm{cdom}(C)) - \mathrm{lo}_p(\mathrm{cdom}(C)) + 1}$$

- *CRS*:
  Identifier of the coverage's CRS ("coordinate reference system"), which **shall** be an empty string ("") for non-georeferenced data; the data type of CRS is an ASCII string of the format

  ```
  urn:ogc:def:crs:EPSG::XXXXX
  ```

  where XXXX is one of the CRS codes defined by EPSG [5].

Example The following is a valid SupportedCRS value:

```
urn:ogc:def:crs:EPSG::4314
```

The DomainSet is optional for a coverage; if the DomainSet is present, its length **shall** be equal to the length of the coverage's CellDomainSet list, i.e., its dimension.

7.4 Accessor functions

A set of so-called *accessor functions* allows to extract the constituents listed above from a given coverage. Table 5 summarises the accessor functions available, plus some convenience functions whose semantics can be based on the core functions.

NOTE These functions serve to define the semantics of **ProcessCoverage** requests in Clause 9.

Table 5 – Coverage constituent accessor functions.

Coverage characteristic	Accessor function for some coverage C	Comment
Cell values	value(C,p) for all $p \in$ cdom(C)	Level 0 and Level 1 functions for defining multi-dimensional data cubes
Cell domain	cdom(C)	
Range list	celltype(C)	
Null value	null(C)	
Interpolation method list	im(C)	
Name	name(C)	Stored coverages only
Domain	dom(C) dom(C,a) for all a,e,t with (a,e,t)$\in$Domain(C)	Level 2 functions for spatio-temporal semantics
CRS	crs(C)	
Dimension	dim(C)	Level 2 convenience functions (redundant)
Axis resolution	res(C,a) for all a,e,t with (a,e,t)$\in$Domain(C)	

7.5 UML model

Tbd once stabilized WCS 1.1 available

8 GetCapabilities operation

The **GetCapabilities** operation is identical to the WCS GetCapabilities operation [4], except that it extends the notion of coverage names to names of coverages and coverage sets, of axis types, and with supported format being a server property, not an individual coverage property.

8.1 GetCapabilities operation request

The GetCapabilities operation request shall be as specified in Subclauses 7.2.2 through 7.2.4 of [OGC 05-008]. The “service”, “request”, and “AcceptVersions” parameters shall be implemented by all WCPS servers. The “sections” and “UpdateSequence” parameters are optional implementation by WCPS servers. All WCPS servers shall implement HTTP GET transfer of the **GetCapabilities** operation request, using KVP encoding. Servers can also implement HTTP POST transfer of the **GetCapabilities** operation request, using XML and/or KVP encoding.

The value of the “service” parameter shall be “WCS”. The allowed set of service metadata (or Capabilities) XML document section names and meanings shall be as specified in Tables 3 and 7 in Subclauses 7.3.3 and 7.4.2 of [OGC 05-008].

The KVP encoding of a WCS **GetCapabilities** operation request shall be as specified in Table 2 in Subclause 7.2.2 of [OGC 05-008].

The XML Schema fragment for encoding a WCPS **GetCapabilities** operation request extends ows:GetCapabilitiesType in owsGetCapabilities.xsd from [OGC 05-008], and is specified in `wcpsGetCapabilities.xsd` as laid out in Annex B.

9 ProcessCoverage operation

9.1 Introduction

A Web Coverage Processing Server evaluates a **ProcessCoverage** request and returns an appropriate response to the client.

While the WCS **GetCoverage** operation allows retrieval of a coverage from a coverage offering, possibly modified through operations like spatial, temporal, and band subsetting and coordinate transformation, the WCPS **ProcessCoverage** extends this functionality through more powerful processing capabilities. This includes, on the one hand, further coverage processing primitives and, on the other hand, nesting of function application, thereby allowing for arbitrarily complex requests.

NOTE WCPS has been designed so as to be "safe in evaluation" – i.e., any valid WCPS request can be evaluated in a finite number of steps based on the primitives. Hence, WCPS implementations can be constructed in a way that no single request can render the service permanently unavailable.

Clients **can** choose whether to phrase **ProcessCoverage** requests based on a coverage's cell coordinates or through spatio-temporal coordinates.

A WCPS response is an ordered sequence of data items. A data item can be a coverage or the result of any other processing expression. The **ProcessCoverage** operation returns a coverage as stored on the server, or a constituent thereof, or a derived coverage, or a constituent thereof (see Section 7.3 for the constituents of a coverage).

NOTE Data items within a WCPS response list can be heterogeneous in size and structure. In particular, the coverages within a response list can have different dimensions, extents, cell types, etc.

9.1.1 WCPS expression language specification

The WCPS primitives plus the nesting capabilities form an expression language; this abstract language collectively is referred to as **WCPS language**. In the following subsections the language elements are detailed. The complete syntax is listed in Appendix A.

A WCPS expression is called **admissible** if and only if it adheres to the WCPS language syntax. WCPS servers **shall** return an exception in response to a WCPS request that is not admissible.

Example The expression

```
C * 2
```

is admissible as it adheres to WCPS syntax whereas the expression

```
C C
```

violates WCPS syntax and, hence, is not admissible.

The semantics of a WCPS expression is defined by indicating, for all admissible expressions, the value of each coverage constituent as laid down in Section 7.4.

An expression is **valid** if and only if it is admissible and it complies with the conditions imposed by the WCPS language semantics.

Example The expression following is valid if and only if the WCPS offers a coverage of name `C` that has a component named `red`.

```
C.red * 2.5
```

9.2 ProcessCoverage request

9.2.1 Request/response overview

A **ProcessCoverage** request follows the conceptual model as described in Subclause 7 and is encoded in one of the structures as described in Subclauses 9.2.2.29 and 9.2.6, resp. In this Subclause, the syntax and semantics of **ProcessCoverage** requests is described in an abstract, encoding-independent manner. For the reader's convenience, for each constituent of a coverage-valued expression it is indicated whether the operation considered changes this constituent value.

A **ProcessCoverage** request **shall** contain exactly one valid WCPS expression.

The server **shall** answer with a response as described in the subsequent Subclauses.

9.2.2 WCPS expressions

9.2.2.1 Overview

A WCPS expression is a **coverageListExpr** (which evaluates to a list of encoded coverages; see Section 9.2.2.2). Each WCPS request shall contain exactly one **coverageListExpr**.

9.2.2.2 coverageListExpr

The **coverageListExpr** element processes a list of coverages in turn. Each coverage is optionally checked first for fulfilling some predicate, and gets selected – i.e., becomes part of the result list – only if the predicate evaluates to true. Each coverage selected then will be processed, and the result will be appended to the result list. This result list, finally, is returned as the **ProcessCoverage** response unless no exception was generated.

The elements in the **coverageList** clause can be names of single coverages or names of coverage sets. The **coverageList** elements **shall** be inspected sequentially in the order given; for a coverage set element, each element **shall** be inspected exactly once without any particular sequence specified by this standard.

Coverage or coverage set names **may** occur more than once in a **coverageList**. In this case the coverage **shall** be inspected each time it is listed, respecting the overall inspection sequence.

Coverage sets **may** be empty.

NOTE Implementers may define their individual iteration sequence on sets.

Let

v be an **iteratorVar**,
L be a **coverageList**,
b be a **booleanScalarExpr** possibly containing occurrences of *v*,
P be a **processingExpr** possibly containing occurrences of *v*.

