Open Geospatial Consortium Inc.

Date: 2009-03-25

Reference number of this OGC® Project Document: **OGC 08-068r2**

Version: 1.0.0

Category: OpenGIS® Interface Standard

Editor: Peter Baumann

Web Coverage Processing Service (WCPS) Language Interface Standard

Document type:	OpenGIS® Interface Standard
Document subtype:	Extension Package
Document stage:	Approved
Document language:	English

Contents

Page

Tables

i. Preface

The OGC® Web Coverage Processing Service (WCPS) defines a protocol-independent language for the extraction, processing, and analysis of multi-dimensional coverages representing sensor, image, or statistics data.

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

ii. Submitting organizations

The following organizations have submitted this Interface Standard to the Open Geospatial Consortium, Inc.

- Jacobs University Bremen

iii. Document Contributor Contact Points

Name	Organization
Peter Baumann	Jacobs University Bremen, rasdaman GmbH

iv. Revision history

Date	Release	Author	Paragraph modified	Description
2008-04-29	0.0.1	Peter Baumann		created from 07-151r1 and 08-059r3
2008-09-01	0.0.2	Peter Baumann	Many	Final version following adoption
2008-09-22	0.0.3	Peter Baumann	Result type specs	More accurate phrasing, not just table reference
2009-01-08	1.0.0	Peter Baumann	many	Reworked scalar operations; incorporated e-vote comments; fixed syntax inconsistencies; final editorial brush-up

v. Changes to the OpenGIS® Abstract Specification

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

vi. Future Work

This WCPS framework will be enhanced and extended incuding the following features:

1) WCPS is based on the conceptual model of OGC WCS and OGC Abstract Topic 6. As the revision of WCS and Abstract Topic 6 proceeds, WCPS will have to be adapted to maintain coherence.

2) In particular, the GeneralDomain is making its way into WCS 1.2; once this is accepted, WCPS has to be adapted to the agreed shape and usage of GeneralDomain.

3) Add support for further coverage types beyond equidistant grids.

4) Add support for inserting, updating, and deleting coverages through expressions (harmonized with WCS-T).

5) Refine metadata probing functions.

6) Extend metadata querying functionality to non-primitive data structure elements.

7) Establish a WPS profile for WCPS (in a separate specification document).

Foreword

The WCPS language standard evolved from an earlier Best Practice Paper [OGC 07-157r1], and supersedes that document. This document does not supersede any other previously approved OGC document.

This document includes two normative Annexes, A and B.

Introduction

The OGC Web Coverage Processing Service (WCPS) defines a language for retrieval and processing of multi-dimensional geospatial coverages representing sensor, image, or statistics data. Services implementing this language provide access to original or derived sets of geospatial coverage information, in forms that are useful for client-side rendering, input into scientific models, and other client applications.

WCPS relies on the coverage model as defined in OGC Abstract Specification Topic 6 “Schema for Coverage Geometry and Functions “ [OGC 07-011] and the OGC Web Coverage Service (WCS) Standard [OGC 07-067r5] where coverages are defined as “digital geospatial information representing space-varying phenomena”, currently constrained to equally spaced grids.

The WCPS language is independent from any particular request and response encoding, as no concrete request/response protocol is specified by WCPS. For setting up a WCPS instance, therefore, a separate, additional specification establishing the concrete protocol is required. This allows embedding of WCPS into different target service frameworks.

One such target framework is OGC WCS. Together with the pertaining request type definition [OGC 08-059r3] WCPS forms an extension of the Web Coverage Service (WCS) version 1.1.2 Standard [OGC 07-067r5]. With small changes, this extension is expected to also apply to subsequent versions of WCS.

NOTE A WPS profile of the WCPS language is under preparation.

Open Geospatial Consortium Interface: Web Coverage Processing Service (WCPS)

1 Scope

This document defines a protocol-independent language for retrieving and processing geospatial coverage data.

Like WCS, WCPS is currently limited to quadrilateral grid coverages, providing information at the grid points, usually with interpolation between these grid points.

NOTE For future versions of this standard it is intended to extend WCPS to incorporate further coverage types defined in the OGC Abstract Specification Topic 6 “Schema for Coverage Geometry and Functions" [OGC document 07-011, in synchronization with WCS.

2 Compliance

Annex A (normative) specifies compliance tests which shall be tested by any service claiming to implement WCPS.

3 Normative references

The following normative documents contain provisions that, through reference in this text, constitute provisions of this standard. 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 2616, *Hypertext Transfer Protocol -- HTTP/1.1*

ISO 8601:2000, *Data elements and interchange formats — Information interchange — Representation of dates and times*

OGC 05-007r7, *Web Processing Service Implementation Standard*, version 1.0.0

OGC 06-121r3, *OpenGIS® Web Services Common Standard*, version 1.1.0

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

OGC 07-011, Abstract Specification Topic 6: Schema for Coverage Geometry and Functions, version 7.0

OGC 07-067r5, *OpenGIS® Web Coverage Service Implementation Standard*, version 1.1.2

NOTE The WCS standard cited contains a list of normative references that are also applicable to this extension standard.

OGC 07-092r1, *Definition identifier URNs in OGC namespace*, version 1.1.2

OGC 08-053r2, *WCS Processing Extension Abstract Test Suite*, version 1.0.0

OGC 08-059r3, *WCS Processing Extension*, version 1.0.0

4 Terms and definitions

For the purposes of this document, the terms and definitions given in the above references (in particular: WCS [OGC 07-067r5]) apply.

5 Conventions

5.1 Symbols (and abbreviated terms)

Most of the abbreviated terms listed in Subclause 5.1 of the OWS Common Standard [OGC 06-121r3] also apply to this document.

Further, this document assumes familiarity with the terms and concepts of the Web Coverage Service Standard [OGC 07-067r5].

5.2 UML notation

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

5.3 Platform-neutral and platform-specific specifications

In terms of Clause 10 of OGC Abstract Specification Topic 12 “OpenGIS Service Architecture” (which contains ISO 19119), this document includes only Distributed Computing Platform-neutral specifications. This document specifies each operation request and response in platform-neutral fashion. This is done using a semi-formal approach to recursively specifying the language syntax and semantics. To assist this, the coverage model is formalized as well (but not changed over WCS OGC 07-067r5]).

The specified platform-neutral data could be encoded in many alternative ways, each appropriate to one or more specific DCPs. One service embedding and encoding is defined in the WCS Processing Extension [08-059r3]. Other encodings may specify an API (Application Programming Interface approach) with actually no networks communication involved between “client” and “server”.

6 Conceptual coverage model

The coverage model of WCPS relies on the coverage model of WCS [OGC 07-067r5] and formalizes it in a way which is suitable for expressing the semantics of the operations in Section 7.1. Subclause 6.2 describes the constituents of a WCPS coverage by defining a set of coverage probing functions. Some restrictions that apply with regard to general WCS coverages are listed in Section 6.3.

NOTE The coverage model specified in this section serves as an interim substitute for a more formalized overall WCS coverage model; once said model is in place it will replace this section.

6.1 Coverage model

6.1.1 Coverages

A coverage consists of a set of locations bearing some value. Following the mathematical notion of a function that maps elements of a domain (here: spatio-temporal[1] coordinates) to a range (here: "pixel", "voxel", … values), the set of coverage locations bearing values is called the coverage domain while the set of possible values, i.e., the coverage value data type, is called the coverage range.

A coverage domain with its set of locations (or *coordinates*) is aligned along some d-dimensional grid where $d>0$ is called the coverage's *dimensionality*. The coordinate space, i.e. the set of all possible coordinates, is spanned by d independent dimension axes. A *dimension axis* (abbreviated also as *dimension* or as *axis*) is identified by its name which is unique within the coverage. The set of all dimension axis names of a coverage C is obtained via the function dimensionNameSet(C).

6.1.2 Dimensions

Each dimension has an dimension type associated, which is one of the elements listed in Table 1. A coverage **can** have at most one *x*, *y*, *z*, and *t* dimension.

Table 1 – Coverage domain dimension types.

Dimension type	Meaning
x	East-West extent, expressed in the coverage's CRS
y	North-South extent, expressed in the coverage's CRS
z	Geographical elevation, i.e., height or depth
t	Time

Each dimension **shall** have one or more coordinate reference systems (CRSs) associated, one of them being – according to WCS – either an ImageCRS or a GridCRS (henceforth

[1] And in future, once the corresponding extension to WCS is available, abstract axes without spatiotemporal semantics

collectively termed the Image CRS). Any number of further CRSs **can** be associated with a coverage dimension, given by the set crsSet(C,a). Image CRS and further CRSs together determine the set of CRSs which can be used in coordinate-aware operations. Time coordinates coordinates **shall** be expressed as strings adhering to [ISO 8601:2000].

NOTE An image CRS always allows to address a coverage in all dimensions. For the other CRSs, however, several CRSs together may be necessary to fully address a coverage – for example, WGS84 only knows x and y and thus does not allow to specify z and t coordinates in a 4-D x/y/z/t climate model.

The WCPS service does **not need** to publish the mapping between coordinates of the different supported CRSs.

6.1.3 Locations

A location L is a set

$$L_C = \{ (a,c,p) \mid a \in \text{dimensionNameSet}(C),\ c \in \text{crsSet}(C),\ p \in \text{DimensionPointValues} \}$$

consisting of dimension names, per dimension the coordinate system used, and a coordinate relative to this dimension and CRS; each of the coverage's dimension name **shall** appear exactly once in this set. The set DimensionPointValues is a generalization of numeric and string values that allows to express all kind of coordinates, including geographic floating-point coordinates and date/time strings.

Example For dimension type *t*, encoding follows [ISO 8601:2000] as described in WCS [OGC 07-067r5] Table 16, 17 and owsTime (that is, the possible values are ASCII strings). For an image CRS, encoding will be integer for all dimension types, and for x/y type geographic coordinates it will usually be float.

On each dimension a total ordering relation "≤" **shall** be available under all CRSs used.

Example On a *t* dimension, this ordering relation will yield true for the following comparison:

```
  "Sun Jan 1 23:59:59 CET 2006"
≤ "Tue Dec 5 22:17:48 CET 2006"
```

Along each dimension a coverage is delimited by a lower and upper bound value, these border values being part of the coverage extent. Location addresses always are relative to a particular coverage.

6.1.4 Domain

The set of all locations contained in a coverage forms its domain. A domain's location set always is non-empty. It can be described, for each dimension, by a lower and upper bound (lo,hi) expressed in one of the coverage's CRSs applicable for this dimension where $lo \leq hi$.

For a coverage C, function domain() describes its domain structure, which is a set of dimension descriptions consisting of dimension name, dimension type, CRS used, and the lower and upper bound of the coverage domain expressed in the CRS at hand:

domain(C) = { (a,t,c,lo,hi) | $a \in$dimensionNameSet(C), $t \in \{x,y,z,time\}$,
$c \in$crsSet(C), lo,$hi \in$DimensionPointValues, $lo \leq hi$ }

A location L is *inside* the domain of a coverage C if its coordinates are inside the domain extent under all CRSs supported:

Let

C be a coverage,
L_C be a location wrt. coverage C
with L_C = { (a,c,p) | $a \in$dimensionNameSet(C), $c \in$crsSet(C), $p \in$DimensionPointValues },
G_C be the domain of coverage C
with G_C = { (a,t,c,lo,hi) | $a \in$dimensionNameSet(C), $t \in \{x,y,z,time\}$,
$c \in$crsSet(C), lo,$hi \in$DimensionPointValues, $lo \leq hi$ }.

Then,

L_C inside G_C
if and only if
for all (a,c,p)$\in L_C$ there is some lo,$hi \in$DimensionPointValues such that:
(a,t,c,lo,hi)$\in G_C$ and $lo \leq p \leq hi$ relative to CRS c

6.1.5 Range values and types

The value associated with a particular location within a coverage, in short: its point value, can be obtained with probing function value(C,l_C) which is defined for every location $l_C \in$imageCrsDomain(C) and l_C inside domain(C).

All grid point values of a coverage share the same type, the coverage's range type. Admissible types consist of named components called fields; each field is identified by a field name unique for the coverage on hand and bears one of the (atomic) numeric or Boolean types enumerated in the set RangeFieldTypes (see Table 2):

RangeFieldTypes = { boolean, char, unsigned char, short, unsigned short, int, unsigned int, long, unsigned long, float, double, complex, complex2 }

NOTE 1 This is a restriction over WCS [OGC 07-067r5], see Subclause 6.3.

NOTE 2 It is not required that all range fields within a coverage are of the same type.

NOTE 3 Range fields are also known as "bands" or "channels".

A coverage's range type description can be obtained by function rangeType() which delivers a set of pairs of field names and field type:

rangeType(C) = { (f,t) | $f \in$ rangeFieldNames(C), $t \in$ RangeFieldTypes }

Table 2 – Coverage range field data types.

Range data type name	Meaning
boolean	Boolean
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
complex	Single precision complex number
complex2	Double precision complex number

6.1.6 Null and interpolation

The set of a coverage's values to be interpreted as null is obtained via probing function nullSet(). This set **can** be empty.

Probing function interpolationSet(*C*) returns a set of pairs (*im*,*nr*) where *im* indicates the interpolation type and *nr* the null resistance employed. This set **can** be empty.

Interpolation method defaults associated with a coverage can be obtained through probing function interpolationDefault().

6.2 Coverage probing functions summary

A set of so-called *probing functions* allows to extract the constituents listed above from a given coverage. These functions are not part of the WCPS interface standard, but serve for the sole purpose of defining the semantics of *ProcessCoverages* requests in Clause 7.1.

Table 3 lists the probing functions provided.

For notational convenience in this document, on the list and set valued items the usual list and set functions are assumed for extraction and manipulation, such as union, intersection. Further, application of some function to a list or set which is defined on the elements denotes simultaneous application of this function to all list or set elements.

Table 3 – Coverage probing functions.

