OGC® DOCUMENT: YY-999

External identifier of this OGC® document: http://www.opengis.net/doc/{doc-type}/{standard}/{m.n}

Open Geospatial Consortium

OGC (ADD TITLE TEXT)

STANDARD

DRAFT

Version: 1.0
Submission Date: 2029-03-30
Approval Date: 2029-03-30
Publication Date: 2029-03-30
Editor: Editor One, Editor Two

Note

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

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

CONTENTS

LIST OF TABLES

LIST OF FIGURES

I KEYWORDS

The following are keywords to be used by search engines and document catalogues.

ogcdoc, OGC document, API, openapi, html

II PREFACE

i. Abstract

CoverageJSON is a format for publishing geo-temporal data to the Web, based on JavaScript Object Notation (JSON). Simplicity, machine and human readability and efficiency of the format were the primary design goals. The primary use case for CoverageJSON is to enable the development of interactive visual websites that display and manipulate environmental data within a web browser, although other use cases are possible.

CoverageJSON has been demonstrated to be an effective, efficient format, friendly to web and application developers, and therefore consistent with the current OGC API developments. The format supports the efficient transfer, from big data stores, of useful quantities of data to lightweight clients, such as browsers and mobile applications. This enables local manipulation of the data in a format familiar to, and popular with, web developers, and readily usable, for example, by science researchers.

It can be used to encode coverages and collections of coverages. Data may be gridded or non-gridded, and data values may represent continuous values (such as temperature) or discrete categories (such as land cover classes). CoverageJSON uses JSON-LD to provide interoperability with RDF and Semantic Web applications and to reduce the potential size of the payload.

Relatively large datasets can be handled efficiently in a “web-friendly” way by partitioning information among several CoverageJSON documents, including a mechanism for tiling. Nevertheless, CoverageJSON is not intended to be a replacement for efficient binary formats such as NetCDF, HDF or GRIB, and is not intended primarily to store or transfer very large datasets in bulk.

The simplest and most common use case is to embed the data values of all variables in a Coverage object within the CoverageJSON document, so that it is completely standalone, which would enable the use of very simple clients.

The next-simplest use case is to put data values for each variable (parameter) in separate array objects in separate CoverageJSON documents, linked from the Coverage object. This is useful for a multi-variable dataset, where one might want temperature, humidity, wind speed, etc., to be recorded in separate files, so the user client only needs to load the variables of interest.

A sophisticated use case is to use tiling objects, where the data values are partitioned spatially and temporally, so that a single variable’s data values would be split among several documents. A simple example of this would be to encode each timestep of a dataset in a separate file, but the tiles could also be divided spatially, like a tiled map server.

It is proposed to be adopted as a Community Standard.

ii. Keywords

The following are keywords to be used by search engines and document catalogues.

ogcdoc, OGC document, JSON, JSON-LD, CoverageJSON, CovJSON, Coverage, spatiotemporal, linked-data

iii. Preface

The specification was an outcome of a European Union project, “Maximizing the Exploitation of Linked Open Data in Enterprise and Science” (MELODIES), run from 2013 to 2016, and released under a Creative Commons 4.0 Licence.

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

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

iv. Submitting organizations

The following organizations submitted this Document to the Open Geospatial Consortium (OGC):

UK Met Office (Technical Member)

University of Reading (Associate/University Member)

Meteorological Service of Canada (Associate Member)

Unidata UCAR (Technical Member)

US NWS/NOAA (Principal Member)

ESIP/JPL/NASA (Strategic Member)

v. Submitters

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

Chris Little (editor) UK Met Office

Jon Blower (editor) University of Reading

Tom Kralidis (contributor) Meteorological Service of Canada

Ethan Davies (contributor) Unidata UCAR

Steve Olson (contributor) US NWS/NOAA

Lewis McGibbney (contributor) ESIP/JPL/NASA

vi. Acknowledgements

This work was inspired by a demonstration of the concept by Joan Masó of CREAF.

III SECURITY CONSIDERATIONS

No security considerations have been made for this document.

IV SUBMITTING ORGANIZATIONS

The following organizations submitted this Document to the Open Geospatial Consortium (OGC):

- Organization One
- Organization Two

2

CONFORMANCE

2 CONFORMANCE

Conformance with this specification shall be checked using all the relevant tests specified in Annex A (normative) of this document. The framework, concepts, and methodology for testing, and the criteria to claim conformance are specified in the OGC Compliance Testing Policies and Procedures and the OGC Compliance Testing web site.

The one Standardization Target for this specification is the format of CoverageJSON.

In order to conform to this OGC® Community Standard, a software implementation shall choose to implement any one of the conformance levels specified in Annex A (normative).

All requirements-classes and conformance-classes described in this document are owned by the standard(s) identified.

1

SCOPE

1 SCOPE

The specification is a logical implementation of the long-established abstract conceptual standard ISO19123:2005 (also known as OGC Abstract Topic 6). The CoverageJSON specification has been stable for several years, with a wide-spread international community in the environmental sciences. The current version of the open specification, on the original GitHub, is labelled 'V0.2-draft'. This document proposes that version, with only editorial changes for presentation, as a Community Standard.