Then,

for any **responseList** *R*,
coverageList *L′*,
where

```
R =  for v in ( L′ )
     where b
     return P
```

and

L′ is that **coverageList** derived from *L* by substituting each occurrence of a coverage set name *N* in *L* by a list of all coverage names contained in the coverage set named *N*, in some arbitrary (not necessary repeatable) sequence.

R is constructed as follows:

- Let *R* be the empty sequence.
- while *L′* is not empty:
- assign the first element in *L′* to *v*
- evaluate *P*, substituting any occurrence of coverage name *v* by the coverage this name refers to
- append the result to *R*
- remove the first element from *L′*

Example Assume a WCPS server offers coverages A and C and coverage set B containing coverages X, Y, Z. Then, the server may execute the following WCPS request:

```
for c in ( A, B, C )
return tiff( c )
```

in a sequence equivalent to this one:

```
for c in ( A, X, Y, Z, C )
return tiff( c )
```

The result list, in this case, will consist of the sequence

```
( tiff(A), tiff(X), tiff(Y), tiff(Z), tiff(C) )
```

…but the server may also choose the evaluation sequence below (with the obvious change to the result list):

```
for c in ( A, Z, Y, X, C )
return tiff( c )
```

…however **not** in this sequence:

```
for c in ( X, Y, Z, A, C )
return tiff( c )
```

NOTE Future versions of this standard may consider nested **coverageListExpr**s.

9.2.2.3 processingExpr

The **processingExpr** element is either a **encodedCoverageExpr** (which evaluates to an encoded coverage; see Section 9.2.2.4), or a **scalarExpr** (which evaluates to coverage description data or coverage summary data; see Section 9.2.2.6).

9.2.2.4 encodedCoverageExpr

The **encodedCoverageExpr** element specifies a coverage result as described by its **C1** sub element, and a data format encoding specified by the format name, **formatName**, and possible extra encoding parameters specified in **extraParams**.

Data format encodings **should**, to the largest extent possible, materialise the coverage's (Level 1 and Level 2) metadata. A service **may** store further information as part of the encoding.

Example A GeoTIFF result file should contain the geo referencing information manifest in the coverage's latitude/longitude axes, if present.

Let

`C` be a **coverageExpr**,
`f` be a string,
where
`f` is the name of a data format listed under **supportedFormats** in the **GetCapabilities** response,
the data format specified by `f` supports encoding of a coverage of `C`'s domain and range.

Then,

for any **byteString** *S*
where
S = **encode (*C* , *f*)**
or *S* = **encode (*C* , *f*, *extraParams*)**
with *extraParams* being a string enclosed in double quotes (‘”’)

S is defined as that byte string which encodes *C* into the data format specified by *formatName* and the optional *extraParams*. Syntax and semantics of the *extraParams* are not specified further in this standard.

Example The following expression specifies retrieval of coverage C encoded in HDF-EOS:

```
encode( C, “hdf-eos” )
```

NOTE The extraParams are data format and implementation dependent.

Example A possible JPEG quality factor of 50% **may** be encoded as the string “50.”.

NOTE some format encodings may lead to a loss of information.

9.2.2.5 booleanExpr

The **booleanExpr** element is a **scalarExpr** (see Section 9.2.2.6) whose result type is Boolean.

NOTE WCPS implementors may extend this to allow, e.g., the usual boolean, arithmetic, and further scalar functions.

9.2.2.6 scalarExpr

The **scalarExpr** element is either a **getMetaDataExpr** (see Section 9.2.2.7) or a **condenseExpr** (see Section 9.2.2.25). It returns a non-coverage result.

9.2.2.7 getMetaDataExpr

The **getMetaDataExpr** element extracts a coverage description element from a coverage.

NOTE The cell value sets can be extracted from a coverage using subsetting operations (see Section 9.2.2.20).

Let

C be a **coverageExpr**.

Then,

The following metadata extraction functions are defined:

Metadata fct.	**Result**	**Description**	**Result type**

cdom(*C* **)**	cdom(*C*)	cell domain of *C*	Cell domain structure as defined in Section 7.3
celltype(*C* **)**	celltype(*C*)	cell type of *C*	Range list structure as defined in Section 7.3
null(*C* **)**	null(*C*)	null value of *C*	celltype(*C*)
crs(*C* **)**	crs(*C*)	CRS of *C*	String
xDomain(*C* **)**	dom(*C*,*a*)	East-West extent of *C*, if *C* has an axis *a* of type *x*	Pair of double numbers
yDomain(*C* **)**	dom(*C*,*a*)	North-South extent of *C*, if *C* has an axis *a* of type *y*	Pair of double numbers
tDomain(*C* **)**	dom(*C*,*a*)	temporal coordinate extent of *C*, if *C* has an axis *a* of type *temporal*	pair of ows:time
dim(*C* **)**	dim(*C*)	dimension of *C*	unsigned integer
res(*C* , *a* **)**	res(*C* ,*a*)	resolution of *C* in axis *a*	double precision floating point number

Whenever one of the conditions mentioned above are not fulfilled the server **shall** respond with an exception.

NOTE Not all information about a coverage can be retrieved this way. Adding the information contained in a **GetCapabilities** response provides complete information about a coverage.

Example For a 3-D coverage `C`, the following expression evaluates to 3:

```
dim( C )
```

Example For an RGB coverage `C` containing color images, the result of the expression `celltype(C)` is the following RangeList (modulo white space):

```
(“red”,   “unsigned char”),
(“green”, “unsigned char”),
(“blue”,  ”unsigned char”)
```

9.2.2.8 coverageExpr

The **coverageExpr** element is either a **coverageName** (see Section 9.2.2.9), or **setMetaDataExpr** (see Section 9.2.2.9), or an **inducedExpr** (see Section 9.2.2.11), or a **subsetExpr** (see Section 9.2.2.20), or a **scaleExpr** (see Section 9.2.2.24), or a **crsTransformExpr** (see Section 9.2.2.29), , or a **coverageConstructorExpr** (see Section 9.2.2.25), or a **condenseExpr** (see Section 9.2.2.26).

A **coverageExpr** always evaluates to a single coverage.

9.2.2.9 coverageName

The **coverageName** element represents the name of a single coverage offered by the server addressed.

Let

> *n* be a **name**
> where there exists a coverage *C* with name *n* offered by the server addressed.

Then,

> for any **coverageExpr** *C′*,
> where
> *C′* = *n*
>
> *C′* is defined as follows:

Coverage constituent	**Changed?**
for all *p* ∈ cdom(*C′*): value(*C′*,*p*) = value(*C* ,*p*)	
cdom(*C′*) = cdom(*C*)	
celltype(*C′*) = celltype(*C*)	
null(*C′*) = null(*C*)	
im(*C′*) = im(*C*)	
name(*C′*) = “” (empty string)	**X**
dom(*C′*) = dom(*C*)	
crs(*C′*) = crs(*C*)	

Example The following coverage expression evaluates to the complete, unchanged coverage *C*, if that is offered by the server:

> *C*

9.2.2.10 setMetaDataExpr

The **setMetaDataExpr** element specifies a change in a coverage’s metadata, leaving untouched the coverage cell values.

Let

C_1 be a **coverageExpr**,
n be a **nullValue** of type celltype(C_1),
m be an **interpolationMethodList**,
a be an **axisName**,
lo, hi be either both **floatNumber**s or both **string**s with *lo*≤*hi*
c be a **crsName**.