Coverage characteristic	Probing function for some coverage C	Comment
Identifier	identifier(*C*)	For original coverages only, not for

		processed coverage results
Grid point values	value(C,p) for all $p \in$ imageCrsDomain(C)	The coverage grid point ("pixel"), "voxel", …) values, of data type rangeType(C)
Domain dimension list	dimensionList(C)	List of all of the coverage's dimension names, in their proper sequence
Domain dimension type	dimensionType(C, a)	Dimension type
Image CRS	imageCRS(C)	Image CRS of the coverage, allowing direct grid point addressing
Domain extent of coverage, expressed in Image CRS	imageCrsDomain(C)	Extent of the coverage in (integer) grid coordinates, relative to the coverage's Image CRS[2]; essentially, the set of all point coordinates inside the coverage
Domain extent of coverage along dimension, expressed in Image CRS	imageCrsDomain(C, a) for some $a \in$ dimensionList(C)	Extent of the coverage in (integer) grid coordinates, relative to the coverage's Image CRS, for a given dimension; essentially, the set of all values inside the extent interval
CRS set	crsSet(C, a) for some $a \in$ dimensionList(C)	Set of all CRSs from the supported CRS (see [OGC 07-067r5] DescribeCoverage)
extent of coverage along dimension, expressed in arbitrary CRS	domain(C, a, c) for some $a \in$ dimensionList(C) and some $c \in$ crsSet(C)	domain of the coverage, expressed in one of its CRSs, for a given (spatial, or temporal) dimension
Range data type	rangeType(C)	The data type of the coverage's grid point values, given as a set of pairs of field Name and (atomic) data type
Range field type	rangeFieldType(C, f) for some $f \in$rangeFieldNames(C)	The data type of one coverage range field, given as some atomic type name
Range field name set	rangeFieldNames(C)	Set all of the coverage's range fields names
Null value set	nullSet(C, r) for all $r \in$ rangeType(C)	The set of all values that represent null as coverage range field value
Default interpolation method	InterpolationDefault(C, r) for all $r \in$ rangeType(C)	Default interpolation method, per coverage field
Interpolation method set	interpolationSet(C, r) for all $r \in$ rangeType(C)	All interpolation methods applicable to the particular coverage range field; must list at least the default interpolation method
Interpolation type	interpolationType(im) for all $im \in$ interpolationList(C)	Interpolation type of a particular interpolation method; possible values are listed in WCS [OGC 07-067r5] Table I.7

[2] Note that the same image CRS is supported by all axes of a coverage.

Null resistance	nullResistance(*im*) for all *im* ∈ interpolationList(*C*)	Null resistance level of a particular interpolation methods; possible values are listed in WCS [OGC 07-067r5] Table I.8

Example For a set of numbers {-1, 0, 1} the abs() function produces:
abs({-1, 0, 1}) = { abs(-1), abs(0), abs(1) } = { 0, 1 }
…while for a list (-1, 0, 1) the abs() function produces:
abs((-1, 0, 1)) = (abs(-1), abs(0), abs(1)) = (1, 0, 1)

NOTE Operations in WCPS rely solely on the structural information when performing semantic checks, i.e., on structural compatibility in operations. Ensuring semantic interoperability of coverage domains and ranges is not within the current scope of WCPS.

6.3 Restrictions relative to the WCS coverage model

The following WCS coverage features are not supported by this version of WCPS:

- Range field types in WCPS are constrained to the set indicated in Table 2 whereas WCS allows any kind of data structure for range fields.

  NOTE 1 This is necessary to concisely fix the semantics of operations on the range types.

  NOTE 2 In practice this should hardly pose a restriction, as at least all numeric types occurring in the applications perceived are provided.

- Range field components in WCPS are atomic.

  NOTE This is believed to cover most practical cases, and allows to emulate the remaining cases.

7 WCPS coverage processing language

The WCPS coverage processing language allows WCPS clients to request processing of one or more coverages available on a WCPS server. A WCPS server evaluates an expression and returns an appropriate response to the client. The result returned to the client upon a successful request consists of an ordered sequence of one or more coverages or scalar values.

A WCPS processing request consists of a **processCoveragesExpr** (see Subclause 7.1.1). Each WCPS server **shall** implement the coverage processing operation as specified in the following subclauses.

NOTE 1 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 language 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 2 WCPS has been designed so as to be “safe in evaluation” – i.e., implementations are possible where any valid WCPS request can be evaluated in a finite number of steps, based on the operation primitives. Hence, WCPS implementations can be constructed in a way that no single request can render the service permanently unavailable. Notwithstanding, it still is possible to send requests that will impose high workload on a server.

NOTE 3 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, domains, range types, etc. However, a response always consists of either coverages or scalar values.

NOTE 4 As the functionality of WCPS centers around coverage processing, metadata are considered only to the extent necessary for a coherent service. This way WCPS keeps orthogonal to other OGC standards.

7.1 Expression syntax

The WCPS primitives plus the nesting capabilities form an expression language, which is independent from any particular encoding, and collectively is referred to as **WCPS language**. In the following subsections the language elements are detailed. The complete syntax is listed in Appendix B.

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

Example The coverage expression

```
C * 2
```

is admissible as it adheres to WCPS syntax whereas

```
C C
```

seen as a coverage expression 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 defined in Subclause 6.2.

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 coverage expression following is valid if and only if the WCPS offers a coverage with identifier `C` that has a numeric field named `red`.

```
C.red * 2.5
```

NOTE In the remainder of this section, tables are used to describe the effect of an operation on each coverage constitutent. For the reader's convenience an extra column "Changed?" is provided containing an "X" character whenever the operation changes the resp. constituent, and left blank whenever the operation does not affect the resp. constituent.

7.1.1 processCoveragesExpr

The **processCoveragesExpr** element processes a list of coverages in turn.

Each coverage is optionally checked first for fulfilling some predicate, and gets selected – i.e., contributes to an element of the result list – only if the predicate evaluates to *true*. Each coverage selected will be processed, and the result will be appended to the result list. This result list, finally, is returned as the *ProcessCoverages* response unless no exception was generated.

Let

> $v_1, \dots v_n$ be n pairwise different **iteratorVar**s ($n \geq 1$),
> $L_1, \dots L_n$ be n **coverageList**s ($n \geq 1$),
> b be a **booleanScalarExpr** possibly containing occurrences of one or more v_i ($1 \leq i \leq n$),
> P be a **processingExpr** possibly containing occurrences of v_i ($1 \leq i \leq n$).

Then,

> for any **processCoveragesExpr** E,
> where

```
E  = for v1 in ( L1 ),
         v2 in ( L2 ),
         … ,
         vn in ( Ln )
     where b
     return P
```

> the result R of evaluating **processCoverageExpr** E is constructed as follows:

```
Let $R$ be the empty sequence;
while $L_1$ is not empty:
{   assign the first element in $L_1$ to $v_1$;
    while $L_2$ is not empty:
    {   assign the first element in $L_2$ to $v_2$;
        …
            while $L_n$ is not empty:
            {   assign the first element in $L_n$ to $v_n$;
                evaluate $b$ and $P$, substituting any occurrence
                of coverage identifier $v_i$ by the corresponding coverage;
                if ($b$)
                then
                    append evaluation result to $R$;
                    remove the first element from $L_n$;
            }
        …
        }
        remove the first element from $L_2$;
    }
    remove the first element from $L_1$;
}
```

The elements contained in the **coverageList** clause, constituting coverage identifiers, are taken from the coverage identifiers advertised by the server.

NOTE In a WCS framework such information can be obtained via a *GetCapabilities* request.

Coverage identifiers **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.

Example Assume a WCPS server offers coverages A, B, and C. Then, the server may execute the following WCPS request:

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

to produce a result list containing three TIFF-encoded coverages tiff(A), tiff(B), tiff(C).

Example Assume a WCPS server offers satellite images A, B, and C and a coverage M acting as a mask (i.e., with range values between 0 and 1). Then, masking each satellite image can be performed with a request like the following:

```
for $s in ( A, B, C ),
    $m in ( M )
return tiff( $s * $m )
```

7.1.2 processingExpr

The **processingExpr** element is either a **encodedCoverageExpr** (which evaluates to an encoded coverage; see Subclause 7.1.4), or a **storeCoverageExpr** (see Subclause 7.1.3), or a **scalarExpr** (which evaluates to coverage description data or coverage summary data; see Subclause 7.1.5).

7.1.3 storeCoverageExpr

The **storeCoverageExpr** element specifies that an encoded coverage result as described by its E sub element is not to be delivered immediately as response to the request, but to be stored on server side for subsequent retrieval. The result of the **storeCoverageExpr** expression is the URL under which the result is provided by the server, and the server returns only the XML response part with the URL(s) being in place of the coverage(s) generated.

Let

E be an **encodedCoverageExpr**.

Then,

for any **URI** `U`
where

`U` = **`store (`** `E` **`)`**

`U` is defined as that URI at which the coverage result is made available by the server.

Example The following expression will deliver the URL under which the server stores the TIFF-encoded result coverage `C`:

```
store( encode( C, “TIFF” ) )
```

NOTE It is not specified in this standard for how long server-side stored coverages remain available; usually they will be deleted after some implementation dependent time to free server space. Future versions of this standard may offer means to address this.

7.1.4 encodedCoverageExpr

The **encodedCoverageExpr** element specifies encoding of a coverage-valued request result by means of a data format and possible extra encoding parameters.

Data format encodings **should**, to the largest extent possible, materialise the coverage’s metadata. A service **may** store further information as part of the encoding.

Example For a georeferenced coverage, a GeoTIFF result file **should** contain the coverage’s geo coordinate and resolution information.

NOTE For materialization of the coverage grid values, the coverage’s image CRS **shall** be used by default. See **crsTransformExpr** (Subclause 7.1.28) for controlling coverage grid values via other CRSs.

Let

`C` be a **coverageExpr**,
`f` be a string,
where

`f` is the name of a data format allowed for `C`,
the data format specified by `f` supports encoding of a coverage of `C`’s domain and range.

Then,

for any **byteString** `S`
where `S` is one of

S_e = **`encode (`** `C` **`,`** `f` **`)`**
S_{ee} = **`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 in this standard.

NOTE 1 In a WCS framework, the data encoding formats supported can be obtained from the **supportedFormats** list contained in the response a *GetCapabilities* request.

NOTE 2 Some format encodings may lead to a loss of information.

NOTE 3 The extraParams are data format and implementation dependent.

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

```
encode( C, ”hdf-eos” )
```

Example A WCPS implementation **may** encode a JPEG quality factor of 50% as the string “`.50`”.

Usage of formats **shall** adhere to the regulations set forth in OGC 07-067r5 Subclause 9.3.2.2. The sequence of axes used for linearizing arrays for encoding **shall** be done as governed by some corresponding data format encoding specification.

Example A data format may specify that x and y axes are linearized to achieve a row-major ordering of coverage cells.

7.1.5 scalarExpr

The **scalarExpr** element is either a **getMetaDataExpr** (see Subclause 7.1.10) or a **condenseExpr** (see Subclause 7.1.31) or a **booleanScalarExpr** (see Subclause 7.1.6) or a **numericScalarExpr** (see Subclause 7.1.7) or a **stringScalarExpr** (see Subclause 7.1.7).

NOTE As such, it returns a result which is not a coverage.

NOTE A future version of WCPS may support further scalar operations beyond those summarized as **scalarExpr**s.

7.1.6 booleanScalarExpr

The **booleanScalarExpr** element is a **scalarExpr** (see Subclause 7.1.5) whose result type is Boolean.

Operations provided are the well-known Boolean functions `and`, `or`, `xor`, and `not` bearing the standard semantics.

7.1.7 numericScalarExpr

The **numericScalarExpr** element is a **scalarExpr** (see Subclause 7.1.5) whose result type is numeric (i.e., an integer, float, or complex number), such as numeric constants or number-valued metadata retrieval functions.

Operations provided are the well-known arithmetic (`+`, `-`, `*`, `/`) and comparison (>, <, >=, <=, =, `!=`) operations bearing the standard mathematical semantics. The rounding function, `round()`, rounds a real (not complex) number to the next integer number towards zero.

7.1.8 stringScalarExpr

The **stringScalarExpr** element is a **scalarExpr** (see Subclause 7.1.5) whose result type is character string of length greater or equal to zero, such as string constants or string-valued metadata retrieval functions.

Operations provided are the well-known string comparison operations = and `!=` bearing the standard semantics. In addition, a string-valued getMetadata operation is a **stringScalarExpr**.

7.1.9 coverageExpr

The **coverageExpr** element is either a **coverageIdentifier** (see Subclause 7.1.12), or **setMetaDataExpr** (see Subclause 7.1.11), or an **inducedExpr** (see Subclause 7.1.13), or a **subsetExpr** (see Subclause 7.1.23), or a **crsTransformExpr** (see Subclause 7.1.28), or a **scaleExpr** (see Subclause 7.1.27), or a **coverageConstructorExpr** (see Subclause 7.1.29), or a **coverageConstantExpr** (see Subclause 7.1.30), or a **condenseExpr** (see Subclause 7.1.31).

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

7.1.10 getMetaDataExpr

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

NOTE The grid point value sets (“pixels”, “voxels”, …) can be extracted from a coverage using subsetting operations (see Subclause 7.1.22).

Let

C be a **coverageExpr**.