A formal JSON schema has been added for all the JSON objects identified, so that conformance to the specification can be checked.

Unused or imprecisely specified components have been removed.

Future improvements have been identified, such as the use of multiple time dimensions, or to use JSON-LD V1.1 which may give better compatibility with the OGC Coverage Implementation Schema (CIS) Version 1.1 (OGC09-146r8), which is a different logical implementation of the abstract conceptual standard ISO19123:2005/OGC Abstract Topic 6. These improvements are out of scope of this document.

This specification does not specify encoding of information in any format other than JSON.

3

NORMATIVE REFERENCES

3 NORMATIVE REFERENCES

There are no normative references in this document.

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

[covjson] CoverageJSON original GitHub repository, https://covjson.org/spec

[netcdf] Network Common Data Form (NetCDF), https://www.unidata.ucar.edu/software/netcdf

[ogc07011] Abstract Specification Topic 6 Schema for coverage geometry and functions, OGC 07-011, https://portal.ogc.org/files/?artifact_id=19820 also known as ISO19123:2005

[ogc18010r7] Lott, R.: Well-Known Text representation of Coordinate Reference Systems, http://docs.opengeospatial.org/is/18-010r7/18-010r7.html

[ogcutc] OGC definition of the UTC Temporal Reference System, http://www.opengis.net/def/trs/BIPM/0/UTC

[rfc2119] Key words “MUST”, “MUST NOT”, “REQUIRED”, “SHALL”, “SHALL NOT”, “SHOULD”, “SHOULD NOT”, “RECOMMENDED”, “MAY”, and “OPTIONAL” are to be interpreted as described in IETF RFC 2119, https://datatracker.ietf.org/doc/html/rfc2119.txt

[rfc2413] Weibel, S., Kunze, J., Lagoze, C., Wolf, M.: IETF RFC 2413, Dublin Core Metadata for Resource Discovery, https://tools.ietf.org/rfc/rfc2413.txt

[rfc3339] Klyne, G., Newman, C.: IETF RFC 3339, Date and Time on the Internet: Timestamps, https://tools.ietf.org/rfc/rfc3339.txt (ISO8601-based lexical representation)

[rfc3896] Berners-Lee, T., Fielding, R., Masinter, L: IETF RFC 3896, Uniform Resource Identifier (URI): Generic Syntax, https://tools.ietf.org/rfc/rfc3896.txt

[rfc4627] Crockford, D.: IETF RFC 4627, The application/json Media Type for JavaScript Object Notation (JSON), and terms `object`, `array`, `name`, `value`, `string`, `number`, and `null`, https://tools.ietf.org/rfc/rfc4627.txt.

[rfc4647] Phillips, A., Ed., and M. Davis, Ed., “Matching of Language Tags”, BCP 47, RFC 4647, September 2006, https://www.rfc-editor.org/info/bcp47

[rfc5646] Phillips, A., Ed., and M. Davis, Ed., “Tags for Identifying Languages”, BCP 47, RFC 5646, September 2009, https://www.rfc-editor.org/info/bcp47

[rfc6570] Gregorio, J., Fielding, R., Hadley, M., Nottingham, M., Orchard, D.: IETF RFC 6570, URI Template, https://datatracker.ietf.org/doc/html/rfc6570

[rfc6906] Wilde, E.: A ‘profile’ Link Relation Type, IETF RFC 6906, https://datatracker.ietf.org/doc/html/rfc6906

[rfc7946] Butler, H., Daly, M., Doyle, A., Gillies, S., Hagen, S., Schaub, T.: IETF RFC 7946, The GeoJSON Format, https://tools.ietf.org/rfc/rfc7946.txt

[w3ccurie] W3C CURIE Syntax 1.0, Syntax for expressing Compact URIs, Working Group Note, 2010-12-16, http://www.w3.org/TR/curie/

[w3cjsonld10] W3C JSON-LD1.0, JSON-based Serialization for Linked Data, 2014-01-16 superseded 2020-11-03, https://www.w3.org/TR/2014/REC-json-ld-20140116/

[UCUM] Schadow, G., McDonald, C. J.: Unified Code for Units of Measure, http://unitsofmeasure.org

4

TERMS AND DEFINITIONS

4 TERMS AND DEFINITIONS

This document uses the terms defined in OGC Policy Directive 49, which is based on the ISO/IEC Directives, Part 2, Rules for the structure and drafting of International Standards. In particular, the word “shall” (not “must”) is the verb form used to indicate a requirement to be strictly followed to conform to this document and OGC documents do not use the equivalent phrases in the ISO/IEC Directives, Part 2.

This document also uses terms defined in the OGC Standard for Modular specifications (OGC 08-131r3), also known as the ‘ModSpec’. The definitions of terms such as standard, specification, requirement, and conformance test are provided in the ModSpec.