Then,

for any **coverageExpr** C_2
where C_2 is one of

C_{null} = **setNull(** C_1 , *n* **)**
C_{im} = **setInterpolation(** C_1 , *m* **)**
C_{crs} = **setCrs(** C_1 , *c* **)**
C_{axis} = **setAxis(** C_1 , a, *lo, hi* **)**

C_2 is defined as follows:

Coverage constituent		**Changed?**
for all p ∈ cdom(C_2): value(C_2, p) = value(C_1, p)		
cdom(C_2) = cdom(C_1)		
celltype(C_2) = celltype(C_1)		
null(C_{null}) = *n* null(C_2) = null(C_1)	 for all C_2 except C_{null}	**X**
im(C_{im}) = *m* im(C_2) = im(C_1)	 for all C_2 except C_{im}	**X**
name(C_i) = *""*	for all C_2	**X**
lo_a(dom(C_{axis},*a*,*lo*,*hi*)) = *lo* hi_a(dom(C_{axis},*a*,*lo*,*hi*)) = *hi* lo_a(dom(C_{axis},*a*,*lo*,*hi*)) = *lo* hi_a(dom(C_{axis},*a*,*lo*,*hi*)) = *hi* dom(C_2) = dom(C_1)	for axis with name *a* and index *i*, for axis with name *a* and index *i*, for axis name not equal to *a*, for axis name not equal to *a*, for all C_2 except C_{axis}	**X**
crs(C_{crs}) = *c* crs(C_2) = crs(C_1)	 for all C_2 except C_{crs}	**X**

Example The following coverage expression evaluates to coverage that has -100 as its null value:

```
setNull( C, -100 )
```

NOTE Those metadata which affect the coverage values themselves (one might call them “technical metadata”) cannot be changed; to accomplish that the operations under **coverageExpr** are available instead. In particular,

- the setNull() operation will not change any preexisting value in the coverage in an attempt to adapt old null values to the new null value;
- the setCrs() operation will not adapt any spatial coordinates in the coverage, i.e., it does not perform any coordinate system transformation.

NOTE Functions setFomain() and setSdomain() allow to set coverage axis extents of type float and string, resp. Among others, this can be used to redefine a coverage’s spatial and temporal coordinates.

NOTE As WCPS focuses on the processing of the coverage values, advanced capabilities for manipulating a coverage’s metadata are currently not foreseen.

9.2.2.11 inducedExpr

The inducedExpr element is either a **unaryInducedExpr** (see Section 9.2.2.12) or a **binaryInducedExpr** (see Section 9.2.2.19).

Induced operations allow to simultaneously apply a function originally working on a single cell value to all cells of a coverage. In case the cell type contains more than one component, the function is applied to each cell component simultaneously.

The result coverage has the same domain, but **may** change its base type.

NOTE The idea is that for each operation available on the cell type, a corresponding coverage operation is provided (“induced from the cell type operation”), a concept first introduced by Ritter et al.

Example Adding two RGB images will apply the “+” operation to each cell, and within a cell to each band in turn.

9.2.2.12 unaryInducedExpr

The **unaryInducedExpr** element specifies a unary induced operation, i.e., an operation where only one coverage argument occurs.

NOTE The term “unary” refers only to coverage arguments; it is well possible that further non-coverage parameters occur, such as an integer number indicating the shift distance in a bit() operation.

A **unaryInducedExprp** is either an **arithmeticExpr** or **exponentialExpr** or **trigonometricExpr** (in which case it evaluates to a coverage with a numeric cell type; see Sections 9.2.2.13, 9.2.2.14, 9.2.2.15), a **boolExpr** (in which case it evaluates to a Boolean expression; see Section 9.2.2.16), a **castExpr** (in which case it evaluates to a coverage with unchanged values, but another cell type; see Section 9.2.2.16), or a **selectExpr** (in which case a component selection is performed; see Section 9.2.2.18).

9.2.2.13 arithmeticExpr

The **arithmeticExpr** element specifies a unary induced arithmetic operation.

Let

C_1 be a **coverageExpr**
where
for all components c in celltype(C_1): c is numeric.

Then,

for any **coverageExpr** C_2
where C_2 is one of
C_{plus} = **+** C_1
or C_{minus} = **-** C_1
or C_{sqrt} = **sqrt(** C_1 **)**
or C_{abs} = **abs(** C_1 **)**

C_2 is defined as follows:

Coverage constituent	**Changed?**
for all p ∈ cdom(C_1): value(C_{plus}, p) = value(C_1, p) value(C_{minus}, p) = - value(C_1, p) value(C_{sqrt}, p) = sqrt(value(C_1, p)) value(C_{abs}, p) = abs(value(C_1, p))	**X**
cdom(C_2) = cdom(C_1)	
celltype(C_2) = celltype(C_1) for all C_2 except C_{sqrt} for all components c of celltype(C_{sqrt}): c = double	**X**
null(C_2) = null(C_1)	
im(C_2) = im(C_1)	
name(C_2) = ""	**X**
dom(C_2) = dom(C_1)	
crs(C_2) = crs(C_1)	

Example The following coverage expression evaluates to a float-type coverage where each cell value contains the square root of the corresponding source coverage's value.

```
sqrt( C + D )
```

9.2.2.14 trigonometricExpr

The **trigonometricExpr** element specifies a unary induced trigonometric operation.

Let

c_1 be a **coverageExpr**
where
for all components c in celltype(C_1): c is numeric.

Then,

for any **coverageExpr** c_2
where c_2 is one of
c_{sin} = **sin(** c_1 **)**
or c_{cos} = **cos(** c_1 **)**
or c_{tan} = **tan(** c_1 **)**
or c_{sinh} = **sinh(** c_1 **)**
or c_{cosh} = **cosh(** c_1 **)**
or c_{arcsin} = **arcsin(** c_1 **)**
or c_{arccos} = **arccos(** c_1 **)**
or c_{arctan} = **arctan(** c_1 **)**

c_2 is defined as follows:

Coverage constituent	**Changed?**
for all p ∈ cdom(c_1): value(c_{sin},p) = sin(value(c_1,p)) value(c_{cos},p) = cos(value(c_1,p)) value(c_{tan},p) = tan(value(c_1,p)) value(c_{sinh},p) = sinh(value(c_1,p)) value(c_{cosh},p) = cosh(value(c_1,p)) value(c_{arcsin},p) = arcsin(value(c_1,p)) value(c_{arccos},p) = arccos(value(c_1,p)) value(c_{arctan},p) = arctan(value(c_1,p))	**X**
cdom(C_2) = cdom(C_1)	
for all components c of celltype(C_2) : c = double	**X**
null(C_2) = null(C_1)	
im(C_2) = im(C_1)	
name(c_i) = "" for all c_2	**X**
dom(C_2) = dom(C_1)	
crs(C_2) = crs(C_1)	

Example The following expression replaces all (numeric) values of coverage C with their sine:

```
sin( C )
```

9.2.2.15 exponentialExpr

The **exponentialExpr** element specifies a unary induced exponential operation.

Let

c_1 be a **coverageExpr**
where
for all components c in celltype(C_1): c is numeric.

Then,

for any **coverageExpr** c_2
where c_2 is one of
c_{exp} = **exp(** c_1 **)**
or c_{log} = **log(** c_1 **)**
or c_{ln} = **ln(** c_1 **)**

c_2 is defined as follows:

Coverage constituent	**Changed?**
for all p ∈ cdom(c_2): value(c_{exp},p) = exp(value(c_1,p)) value(c_{log},p) = log(value(c_1,p)) value(c_{ln},p) = ln(value(c_1,p))	**X**
cdom(C_2) = cdom(C_1)	
for all components c of celltype(C_2) : c = double	**X**
null(C_2) = null(C_1)	
im(C_2) = im(C_1)	
name(c_2) = "" for all c_2	**X**
dom(C_2) = dom(C_1)	
crs(C_2) = crs(C_1)	

Example The following expression replaces all (nonnegative numeric) values of coverage C with their natural logarithm:

```
ln( C )
```

9.2.2.16 boolExpr

The **boolExpr** element specifies a unary induced Boolean operation. The only operation available is logical negation (logical "not").