Then,

The following metadata extraction functions are defined, whereby the result is specified in terms of the coverage’s probing functions as defined in Table 3:

Metadata function (for some coverage *C*, dimension *a*, range field *r*)	**Result** (in terms of probing functions)	**Result type**
`identifier(`*`C`*`)`	identifier(*C*)	Name
`imageCrs`(*C*)	imageCRS(*C*)	URN
`imageCrsDomain(`*`C`*,*`a`*`)`	imageCrsDomain(*C*,*a*)	(lower bound, upper bound) integer pair
`crsSet(`*`C`*`)`	crsSet(*C*,*a*)	Set of URNs

domain(*C*,*a*,*c***)**	domain(*C*,*a*,*c*)	(lower bound, upper bound) numeric / string pair
nullSet(*C***)**	nullSet(*C*)	Set of values, each structured according to rangeType(*C*); set may be empty.
interpolationDefault(*C*,*r***)**	interpolationDefault(*C*)	Pair of enumeration values
interpolationSet(*C*,*r***)**	interpolationSet(*C*,*a*)	List of pairs of enumeration values

NOTE Not all information about a coverage can be retrieved this way. In a WCS framework, adding the information supplied in a *GetCapabilities* and *DescribeCoverage* response provides complete information about a coverage.

Example For some stored coverage named C, the following expression evaluates to “C”:

```
identifier( C )
```

7.1.11 setMetaDataExpr

The **setMetaDataExpr** element allows to derive a coverage with modified metadata, leaving untouched the coverage range values and all metadata not addressed.

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

Let

C_1 be a **coverageExpr**,
s be a **stringConstant**,
m, *n*, *p* be integers with $m \geq 0$ and $n \geq 0$ and $p \geq 0$,
null be a **rangeExpr** with $null \in$ nullSet(C_1),
$null_1, \ldots, null_m$ be **rangeExpr**s which are cast-compatible with type rangeType(C_1),
f be an **identifier**, *it* an **interpolationType**, *nr* a **nullResistance** with $f \in$ rangeFieldNames(C_1) and $(im, nr) \in$ interpolationSet(C_1,f),
$it_1, \ldots, it_n$ be **interpolationType**s, and $nr_1, \ldots, nr_n$ be **nullResistance**s with $f_i \in$ rangeFieldNames(C_1) for $1 \leq i \leq n$ and $im_i \in$ interpolationSet(C_1,f_i),
$crs_1, \ldots, crs_p$ be **crsName**s.

Then,

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

C_{id} = **setIdentifier(** C_1, **s)**
C_{null} = **setNullSet(** C_1, **{** $null_1, \ldots, null_m$ **})**
C_{intDef} = **setInterpolationDefault(** C_1, f, **(**im ,nr**))**
C_{int} = **setInterpolationSet(** C_1, f,
{ (im_1,nr_1), … , **(**im_n,nr_n) **})**
C_{crs} = **setCrsSet(** C_1, **{** $crs_1, \ldots, crs_p$ **},** a **)**

C_2 is defined as follows:

Coverage constituent	Changed?
identifier(C_2) = s for C_2=C_{id}, identifier(C_2) = identifier(C_1) otherwise	X

for all $p \in$ imageCrsDomain(C_2):
value(C_2, p) = value(C_1, p)	

imageCrs(C_2) = imageCrs(C_1)	
imageCrsDomain(C_2) = imageCrsDomain(C_1)	
dimensionList(C_2) = dimensionList(C_1)	

for all $a \in$ dimensionList(C_2):
crsSet(C_{null}, a) = crsSet(C_1, a)
crsSet(C_{intDef}, a) = crsSet(C_1, a)
crsSet(C_{int}, a) = crsSet(C_1, a)
crsSet(C_{crs}, a) = { $crs_1, \ldots, crs_p$ }
dimensionType(C_2, a) = dimensionType(C_1, a)
	X

for all $a \in$ dimensionList(C_2), $c \in$ crsSet(C_2, a):
domain(C_2, a, c) = domain(C_1, a, c)
	X

for all fields $r \in$ rangeFieldNames(C_2):
rangeFieldType(C_2, r) = rangeFieldType(C_1, r)	

nullSet(C_{null}) = { $null_1, \ldots, null_m$ }
nullSet(C_{intDef}) = nullSet(C_1)
nullSet(C_{int}) = nullSet(C_1)
nullSet(C_{crs}) = nullSet(C_1)
	X

for all $r \in$ rangeFieldNames(C_2):
interpolationDefault(C_{null} , r) = interpolationDefault(C_1, r)
interpolationDefault(C_{intDef} , r) = (it,nr)
interpolationDefault(C_{int} , r) =
if interpolationDefault(C_1)$\in${(im_1, nr_1),…,(im_n, nr_n)}
	X

then interpolationDefault(C_1, r)
else undefined[3]
interpolationDefault(C_{crs}, r) = interpolationDefault(C_1, r)	

for all $r \in$ rangeFieldNames(C_2):
interpolationSet(C_{null}, r) = interpolationSet(C_1, r)
interpolationSet(C_{intDef}, r) = interpolationSet(C_1, r)
interpolationSet(C_{int}, r) = interpolationSet(C_1, r)
interpolationSet(C_{crs}, r) =
if $r=f$ then { (im_1,nr_1), …, (im_n,nr_n) }
else interpolationSet(C_1, r)
interpolationSet(C_{crs}, r) = interpolationSet(C_1, r)
	X

Example The following coverage expression evaluates to a coverage that, in its data, resembles *C* , but has no interpolation method available on its range field *landUse*, allows *linear* interpolation with full null resistance, and *quadratic* interpolation with half null resistance on *C*'s range field *panchromatic*:

```
setInterpolation( setInterpolation( C, landUse, { } ), pan-
chromatic, { linear:full, quadratic:half } )
```

The *setNullSet()* operation **shall** not change any preexisting value in the coverage (such as in an attempt to adapt old null values to the new ones).

NOTE Obviously changing a coverage's null values can render its contents inconsistent.

A server may respond with an exception if it does not support a CRS specified in a *setCrsSet()* call.

7.1.12 coverageIdentifier

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

Let

id be the **identifier** of a coverage C_1 offered by the server.

Then,

for any **coverageExpr** C_2,
where
C_2 = *id*

C_2 is defined as follows:

[3] An undefined default interpolation method **shall** lead to a runtime exception whenever it needs to be applied (see Clause 6).

Coverage constituent	Changed?
identifier(C_2) = identifier(C_1) = `id`	

for all $p \in$ imageCrsDomain(C_2):
value(C_2,p) = value(C_1,p)	

imageCrs(C') = imageCrs(C_1)	
imageCrsDomain(C_2) = imageCrsDomain(C_1)	
dimensionList(C_2) = dimensionList(C_1)	

for all $a \in$ dimensionList(C_2):
crsSet(C_2, a) = crsSet(C_1, a)
dimensionType(C_2, a) = dimensionType(C_1, a)	

for all $a \in$ dimensionList(C_2), $c \in$ crsSet(C_2, a):
domain(C_2, a, c) = domain(C_2, a, c)	

for all fields $r \in$ rangeFieldNames(C$_2$):
rangeFieldType(C_2, r) = rangeFieldType(C_1, r)	

nullSet(C_2) = nullSet(C_1)	

for all $r \in$ rangeFieldNames(C_2):
interpolationDefault(C_2, r) = interpolationDefault(C_1, r)	

for all $r \in$ rangeFieldNames(C_2):
interpolationSet(C_2, r) = interpolationSet(C_1, r)	

Example The following coverage expression evaluates to the complete, unchanged coverage `C`, assuming it is offered by the server:

```
C
```

7.1.13 inducedExpr

The inducedExpr element is either a **unaryInducedExpr** (see Subclause 7.1.14) or a **binaryInducedExpr** (see Subclause 7.1.21) or a **rangeConstructorExpr** (see Subclause 7.1.22).

Induced operations allow to simultaneously apply a function originally working on a single value to all grid point values of a coverage. In case the range type contains more than one component, the function **shall** be applied to each point simultaneously.

Whenever a numeric argument is expected (such as a coverage with numeric range fields), Boolean false and true **shall** be interpreted as 0 and 1, resp. Conversely, whenever a Boolean argument is expected (such as a coverage with numeric range fields), then 0 and 1 **shall** be interpreted as Boolean false and true, resp.

Whenever one of the point values (“pixels”, etc.) participating in an induced operation is equal to one of the null values of its coverage then the result of the value combination **shall** be one of the values in the participating coverage’s null value set (for a unary induced operation) or one of the values in the null value set intersection of both participating coverages (for a binary induced operation) if said intersection is not empty. If no null value is available (for a unary induced operation) or the intersection of both input coverages’ null values is empty (for a binary induced operation) then the server **shall** respond with a service exception.

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

NOTE The idea is that for each operation available on the range type, a corresponding coverage operation is provided (“induced from the range type operation”) [1] [2].

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

7.1.14 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 **unaryInducedExpr** is either a **unaryArithmeticExpr**, or **trigonometricExpr**, or **exponentialExpr** (in which case it evaluates to a coverage with a numeric range type; see Subclauses 7.1.15, 7.1.16, 7.1.17), a **boolExpr** (in which case it evaluates to a Boolean expression; see Subclause 7.1.18), a **castExpr** (in which case it evaluates to a coverage with unchanged values, but another range type; see Subclause 7.1.19), or a **fieldExpr** (in which case a range field selection is performed; see Subclause 7.1.20).

7.1.15 unaryArithmeticExpr

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

Let

C_1 be a **coverageExpr**

Then,

for any **coverageExpr** C_2
where C_2 is one of
C_{plus} = `+` C_1
C_{minus} = `-` C_1

C_{sqrt} = **`sqrt(`** C_1 **`)`**
C_{abs} = **`abs(`** C_1 **`)`**

C_2 is defined as follows:

Coverage constituent	**Changed?**
identifier(C_2) = “” (empty string)	**X**

for all $p \in$ imageCrsDomain(C_2):
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))
if value(C_1,p) $\notin$ nullSet(C_1)
	X

imageCrs(C_2) = imageCrs(C_1)	
imageCrsDomain(C_2) = imageCrsDomain(C_1)	
dimensionList(C_2) = dimensionList(C_1)	

for all $a \in$ dimensionList(C_2):
crsSet(C_2, a) = crsSet(C_1, a)
dimensionType(C_2, a) = dimensionType(C_1, a)	

for all $a \in$ dimensionList(C_2), $c \in$ crsSet(C_2, a):
domain(C_2, a, c) = domain(C_2, a, c)	

for all fields $r \in$ rangeFieldNames(C$_2$):
 	
rangeFieldType(C_{plus},r) = rangeFieldType(C_1,r)
 	
rangeFieldType(C_{minus},r) = rangeFieldType(C_1,r)
if rangeFieldType(C_{minus},r) $\in$ { char, short, int, long, float, double, complex, complex2 }
rangeFieldType(C_{minus},r) = char
if rangeFieldType(C_1,r) = unsigned char,
rangeFieldType(C_{minus},r) = short
if rangeFieldType(C_1,r) = unsigned short,
rangeFieldType(C_{minus},r) = int
if rangeFieldType(C_1,r) = unsigned int,
rangeFieldType(C_{minus},r) = long
if rangeFieldType(C_1,r) = unsigned long
 	
rangeFieldType(C_{sqrt},r) = double
if rangeFieldType(C_1,r) $\notin$ { complex, complex2 }
and $C_1.r \geq 0$,
	X

rangeFieldType(C_{sqrt},r) = complex2 otherwise rangeFieldType(C_{abs},r) = rangeFieldType(C_1,r) if rangeFieldType(C_1,r) $\in$ { boolean, unsigned char, unsigned short, unsigned int, unsigned long, float, double } rangeFieldType(C_{abs},r) = unsigned char if rangeFieldType(C_1,r) = char, rangeFieldType(C_{abs},r) = unsigned short if rangeFieldType(C_1,r) = short, rangeFieldType(C_{abs},r) = unsigned int if rangeFieldType(C_1,r) = int, rangeFieldType(C_{abs},r) = unsigned long if rangeFieldType(C_1,r) = long, rangeFieldType(C_{abs},r) = float if rangeFieldType(C_1,r) $\in$ { float, complex }, rangeFieldType(C_{abs},r) = double if rangeFieldType(C_1,r) $\in$ { double, complex2}	

nullSet(C_{plus}) = nullSet(C_1)
nullSet(C_{minus}) = - nullSet(C_1)
nullSet(C_{sqrt}) = sqrt(nullSet(C_1))
nullSet(C_{abs}) = abs(nullSet(C_1))
where the resp. operation is applied componentwise to the set. An invalid operand (such as negative numbers in case of a square root operation) **shall** lead to a service exception.
	X

for all $r \in$ rangeFieldNames(C_2):
interpolationDefault(C_2, r) = interpolationDefault(C_1, r)	

for all $r \in$ rangeFieldNames(C_2):
interpolationSet(C_2, r) = interpolationSet(C_1, r)	

The server **shall** respond with an exception if one of the coverage's grid point values or its null values is negative.

Example The following coverage expression evaluates to a float-type coverage where each range value contains the square root of the sum of the corresponding source coverages' values.