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

The key words “MUST”, “MUST NOT”, “REQUIRED”, “SHALL”, “SHALL NOT”, “SHOULD”, “SHOULD NOT”, “RECOMMENDED”, “MAY”, and “OPTIONAL” in this document are to be interpreted as described in IETF RFC 2119.

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

4.1. RFC

Request For Comment (IETF)

4.2. BCP

Best Current Practice. It may include several RFCs (IETF)

4.3. Coverage

feature that acts as a function to return values from its range for any direct position within its spatial, temporal or spatiotemporal domain (ISO19123-1)

4.4. **i18n object**

string in multiple languages with tags as defined in [BCP 47](http://tools.ietf.org/html/bcp47), and the value is the string in that language. The special language tag `"und"` can be used to identify a value whose language is unknown or undetermined. (IETF BCP47)

5

CONVENTIONS

5 CONVENTIONS

This sections provides details and examples for any conventions used in the document. Examples of conventions are symbols, abbreviations, use of XML schema, or special notes regarding how to read the document.

5.1. Identifiers

The normative provisions in this specification are denoted by the URI

http://www.opengis.net/spec/{specification}/{m.n}

All requirements and conformance tests that appear in this document are denoted by partial URIs which are relative to this base.

6

FULL SPECIFICATION: NORMATIVE AND INFORMATIVE INTERMIXED

6 FULL SPECIFICATION: NORMATIVE AND INFORMATIVE INTERMIXED

7

1. INTRODUCTION

7

1. INTRODUCTION

CoverageJSON is a format for encoding coverage data like grids, time series, and vertical profiles, distinguished by the geometry of their spatiotemporal domain. A CoverageJSON object represents a domain, a range, a coverage, or a collection of coverages. A range in CoverageJSON represents coverage values. A coverage in CoverageJSON is the combination of a domain, parameters, ranges, and additional metadata. A coverage collection represents a list of coverages.

A complete CoverageJSON data structure is always an object (in JSON terms). In CoverageJSON, an object consists of a collection of name/value pairs — also called members. For each member, the name is always a string. Member values are either a string, number, object, array or one of the literals: true, false, and null. An array consists of elements where each element is a value as described above.

7.1. 1.1. Example

A CoverageJSON grid coverage of global air temperature:

```
{
  "type" : "Coverage",
  "domain" : {
    "type": "Domain",
    "domainType": "Grid",
    "axes": {
      "x": { "start": -179.5, "stop": 179.5, "num": 360 },
      "y": { "start": -89.5, "stop": 89.5, "num": 180 },
      "t": { "values": ["2013-01-13T00:00:00Z"] }
    },
    "referencing": [{
      "coordinates": ["x","y"],
      "system": {
        "type": "GeographicCRS",
        "id": "http://www.opengis.net/def/crs/OGC/1.3/CRS84"
      }
    }, {
      "coordinates": ["t"],
      "system": {
        "type": "TemporalRS",
        "calendar": "Gregorian"
      }
    }]
  },
  "parameters" : {
    "TEMP": {
      "type" : "Parameter",
      "description" : {
        "en": "The air temperature measured in degrees Celsius."
      },
      "unit" : {
        "label": {
```

```
          "en": "Degree Celsius"
        },
        "symbol": {
          "value": "Cel",
          "type": "http://www.opengis.net/def/uom/UCUM/"
        }
      },
      "observedProperty" : {
        "id" : "http://vocab.nerc.ac.uk/standard_name/air_temperature/",
        "label" : {
          "en": "Air temperature",
          "de": "Lufttemperatur"
        }
      }
    }
  },
  "ranges" : {
    "TEMP" : "http://example.com/coverages/123/TEMP"
  }
}
```

Figure 1

where `"http://example.com/coverages/123/TEMP"` points to the following document:

```
{
  "type" : "NdArray",
  "dataType": "float",
  "axisNames": ["t","y","x"],
  "shape": [1, 180, 360],
  "values" : [ 27.1, 24.1, null, 25.1, ... ]
}
```

Figure 2

Range data can also be directly embedded into the main CoverageJSON document, making it standalone.

8

2. I18N OBJECTS

8 2. I18N OBJECTS

An i18n object represents a string in multiple languages where each key is a language tag as defined in BCP 47, and the value is the string in that language. The special language tag `"und"` can be used to identify a value whose language is unknown or undetermined.

Example:

```
{
  "en": "Temperature",
  "de": "Temperatur"
}
```

Figure 3

9

3. PARAMETER OBJECTS

3. PARAMETER OBJECTS

Parameter objects represent metadata about the values of the coverage in terms of the observed property (like water temperature), the units, and others.