Let

c_1 be a **coverageExpr**
where
for all components c in celltype(C_1): c = Boolean.

Then,

for any **coverageExpr** c_2
where c_2 is one of
c_{not} = **not** c_1
or c_{bit} = **bit(** C_1 , n **)**
where n is an expression evaluating to a nonnegative integer value

c_2 is defined as follows:

Coverage constituent	**Changed?**
for all p ∈ cdom(c_2): value(c_{not} , p) = not(value(c_1,p)) value(c_{bit}, p) = (value(c_1,p) >> value(n)) mod 2	**X**
cdom(C_2) = cdom(C_1)	
for all components c of celltype(C_2): c = Boolean	
null(C_2) = null(C_1)	
im(C_2) = im(C_1)	
name(c_2) = ""	**X**
dom(C_2) = dom(C_1)	
crs(C_2) = crs(C_1)	

Example The following expression inverts all (Boolean) values of coverage C:

```
not C
```

NOTE The operation `bit(a,b)` extracts bit position *b* (assuming a binary representation) from integer number *a* and shifts the resulting bit value to bit position 0. Hence, the resulting value is either 0 or 1.

9.2.2.17 castExpr

The **castExpr** element specifies a unary induced cast operation, that is: to change the cell type of the coverage while leaving all other constituents unchanged, however possibly suffering from a loss of accuracy through data type conversion.

Let

c_1 be a **coverageExpr**,
t be a cell type name.

Then,

for any **coverageExpr** c_2
where
c_2 = **(** t **)** c_1

c_2 is defined as follows:

Coverage constituent	**Changed?**
for all p ∈ cdom(c_2): value(c_2 , p) = (t) value(c_1,p)	**X**
cdom(C_2) = cdom(C_1)	
for all components c of celltype(C_2): $c = t$	**X**
null(C_2) = null(C_1)	
im(C_2) = im(C_1)	
name(c_2) = ""	**X**
dom(C_2) = dom(C_1)	
crs(C_2) = crs(C_1)	

Example the result cell type of the following expression will be char, i.e., 8 bit:

```
(char) ( C / 2 )
```

9.2.2.18 selectExpr

The **selectExpr** element specifies a unary induced record ("struct") selection operation. Selection can be done either using the component's name or its position, starting with position number 0.

Let

c_1 be a **coverageExpr**,
comp be a component ("band") of type *t* within celltype(C_1), either identified by band name or by its position number, starting from 0.

Then,

for any **coverageExpr** c_2
where
$c_2 = c_1.comp$

c_2 is defined as follows:

Coverage constituent	**Changed?**
for all p ∈ cdom(c_2): value(C_2, p) = value(C_1,p).*comp*	**X**
cdom(C_2) = cdom(C_1)	
celltype(C_2) = *t*	**X**
null(C_2) = null(C_1)	
im(C_2) = im(C_1)	
name(c_2) = *""*	**X**
dom(C_2) = dom(C_1)	
crs(C_2) = crs(C_1)	

Example Let C be a coverage with cell type char. Then the following request snippet describes a char-type coverage where each cell value contains the nonnegative difference between red and green band:

```
C.red - C.green
```

9.2.2.19 binaryInducedExpr

The **binaryInducedExpr** element specifies a binary induced operation, i.e., an operation involving two coverage-valued arguments.

The coverage cell types **shall** be atomic and numeric.

Let

C_1, C_2 be **coverageExpr**s
where
dim(C_1) = dim(C_2),
sdom(C_1) = sdom(C_2),
crs(C_1) = crs(C_2),
tdom(C_1) = tdom(C_2),
res(C_1) = res(C_2),
celltype(C_1) = celltype(C_2),
null(C_1) = null(C_2).

Then,

for any **coverageExpr** C_3
where

	C_{plus} = C_1 **+** C_2	and celltype(C_1), celltype(C_2) numeric
or	C_{min} = C_1 **-** C_2	and celltype(C_1), celltype(C_2) numeric
or	C_{mult} = C_1 ***** C_2	and celltype(C_1), celltype(C_2) numeric
or	C_{div} = C_1 **/** C_2	and celltype(C_1), celltype(C_2) numeric
or	C_{and} = C_1 **and** C_2	and celltype(C_1)=celltype(C_2)=Boolean
or	C_{or} = C_1 **or** C_2	and celltype(C_1)=celltype(C_2)=Boolean
or	C_{xor} = C_1 **xor** C_2	and celltype(C_1)=celltype(C_2)=Boolean
or	C_{eq} = C_1 **= =** C_2	and celltype(C_1), celltype(C_2) numeric or Boolean
or	C_{lt} = C_1 **<** C_2	and celltype(C_1), celltype(C_2) numeric or Boolean
or	C_{gt} = C_1 **>** C_2	and celltype(C_1), celltype(C_2) numeric or Boolean
or	C_{le} = C_1 **<=** C_2	and celltype(C_1), celltype(C_2) numeric or Boolean
or	C_{ge} = C_1 **>=** C_2	and celltype(C_1), celltype(C_2) numeric or Boolean
or	C_{ne} = C_1 **!=** C_2	and celltype(C_1), celltype(C_2) numeric or Boolean
or	C_{ovl} = C_1 **overlay** C_2	and celltype(C_1), celltype(C_2) numeric or Boolean

C_2 is defined as follows:

Coverage constituent	**Changed?**
for all p ∈ cdom(C_3): value(C_{plus}, p) = value(C_1) + value(C_2) value(C_{min}, p) = value(C_1) - value(C_2) value(C_{mult}, p) = value(C_1) * value(C_2) value(C_{div}, p) = value(C_1) / value(C_2)	**X**

value(c_{and}, p) = value(C_1) and value(C_2) value(c_{or}, p) = value(C_1) or value(C_2) value(c_{xor}, p) = value(C_1) xor value(C_2) value(c_{eq}, p) = value(C_1) = = value(C_2) value(c_{lt}, p) = value(C_1) < value(C_2) value(c_{gt}, p) = value(C_1) > value(C_2) value(c_{le}, p) = value(C_1) <= value(C_2) value(c_{ge}, p) = value(C_1) >= value(C_2) value(c_{ne}, p) = value(C_1) != value(C_2) value(c_{ovl}, p) = value(C_2) if value(C_1)=0 value(C_1) otherwise	
cdom(C_3) = cdom(C_1)	
for all components c of celltype(C_3): value(c_{plus}, p) = value(C_1) + value(C_2) value(c_{min}, p) = value(C_1) - value(C_2) value(c_{mult}, p) = value(C_1) * value(C_2) value(c_{div}, p) = value(C_1) / value(C_2) value(c_{and}, p) = value(C_1) and value(C_2) value(c_{or}, p) = value(C_1) or value(C_2) value(c_{xor}, p) = value(C_1) xor value(C_2) value(c_{eq}, p) = value(C_1) = = value(C_2) value(c_{lt}, p) = value(C_1) < value(C_2) value(c_{gt}, p) = value(C_1) > value(C_2) value(c_{le}, p) = value(C_1) <= value(C_2) value(c_{ge}, p) = value(C_1) >= value(C_2) value(c_{ne}, p) = value(C_1) != value(C_2) value(c_{ovl}, p) = celltype(C_2) = t where c is numeric for **+**, **-**, *****, **/**, **overlay** c is Boolean for **==**, **<**, **>**, **<=**, **>=**, **!=**	**X**
null(C_3) = null(C_1)	
im(C_3) = im(C_1)	
name(C_3) = ""	**X**
dom(C_3) = dom(C_1)	
crs(C_3) = crs(C_1)	

Example The following expression describes a coverage composed of the sum of the red, green, and blue components of coverage C:

```
C.red + C.green + C.blue
```

9.2.2.20 subsetExpr

The **subsetExpr** element specifies spatial and temporal domain subsetting. It encompasses spatial and temporal trimming (i.e., constraining the result coverage domain to a subinterval) and sectioning (i.e., cutting out a hyperplane from a coverage).