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

7.1.16 trigonometricExpr

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

Let

C_1 be a **coverageExpr**

Then,

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

C_{sin}	=	**sin(** C_1 **)**
C_{cos}	=	**cos(** C_1 **)**
C_{tan}	=	**tan(** C_1 **)**
C_{sinh}	=	**sinh(** C_1 **)**
C_{cosh}	=	**cosh(** C_1 **)**
C_{arcsin}	=	**arcsin(** C_1 **)**
C_{arccos}	=	**arccos(** C_1 **)**
C_{arctan}	=	**arctan(** C_1 **)**

C_2 is defined as follows:

Coverage constituent	**Changed?**
identifier(C_2) = “” (empty string)	**X**

for all $p \in$ imageCrsDomain(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))
if value(C_1,p) $\notin$ nullSet(C_1)
	X

ImageCrs(C_2) = imageCrs(C_1)	
imageCrsDomain(C_2) = imageCrsDomain(C_1)	
dimensionList(C_2) = dimensionList(C_1)	

for all $a \in$ dimensionList(C_2):
crsSet(C_2, a) = crsSet(C_1, a)
dimensionType(C_2, a) = dimensionType(C_1, a)	

for all $a \in$ dimensionList(C_2), $c \in$ crsSet(C_2, a):
domain(C_2, a, c) = domain(C_2, a, c)	

for all fields $r \in$ rangeFieldNames(C_2):
rangeFieldType(C_2,r) = complex2
if rangeFieldType(C_1,r) $\in$ { complex, complex2 }, rangeFieldType(C_2,r) = double
	X

otherwise	

nullSet(C_{sin}) = sin(nullSet(C_1))
nullSet(C_{cos}) = cos(nullSet(C_1))
nullSet(C_{tan}) = tan(nullSet(C_1))
nullSet(C_{sinh}) = sinh(nullSet(C_1))
nullSet(C_{cosh}) = cosh(nullSet(C_1))
nullSet(C_{arcsin}) = arcsin(nullSet(C_1))
nullSet(C_{arccos}) = arccos(nullSet(C_1))
nullSet(C_{arctan}) = arctan(nullSet(C_1))
where the resp. operation is applied componentwise to the set. An invalid operand (such as negative numbers in case of a square root operation) **shall** lead to a service exception.
	X

for all $r \in$ rangeFieldNames(C_2):
interpolationDefault(C_2, r) = interpolationDefault(C_1, r)	

for all $r \in$ rangeFieldNames(C_2):
interpolationSet(C_2, r) = interpolationSet(C_1, r)	

The server **shall** respond with an exception if one of the coverage's grid point values or its null values is not within the domain of the function to be applied to it.

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

```
sin( C )
```

Example To enforce a complex result for real-valued arguments the input coverage can be cast to complex:

```
arcsin( (complex) C )
```

7.1.17 exponentialExpr

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

Let

C_1 be a **coverageExpr**,
p be a **floatConstant**

Then,

for any **coverageExpr** C_2
where C_2 is one of
C_{exp} = **exp(** C_1 **)**
C_{log} = **log(** C_1 **)**
C_{ln} = **ln(** C_1 **)**
C_{pow} = **pow(** C_1, p **)**

C_2 is defined as follows:

Coverage constituent	Changed?
identifier(C_2) = “” (empty string)	X

for all $p \in$ imageCrsDomain(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))
value(C_{pow}, p) = value(C_1,p)^p
	X

imageCrs(C_2) = imageCrs(C_1)	
imageCrsDomain(C_2) = imageCrsDomain(C_1)	
dimensionList(C_2) = dimensionList(C_1)	

for all $a \in$ dimensionList(C_2):
crsSet(C_2, a) = crsSet(C_1, a)
dimensionType(C_2, a) = dimensionType(C_1, a)	

for all $a \in$ dimensionList(C_2), $c \in$ crsSet(C_2, a):
domain(C_2, a, c) = domain(C_2, a, c)	

for all fields $r \in$ rangeFieldNames(C_2): rangeFieldType(C_2,r) = complex2 if rangeFieldType(C_1,r) $\in$ { complex, complex2 }, ran- geFieldType(C_2,r) = double otherwise	X
nullSet(C_2) = nullSet(C_1)	

for all $r \in$ rangeFieldNames(C_2):
interpolationDefault(C_2, r) = interpolationDefault(C_1, r)	

for all $r \in$ rangeFieldNames(C_2):
interpolationSet(C_2, r) = interpolationSet(C_1, r)	

The server **shall** respond with an exception if one of the coverage’s grid point values or its null values is not within the domain of the function to be applied to it.

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

```
ln( C )
```

7.1.18 boolExpr

The **boolExpr** element specifies a unary induced Boolean operation.

Let

C_1 be a **coverageExpr**

Then,

for any **coverageExpr** C_2
where C_2 is one of
C_{not} = **`not`** C_1
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?**
identifier(C_2) = “” (empty string)	**X**

for all p ∈ imageCrsDomain(C_2):
value(C_{not} , p) = not(value(C_1,p))
value(C_{bit}, p) = (value(C_1,p) >> value(n)) mod 2
	X

imageCrs(C_2) = imageCrs(C_1)	
imageCrsDomain(C_2) = imageCrsDomain(C_1)	
dimensionList(C_2) = dimensionList(C_1)	

for all a ∈ dimensionList(C_2):
crsSet(C_2, a) = crsSet(C_1, a)
dimensionType(C_2, a) = dimensionType(C_1, a)	

for all a ∈ dimensionList(C_2), c ∈ crsSet(C_2, a):
domain(C_2, a, c) = domain(C_2, a, c)	

for all fields r ∈ rangeFieldNames(C_2): rangeieldType(C_2,r) = boolean	**X**
nullSet(C_{not}) = not(nullSet(C_1)) nullSet(C_{bit}) = (nullSet(C_1) >> value(n)) mod 2 where the resp. operation is applied componentwise to the set. An invalid operand (such as negative numbers in case of a square root operation) **shall** lead to a service exception.	**X**
for all r ∈ rangeFieldNames(C_2):	

interpolationDefault(C_2, r) = interpolationDefault(C_1, r)	
for all $r \in$ rangeFieldNames(C_2): interpolationSet(C_2, r) = interpolationSet(C_1, r)	

Example The following expression inverts all (assumed: Boolean) range field 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.

7.1.19 castExpr

The **castExpr** element specifies a unary induced cast operation, that is: to change the range type of the coverage while leaving all other properties unchanged.

NOTE Depending on the input and output types result possibly may suffer from a loss of accuracy through data type conversion.

Let

C_1 be a **coverageExpr**,
t be a range field type name.

Then,

for any **coverageExpr** C_2
where
C_2 = **(** t **)** C_1

C_2 is defined as follows:

Coverage constituent	**Changed?**
identifier(C_2) = “” (empty string)	**X**
for all p $\in$ imageCrsDomain(C_2): value(C_2 , p) = (t) value(C_1,p)	**X**
imageCrs(C_2) = imageCrs(C_1)	
imageCrsDomain(C_2) = imageCrsDomain(C_1)	
dimensionList(C_2) = dimensionList(C_1)	
for all $a \in$ dimensionList(C_2): crsSet(C_2, a) = crsSet(C_1, a) dimensionType(C_2, a) = dimensionType(C_1, a)	

for all $a \in$ dimensionList(C_2), $c \in$ crsSet(C_2, a):
domain(C_2, a, c) = domain(C_2, a, c)	

for all fields $r \in$ rangeFieldNames(C_2): rangeFieldType(C_2,r) = t	**X**
nullSet(C_2) = (t) nullSet(C_1 where the resp. operation is applied componentwise to the set. An invalid operand (such as negative numbers in case of a square root operation) **shall** lead to a service exception.	**X**

for all $r \in$ rangeFieldNames(C_2):
interpolationDefault(C_2, r) = interpolationDefault(C_1, r)	

for all $r \in$ rangeFieldNames(C_2):
interpolationSet(C_2, r) = interpolationSet(C_1, r)	

The server **shall** respond with an exception if one of the coverage's grid point values or its null values cannot be cast to the type specified (see Subclause 7.2.5).

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

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

7.1.20 fieldExpr

The **fieldExpr** element specifies a unary induced field selection operation. Fields are selected by their name, in accordance with the WCS range field subsetting operation.

NOTE Due to the current restriction to atomic range fields, the result of a field selection has atomic values too.

Let

C_1 be a **coverageExpr**,
f be a **fieldName** appearing in rangeFieldNames(C_1).

Then,

for any **coverageExpr** C_2
where
$C_2 = C_1 . f$

C_2 is defined as follows:

Coverage constituent	Changed?
identifier(C_2) = "" (empty string)	**X**

for all $p \in$ imageCrsDomain(C_2): value(C_2, p) = value($C_1.f$, p)	
imageCrs(C_2) = imageCrs(C_1)	
ImageCrsDomain(C_2) = imageCrsDomain(C_1)	
dimensionList(C_2) = dimensionList(C_1)	
for all $a \in$ dimensionList(C_2): crsSet(C_2, a) = crsSet(C_1, a) dimensionType(C_2, a) = dimensionType(C_1, a)	
for all $a \in$ dimensionList(C_2), $c \in$ crsSet(C_2, a): domain(C_2, a, c) = domain(C_2, a, c)	
rangeFieldType(C_2,f) = rangeFieldType(C_1,f)	**X**
nullSet(C_2) = nullSet(C_1).f where the resp. operation is applied componentwise to the set. An invalid operand (such as negative numbers in case of a square root operation) **shall** lead to a service exception.	**X**
interpolationDefault(C_2, $comp$) = interpolationDefault(C_1, f)	
interpolationSet(C_2, f) = interpolationSet(C_1, f)	

Example Let C be a coverage with range type integer. Then the following request snippet describes a single-field, integer-type coverage where each grid point value contains the difference between red and green band:

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

7.1.21 binaryInducedExpr

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

Both participating coverages **must** have the same number of range components; otherwise the server **shall** respond with a service exception.

The coverage range types **shall** be numeric.

Let

C_1, C_2 be **coverageExpr**s,
S_1, S_2 be **scalarExpr**s,
where
imageCrsDomain(C_1,a) = imageCrsDomain(C_2,a),

imageCrs(C_1,a) = imageCrs(C_2,a),
domain(C_1,a) = domain(C_2,a),
crsSet(C_1,a) = crsSet(C_2,a) for all $a \in$ dimensionList(C_2),
rangeFieldNames(C_1) = rangeFieldNames(C_2),
rangeType(C_1,f) is cast-compatible with rangeType(C_2,f) or
rangeType(C_2,f) is cast-compatible with rangeType(C_1,f)
for all $f \in$ rangeFieldNames(C_1),
nullSet(C_1) $\cap$ nullSet(C_2) $\neq$ {}.

Then,

for any **coverageExpr** C_3
where C_3 is one of

C_{plusCC} = C_1 **+** C_2
C_{minCC} = C_1 **-** C_2
C_{multCC} = C_1 ***** C_2
C_{divCC} = C_1 **/** C_2
C_{andCC} = C_1 **and** C_2
C_{orCC} = C_1 **or** C_2
C_{xorCC} = C_1 **xor** C_2
C_{eqCC} = C_1 **=** C_2
C_{ltCC} = C_1 **<** C_2
C_{gtCC} = C_1 **>** C_2
C_{leCC} = C_1 **<=** C_2
C_{geCC} = C_1 **>=** C_2
C_{neCC} = C_1 **!=** C_2
C_{ovlCC} = C_1 **overlay** C_2

C_{plusSC} = S_1 **+** C_2
C_{minSC} = S_1 **-** C_2
C_{multSC} = S_1 ***** C_2
C_{divSC} = S_1 **/** C_2
C_{andSC} = S_1 **and** C_2
C_{orSC} = S_1 **or** C_2
C_{xorSC} = S_1 **xor** C_2
C_{eqSC} = S_1 **=** C_2
C_{ltSC} = S_1 **<** C_2
C_{gtSC} = S_1 **>** C_2
C_{leSC} = S_1 **<=** C_2
C_{geSC} = S_1 **>=** C_2
C_{neSC} = S_1 **!=** C_2
C_{ovlSC} = S_1 **overlay** C_2

C_{plusCS} = C_1 **+** S_2
C_{mincS} = C_1 **-** S_2
C_{multCS} = C_1 ***** S_2

$C_{\text{divCS}} = C_1 \;\mathtt{/}\; S_2$
$C_{\text{andCS}} = C_1 \;\mathtt{and}\; S_2$
$C_{\text{orCS}} = C_1 \;\mathtt{or}\; S_2$
$C_{\text{xorCS}} = C_1 \;\mathtt{xor}\; S_2$
$C_{\text{eqCS}} = C_1 \;\mathtt{=}\; S_2$
$C_{\text{ltCS}} = C_1 \;\mathtt{<}\; S_2$
$C_{\text{gtCS}} = C_1 \;\mathtt{>}\; S_2$
$C_{\text{leCS}} = C_1 \;\mathtt{<=}\; S_2$
$C_{\text{geCS}} = C_1 \;\mathtt{>=}\; S_2$
$C_{\text{neCS}} = C_1 \;\mathtt{!=}\; S_2$
$C_{\text{ovlCS}} = C_1 \;\mathtt{overlay}\; S_2$

C_3 is defined as follows:

Coverage constituent	**Changed?**
identifier(C_3) = “” (empty string)	**X**

for all $p \in$ imageCrsDomain(C_3):
value(C_{plusCC}, p) = value(C_1, p) + value(C_2, p)
value(C_{minCC}, p) = value(C_1, p) - value(C_2, p)
value(C_{multCC}, p) = value(C_1, p) * value(C_2, p)
value(C_{divCC}, p) = value(C_1, p) / value(C_2, p)
value(C_{andCC}, p) = value(C_1, p) and value(C_2, p)
value(C_{orCC}, p) = value(C_1, p) or value(C_2, p)
value(C_{xorCC}, p) = value(C_1, p) xor value(C_2, p)
value(C_{eqCC}, p) = value(C_1, p) = value(C_2, p)
value(C_{ltCC}, p) = value(C_1, p) < value(C_2, p)
value(C_{gtCC}, p) = value(C_1, p) > value(C_2, p)
value(C_{leCC}, p) = value(C_1, p) <= value(C_2, p)
value(C_{geCC}, p) = value(C_1, p) >= value(C_2, p)
value(C_{neCC}, p) = value(C_1, p) != value(C_2, p)
value(C_{ovlCC}, p) = value(C_2, p) if value(C_1, p)=0
value(C_1, p) otherwise
 	
value(C_{plusSC}, p) = S_1 + value(C_2, p)
value(C_{minSC}, p) = S_1 - value(C_2, p)
value(C_{multSC}, p) = S_1 * value(C_2, p)
value(C_{divSC}, p) = S_1 / value(C_2, p)
value(C_{andSC}, p) = S_1 and value(C_2, p)
value(C_{orSC}, p) = S_1 or value(C_2, p)
value(C_{xorSC}, p) = S_1 xor value(C_2, p)
value(C_{eqSC}, p) = S_1 = value(C_2, p)
value(C_{ltSC}, p) = S_1 < value(C_2, p)
value(C_{gtSC}, p) = S_1 > value(C_2, p)
value(C_{leSC}, p) = S_1 <= value(C_2, p)
value(C_{geSC}, p) = S_1 >= value(C_2, p)
	X

value(C_{neSC}, p) = S_1 != value(C_2, p) value(C_{ovlSC}, p) = value(C_2, p) if S_1=0 S_1 otherwise value(C_{plusCS}, p) = value(C_1, p) + S_2 value(C_{minCS}, p) = value(C_1, p) - S_2 value(C_{multCS}, p) = value(C_1, p) * S_2 value(C_{divCS}, p) = value(C_1, p) / S_2 value(C_{andCS}, p) = value(C_1, p) and S_2 value(C_{orCS}, p) = value(C_1, p) or S_2 value(C_{xorCS}, p) = value(C_1, p) xor S_2 value(C_{eqCS}, p) = value(C_1, p) = S_2 value(C_{ltCS}, p) = value(C_1, p) < S_2 value(C_{gtCS}, p) = value(C_1, p) > S_2 value(C_{leCS}, p) = value(C_1, p) <= S_2 value(C_{geCS}, p) = value(C_1, p) >= S_2 value(C_{neCS}, p) = value(C_1, p) != S_2 value(C_{ovlCS}, p) = S_2 if value(C_1, p)=0 value(C_1, p) otherwise Whenever necessary, appropriate cast operations are performed on the values prior to performing the binary value operation (cf. Subclause 7.2.5).	
ImageCrs(C_3) = imageCrs(C_1)	
imageCrsDomain(C_3) = imageCrsDomain(C_1)	
dimensionList(C_2) = dimensionList(C_1)	

for all $a \in$ dimensionList(C_2):
crsSet(C_3, a) = crsSet(C_1, a)
dimensionType(C_2, a) = dimensionType(C_1, a)	

for all $a \in$ dimensionList(C_3), $c \in$ crsSet(C_3, a):
domain(C_3, a, c) = domain(C_1, a, c)	

for all $r \in$ rangeFieldNames(C_3): rangeFieldType(C_{plusCC}, r) is given by Section 7.2.5 rangeFieldType(C_{minCC}, r) is given by Section 7.2.5 rangeFieldType(C_{multCC}, r) is given by Section 7.2.5 rangeFieldType(C_{divCC}, r) is given by Section 7.2.5 rangeFieldType(C_{andCC}, r) = boolean rangeFieldType(C_{orCC}, r) = boolean rangeFieldType(C_{xorCC}, r) = boolean rangeFieldType(C_{eqCC}, r) = boolean rangeFieldType(C_{ltCC}, r) = boolean rangeFieldType(C_{gtCC}, r) = boolean	**X**

rangeFieldType(C_{leCC}, r) = boolean rangeFieldType(C_{geCC}, r) = boolean rangeFieldType(C_{neCC}, r) = boolean rangeFieldType(C_{ovlCC}, r) = rangeFieldType(C_1, r) rangeFieldType(C_{plusSC}, r) is given by Section 7.2.5 rangeFieldType(C_{minSC}, r) is given by Section 7.2.5 rangeFieldType(C_{multSC}, r) is given by Section 7.2.5 rangeFieldType(C_{divSC}, r) is given by Section 7.2.5 rangeFieldType(C_{andSC}, r) = boolean rangeFieldType(C_{orSC}, r) = boolean rangeFieldType(C_{xorSC}, r) = boolean rangeFieldType(C_{eqSC}, r) = boolean rangeFieldType(C_{ltSC}, r) = boolean rangeFieldType(C_{gtSC}, r) = boolean rangeFieldType(C_{leSC}, r) = boolean rangeFieldType(C_{geSC}, r) = boolean rangeFieldType(C_{neSC}, r) = boolean rangeFieldType(C_{ovlSC}, r) = rangeFieldType(C_2) rangeFieldType(C_{plusCS}, r) is determined by Table 5 rangeFieldType(C_{minCS}, r) is determined by Table 5 rangeFieldType(C_{multCS}, r) is determined by Table 5 rangeFieldType(C_{divCS}, r) is determined by Table 5 rangeFieldType(C_{andCS}, r) = boolean rangeFieldType(C_{orCS}, r) = boolean rangeFieldType(C_{xorCS}, r) = boolean rangeFieldType(C_{eqCS}, r) = boolean rangeFieldType(C_{ltCS}, r) = boolean rangeFieldType(C_{gtCS}, r) = boolean rangeFieldType(C_{leCS}, r) = boolean rangeFieldType(C_{geCS}, r) = boolean rangeFieldType(C_{neCS}, r) = boolean rangeFieldType(C_{ovlCS}, r) = boolean	
nullSet(C_3) = $\emptyset$ if the result is boolean, nullSet(C_3) = nullSet(C_1) $\cap$ nullSet(C_2) otherwise	**X**
for all $r \in$ rangeFieldNames(C_3): interpolationDefault(C_3, r) = **none**	**X**

for all $r \in$ rangeFieldNames(C_3):
interpolationSet(C_3, r) = { **none** }
	X

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

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

7.1.22 rangeConstructorExpr

The **rangeConstructorExpr**, an n-ary induced operation, allows to build coverages with compound range structures. To this end, coverage range field expressions enumerated are combined into one coverage.

All input coverages must match wrt. domains and CRSs. An input coverage range field **may** be listed more than once.

The names of the range fields generated by the operation **shall** be given by the names prefixed to each component expression.

Let

n be an **integer** with $n \geq 1$,
$C_1, \ldots, C_n$ be **coverageExpr**s,
$f_1, \ldots, f_n$ be **fieldName**s
where, for $1 \leq i,j \leq n$,
$f_i \in$ rangeFieldNames(C_i),
imageCrs(C_i) = imageCrs(C_j),
imageCrsDomain(C_i) = imageCrsDomain(C_j),
crsSet(C_i) = crsSet(C_j),
domain(C_i,a_i,c_i) = domain(C_j,a_j,c_j)
for all $a_i \in$dimensionList(C_i), $a_j \in$dimensionList(C_j), $c_i \in$crsSet(C_i), $c_j \in$crsSet(C_j).

Then,

for any **coverageExpr** C'
where C' is one of
{ f_1 : C_1 ; … ; f_n : C_n }
struct { f_1 : C_1 ; … ; f_n : C_n }

C' is defined as follows:

Coverage constituent	**Changed?**
identifier(C') = "" (empty string)	**X**

for all $p \in$ imageCrsDomain(C'), $i \in \{1, \ldots, n\}$:
value(C'.**(**i**-1)**, p) = value(C_i.f_i, p)
	X

imageCrs(C') = imageCrs(C_1)	
imageCrsDomain(C') = imageCrsDomain(C_1)	

dimensionList(C_2) = dimensionList(C_1)	

for all $a \in$ dimensionList(C_2):
crsSet(C', a) = crsSet(C_1, a)
dimensionType(C_2, a) = dimensionType(C_1, a)	

for all $a \in$ dimensionList(C_1), $c \in$ crsSet(C_1):
domain(C', a, c) = domain(C_1, a, c)	

for all fields $r \in \{ f_1, \ldots, f_n\}$:
rangeFieldType(C_3, r) = rangeFieldType(C_i, f_i)
	X

nullSet(C') = nullSet(C_1) $\times \ldots \times$ nullSet(C_n)	**X**

for all i in $\{1, \ldots, n\}$:
interpolationDefault(C', f_i) = interpolationDefault(C_i, f_i)
	X

for all i in $\{1, \ldots, n\}$:
interpolationSet(C', f_i) = interpolationSet(C_i, f_i)
	X

Example 1: The expression below does a false color encoding by combining near-infrared, red, and green bands into a 3-band image of 8-bit channels each, which can be visually interpreted as RGB:

```
struct
{ red:   (char) L.nir;
  green: (char) L.red;
  blue:  (char) L.green
}
```

Example 2: The following expression transforms a greyscale image `G` containing a range field `panchromatic` into an RGB-structured image:

```
{ red:   G.panchromatic;
  green: G.panchromatic;
  blue:  G.panchromatic
}
```

7.1.23 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, Subclause 7.1.24), slicing (i.e., cutting out a hyperplane from a coverage, Subclause 7.1.26), extending (Subclause 7.1.25), and scaling (Subclause 7.1.27) of a coverage expression.

All of the **subsetExpr** elements allow to make use of coordinate reference systems other than a coverage's image CRS. A coverage's individual mapping from some supported CRS coordinates to its ImageCRS coordinates does not need to be disclosed by the server, hence coordinate transformation **should** be considered a "black box" by the client.

NOTE 1 The special case that subsetting leads to a single point remaining still resembles a coverage by definition; this coverage is viewed as being of dimension 0.

NOTE 2 Range subsetting is accomplished via the unary induced **fieldExpr** (cf. Subclause 7.1.20).

7.1.24 trimExpr

The **trimExpr** element extracts a subset from a given coverage expression along the dimension indicated, specified by a lower and upper bound for each dimension affected. Interval limits can be expressed in the coverage's image CRS or any CRS which the the coverage supports.

Lower as well as upper limits **must** lie inside the coverage's domain.

For syntactic convenience, both array-style addressing using brackets and function-style syntax are provided; both are equivalent in semantics.

Let

> C_1 be a **coverageExpr**,
> n be an **integer** with $0 \leq n$,
> $a_1, \ldots, a_n$ be pairwise distinct **dimensionName**s with $a_i \in$ dimensionNameSet(C_1) for $1 \leq i \leq n$,
> $crs_1, \ldots, crs_n$ be **crsName**s with $crs_i \in$ crsList(C_1) for $1 \leq i \leq n$,
> $(lo_1 : hi_1), \ldots, (lo_n : hi_n)$ be **dimensionIntervalExpr**s with $lo_i \leq hi_i$ for $1 \leq i \leq n$.

Then,

> for any **coverageExpr** C_2
> where C_2 is one of
>> $C_{bracket} = C_1$ **[** p_1, …, p_n **]**
>> C_{func} = **trim (** C_1, **{** p_1, …, p_n **})**
>
> with
>> p_i is one of
>> $p_{img,i} = a_i$ **(** lo_i **:** hi_i **)**
>> $p_{crs,i} = a_i : crs_i$ **(** lo_i **:** hi_i **)**
>
> C_2 is defined as follows:

Coverage constituent	Changed?
identifier(C_2) = "" (empty string)	X
for all $p \in$ imageCrsDomain(C_2):	

value(C_2, p) = value(C_1,p)	
imageCrs(C_2) = imageCrs(C_1)	
dimensionList(C_2) = dimensionList(C_1)	

for all $a \in$ dimensionList(C_2):
if $a = a_i$ for some i
then imageCrsDomain(C_2 , a) = ($lo_{i,img}$: $hi_{i,img}$)
else imageCrsDomain(C_2 , a) = imageCrsDomain(C_1 , a)
where ($lo_{i,img}$: $hi_{i,img}$) = (lo_i:hi_i) if no CRS is indicated, and the transform from crs_i into the image CRS if crs_i is indicated.
	X

for all $a \in$ dimensionList(C_2):
crsSet(C_2, a) = crsSet(C_1, a)
dimensionType(C_2, a) = dimensionType(C_1, a)	

for all $a \in$ dimensionList(C_2), $c \in$ crsSet(C_2):
if $a = a_i$ for some i
then domain(C_2 , a, c) = ($lo_{i,c}$: $hi_{i,c}$)
else domain(C_2 , a, c) = domain(C_1 , a, c)
where ($lo_{i,c}$:$hi_{i,c}$) represent the dimension boundaries (lo_i:hi_i) transformed of (lo_i:hi_i) from the C_2 image CRS into CRS c.
	X

for all $r \in$ rangeFieldNames(C_2):
rangeFieldType(C_2, r) = rangeFieldType(C_1, r)	

nullSet(C_2) = nullSet(C_1)	

for all $r \in$ rangeFieldNames(C_2):
interpolationDefault(C_2, r) = interpolationDefault(C_1, r)	

for all $r \in$ rangeFieldNames(C_2):
interpolationSet(C_2, r) = interpolationSet(C_1, r)	

NOTE 1 It is possible to mix different CRSs in one trim operation, however each dimension must be addressed in exactly one CRS (either image CRS or another supported CRS).

NOTE 2 A trim operation might simultaneously perform x/y trimming expressed in some geographic coordinate CRS and time trimming in a time CRS.

Example The following are syntactically valid, equivalent trim expressions:

```
C[ x : ”urn:ogc:def:crs:EPSG::4326” (-180 : +180),
   y : ”urn:ogc:def:crs:EPSG::4326” (-90 : 90) ]

trim( C,
      { x: ”urn:ogc:def:crs:EPSG::4326” (-180 : +180),
        y: ”urn:ogc:def:crs:EPSG::4326” (-90 : 90)
```

```
            }
        )
```

7.1.25 extendExpr

The **extendExpr** element extends a coverage to the bounding box indicated. The new grid points are filled with one of the coverage's null values. If the coverage's null value set is empty then the server **shall** throw an exception.

There is no restriction on the position and size of the new bounding box; in particular, it does not need to lie outside the coverage; it may intersect with the coverage; it may lie completely inside the coverage; it may not intersect the coverage at all (in which case a coverage completely filled with null values will be generated).

NOTE In this sense the **extendExpr** is a generalization of the **trimExpr**; still the **trimExpr** should be used whenever the application needs to be sure that a proper subsetting has to take place.

Let

C_1 be a **coverageExpr**,
n be an **integer** with $0 \leq n$,
$a_1, \ldots, a_n$ be pairwise distinct **dimensionName**s with $a_i \in$ dimensionNameSet(C_1) for $1 \leq i \leq n$,
$crs_1, \ldots, crs_n$ be **crsName**s with $crs_i \in$ crsList(C_1) for $1 \leq i \leq n$,
$(lo_1:hi_1), \ldots, (lo_n:hi_n)$ be **dimensionIntervalExpr**s with $lo_i \leq hi_i$ for $1 \leq i \leq n$.