- A parameter object MAY have any number of members (name/value pairs).
- A parameter object MUST have a member with the name `"type"` and the value `"Parameter"`.
- A parameter object MAY have a member with the name `"id"` where the value MUST be a string and SHOULD be a common identifier.
- A parameter object MAY have a member with the name `"label"` where the value MUST be an i18n object that is the name of the parameter and which SHOULD be short. Note that this SHOULD be left out if it would be identical to the `"label"` of the `"observedProperty"` member.
- A parameter object MAY have a member with the name `"description"` where the value MUST be an i18n object which is a, perhaps lengthy, textual description of the parameter.
- A parameter object MUST have a member with the name `"observedProperty"` where the value is an object which MUST have the member `"label"` and which MAY have the members `"id"`, `"description"`, and `"categories"`. The value of `"label"` MUST be an i18n object that is the name of the observed property and which SHOULD be short. If given, the value of `"id"` MUST be a string and SHOULD be a common identifier. If given, the value of `"description"` MUST be an i18n object with a textual description of the observed property. If given, the value of `"categories"` MUST be a non-empty array of category objects. A category object MUST an `"id"` and a `"label"` member, and MAY have a `"description"` member. The value of `"id"` MUST be a string and SHOULD be a common identifier. The value of `"label"` MUST be an i18n object of the name of the category and SHOULD be short. If given, the value of `"description"` MUST be an i18n object with a textual description of the category.
- A parameter object MAY have a member with the name `"categoryEncoding"` where the value is an object where each key is equal to an `"id"` value of the `"categories"` array within the `"observedProperty"` member of the parameter object. There MUST be no duplicate keys. The value is either an integer or an array of integers where each integer MUST be unique within the object.
- A parameter object MAY have a member with the name `"unit"` where the value is an object which MUST have either or both the members `"label"` or/and “symbol”, and which MAY have the member `"id"`. If given, the value of `"symbol"` MUST either be a string of the symbolic notation of the unit, or an object with the members `"value"` and `"type"` where `"value"` is the symbolic unit notation and `"type"` references the unit serialization scheme that is used. `"type"` MUST HAVE the value `"http://www.opengis.net/def/uom/UCUM/`" if UCUM is used, or a custom value as recommended in section "Extensions". If given, the value of `"label"` MUST be an i18n object of the name of the unit and SHOULD be short. If given, the value of `"id"` MUST be a string and

SHOULD be a common identifier. It is RECOMMENDED to reference a unit serialization scheme to allow automatic unit conversion.

- A parameter object MUST NOT have a "unit" member if the "observedProperty" member has a "categories" member.

Example for a continuous-data parameter:

```
{
  "type" : "Parameter",
  "description" : {
    "en": "The sea surface temperature in degrees Celsius."
  },
  "observedProperty" : {
    "id" : "http://vocab.nerc.ac.uk/standard_name/sea_surface_temperature/",
    "label" : {
      "en": "Sea Surface Temperature"
    },
    "description" : {
      "en": "The temperature of sea water near the surface (including the part
under sea-ice, if any), and not the skin temperature."
    }
  },
  "unit" : {
    "label" : {
      "en": "Degree Celsius"
    },
    "symbol": {
      "value": "Cel",
      "type": "http://www.opengis.net/def/uom/UCUM/"
    }
  }
}
```

Figure 4

Example for a categorical-data parameter:

```
{
  "type" : "Parameter",
  "description" : {
    "en": "The land cover category."
  },
  "observedProperty" : {
    "id" : "http://example.com/land_cover",
    "label" : {
      "en": "Land Cover"
    },
    "description" : {
      "en": "longer description..."
    },
    "categories": [{
      "id": "http://example.com/land_cover/categories/grass",
      "label": {
        "en": "Grass"
      },
      "description": {
        "en": "Very green grass."
      }
    }, {
      "id": "http://example.com/land_cover/categories/forest",
```

```json
        "label": {
          "en": "Forest"
        }
      }]
    },
    "categoryEncoding": {
      "http://example.com/land_cover/categories/grass": 1,
      "http://example.com/land_cover/categories/forest": [2,3]
    }
  }
```

Figure 5

10

4. PARAMETERGROUP OBJECTS

10

4. PARAMETERGROUP OBJECTS

Parameter group objects represent logical groups of parameters, for example vector quantities.

- A parameter group object MAY have any number of members (name/value pairs).
- A parameter group object MUST have a member with the name `"type"` and the value `"ParameterGroup"`.
- A parameter group object MAY have a member with the name `"id"` where the value MUST be a string and SHOULD be a common identifier.
- A parameter group object MAY have a member with the name `"label"` where the value MUST be an i18n object that is the name of the parameter group and which SHOULD be short. Note that this SHOULD be left out if it would be identical to the `"label"` of the `"observedProperty"` member.
- A parameter group object MAY have a member with the name `"description"` where the value MUST be an i18n object which is a, perhaps lengthy, textual description of the parameter group.
- A parameter group object MAY have a member with the name `"observedProperty"` where the value is an object as specified for parameter objects.
- A parameter group object MUST have either or both the members `"label"` or/and `"observedProperty"`.
- A parameter group object MUST have a member with the name `"members"` where the value is a non-empty array of parameter identifiers (see 6.3 Coverage objects).

Example of a group describing a vector quantity:

```
{
  "type": "ParameterGroup",
  "observedProperty": {
    "label": {
      "en": "Wind velocity"
    }
  },
  "members": ["WIND_SPEED", "WIND_DIR"]
}
```

Figure 6

where `"WIND_SPEED"` and `"WIND_DIR"` reference existing parameters in a CoverageJSON coverage or collection object by their short identifiers.