NOTE The special case that subsetting leads to a single cell remaining still resembles a coverage by definition; this coverage is viewed as being of dimension 0, with a spatial and temporal domain defined by the spatial and temporal resolution.

NOTE Range subsetting is accomplished via the unary induced operation `structSelection` (cf. Subclause 0).

9.2.2.21 trimExpr

The **trimExpr** element extracts a subset from a given coverage expression along the axis indicated, specified by a lower and upper bound.

Lower as well as upper limits **can** lie outside the coverage's domain, in which case the resulting coverage **shall** be completed with the coverage's null values.

Let

C_1 be a **coverageExpr**,
n be an integer with $0 \le n < \dim(C)$,
lo, hi be **integer**s with $lo \le hi$.

Then,

for any **coverageExpr** C_2
where
C_2 = **trim(** C_1, *axis, lo, hi* **)**

C_2 is defined as follows:

Coverage constituent	**Changed?**
for all p ∈ cdom(C_2): value(C_2, p) = value(C_1,p) for p ∈ sdom(C_1), value(C_2, p) = null(C_1) otherwise.	**X**
for all i with $0 \le i < \dim(C_2)$: cdom(C_2, i) = cdom(C_1,i) for $i \ne n$ cdom(C_2, i) = lo:hi for $i = n$	**X**
celltype(C_2) = celltype(C_1)	
null(C_2) = null(C_1)	

im(C_2) = im(C_1)	
name(C_2) = ""	**X**
for all (a,e,t) ∈ dom(C_2): dom(C_2, a) = Lo:$Hi` if ($a$,$e$,$t$) has position n in dom(C_2), dom(C_2, a) = dom(C_1, a) otherwise where lo = **stransform(**C_2 ,Lo**)**, hi = **stransform(**C_2 ,Hi**)** for t=*latitude* or t=*longitude*, lo = **ttransform(**C_2 ,Lo**)**, hi = **ttransform(**C_2 ,Hi**)** for t=*temporal*, lo, hi undefined otherwise.	**X**
crs(C_2) = crs(C_1)	

9.2.2.22 sectionExpr

The **sectionExpr** element extracts a spatial slice (hyperplane) from a given coverage expression along one of its axes, specified by section axis and section position. The resulting coverage has a dimension reduced by 1; its axes are the axes of the original coverage, in the same sequence, with the section axis being removed from the list.

The section point **can** lie outside the coverage's domain, in which case the resulting coverage **shall** be completed with the coverage's null values.

Let

C_1 be a **coverageExpr**,
n be an integer with $0 \le n < \dim(C)$,
s be an **integer**.

Then,

for any **coverageExpr** C_2
where
C_2 = **sect(** C_1, $axis$, s **)**

C_2 is defined as follows:

Coverage constituent		**Changed?**
for all p ∈ cdom(C_2): value(C_2, p) = value(C_1,p')	for p ∈ sdom(C_1) and p' is componentwise identical to p	**X**

value(C_2, p) = null(C_1) — except that p' does not contain a component for axis $axis$ and p has value s at the position of axis $axis$; otherwise	
let cdom(C_1) be given as (lo_0,hi_0),…,(lo_{d-1},hi_{d-1}) where d=dim(C_1). Then, cdom(C_2) = (lo_0,hi_0),…, (lo_{n-1},hi_{n-1}), (lo_{n+1},hi_{n+1}) ,…,(lo_{d-1},hi_{d-1})	**X**
celltype(C_2) = celltype(C_1)	
null(C_2) = null(C_1)	
im(C_2) = im(C_1)	
name(C_2) = ""	**X**
Let (a,e,t) ∈ dom(C_2, a) at position n. Then, dom(C_2) = dom(C_1) \ (a,e,t)	**X**
crs(C_2) = crs(C_1)	

9.2.2.23 crsTransformExpr

The **crsTransformExpr** element transforms a coverage into the specified co-oordinate reference system (CRS). The resulting coverage has its spatial domain transformed to the new CRS; further, the coverage values are resampled/interpolated so as to make them a well-formed grid in the new CRS.

The specified CRS **shall** be a string containing one of the CRSs listed in GetCapabilities **supportedCRS**.

Let

C be a **coverageExpr**,
crs be a **crsName,**
m be an **interpMethod**.

Then,

For any **coverageExpr** C_2,
where
C_2 = **ctransform(** C, crs, m **)**

C_2 is defined as follows:

Coverage constituent	Changed?
for all p ∈ cdom(C_2): value(C_2, p) is defined by the resampling of the coverage from crs(C_1) into crs while using interpolation method m.	X
cdom(C_2) = cdom(C_1)	
celltype(C_2) = celltype(C_1)	
null(C_2) = null(C_1)	
im(C_2) = im(C_1)	
name(C_2) = “”	X
dom(C_2) = dom(C_1) where all axes with axis type *latitude* and *longitude* have been transformed from crs(C_1) into crs	X
crs(C_2) = crs	X

Example the following expression will evaluate to a coverage that resembles C, however spatial information expressed in the coordinate reference system known by the name “EPSG:12345“; interpolation nearest-neighbor will be used for the interpolation occurring during request evaluation:

```
ctransform( C, “EPSG:12345“, “nearest-neighbor” )
```

NOTE the transformation regularly involves cell interpolation, hence potential numerical effects have to be taken into account.

9.2.2.24 scaleExpr

The **scaleExpr** element performs scaling in one axis of the source coverage using the interpolation method indicated.

Let

C_1 be a **coverageExpr**,
n be an **integer** with $0 \le n < \dim(C_1)$,
lo and hi be **integer**s with $lo \le hi$,
m be an **interpMethod.**

Then,

For any **coverageExpr** C_2,
where
C_2 = **scale (** C_1, n, lo, hi, m **)**

c_2 is defined as follows:

Coverage constituent	**Changed?**
for all p ∈ cdom(c_2): value(c_2, p) is defined by the rescaling the coverage along axis n, using interpolation method m, such that, while the geographic extent remains unchanged, the coverage's extent along axis n now is (lo,hi).	**X**
let cdom(c_1) be given as (lo_0,hi_0),…,(lo_{d-1},hi_{d-1}) where d=dim(c_1). Then, cdom(c_2) = (lo_0,hi_0),…, (lo_{n-1},hi_{n-1}), (lo,hi), (lo_{n+1},hi_{n+1}) ,…,(lo_{d-1},hi_{d-1})	
celltype(c_2) = celltype(c_1)	
null(C_2) = null(C_1)	
im(C_2) = im(C_1)	
name(c_2) = ""	**X**
Let (a,e,t) ∈ dom(c_2, a) at position n. Let further e' be the rescaled extent of c_2 in axis n. Then, dom(C_2) = dom(C_1) \ (a,e,t) ∪ (a,e',t)	**X**
crs(c_2) = crs(c_1)	

NOTE The transformation regularly involves cell interpolation, hence potential numerical instabilities have to be taken into account.