Then,

for any **coverageExpr** C_2
where
C_2 = **extend (** C_1, **{** p_1, …, p_n **})**
with
p_i is one of
$p_{img,i} = a_i$ **(** lo_i **:** hi_i **)**
$p_{crs,i} = a_i : crs_i$ **(** lo_i **:** hi_i **)**

C_2 is defined as follows:

Coverage constituent	**Changed?**
identifier(C_2) = "" (empty string)	**X**

for all $p \in$ imageCrsDomain(C_2):
value(C_2, p) = value(C_1,p) for $p \in$ imageCrsDomain(C_1)
value(C_2, p) = n for $n \in$nullSet(C_1), nullSet(C_1)$\neq\emptyset$
	X

imageCrs(C_2) = imageCrs(C_1)	

dimensionList(C_2) = dimensionList(C_1)	

for all $a \in$ dimensionList(C_2):
if $a = a_i$ for some i
then imageCrsDomain(C_2 , a) = ($lo_{i,img}$: $hi_{i,img}$)
else imageCrsDomain(C_2 , a) = imageCrsDomain(C_1 , a)
where ($lo_{i,img}$: $hi_{i,img}$) = (lo_i:hi_i) if no CRS is indicated, and the transform of (lo_i:hi_i) from crs_i into the C_2 image CRS if crs_i is indicated.
	X

for all $a \in$ dimensionList(C_2):
crsSet(C_2, a) = crsSet(C_1, a)
dimensionType(C_2, a) = dimensionType(C_1, a)	

for all $a \in$ dimensionList(C_2), $c \in$ crsSet(C_2):
if $a = a_i$ for some i
then domain(C_2 , a, c) = ($lo_{i,c}$: $hi_{i,c}$)
else domain(C_2 , a, c) = domain(C_1 , a, c)
where ($lo_{i,c}$:$hi_{i,c}$) represent the dimension boundaries (lo_i:hi_i) transformed from their image CRS into CRS c.
	X

for all $r \in$ rangeFieldNames(C_2):
rangeFieldType(C_2, r) = rangeFieldType(C_1, r)	

nullSet(C_2) = nullSet(C_1)	

for all $r \in$ rangeFieldNames(C_2):
interpolationDefault(C_2, r) = interpolationDefault(C_1, r)	

for all $r \in$ rangeFieldNames(C_2):
interpolationSet(C_2, r) = interpolationSet(C_1, r)	

NOTE A server **may** decide to restrict the CRSs available on the result, as not all CRSs may be technically appropriate any more.

Example The following is a syntactically valid extend expression:

```
extend( C, { x ( -200 : +200 ) } )
```

7.1.26 sliceExpr

The **sliceExpr** element extracts a spatial slice (i.e., a hyperplane) from a given coverage expression along one of its dimensions, specified by one or more slicing dimensions and a slicing position thereon. For each slicing dimension indicated, the resulting coverage has a dimension reduced by 1; its dimensions are the dimensions of the original coverage, in the same sequence, with the section dimension being removed from the list. CRSs not used by any remaining dimension are removed from the coverage's CRS set.

The slicing coordinates **shall** lie inside the coverage's domain.

For syntactic convenience, both array-style addressing using brackets and function-style syntax are provided; both are equivalent in semantics.

Let

> C_1 be a **coverageExpr**,
> n be an **integer** with $0 \leq n$,
> $a_1, \ldots, a_n$ be pairwise distinct **dimensionName**s with $a_i \in$ dimensionName-Set(C_1) for $1 \leq i \leq n$,
> $crs_1, \ldots, crs_n$ be **crsName**s with $crs_i \in$ crsList(C_1) for $1 \leq i \leq n$,
> $s_1, \ldots, s_n$ be **dimensionPoint**s for $1 \leq i \leq n$.

Then,

> for any **coverageExpr** C_2
> where C_2 is one of
> $\quad C_{bracket} = C_1$ **[** S_1**,** … **,** S_n **]**
> $\quad C_{func} =$ **slice(** C_1, **, {** S_1**,** … **,** S_n **})**
> with
> $\quad S_i$ is one of
> $\quad S_{img,i} = a_i$**(**s_i**)**
> $\quad S_{crs,i} = a_i$**(**s_i**)**crs_i
>
> C_2 is defined as follows:

Coverage constituent	Changed?
identifier(C_2) = "" (empty string)	X
for all $p \in$ imageCrsDomain(C_1) such that for all $a \in$ dimensionList(C_1): if $a \in \{a_1, \ldots, a_n\}$ then let p_a be that component of p addressing dimension a $p_a' = s_i$ for $S_{img,i}$ $p_a' = s_i$ transformed from crs_i for $S_{crs,i}$ else let p_a be that component of p addressing dimension a let p_a' be that component of p' addressing dimension a $p_a, p_a' \in$ imageCrsDomain(C_1,a)	

value(C_2, p) = value(C_1,p')	
imageCrs(C_2) = imageCrs(C_1)	
dimensionList(C_2) = dimensionList(C_1) \ {$a_1, …, a_n$}	**X**

for all $a \in$ dimensionList(C_2):
imageCrsDomain(C_2, a) = imageCrsDomain(C_1, a)
	X

for all $a \in$ dimensionList(C_2):
crsSet(C_2, a) = crsSet(C_1, a)
\ ({$crs_1, …, crs_n$} \ crsSet(C_1, a))
dimensionType(C_2, a) = dimensionType(C_1, a)
	X

for all $a \in$ dimensionList(C_1) \ {$a_1, …, a_n$}, $c \in$ crsSet(C_2, a):
domain(C_2, a, c) = domain(C_1, a, c)
	X

for all $r \in$ rangeFieldNames(C_2):
rangeFieldType(C_2, r) = rangeFieldType(C_1, r)	

NullSet(C_2) = nullSet(C_1)	

for all $r \in$ rangeFieldNames(C_2):
interpolationDefault(C_2, r) = interpolationDefault(C_1, r)	

for all $r \in$ rangeFieldNames(C_2):
interpolationSet(C_2, r) = interpolationSet(C_1, r)	

NOTE 1 A server **may** decide to restrict the CRSs available on the result, as not all CRSs may be appropriate any more.

NOTE 2 In a future version of this document this function is likely to be .extended with multi-dimensional slicing.

Example The following are syntactically valid, equivalent slice expressions:

```
C[ x ( 120 ) ]

slice( C, { x ( 120 ) } )
```

7.1.27 scaleExpr

The **scaleExpr** element performs scaling along a subset of the source coverage's dimensions. For each of the coverage's range fields, an interpolation method **can** be chosen from the coverage's interpolation method list. If no interpolation is indicated for a field, then this field's default interpolation method **shall** be used.

A service exception **shall** be raised if for any of the coverage's range fields no appropriate interpolation method is available for the resampling/interpolation performed in the course of the transformation.

Let

> C_1 be a **coverageExpr**,
> m, n be **integer**s with $0 \leq m$ and $0 \leq n$,
> $a_1, \ldots, a_m$ be pairwise distinct **dimensionName**s with $a_i \in$ dimensionNameSet(C_1) for $1 \leq i \leq m$,
> $(lo_1{:}hi_1), \ldots, (lo_m{:}hi_m)$ be **dimensionPoint** pairs with $lo_i \leq hi_i$ for $1 \leq i \leq m$,
> $f_1, \ldots, f_n$ be pairwise distinct **fieldName**s, $it_1, \ldots, it_n$ be **interpolationType**s,
> $nr_1, \ldots, nr_n$ be **nullResistance**s with $f_i \in$ rangeFieldNames(C_1)
> and $(it_i, nr_i) \in$ interpolationSet(C_1, f_i) for $1 \leq i \leq n$.

Then,

> For any **coverageExpr** C_2,
> where

```
        C2 = scale (
                    C1,
                    { p1, …, pm } ,
                    { f1(it1,nr1), …, fn(itn,nrn) }
                )
```

> with
>
> p_i is one of
> $p_{img,i} = a_i(lo_i{:}hi_i)$
> $p_{crs,i} = a_i(lo_i{:}hi_i)crs_i$
>
> C_2 is defined as follows:

Coverage constituent	**Changed?**
identifier(C_2) = "" (empty string)	**X**
for all $p \in$ imageCrsDomain(C_2): value(C_2, p) is obtained by rescaling the coverage along dimensions a_i such that the coverage's extent along dimension a_i is set to $(lo_i{:}hi_i)$, expressed in the coverage's image CRS; all other dimensions remain unaffected. For every range field f_i listed, interpolation type it_i and null resistance nr_i are applied during evaluation; for all range fields not listed their resp. default interpolation is applied.	**X**
imageCrs(C_2) = imageCrs(C_1)	
dimensionList(C_2) = dimensionList(C_1)	
for all $a \in$ dimensionList(C_1): if $a = a_i$ for some i then imageCrsDomain(C_2, a) = $(lo_i{:}hi_i)$	**X**

else imageCrsDomain(C_2 , a) = imageCrsDomain(C_1 , a)	

for all $a \in$ dimensionList(C_2):
crsSet(C_2, a) = crsSet(C_1, a)
dimensionType(C_2, a) = dimensionType(C_1, a)	

for all $a \in$ dimensionList(C_2), $c \in$ crsSet(C_2, a):
domain(C_2,a,c) = domain(C_1,a,c)	

for all $r \in$ rangeFieldNames(C_2):
rangeFieldType(C_2, r) = rangeFieldType(C_1, r)	

nullSet(C_2) = nullSet(C_1)	

for all $r \in$ rangeFieldNames(C_2):
interpolationDefault(C_2, r) = interpolationDefault(C_1, r)	

for all $r \in$ rangeFieldNames(C_2):
interpolationSet(C_2, r) = interpolationSet(C_1, r)	

NOTE Scaling regularly involves range interpolation, hence numerical effects have to be expected.

Example The following expression performs x/y scaling of some coverage C – which has one single range field, `temperature` – using interpolation type `cubic` and null resistance `full` in both `x` and `y` dimension, assuming that the range field supports this method:

```
scale(
    C,
    { x ( lox : hix ) , y ( loy : hiy )  },
    { red ( cubic , full ), nir ( linear, half ) }
)
```

If the default interpolation method is undefined and no interpolation method is indicated expressly then the server **shall** respond with a runtime exception.

7.1.28 crsTransformExpr

The element performs reprojection of a coverage. For each dimension, a separate CRS can be indicated; for any dimension for which no CRS is indicated, no reprojection will be performed. For the resampling which usually is incurred the interpolation method and null resistance can be indicated per range field; for fields not mentioned the default will be applied.

NOTE 1 This changes the range values (e.g., pixel radiometry).

NOTE 2 A service may refuse to accept some CRS combinations (e.g., different CRSs handling for x and y dimension).

NOTE 3 As any coverage bearing a CRS beyond its image CRS is stored in some CRS, there will normally be a parameter combination which retrieves the coverage as stored, without any reprojection operation required.

Let

C_1 be a **coverageExpr**,
m, n be **integer**s with $1 \le m$ and $0 \le n$,
$a_1, \dots, a_m$ be pairwise distinct **dimensionName**s with $a_i \in$ dimensionNameSet(C_1) for $1 \le i \le m$,
$crs_1, \dots, crs_m$ be pairwise distinct **crsName**s with $crs_i \in$ crsList(C_1) for $1 \le i \le m$,
$f_1, \dots, f_n$ be pairwise distinct **fieldName**s,
$it_1, \dots, it_n$ be **interpolationType**s,
$nr_1, \dots, nr_n$ be **nullResistance**s with $f_i \in$rangeFieldNames(C_1) and $(it_i, nr_i) \in$ interpolationSet(C_1,f_i) for $1 \le i \le n$.

Then,

for any **coverageExpr** C_2
where
C_2 = **crsTransform(**
C_1, ,
{ a_1**:**crs_1, …, a_m**:**crs_m **})**
{ f_1**(**it_1,nr_1**)**, …, f_n**(**it_n,nr_n**) }**
)

C_2 is defined as follows:

Coverage constituent	Changed?
identifier(C_2) = “” (empty string)	X
for all $p \in$ imageCrsDomain(C_2): value(C_2, p) is obtained by reprojecting coverage C_1 along dimensions a_i into CRS crs_i; all other dimensions remain unaffected. For every range field f_i listed, interpolation type it_i and null resistance nr_i are applied during evaluation; for all range fields not listed their resp. default interpolation is applied.	X
imageCrs(C_2) = imageCrs(C_1)	
dimensionList(C_2) = dimensionList(C_1)	
for all $a \in$ dimensionList(C_1): imageCrsDomain(C_2 , a) = imageCrsDomain(C_1 , a)	

for all $a \in$ dimensionList(C_2):
crsSet(C_2, a) = crsSet(C_1, a)
dimensionType(C_2, a) = dimensionType(C_1, a)	

for all $a \in$ dimensionList(C_2), $c \in$ crsSet(C_2, a):
domain(C_2,a,c) = domain(C_1,a,c)	

for all $r \in$ rangeFieldNames(C_2):
rangeFieldType(C_2, r) = rangeFieldType(C_1, r)	

nullSet(C_2) = nullSet(C_1)	

for all $r \in$ rangeFieldNames(C_2):
interpolationDefault(C_2, r) = interpolationDefault(C_1, r)	

for all $r \in$ rangeFieldNames(C_2):
interpolationSet(C_2, r) = interpolationSet(C_1, r)	

Example The following expression transforms coverage C (which is assumed to have x and y dimensions) into the CRS identified by urn:ogc:def:crs:EPSG:6.6:63266405. Linear interpolation with null resistance “none” is applied to range field red if necessary, to all other range fields the default itenrpolation and null resistance is applied.

```
crsTransform( C,
              { x: “ urn:ogc:def:crs:EPSG::63266405”,
                y: “ urn:ogc:def:crs:EPSG::63266405”
              },
              { red(linear,none)
            }
```

7.1.29 coverageConstructorExpr

The **coverageConstructorExpr** element allows to create a d-dimensional coverage for some $d \geq 1$.