Example of a group describing uncertainty of a parameter:

```
{
  "type": "ParameterGroup",
  "label": {
    "en": "Daily sea surface temperature with uncertainty information"
```

```
  },
  "observedProperty": {
    "id": "http://vocab.nerc.ac.uk/standard_name/sea_surface_temperature/",
    "label": {
      "en": "Sea surface temperature"
    }
  },
  "members": ["SST_mean", "SST_stddev"]
}
```

Figure 7

where "`SST_mean`" references the following parameter:

```
{
  "type" : "Parameter",
  "observedProperty" : {
    "label" : {
      "en": "Sea surface temperature daily mean"
    },
    "statisticalMeasure": "http://www.uncertml.org/statistics/mean",
    "statisticalPeriod": "P1D",
    "narrowerThan": ["http://vocab.nerc.ac.uk/standard_name/sea_surface_
temperature/"]
  },
  "unit" : {
    "label": {
      "en": "Kelvin"
    },
    "symbol": {
      "value": "K",
      "type": "http://www.opengis.net/def/uom/UCUM/"
    }
  }
}
```

Figure 8

and "`SST_stddev`":

```
{
  "type" : "Parameter",
  "observedProperty" : {
    "label" : {
      "en": "Sea surface temperature standard deviation of daily mean"
    },
    "statisticalMeasure": "http://www.uncertml.org/statistics/standard-
deviation",
    "narrowerThan": ["http://vocab.nerc.ac.uk/standard_name/sea_surface_
temperature/"]
  },
  "unit" : {
    "label": {
      "en": "Kelvin"
    },
    "symbol": {
      "value": "K",
      "type": "http://www.opengis.net/def/uom/UCUM/"
    }
  }
```

```
}
```

Figure 9

11

5. REFERENCE SYSTEM OBJECTS

11 5. REFERENCE SYSTEM OBJECTS

Reference system objects are used to provide information about how to interpret coordinate values within the domain. Coordinates are usually geospatial or temporal in nature, but may also be categorical (based on identifiers). All reference system objects MUST have a member `"type"`, the possible values of which are given in the sections below. Custom values MAY be used as detailed in the "Extensions" section below.

11.1. 5.1. Geospatial Coordinate Reference Systems

Geospatial coordinate reference systems (CRSs) link coordinate values to the Earth.

11.1.1. 5.1.1 Geographic Coordinate Reference Systems

Geographic CRSs anchor coordinate values to an ellipsoidal approximation of the Earth. They have coordinate axes of geodetic longitude and geodetic latitude, and perhaps height above the ellipsoid (i.e. they can be two- or three-dimensional). The origin of the CRS is on the surface of the ellipsoid.

- The value of the `"type"` member MUST be "GeographicCRS"
- The object MAY have an `"id"` member, whose value MUST be a string and SHOULD be a common identifier for the reference system.
- The object MAY have a `"description"` member, where the value MUST be an i18n object, but no standardised content is interpreted from this description.

Note that sometimes (e.g. for numerical model data) the exact CRS may not be known or may be undefined. In this case the `"id"` may be omitted, but the `"type"` still indicates that this is a geographic CRS. Therefore clients can still use geodetic longitude, geodetic latitude (and maybe height) axes, even if they can't accurately georeference the information.

If a Coverage conforms to one of the defined [domain types][domain-types] then the coordinate identifier `"x"` is used to denote geodetic longitude, `"y"` is used for geodetic latitude and `z` for ellipsoidal height.

Example of a two-dimensional geographic CRS (longitude-latitude):

```
{
  "type": "GeographicCRS",
  "id": "http://www.opengis.net/def/crs/OGC/1.3/CRS84"
}
```

Figure 10

Example of a three-dimensional geographic CRS (latitude-longitude-height):

```
{
  "type": "GeographicCRS",
  "id": "http://www.opengis.net/def/crs/EPSG/0/4979"
}
```

Figure 11

11.1.2. 5.1.2 Projected Coordinate Reference Systems

Projected CRSs use two coordinates to denote positions on a Cartesian plane, which is derived from projecting the ellipsoid according to some defined transformation.

- The value of the `"type"` member MUST be "ProjectedCRS"
- The object MAY have an `"id"` member, whose value MUST be a string and SHOULD be a common identifier for the reference system.
- The object MAY have a `"description"` member, where the value MUST be an i18n object, but no standardised content is interpreted from this description.

If a Coverage conforms to one of the defined [domain types][domain-types] then the coordinate identifier `"x"` is used to denote easting and `"y"` is used for northing.

Example of a projected CRS using the British National Grid:

```
{
  "type": "ProjectedCRS",
  "id": "http://www.opengis.net/def/crs/EPSG/0/27700"
}
```

Figure 12

11.1.3. 5.1.3 Vertical Coordinate Reference Systems

Vertical CRSs use a single coordinate to denote some measure of height or depth, usually approximately oriented with gravity.