Example The following expression performs a scaling of coverage C by factor s in x and y: using interpolation method *bicubic*:

```
scale( scale( C, x, s, bicubic ), y, s, bicubic )
```

9.2.2.25 coverageConstructorExpr

The **coverageConstructorExpr** element allows to create an n-dimensional coverage with its content defined by a general expression. Only Level 0 and Level 1 data are set, all Level 2 data are left empty or undefined, resp.

NOTE This constructor is useful

- whenever the coverage is too large to be described as a constant or

- when the coverage's cell values are derived from some other source (such as a histogram computation, see example below).

Let

x be a **name**,
D be a **cell domain**,
V be a **scalarExpr** possibly containing occurrences of *x*.

Then,

For any **coverageExpr** *C*
where
C = **coverage** *x* **in** *D* **values** *V*

C is defined as follows:

Coverage constituent	**Changed?**
for all $p \in \text{cdom}(C)$: value(C, p) = $V\|_{x \to p}$ i.e., the cell value at position *p* is obtained by evaluating expression *V* after substituting all occurrences of *x* by *p*	**X**
cdom(*C*) = *D*	**X**
celltype(*C*) = type(*V*)	**X**
null(C) is false for boolean, 0 for numeric types	**X**
im(C_2) = *none*	**X**
name(C_2) = *""*	**X**
dom(C_2) is empty	**X**
crs(C_2) = undefined	**X**

Example The expression below computes a 256-bucket histogram over some n-dimensional coverage *C*.

```
coverage n in [0:255]
values count( C = n )
```

9.2.2.26 condenseExpr

A **condenseExpr** is either a **reduceExpr** (see Section 9.2.2.27) or a **generalCondense-Expr** (see Section 9.2.2.27). It takes a coverage and summarizes its values using some summarization function. The value returned is scalar.

9.2.2.27 generalCondenseExpr

The general **generalCondenseExpr** consolidates cell values of a coverage to a scalar value based on the condensing operation indicated. It iterates over a given domain while combining the result values of the **scalarExpr**s through the **condenseOpType** indicated.

Any summarisation function s() is admissible for a **generalCondenseExpr** over some coverage if it has the following properties:

- s() is a binary function between values of the coverage cell type;
- s() is commutative and associative.

Let

op be a **condenseOpType**,
x be a **name**,
D be a **cell domain**,
P be a **booleanExpr** possibly containing occurrences of *x*,
V be a **scalarExpr** possibly containing occurrences of *x*.

Then,

For any **scalarExpr** *S*
where

S = **condense** *op* **over** x **in** *D* [**where** *P*] **using** *V*

S is constructed as follows:

- Let *S* = neutral element of type(*V*)
- for all $x \in$ value(*D*) where ($P|_{x \to p}$):
- *S* = *S* value(*op*) value($V|_{x \to p}$)
  (i.e., the cell value at position *p* is obtained by evaluating expression *V* after substituting all occurrences of *x* by *p*, provided predicate *P* is fulfilled for this position)

Null values encountered **shall** be treated as follows:

- if at least one non-null value is encountered in the repeated evaluation of *V*, then all null values **shall** be ignored;

- if V is not evaluated at least once, or if there are only null-valued results, then the overall result **shall** be null.

Example Binary "+" on floating point numbers is admissible for a condenser on a float coverage, while binary "-" is not.

Example For a filter kernel, the condenser must summarise not only over the cell under inspection, but also some neighbourhood. The following applies a filter kernel to some coverage C:

```
coverage x in cdom(C)
  values condense +
         over y in cdom(kernel)
         using C[x+y] * kernel[y]
```

where `kernel` is a 3x3 matrix like

1	3	1
0	0	0
-1	-3	-1

NOTE Condensers are heavily used in two situations:

- To collapse Boolean-valued coverage expressions into scalar Boolean values so that they can be used in predicates.
- In conjunction with the **coverageConstructorExpr** (see Section 9.2.2.25) to phrase high-level imaging, signal processing and statistical operations.

NOTE The additional expressive power of **condenseExpr** over **reduceExpr** is twofold:

- A WCPS implementation may offer further summarisation functions.
- The condenseExpr gives explicit access to the coordinate values; this makes summarisation considerably more powerful (see example below).

9.2.2.28 reduceExpr

A **reduceExpr** element derives a summary value from the coverage passed; in this sense it "reduces" a coverage to a scalar value. A **reduceExpr** is either an `add`, `avg`, `min`, `max`, `count`, `some`, or `all` operation.

Table 6 – reduceExpr definition via generalCondenseExpr (*a* is a numeric, *b* a Boolean coverageExpr)

reduceExpr definition	Meaning

`add(a) =`
`  condense +`
`  over x in sdom(a)`
`  using a[x]`
	sum over all cells in *a*

`avg(a) =` `  add(a) / | cdom(a) |`	Average of all cells in *a*

`min(a) =`
`  condense min`
`  over x in sdom(a)`
`  using a[x]`
	Minimum of all cells in *a*

`max(a) =`
`  condense max`
`  over x in sdom(a)`
`  using a[x]`
	Maximum of all cells in *a*

`count(b) =`
`  condense +`
`  over x in sdom(b)`
`  where b[x]`
`  using 1`
	Number of cells in *b*

`some(b) =`
`  condense or`
`  over x in sdom(b)`
`  using b[x]`
	is there any cell in *b* with value true?

`all(b) =`
`  condense and`
`  over x in sdom(b)`
`  using b[x]`
	do all cells of *b* have value true?

9.2.2.29 coordinateTransformExpr

The **coordinateTransformExpr** element specifies translation from a spatial (**spatial-TransformExpr**) or temporal (**temporalTransformExpr**) reference system into cell (grid) coordinates.

9.2.2.30 spatialTransformExpr

The **spatialTransformExpr** element specifies the transformation of a geographic point coordinate given in some reference system into a coverage's cell (grid) coordinates.

Let

c be a **coverageExpr** whose Domain has axes of axis type *latitude* and *longitude*,
g be a **spatialPointCoordinate**,
c be a **crsName**.

Then,

For any **gridPointCoordinate** *p*
where
p = **stransform(** *C*, *g*, *c* **)**

p is defined as that integer point coordinate which, given the coordinate mapping of coverage *C*, is equivalent to the location expressed by geo coordinate *g* relative to coordinate reference system *c*, rounded to the spatially closest cell coordinate.

NOTE A coverage's individual mapping from geo to cell coordinates does not need to be disclosed by the server, hence this transformation should be considered a "black box" by the client.

Example the following expression yields a 2-D integer coordinate:

```
stransform( C, [ 15.001, 19.999 ], "EPSG:0815" )
```

9.2.2.31 temporalTransformExpr

The **temporalTransformExpr** element specifies the transformation of a time point coordinate into a coverage's cell (grid) coordinates.

Let

C be a **coverageExpr** whose Domain has an axis of type temporal,
t be a **temporalPointCoordinate**.

Then,

For any **coverageExpr** C_2
integer value *p*,
where
p = **ttransform(** *C* , *t* **)**

p is defined as the integer value which, given the coordinate mapping of coverage *C*, is equivalent to the point in time expressed by *t*, rounded to the temporally closest cell coordinate.

Example Given a coverage C with a time axis, the following expression will evaluate to an integer number representing the coordinate corresponding to the indicated point in time in the coverage:

```
ttransform( C, "Thu Nov 24 01:33:27 CET 2005" )
```

9.2.3 Expression evaluation

9.2.3.1 Evaluation sequence

A Web Coverage Processing Server **shall** evaluate coverage expressions from left to right.