The domain definition consists, for each dimension, of a unique dimension name plus lower and upper bound of the coverage, expressed in a fixed image CRS and using integer coordinates; for this image CRS one of the identifiers listed in [05-096r1] Table 1 **shall** be used.

The coverage’s content is defined by an expression whose type defines the coverage range type.

This coverage has no other CRS associated beyond the abovementioned image CRS; further, it has no null values and interpolation methods associated. Finally, all other metadata are undefined. To set specific metadata for this new coverage the **setMatadataExpr** (Subclause 7.1.11) is available.

NOTE This constructor is useful

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

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

Let

f be a **fieldName**,
d be an **integer** with d>0,
$axis_i$ be pairwise distinct **dimensionType**s for $1 \le i \le d$[4],
$name_i$ be pairwise distinct **variableName**s for $1 \le i \le d$, which additionally, in the request on hand, are not used already as a variable in this expression's scope,
lo_i and hi_i be **integer**s for $1 \le i \le d$ with $lo_i \le hi_i$,
V be a **scalarExpr** possibly containing occurrences of $name_i$.

Then,

For any **coverageExpr** C
where

```
C  = coverage f
     over     name1 axis1 (lo1:hi1),
              …,
              named axisd (lod:hid)
     values   V
```

C is defined as follows:

Coverage constituent	Changed?
identifier(C) = “” (empty string)	X
for all $p \in$ imageCrsDomain(C): value(C, p) = V’ where expression V' is obtained from expression V by substituting all occurrences of $name_i$ by v where $(name_i, v) \in p$	X
imageCrs(C) = c_0 (i.e., the WCPS standard image CRS, see Clause 6)	X
imageCrsDomain(C) is set to a d-dimensional cube with dimension names $name_1 … name_d$ where the extent of dimension $name_i$ ranges from lo_i to hi_i (including these boundary values).	X
dimensionList(C_2) = { $name_1$,…, $name_d$ }	X
for all $a \in$ dimensionList(C): crsSet(C, a) = {},	X

[4] In the future, introduction of a GeneralDomain concept is planned for WCS which, among others, will allow an arbitrary number of so-called “abstract axes”, i.e., axes without spatio-temporal semantics. More than one dimension of this type will be allowed.

dimensionType(C, a) = if a = $name_i$ then $axis_1$ for $1 \leq i \leq d$	
for all $a \in$ dimensionList(C), $c \in$ crsSet(C, a): domain(C, a, c) = undefined[5]	**X**
for $r \in$ { f }, rangeFieldType(C,r) = type(V) i.e., the single range field's type is equal to the result type of expression V	**X**
nullSet(C) = {}	**X**
for all $r \in$ rangeFieldNames(C): interpolationDefault(C, r) = **none**	**X**
for all $r \in$ rangeFieldNames(C): interpolationSet(C, r) = {}	**X**

Example The expression below represents a 2-D greyscale image with a diagonal shade from white to black (the cast operator forces the floating point division result into an integer):

```
coverage greyshade
over     px x ( 0 : 255 ),
         py y ( 0 : 255 )
values   (unsigned char) ( px + py ) / 2
```

Example The expression below computes a 256-bucket histogram over band b some coverage C of unknown domain and dimension[6]:

```
coverage histogram
over     bucket x ( 0 : 255 )
values   count( C.b = bucket )
```

NOTE In future, WCPS is expected to support “abstract” axes with non-spatiotemporal semantics, which is the better choice for a histogram. Until then, some existing axis type like “x” is recommended.

7.1.30 coverageConstantExpr

The **coverageConstantExpr** element allows to create a d-dimensional coverage, for some $d \geq 1$, having one range field component with point values immediately given in the expression.

The domain definition consists, for each dimension, of a unique dimension name plus lower and upper bound of the coverage, expressed in a fixed image CRS and using integ-

[5] Note that, due to the empty crsSet, this “loop” anyway will not be “entered”.

[6] In the future, introduction of a GeneralDomain concept is planned for WCS which, among others, will allow an arbitrary number of so-called “abstract axes”, i.e., axes without spatio-temporal semantics. With the availability of abstract axes, dimension type “x” will be replaced by dimension type “abstract” in the code piece displayed.

er coordinates; for this image CRS one of the identifiers listed in [05-096r1] Table 1 **shall** be used.

The coverage's content is defined by a sequence of values. The narrowest range type encompassing all values encountered **shall** be its range type.

If the number of range components or their type turns out incompatible for any two points, or if the number of point values provided does not match with the domain extent specified, then the server **shall** respond with an exception.

This coverage has no other CRS associated beyond the abovementioned image CRS; further, it has no null values and interpolation methods associated. Finally, all other metadata are undefined. To set specific metadata for this new coverage the **setMetadataExpr** (Subclause 7.1.11) is available.

NOTE This constructor is useful for supplying a moderately sized coverage,e.g., as filter kernel.

Let

> *f* be a **fieldName**,
> *d* be an **integer** with *d*>0,
> $axis_i$ be pairwise distinct **dimensionType**s for $1 \leq i \leq d$[7],
> $name_i$ be pairwise distinct **variableName**s for $1 \leq i \leq d$, which additionally, in the request on hand, are not used already as a variable in this expression's scope,
> lo_i and hi_i be **integer**s for $1 \leq i \leq d$ with $lo_i \leq hi_i$,
> *S* be a **scalarExpr** possibly**.**

Then,

> For any **coverageExpr** *C*
> where

```
C  = coverage f
     over     name1 axis1 (lo1:hi1),
              …,
              named axisd (lod:hid)
     valueset S
```

> *C* is defined as follows:

Coverage constituent	Changed?
identifier(*C*) = "" (empty string)	**X**
for all $p \in$ imageCrsDomain(*C*): value(*C*, *p*) is determined by assigning each value in turn to a	**X**

[7] In the future, introduction of a GeneralDomain concept is planned for WCS which, among others, will allow an arbitrary number of so-called "abstract axes", i.e., axes without spatio-temporal semantics. More than one dimension of this type will be allowed.

grid point location, whereby assignment proceeds per dimension from the lowest to the highest coordinate, and loops over the grid points with the first dimension making up for the innermost loop, the second dimension for the next-to-innermost loop, etc., and the last dimension for the outermost loop.	
imageCrs(*C*) = c_0 (i.e., the WCPS standard image CRS, see Clause 6)	**X**
imageCrsDomain(*C*) is set to a d-dimensional cube with dimension names $name_1 … name_d$ where the extent of dimension $name_i$ ranges from lo_i to hi_i (including these boundary values).	**X**
dimensionList(C_2) = { $name_1$,…, $name_d$ }	**X**
for all *a* ∈ dimensionList(*C*): crsSet(*C*, *a*) = {}, dimensionType(*C*, *a*) = if *a* = $name_i$ then $axis_1$ for $1 \le i \le d$	**X**
for all *a* ∈ dimensionList(*C*), *c* ∈ crsSet(*C*, *a*): domain(*C*, *a*, *c*) = undefined[8]	**X**

for *r* ∈ { *f* },
rangeFieldType(*C*,*r*) = type(*V*)
i.e., the single range field’s type is equal to the result type of expression *V*
	X

nullSet(C) = {}	**X**
for all *r* ∈ rangeFieldNames(*C*): interpolationDefault(*C*, *r*) = **none**	**X**

for all *r* ∈ rangeFieldNames(*C*):
interpolationSet(*C*, *r*) = {}
	X

Example For a Sobel filter, a 3x3 filter kernel can be provided by the expression below.

```
coverage   Sobel3x3
over       px x ( -1 : 1 ),
           py y ( -1 : 1 )
value list <  1,  2,  1,
              0,  0,  0,
             -1, -2, -1
           >
```

[8] Note that, due to the empty crsSet, this “loop” anyway will not be “entered”.

7.1.31 condenseExpr

A **condenseExpr** is either a **reduceExpr** (see Subclause7.1.33) or a **generalCondense-Expr** (see Subclause 7.1.32). It takes a coverage and summarizes its values using some summarization function. The value returned is scalar.

Whenever one of the point values ("pixels", etc.) participating in a condense operation is equal to one of the null values of its coverage then the result of the operation **shall** be one of the values in the coverage's null value set. If no null value is available then the the server **shall** respond with a service exception.

7.1.32 generalCondenseExpr

The general **generalCondenseExpr** consolidates the grid point values of a coverage along selected dimensions 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. Admissible **condenseOpType**s are the binary operations `+`, `*`, `max`, `min`, `and`, and `or`.

Let

> *op* be a **condenseOpType**,
> n be some **integer** with $n \geq 0$,
> d be some **integer** with $d > 0$,
> $axis_i$ be **dimensionName**s for $1 \leq i \leq d$,
> $name_i$ be pairwise distinct **variableName**s for $1 \leq i \leq d$ which, in the request on hand, are not used already as a variable in this expression's scope,
> lo_i and hi_i be **integerExpr**s for $1 \leq i \leq d$ with $lo_i \leq hi_i$,
> C_j be **coverageExpr**s for $1 \leq j \leq n$,
> P be a **booleanExpr** possibly containing occurrences of $name_i$ and C_j,
> V be a **scalarExpr** possibly containing occurrences of $name_i$ and C_j
> where
> $1 \leq i \leq d$.

Then,

> For any **scalarExpr** *S*
> where
>
> *S* is one of

```
S′  =  condense op
       over name₁ axis₁ (lo₁:hi₁),
            …,
            name_d axis_d (lo_d:hi_d)
       using V
S″  =  condense op
       over name₁ axis₁ (lo₁:hi₁),
            …,
            name_d axis_d (lo_d:hi_d)
       where P
       using V
```

S is constructed as follows:

```
Let S = neutral element of type(V);
for all $name_1 \in \{lo_1, \ldots, hi_1\}$
    for all $name_2 \in \{lo_2, \ldots, hi_2\}$
        …
            for all $name_d \in \{lo_d, \ldots, hi_d\}$
                if (expression P is present)
                then
                    let predicate P′ be obtained from evaluating expression
                    P by substituting all occurrences of $name_i$ by its current
                    value where $name_i$ occurring in a coordinate
                    position of $C_j$ is interpreted as coordinates in the image
                    CRS of $C_j$;
                else
                    P’ = true;
                if (P’)
                then
                    let V′ be obtained from evaluating expression V
                    by substituting all occurrences of $name_i$ by its current
                    value where $name_i$ occurring in a coordinate
                    position of $C_j$ is interpreted as coordinates in the image
                    CRS of $C_j$;
                    S = S op value(V’);
return S
```

NOTE 1 Condensers are heavily used, among others, in these 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 Subclause 7.1.29) to phrase high-level imaging, signal processing, and statistical operations.

NOTE 2 The additional expressive power of **condenseExpr** over **reduceExpr** is twofold:

- A WCPS implementation **may** offer further summarisation functions, as long as these are commutative and associative.
- The **condenseExpr** gives explicit access to the coordinate values; this makes summarisation considerably more powerful (see example below).

Example For a filter kernel k, the condenser must summarise not only over the grid point under inspection, but also some neighbourhood. The following applies a 3x3 filter kernel to band b of some coverage C with extent x0…x1/y0…y1; note that the result image is defined to have an *x* and *y* dimension.

```
coverage filteredImage
over     px x ( x0 : x1 ),
         py y ( y0 : y1 )
values   condense +
         over    kx x ( -1 : +1 ),
                 ky y ( -1 : +1 )
         using   C.b[ kx + px , ky + py ] * k[ kx , ky ]
```

where *k* is a 3x3 matrix like

1	2	1
0	0	0
-1	-2	-1

NOTE See **coverageConstantExpr** for a way to specify the *k* matrix.

7.1.33 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 4 – reduceExpr definition via generalCondenseExpr

reduceExpr definition[9]	Meaning

add(a) =
   condense +
   over p$_1$ D$_1$(imageCrsDomain(a,D$_1$)),
       …,
       p$_d$ D$_d$(imageCrsDomain(a,D$_1$)),
   using a[**p**$_1$, …, **p**$_d$]
	sum over all points in *a*

avg(a) = add(a) / | imageCrsDomain(a) |	Average of all points in *a*

min(a) =
   condense min
   over p$_1$ D$_1$(imageCrsDomain(a,D$_1$)),
       …,
       p$_d$ D$_d$(imageCrsDomain(a,D$_1$))
   using a[px]
	Minimum of all points in *a*

max(a) =
   condense max
   over p$_1$ D$_1$(imageCrsDomain(a,D$_1$)),
       …,
       p$_d$ D$_d$(imageCrsDomain(a,D$_1$))
	Maximum of all points in *a*

[9] *a* is a numeric, *b* a Boolean **coverageExpr**

using a[x]	
count(b) = **condense** + **over p**$_1$ D$_1$(imageCrsDomain(a,D$_1$)), …, **p**$_d$ D$_d$(imageCrsDomain(a,D$_1$)) **where** b[x] **using** 1	Number of points in *b*
some(b) = **condense** or **over p**$_1$ D$_1$(imageCrsDomain(a,D$_1$)), …, **p**$_d$ D$_d$(imageCrsDomain(a,D$_1$)) **using** b[x]	is there any point in *b* with value true?
all(b) = **condense** and **over p**$_1$ D$_1$(imageCrsDomain(a,D$_1$)), …, **p**$_d$ D$_d$(imageCrsDomain(a,D$_1$)) **using** b[x]	do all points of *b* have value true?

7.2 Expression evaluation

This Sublause defines additional rules for *ProcessCoverages* expression evaluation.

7.2.1 Evaluation sequence

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

7.2.2 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.

7.2.3 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 yielding the same result as C_1.

Example `C * ( C > 0 )`

7.2.4 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):

- Range field selection, trimming, slicing
- 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.