- The value of the `"type"` member MUST be "VerticalCRS"
- The object MAY have an `"id"` member, whose value MUST be a string and SHOULD be a common identifier for the reference system.
- The object MAY have a `"description"` member, where the value MUST be an i18n object, but no standardised content is interpreted from this description.

Example of a vertical CRS, here representing height above the NAV88 datum:

```
{
  "type": "VerticalCRS",
  "id": "http://www.opengis.net/def/crs/EPSG/0/5703"
```

```
}
```

Figure 13

11.2. 5.2. Temporal Reference Systems

Time is referenced by a temporal reference system (temporal RS). In the current specification, only a string-based notation for time values is defined. Future versions of this specification may allow for alternative notations, such as recording time values as numeric offsets from a given temporal datum (e.g. “days since 1970-01-01”).

- A temporal RS object MUST have a member `"type"`. The only currently defined value of it is `"TemporalRS"`.
- A temporal RS object MUST have a member `"calendar"` with value `"Gregorian"` or a URI.
- If the Gregorian calendar is used, then `"calendar"` MUST have the value `"Gregorian"` and cannot be a URI.
- A temporal RS object MAY have a member `"timeScale"` with a URI as value. If omitted, the time scale defaults to `"UTC"`. If the time scale is UTC, the `"timeScale"` member MUST be omitted.
- If the calendar is based on years, months, days, then the referenced values SHOULD use one of the following ISO8601-based lexical representations:
- YYYY
- ±XYYYY (where X stands for extra year digits)
- YYYY-MM
- YYYY-MM-DD
- YYYY-MM-DDTHH:MM:SS[.F]Z where Z is either “Z” or a time scale offset +|-HH:MM
- If calendar dates with reduced precision are used in a lexical representation (e.g. `"2016"`), then a client SHOULD interpret those dates in that reduced precision.
- If `"type"` is `"TemporalRS"` and `"calendar"` is `"Gregorian"`, then the above lexical representation MUST be used.

Example:

```
{
  "type": "TemporalRS",
  "calendar": "Gregorian"
```

```
}
```

Figure 14

11.3. 5.3. Identifier-based Reference Systems

Identifier-based reference systems (identifier RS) .

- An identifier RS object MUST have a member `"type"` with value `"IdentifierRS"`.
- An identifier RS object MAY have a member `"id"` where the value MUST be a string and SHOULD be a common identifier for the reference system.
- An identifier RS object MAY have a member `"label"` where the value MUST be an i18n object that is the name of the reference system.
- An identifier RS object MAY have a member `"description"` where the value MUST be an i18n object that is the (perhaps lengthy) description of the reference system.
- An identifier RS object MUST have a member `"targetConcept"` where the value is an object that MUST have a member `"label"` and MAY have a member `"description"` where the value of each MUST be an i18n object that is the name or description, respectively, of the concept which is referenced in the system.
- An identifier RS object MAY have a member `"identifiers"` where the value is an object where each key is an identifier referenced by the identifier RS and each value an object describing the referenced concept, equal to `"targetConcept"`.
- Coordinate values associated with an identifier RS MUST be strings.

Example of a geographic identifier reference system:

```
{
  "type": "IdentifierRS",
  "id": "https://en.wikipedia.org/wiki/ISO_3166-1_alpha-2",
  "label": { "en": "ISO 3166-1 alpha-2 codes" },
  "targetConcept": {
    "id": "http://dbpedia.org/resource/Country",
    "label": {"en": "Country", "de": "Land" }
  },
  "identifiers": {
    "de": {
      "id": "http://dbpedia.org/resource/Germany",
      "label": { "de": "Deutschland", "en": "Germany" }
    },
    "gb": {
      "id": "http://dbpedia.org/resource/United_Kingdom",
      "label": { "de": "Vereinigtes Königreich", "en": "United Kingdom" }
    }
  }
```

```
}
```

Figure 15

The domain values in the above example would be `"de"` and `"gb"`.

12

6. COVERAGEJSON OBJECTS

12 6. COVERAGEJSON OBJECTS

CoverageJSON documents always consist of a single object. This object (referred to as the CoverageJSON object below) represents a domain, range, coverage, or collection of coverages.

- The CoverageJSON object MAY have any number of members (name/value pairs).
- The CoverageJSON object MUST have a member with the name `"type"` whose value is one of: `"Domain"`, `"NdArray"` (a range encoding), `"TiledNdArray"` (a range encoding), `"Coverage"`, or `"CoverageCollection"`. The case of the type member values MUST be as shown here.