9.2.3.2 Nesting

A Web Coverage Processing Server **shall** allow to nest all operators, constructors, and functions arbitrarily, provided that each sub-expression's result type matches the required type at the position where the sub-expression occurs. This holds without limitation for all arithmetic, Boolean, String, and coverage-valued expressions.

9.2.3.3 Parentheses

A Web Coverage Processing Server **shall** allow use of parentheses to enforce a particular evaluation sequence.

Let

C_1 and C_2 be **coverageExpr**s

Then,

For any **coverageExpr** C_2
where
C_2 = **(** C_1 **)**

C_2 is defined as being equivalent to C_1.

Example `C * ( C > 0 )`

9.2.3.4 Operator precedence rules

In case of ambiguities in the syntactical analysis of a request, operators **shall** have the following precedence (listed in descending strength of binding):

- dot ".", trimming, section
- unary –
- unary arithmetic, trigonometric, and exponential functions
- `*`, `/`
- `+`, `-`
- `<`, `<=`, `>`, `>=`, `!=`, `=`
- `and`
- `or`, `xor`
- `":"` (interval constructor), condense, marray
- overlay

In all remaining cases evaluation is done left to right.

9.2.3.5 Type extension

Whenever coverages of different cell type are combined in one operation then the cell types of both coverage operands shall be repeatedly extended until both types match. Extending a type is defined as replacing a type which appears on the left-hand side in Table 7 by the type to its right.

Table 7 – Type extension sequence

Type extension rules
`Boolean > short`
`Boolean > unsigned short`
`short > int`
`short > unsigned int`
`unsigned short > int`
`unsigned short > unsigned int`
`int > long`
`int > unsigned long`
`unsigned int > long`
`unsigned int > unsigned long`
`long > float`
`float > double`

Extending Boolean to (unsigned) short **shall** map `false` to 0 and `true` to 1.

In case of a matching an implicit cast shall be performed to change one or both coverages to the extended cell type.

If such a matching is not possible an exception **shall** be reported.

Example For coverages `F`, `I`, and `B` of cell type `float`, `integer`, and `Boolean`, resp., the result type of the following expression is `float`:

```
F + I + B
```

NOTE This behavior is similar to programming languages and database query languages.

9.2.4 Response overview

The response to a valid ProcessCoverage request **shall** consist of one of the following alternatives:

- A coverage, encoded in a particular data format, or a sequence of encoded coverages
- A component of a coverage, or a sequence of coverage components
- A scalar numeric value, or a sequence of such values

Responses **shall** be formed as laid out in Subclause 9.2.6.1.

In an HTTP environment, the returned value **shall** have a Content-type entity header that matches the format of the return value.

9.2.4.1 Response structure

A **ProcessCoverage** response consists of an XML structure plus, if URL forwarding has been specified for the result provision, of one or more data files accessible through the URLs communicated by the server. The XML response type is `ProcessCoverage-ResponseType` (see Annex B).

Depending on the response type, the response to a WCPS request **shall** be one of the following:

- An encoded coverage or a sequence of encoded coverages where the response forwarding requested was *direct transmission* **shall** be transmitted to the client in multi-part MIME encoding, without any additional XML response.
- An encoded coverage or a sequence of encoded coverages where the response forwarding requested was *URL forwarding* **shall** be transmitted to the client in an XML response of type `ProcessCoverageResponseUrlType` containing a sequence of URLs where each URL refers to one response list element, in proper sequence. The server shall provide each encoded coverage result at the resp. URL at least until the specified expiration period.
- A component of a coverage or a sequence of coverage components **shall** be transmitted to the client in an XML structure of type `ProcessCoverage-ResponseComponentType`.
- A scalar numeric value, or a sequence of such values **shall** be transmitted to the client in an XML structure of type `ProcessCoverageResponseScalarType`.

9.2.4.2 Exceptions

An invalid **ProcessCoverage** request **shall** yield an error output, either as a WCPS exception reported in the requested Exceptions format, or as a network protocol error response.

A Web Coverage Processing server throwing an exception **shall** adhere to the value of the Exceptions parameter. Nonetheless, a Web Coverage Processing server **may**, due to circumstances beyond its control, return nothing (this might result from the HTTP server's behavior caused by a malformed request, by an invalid HTTP request, by access violations, or any of several other conditions). Web Coverage Processing clients should be prepared for this eventuality.

9.2.5 Key-value pair encoding

The key-value pair encoding allows clients to use the HTTP GET method for transmitting **ProcessCoverage** requests.

9.2.5.1 Overview

Table 8 specifies the complete **ProcessCoverage** Request.

Table 8 – The ProcessCoverage Request expressed as Key-Value Pairs

URL Component	Description	Multiplicity
http://server_address/path/script?	URL of WCS server.	*Required*
SERVICE=WCPS	Service name: Must be "WCPS".	*Required*
VERSION=*m.n.p*	Request protocol version, *m*, *n*, *p* being non-negative integer numbers.	*Required*
REQUEST=ProcessCoverage	Name of the request. Must be "ProcessCoverage".	*Required*
RESULT=*expr*	The expression describing the result coverage(s) derived from the coverage offering. Must be conformant to Section 0 below.	*Required*
STORE=*f*	Determines whether response is sent back immediately (*f=false*) or stored on the server (*f=true*). Default: *false*	*Optional*
KEEPTIME=*k*	Time until response may become unavailable at the server, if STORE=*true*; for STORE=*false*, a KEEPTIME parameter is not allowed. *k* is a valid time specification Default: respone becomes unavailable at the server's discretion	*Optional*
EXCEPTIONS= application/vnd.ogc.se_xml	The format in which exceptions are to be reported by the server. The currently only allowed format is XML. Default: application/vnd.ogc.se_xml	*Optional*
(Vendor-specific parameters)	Default: none	*Optional*

9.2.5.2 SERVICE=WCPS / VERSION=*version*

These parameters are defined as for **GetCapabilities** in Subclause 8.

9.2.5.3 REQUEST=ProcessCoverage

The Basic Service Elements clause defines this parameter. For **ProcessCoverage**, the value "ProcessCoverage" **shall** be used.

9.2.5.4 RESULT=*expr*

The RESULT argument is a valid WCPS expression, in the abstract syntax as specified in Section 9.2.2.

For the URL encoding the pertaining IETF rules [6] **shall** be used.

9.2.5.5 EXCEPTIONS

A Web Coverage Processing Service **shall** offer the exception reporting format *application/vnd.ogc.se_xml* by listing it in its **GetCapabilities** XML response. The entire MIME type string in **Capability / Exceptions / Format** is used as the value of the EXCEPTIONS parameter.

Errors are reported using Service Exception XML, as specified in Subclause B.3. This is the default exception format if none is specified in the request.

9.2.6 XML encoding

9.2.6.1 Overview

The XML encoding is an alternative to the KVP encoding (Subclause 9.2.2.29), with the same semantics and expressive power. See Annex B for the XML definitions.

Annex A
(normative)

WCPS Language Syntax

A.1 Overview

The WCPS expression syntax is described below in BNF grammar syntax.

Boldface tokens represent literals which appear as is in a valid WCPS expression ("terminal symbols", tokens in italics represent sub-expressions to be substituted according to the grammar production rules ("non-terminals").