7.2.5 Range type compatibility and extension

A range type t_1 is said to be **cast-compatible** with a range type t_2 iff the following conditions hold:

- Both range types, t_1 and t_2, have the same number of field elements, say d;
- For each range field element position i with $1 \leq i \leq d$, the ith range field type $f_{1,i}$ of t_1 is **cast-compatible** with the ith range field type $f_{2,i}$ of t_2.

A range field type f_1 is said to be **cast-compatible** with a range field type f_2 iff f_2 can be cast to f_1, whereby **casting** of f_2 to f_1 is defined as looking up f_2 in Table 5 and replacing it by its right-hand neighbour type or, if it is the last type in line, by the first type of the next line. This is repeated until either f_1 is matched, or the end of the Table 5 is reached. Type f_1 can be cast to type f_2 if the casting procedure terminates with finding t_2, otherwise the cast is not possible.

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

On both arguments to binary operations casting **shall** be attempted until both argument types are equal; if such a common type can be found, then this **shall** be the binary operation result type; otherwise, an exception **shall** be thrown.

Example For three single-field coverages `F`, `I`, and `B` with range types `float`, `integer`, and `Boolean`, resp., the result type of the following expression is `float`:

```
F + I + B
```

Table 5 – Type extension sequence.

Type extension rules
Boolean > char
char > Boolean
Boolean > unsigned char
unsigned char > Boolean
char > short
char > unsigned short
unsigned char > short
unsigned char > 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
float > complex
double > complex2
complex > complex2

The type of each of the operands of a multiplicative operator (+, -, *, /) **shall** be a type that can be extended to a numeric numeric type, or an exception occurs. The type of a multiplicative expression is the extended type of its operands. If this promoted type is an integer type, then integer arithmetic **shall** be performed; if this promoted type is a floating-point type, then floating-point arithmetic **shall** be performed; if this promoted type is a complex type, then complex arithmetic shall be performed.

NOTE Explicit and implicit casts should be used with caution, as unintended consequences can arise. Data can be lost when floating-point representations are converted to integral representations as the fractional components of the floating-point values will be truncated (rounded down). Conversely, converting from an integral representation to a floating-point one can also lose precision, since the floating-point type may be unable to represent the integer exactly (for example, float might be an IEEE 754 single precision type, which cannot represent the integer 16777217 exactly, while a 32-bit integer type can). This can lead to situations such as storing the same integer value into two variables of type int and type float which return false if compared for equality.

Whenever rounding from floating-point to integer numbers is required, rounding towards zero **shall** be applied.

Example For a Boolean single-field coverage `B`, and an integer single-field coverage `I`, the following expression will evaluate to some integer value:

```
count( I * B )
```

Before executing any binary operation where the two operands are of different type a cast operation **shall** be attempted to achieve equal types. The result type of each of the binary induced operations (see Section 7.1.21) addition, subtraction, multiplication, and division **shall** be equal to the common type of the input operands.

If a cast is attempted or implicitly needed, but not possible according to the above rules, then an exception **shall** be reported.

NOTE The cast operation is similar to that found in many programming languages, such as C/C++.

7.3 Evaluation exceptions

Whenever a coverage expression cannot be evaluated according to the rules specified in Clauses 7.1 and 7.2, a Web Coverage Processing Server **shall** respond with an exception.

Example The following expressions will lead to an exception when used in a *ProcessCoverages* request (reasons: division by zero; square root of a negative number):

```
C / 0

sqrt( - abs( C ) )
```

7.4 processCoveragesExpr response

The response to a request sending a **processCoveragesExpr** is one of the following:

Depending on its result type, the normal response to a valid *ProcessCoverages* request **shall** consist of one of the following alternatives:

- A (possibly empty) set of coverages.
- A scalar (where scalar summarizes all non-coverage type data, such as numbers, strings, URLs) or a structure composed of scalars (via structs, lists, etc., possibly nested).

- An exception.

Encoding of single coverages is governed by the **encodedCoverageExpr** (see Section 7.1.4). Encoding of scalar structures and exceptions as well as the representation of the overall response **shall** be governed by a separate, additional protocol specification.

NOTE WCS [OGC 08-059r3] specifies a response protocol suitable for WCPS results. An embedding of WCPS into WPS, including its specific request and delivery structures, is under work.

Annex A
(normative)

Abstract Test Suite

The Abstract Test Suite for WCPS is provided in [OGC 08-069r2].

Annex B (normative)

WCPS Expression Syntax

B.1 Overview

This Annex summarizes the WCPS expression syntax. It is described in EBNF grammar syntax according to [IETF RFC 2616]. Underlined tokens represent literals which appear "as is" in a valid WCPS expression ("terminal symbols"), underlined tokens in italics represent sub-expressions to be substituted according to the grammar production rules ("non-terminals"). Any number of whitespace characters (blank, tabulator, newline) **may** appear between tokens as long as parsing is unambiguous.

Example Between language tokens (such as "for") and names there must be at least one whitespace character, whereas between names and non-alphanumeric tokens (such as opening parenthesis, "("), no whitespace is required.

Meta symbols used are as follows:

- brackets ("`[…]`") denote optional elements which **may** occur or be left out;
- an asterisk ("`*`") denotes that an arbitrary number of repetitions of the following 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.

B.2 WCPS syntax

```
processCoveragesExpr:
        for variableName in ( coverageList )
            *( , variableName in ( coverageList ) )
        [ where booleanScalarExpr ]
        return processingExpr

coverageList:
        coverageName *( , coverageName )

processingExpr:
        encodedCoverageExpr
      | storeExpr
      | scalarExpr

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

```
formatName:
         stringConstant

extraParams:
         stringConstant

storeExpr:
         store ( encodedCoverageExpr )

scalarExpr:
         metaDataExpr
       | generalCondenseExpr
       | booleanScalarExpr
       | numericScalarExpr
       | stringScalarExpr
       | ( scalarExpr )

booleanScalarExpr:
       | booleanScalarExpr and booleanScalarTerm
       | booleanScalarExpr or booleanScalarTerm
       | booleanScalarExpr xor booleanScalarTerm
       | booleanScalarTerm

booleanScalarTerm:
       | booleanScalarTerm and booleanScalarFactor
       | booleanScalarTerm or booleanScalarFactor
       | booleanScalarTerm xor booleanScalarFactor
       | booleanScalarFactor

booleanScalarFactor:
       | numericScalarExpr compOp numericScalarExpr
       | stringScalarExpr compOp stringScalarExpr
       | not booleanScalarExpr
       | booleanConstant

numericScalarExpr:
       | numericScalarExpr addOp numericScalarTerm
       | numericScalarTerm

numericScalarTerm:
         numericScalarTerm multOp numericScalarFactor
       | numericScalarFactor
       | - numericScalarFactor
       | round ( numericScalarFactor )

numericScalarFactor:
       | ( numericScalarExpr )
       | integerConstant
       | floatConstant
       | complexConstant
       | condenseExpr
```

```
compOp:
        compOpEqual
      | compOpGreaterLess

compOpEqual:
        =
      | !=

compOpGreaterLess:
        >
      | >=
      | <
      | <=

multOp:
        *
      | /

addOp:
        +
      | -

stringScalarExpr:
        metaDataExpr              // currently only identifier() allowed
      | stringConstant


metaDataExpr:
        identifier ( coverageExpr )
      | imageCrs ( coverageExpr )
      | imageCrsDomain ( coverageExpr )
      | imageCrsDomain ( coverageExpr , axisName )
      | crsSet ( coverageExpr )
      | domain ( coverageName , axisName , crsName )
      | nullSet ( coverageExpr )
      | interpolationDefault ( coverageExpr , fieldName )
      | interpolationSet ( coverageExpr , fieldName )

 setMetaDataExpr:
        setIdentifier ( stringConstant )
      | setCrsSet ( coverageExpr ,
          { [ crsName *( , crsName ) ] } )
      | setNullSet ( coverageExpr ,
          { [ rangeExpr *( , rangeExpr ) ] } )
      | setInterpolationDefault ( coverageExpr , fieldName
          , interpolationMethod )
      | setInterpolationSet ( coverageExpr , fieldName ,
          { [ interpolationMethod
              *( , interpolationMethod ) ] } )

rangeExpr:
        struct { fieldname : scalarExpr
                *( , fieldname : scalarExpr ) }
```

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

inducedExpr:
       | unaryInducedExpr
       | binaryInducedExpr
       | rangeConstructorExpr

unaryInducedExpr:
         unaryArithmeticExpr
       | exponentialExpr
       | trigonometricExpr
       | booleanExpr
       | castExpr
       | fieldExpr

unaryArithmeticExpr:
         + 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:
         ( rangeType ) coverageExpr
```

```
rangeType:
         bool
       | char
       | unsigned char
       | short
       | unsigned short
       | long
       | unsigned long
       | float
       | double
       | complex
       | complex2

fieldExpr:
         coverageExpr . fieldName

binaryInducedExpr:
       | coverageExpr binaryInducedOp coverageExpr
       | coverageExpr binaryInducedOp scalarExpr
       | scalarExpr binaryInducedOp coverageExpr

binaryInducedOp:
       | +
       | -
       | *
       | /
       | and
       | or
       | xor
       | =
       | <
       | >
       | <=
       | >=
       | !=
       | overlay


rangeConstructorExpr:
         [ struct ] { fieldName : coverageExpr
                    *( ; fieldName : coverageExpr ) }

subsetExpr:
       | trimExpr
       | sliceExpr
       | extendExpr

trimExpr:
         coverageExpr [ dimensionIntervalList ]
       | trim ( coverageExpr , { dimensionIntervalList } )

sliceExpr:
         coverageExpr [ dimensionPointList ]
       | slice ( coverageExpr , { dimensionPointList } )
```

```
extendExpr:
        extend ( coverageExpr , dimensionIntervalList )

scaleExpr:
        scale ( coverageExpr , dimensionIntervalList ,
                fieldInterpolationList )

crsTransformExpr:
        crsTransform ( coverageExpr ,
                       dimensionIntervalList ,
                       fieldInterpolationList )

dimensionPointList:
        dimensionPointElement *( , dimensionPointElement )

dimensionPointElement:
        axisName [ : crsName ] ( dimensionPoint )

dimensionPoint:
        scalarExpr

dimensionIntervalList:
        dimensionIntervalElement
        *( , dimensionIntervalElement )

dimensionIntervalElement:
       axisName [ : crsName ] ( dimensionIntervalExpr )

dimensionIntervalExpr:
        scalarExpr : scalarExpr
       | domain ( coverageName , axisName , crsName )

fieldInterpolationList:
        { fieldInterpolationListElement
          *( , fieldInterpolationListElement ) }

fieldInterpolationListElement:
        fieldName : interpolationMethod

interpolationMethod:          // taken from WCS [OGC 07-067r5]
        ( interpolationType : nullResistance )

interpolationType:            // taken from WCS [OGC 07-067r5] Table I.7
        nearest
       | linear
       | quadratic
       | cubic

nullResistance:               // taken from WCS [OGC 07-067r5]
        full
       | none
       | half
       | other
```

```
coverageConstructorExpr:
         coverage coverageName
         over axisIterator *( , axisIterator )
         values scalarExpr

axisIterator:
          variableName axisName ( intervalExpr )

intervalExpr:
         integerExpr : integerExpr
       | imageCrsDomain ( coverageName , axisName )

coverageConstantExpr:
         coverage coverageName
         over axisIterator *( , axisIterator )
         value list < constant *( ; constant ) >

condenseExpr:
         reduceExpr
       | generalCondenseExpr

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

generalCondenseExpr:
         condense condenseOpType
         over axisIterator *( , axisIterator )
         [ where booleanScalarExpr ]
         using scalarExpr

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

variableName:
         name                // see below for restrictions

coverageName:
         name                // coverage identifier as in WCS [OGC 07-067r5]

crsName:
        stringConstant       // containing a valid CRS name
```

```
axisName[10]:
          x
        | y
        | z
        | t
        | name              // where name is none of the other variants

fieldName:
          name              // as defined in WCS [OGC 07-067r5] Table 19

constant:
          stringConstant | booleanConstant
        | integerConstant | floatConstant | complexConstant

complexConstant:
          ( floatConstant , floatConstant )
```

A `variableName` **shall** be a consecutive sequence of characters where the first character **shall** be either an alphabetical character or the "$" character and the remaining characters consist of decimal digits, upper case alphabetical characters, lower case alphabetical characters, underscore ("_"), and nothing else. The length of an identifier **shall** be at least 1.

NOTE 1 The regular expression describing an identifier is: [$a-zA-Z_][0-9a-zA-Z_]*.

NOTE 2 WCS [OGC 07-067r5] allows more freedom in the choice of identifiers; for the sake of simplicity this is tightened for now, but may be adapted to the WCS identifier definition in a future version of this standard.

A *`booleanConstant`* **shall** represent a logical truth value expressed as one of the literals "true" and "false" resp., whereby upper and lower case characters **shall** not be distinguished.

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

A *`floatConstant`* **shall** represent a floating point number following the syntax of the Java programming language.

A *`stringConstant`* **shall** represent a character sequence expressed by enclosing it into double quotes ('"´).

[10] In the future, introduction of a General Domain concept is planned for WCS which, among others, will allow an arbitrary number of so-called "abstract axes", i.e., axes without spatio-temporal semantics. DimensionType, then, will be extended with the additional dimension type "abstract".

Bibliography

[1] Ritter, G., Wilson, J., Davidson, J.: Image Algebra: An Overview. Computer Vision, Graphics, and Image Processing, 49(3)1990, pp. 297-331

[2] Baumann, P.: A Database Array Algebra for Spatio-Temporal Data and Beyond. The Fourth International Workshop on Next Generation Information Technologies and Systems (NGITS '99), July 5-7, 1999, Zikhron Yaakov, Israel, Lecture Notes on Computer Science 1649, Springer Verlag, pp. 76 - 93