12.1. 6.1. Domain Objects

A domain object is a CoverageJSON object which defines a set of positions and their extent in one or more referencing systems. Its general structure is:

```
{
  "type": "Domain",
  "domainType": "...",
  "axes": { ... },
  "referencing": [...]
}
```

Figure 16

- The value of the `"type"` member MUST be `"Domain"`.
- For interoperability reasons it is RECOMMENDED that a domain object has the member `"domainType"` with a string value to indicate that the domain follows a certain structure (e.g. a time series, a vertical profile, a spatio-temporal 4D grid). See the ["Common CoverageJSON Domain Types Specification"][domain-types], which forms part of this specification, for details. Custom domain types may be used as recommended in the section "Extensions".
- A domain object MUST have the member `"axes"` which has as value an object where each key is an axis identifier and each value an axis object as defined below.
- The `"axes"` member MUST NOT be empty.
- A domain object MAY have the member `"referencing"` where the value is an array of reference system connection objects as defined below.
- A domain object MUST have a `"referencing"` member if the domain object is not part of a coverage collection or if the coverage collection does not have a `"referencing"` member.

12.1.1. 6.1.1. Axis Objects

- An axis object MUST have either a `"values"` member or, as a compact notation for a regularly spaced numeric axis, all the members `"start"`, `"stop"`, and `"num"`.
- The value of `"values"` is a non-empty array of axis values.
- The values of `"start"` and `"stop"` MUST be numbers, and the value of `"num"` an integer greater than zero. If the value of `"num"` is `1`, then `"start"` and `"stop"` MUST have identical values. For `num > 1`, the array elements of `"values"` MAY be reconstructed with the formula `start + i * step` where `i` is the ith element and in the interval `[0, num-1]` and `step = (stop - start) / (num - 1)`. If `num = 1` then `"values"` is `[start]`. Note that `"start"` can be greater than `"stop"` in which case the axis values are descending.
- The value of `"dataType"` determines the structure of an axis value and its coordinates that are made available for referencing. The values of `"dataType"` defined in this specification are `"primitive"`, `"tuple"`, and `"polygon"`. Custom values MAY be used as detailed in the “Extensions” section. For `"primitive"`, there is a single coordinate identifier and each axis value MUST be a number or string. For `"tuple"`, each axis value MUST be an array of fixed size of primitive values in a defined order, where the tuple size corresponds to the number of coordinate identifiers. For `"polygon"`, each axis value MUST be a GeoJSON Polygon coordinate array, where the order of coordinates is given by the `"coordinates"` array.
- If missing, the member `"dataType"` defaults to `"primitive"` and MUST not be included for that default case.
- If `"dataType"` is `"primitive"` and the associated reference system (see 6.1.2) defines a natural ordering of values then the array values in `"values"`, if existing, MUST be ordered monotonically, that is, increasing or decreasing.
- The value of `"coordinates"` is a non-empty array of coordinate identifiers corresponding to the order of the coordinates defined by `"dataType"`.
- If missing, the member `"coordinates"` defaults to a one-element array of the axis identifier and MUST NOT be included for that default case.
- A coordinate identifier SHALL NOT be defined more than once in all axis objects of a domain object.
- An axis object MAY have axis value bounds defined in the member `"bounds"` where the value is an array of values of length `len*2` with `len` being the length of the `"values"` array. For each axis value at array index `i` in the `"values"` array, a lower and upper bounding value at positions `2*i` and `2*i+1`, respectively, are given in the bounds array.
- If a domain axis object has no `"bounds"` member then a bounds array MAY be derived automatically.

Example of an axis object with bounds:

```
{
  "values": [20,21],
  "bounds": [19.5,20.5,
             20.5,21.5]
}
```

Figure 17

Example of an axis object with regular axis encoding:

```
{
  "start": 0,
  "stop": 5,
  "num": 6
}
```

Figure 18

The axis values in the above example are equal to `"values": [0,1,2,3,4,5]`.

Example of an axis object with tuple values:

```
{
  "dataType": "tuple",
  "coordinates": ["t","x","y"],
  "values": [
    ["2008-01-01T04:00:00Z",1,20],
    ["2008-01-01T04:30:00Z",2,21]
  ]
}
```

Figure 19

Example of an axis object with Polygon values:

```
{
  "dataType": "polygon",
  "coordinates": ["x","y"],
  "values": [
    [ [ [100.0, 0.0], [101.0, 0.0], [101.0, 1.0], [100.0, 1.0], [100.0, 0.0] ]
 ]
  ]
}
```

Figure 20

12.1.2. 6.1.2. Reference System Connection Objects

A reference system connection object creates a link between values within domain axes and a reference system to be able to interpret those values, e.g. as coordinates in a certain coordinate reference system.

- A reference system connection object MUST have a member `"coordinates"` which has as value an array of coordinate identifiers that are referenced in this object. Depending on the type of referencing, the ordering of the identifiers MAY be relevant, e.g. for 2D/3D coordinate reference systems. In this case, the order of the identifiers MUST match the order of axes in the coordinate reference system.

- A reference system connection object MUST have a member `"system"` whose value MUST be a Reference System Object (defined in section 5 above).