Meta symbols used are as follows:

- brackets ("[…]") denote optional elements which **may** occur or not;
- an asterisk ("*") denotes that an arbitrary number of repetitions of the preceding element **can** be chosen, including none at all;
- a vertical bar ("|") denotes alternatives from which exactly one **must** be chosen;
- Double slashes ("//") begin comments which continue until the end of the line.

A.2 WCPS syntax

```
coverageListExpr:
        for iteratorVar in ( coverageList )
        [ where booleanScalarExpr ]
        return processingExpr

iteratorVar: name

coverageList:
        coverageOrSetName [ , coverageOrSetName ]*

coverageOrSetName:
        name

booleanScalarExpr:
        scalarExpr   // …whose result type is Boolean

processingExpr:
        encodedCoverageExpr
      | scalarExpr

encodedCoverageExpr:
        encode ( coverageExpr, formatName )
      | encode ( coverageExpr, formatName, extraParams )
```

```
formatName:
        name

extraParams:
        string

scalarExpr:
        getMetaDataExpr
      | generalCondenseExpr
      | ( scalarExpr )

getMetaDataExpr:
        sDomain( coverageExpr )
      | crs( coverageExpr )
      | xDomain( coverageExpr )
      | yDomain( coverageExpr )
      | tDomain( coverageExpr )
      | res( coverageExpr )
      | dim( coverageExpr )
      | null( coverageExpr )
      | cellType( coverageExpr )

coverageExpr:
        coverageName
      | setMetaDataExpr
      | inducedExpr
      | subsetExpr
      | crsTransformExpr
      | scaleExpr
      | coverageConstExpr
      | coverageConstructorExpr
      | ( processingExpr )

 setMetaDataExpr:
      | setNull ( coverageExpr , scalarLit )
      | setInterpolation ( coverageExpr , interpMethod )
      | setCrs ( coverageExpr , crsName )
      | setAxis ( coverageExpr , axisName ,
                  floatExpr , floatExpr )

inducedExpr:
      | unaryInducedExpr
      | binaryInducedExpr

unaryInductionExpr:
        arithmeticExpr
      | exponentialExpr
      | trigonometricExpr
      | booleanExpr
      | castExpr
      | selectExpr
```

```
arithmeticExpr:
          + coverageExpr
        | - coverageExpr
        | sqrt ( coverageExpr )
        | abs ( coverageExpr )

exponentialExpr:
          exp ( coverageExpr )
        | log ( coverageExpr )
        | ln ( coverageExpr )

trigonometricExpr:
          sin ( coverageExpr )
        | cos ( coverageExpr )
        | tan ( coverageExpr )
        | sinh ( coverageExpr )
        | cosh ( coverageExpr )
        | tanh ( coverageExpr )
        | arcsin ( coverageExpr )
        | arccos ( coverageExpr )
        | arctan ( coverageExpr )

booleanExpr:
          not coverageExpr
        | bit ( coverageExpr , integerExpr )

castExpr:
          ( cellType ) coverageExpr

cellType:
          bool
        | char
        | unsigned char
        | short
        | unsigned short
        | long
        | unsigned long
        | float
        | double

selectExpr:
          coverageExpr structSelection

structSelection:
          . selector
        | . integerExpr

binaryInducedExpr:
        | coverageExpr plus coverageExpr
        | coverageExpr minus coverageExpr
        | coverageExpr mult coverageExpr
        | coverageExpr div coverageExpr
```

```
        | coverageExpr and coverageExpr
        | coverageExpr or coverageExpr
        | coverageExpr xor coverageExpr
        | coverageExpr = = coverageExpr
        | coverageExpr < coverageExpr
        | coverageExpr > coverageExpr
        | coverageExpr <= coverageExpr
        | coverageExpr >= coverageExpr
        | coverageExpr != coverageExpr
        | coverageExpr overlay coverageExpr

subsetExpr:
        | trimExpr
        | sectExpr

trimExpr:
          trim ( coverageExpr , axisName ,
                 integer , integer )

sectExpr:
          sect ( coverageExpr , axisName ,
                 integer )

crsTransformExpr:
          ctransform( coverageExpr , crsName , interpMethod )

scaleExpr:
          scale ( coverageExpr , axisName ,
                  integer , integer , interpMethod )

coverageConstructorExpr:
          coverage variable in cellDomainExpr
          values scalarExpr

condenseExpr:
          reduceExpr
        | generalCondenseExpr

reduceExpr:
          all ( coverageExpr )
        | some ( coverageExpr )
        | count ( coverageExpr )
        | add ( coverageExpr )
        | avg ( coverageExpr )
        | min ( coverageExpr )
        | max ( coverageExpr )

generalCondenseExpr:
          condense condenseOpType
          over variable in cellDomainExpr
          [ where booleanScalarExpr ]
          using scalarExpr
```

```
condenseOpType:
        +
      | *
      | max
      | min
      | and
      | or

coordinateTransformExpr:
      | spatialTransformExpr
      | temporalTransformExpr

spatialTransformExpr:
        stransform ( coverageExpr , gridPointCoordinate ,
                     crsName )

temporalTransformExpr:
        ttransform ( coverageExpr ,
                     temporalPointCoordinate )

interpMethod:
        nearest-neighbor
      | bilinear
      | bicubic
      | lost-area
      | barycentric

spatialPointCoordinate:
        [ floatExpr [ , floatExpr ]* ]

temporalPointCoordinate:
        " string "      // ows:Time

cellDomainExpr:


gridPointCoordinate:
        [ integerExpr [ , integerExpr ]* ]

crsName:
        string  // containing a valid EPSG CRS name of format "EPSG:nnnn"
                // where n is a decimal digit

coverageName:
        name

axisName:
        name

variable:
        name

name:  // nonempty sequence of printable ASCII characters, not starting with digits
```

```
scalarLit:
          complexLit
        | atomicLit

complexLit:
          { scalarLitList }
        | struct { scalarLitList }

atomicLit:
          booleanExpr
        | integerExpr
        | floatExpr

booleanExpr:
          // an expression resulting in a Boolean value

integerExpr:
          // an expression resulting in an integer value

floatExpr:
          an expression resulting in a floating point value
```

A name **shall** be a consecutive sequence consisting of decimal digits, upper case alphabetical characters, lower case alphabetical charaters, underscore ("_"), and nothing else. The length of a name **shall** be at least 1, and the first character **shall not** be a decimal digit.

An integer number **shall** be expressed in either decimal, octal (with a "0" prefix), or hexadecimal notation (with a "0x" or "0X" prefix).

A floating point number **shall** be expressed as in a Java programming language source code.

Annex B
(**normative**)

WCPS XML Schemas

B.1 GetCapabilities request Schema

See file wcpsCapabilities.xsd

B.2 GetCapabilities response schema

See file wcpsCapabilities.xsd

B.3 DescribeCoverage request schema

See file wcpsDescribeCoverage.xsd

B.4 DescribeCoverage response schema

See file wcpsDescribeCoverage.xsd

B.5 GetCoverage request schema

See file wcpsGetCoverage.xsd

B.6 Service exception schema

See file wcpsException.xsd.

Annex D
(normative)

Conformance

D.1 Introduction

TBD.

Bibliography

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

[2] ISO 19103, Geographic Information – Conceptual schema language

[3] OMG Unified Modeling Language Specification (UML), Version 1.5, March 2003, http://www.omg.org/docs/formal/03-03-01.pdf

[4] OGC 03-065r6, Web Coverage Service (WCS), Version 1.0.0

[5] European Petroleum Survey Group, EPSG Geodetic Parameter Set, Version 6.8

[6] IETF RFC 2396

[7] IETF RFC 2616