Example of a reference system connection object:

```
{
  "coordinates": ["y","x","z"],
  "system": {
    "type": "GeographicCRS",
    "id": "http://www.opengis.net/def/crs/EPSG/0/4979"
  }
}
```

Figure 21

12.1.3. 6.1.3. Examples

Example of a domain object with [`"Grid"`][domain-types] domain type:

```
{
  "type": "Domain",
  "domainType": "Grid",
  "axes": {
    "x": { "values": [1,2,3] },
    "y": { "values": [20,21] },
    "z": { "values": [1] },
    "t": { "values": ["2008-01-01T04:00:00Z"] }
  },
  "referencing": [{
    "coordinates": ["t"],
    "system": {
      "type": "TemporalRS",
      "calendar": "Gregorian"
    }
  }, {
    "coordinates": ["y","x","z"],
    "system": {
      "type": "GeographicCRS",
      "id": "http://www.opengis.net/def/crs/EPSG/0/4979"
    }
  }]
}
```

Figure 22

Example of a domain object with [`"Trajectory"`][domain-types] domain type:

```
{
  "type": "Domain",
  "domainType": "Trajectory",
  "axes": {
    "composite": {
      "dataType": "tuple",
      "coordinates": ["t","x","y"],
      "values": [
        ["2008-01-01T04:00:00Z", 1, 20],
        ["2008-01-01T04:30:00Z", 2, 21]
      ]
    }
```

```
  },
  "referencing": [{
    "coordinates": ["t"],
    "system": {
      "type": "TemporalRS",
      "calendar": "Gregorian"
    }
  }, {
    "coordinates": ["x","y"],
    "system": {
      "type": "GeographicCRS",
      "id": "http://www.opengis.net/def/crs/OGC/1.3/CRS84"
    }
  }]
}
```

Figure 23

12.2. 6.2. NdArray Objects

A CoverageJSON object with the type `"NdArray"` is an NdArray object. It represents a multidimensional (>= 0D) array with named axes, encoded as a flat one-dimensional array in row-major order.

- An NdArray object MUST have a member with the name `"values"` where the value is a non-empty array of numbers and nulls, or strings and nulls, where nulls represent missing data.
- Zero-dimensional NdArrays MUST have exactly one item in the `"values"` array.
- An NdArray object MUST have a member with the name `"dataType"` where the value is either `"float"`, `"integer"`, or `"string"` and MUST correspond to the data type of the non-null values in the `"values"` array.
- An NdArray object MAY have a member with the name `"shape"` where the value is an array of integers. For 0D arrays, `"shape"` MAY be omitted (defaulting to `[]`), for >= 1D arrays it MUST be included.
- An NdArray object MAY have a member with the name `"axisNames"` where the value is a string array of the same length as `"shape"`. For 0D arrays, `"axisNames"` MAY be omitted (defaulting to `[]`), for >= 1D arrays it MUST be included.
- Note that common JSON implementations use 64-bit floating point numbers as data type for `"values"`, therefore precision has to be taken into account. For example, only integers within the extent [-2^32, 2^32] can be accurately represented with 64-bit floating point numbers.

Example:

```
{
  "type": "NdArray",
  "dataType": "float",
```

```
  "shape": [4, 2],
  "axisNames": ["y", "x"],
  "values": [
    12.3, 12.5, 11.5, 23.1,
    null, null, 10.1, 9.1
  ]
}
```

Figure 24

12.3. 6.3. TiledNdArray Objects

A CoverageJSON object with the type `"TiledNdArray"` is a TiledNdArray object. It represents a multidimensional (>= 1D) array with named axes that is split up into sets of linked NdArray documents. Each tileset typically covers a specific data access scenario, for example, loading a single time slice of a grid vs. loading a time series of a spatial subset of a grid.

- A TiledNdArray object MUST have a member with the name `"dataType"` where the value is either `"float"`, `"integer"`, or `"string"`.
- A TiledNdArray object MUST have a member with the name `"shape"` where the value is a non-empty array of integers.
- A TiledNdArray object MUST have a member with the name `"axisNames"` where the value is a string array of the same length as `"shape"`.
- A TiledNdArray object MUST have a member with the name `"tileSets"` where the value is a non-empty array of TileSet objects.
- A TileSet object MUST have a member with the name `"tileShape"` where the value is an array of the same length as `"shape"` and where each array element is either null or an integer lower or equal than the corresponding element in `"shape"`. A null value denotes that the axis is not tiled.
- A TileSet object MUST have a member with the name `"urlTemplate"` where the value is a Level 1 URI template as defined in RFC 6570 . The URI template MUST contain a variable for each axis name whose corresponding element in `"tileShape"` is not null. A variable for an axis of total size `totalSize` (from `"shape"`) and tile size `tileSize` (from `"tileShape"`) has as value one of the integers `0, 1, …, q + r - 1` where `q` and `r` are the quotient and remainder obtained by dividing `totalSize` by `tileSize`. Each URI that can be generated from the URI template MUST resolve to an NdArray CoverageJSON document where the members `"dataType"` and `"axisNames`" are identical to the ones of the TiledNdArray object, and where each value of `"shape"` is an integer equal, or lower if an edge tile, to the corresponding element in `"tileShape"` while replacing null with the corresponding element of `"shape"` of the TiledNdArray.

Example:

```
{
  "type" : "TiledNdArray",
```

```
  "dataType": "integer",
  "axisNames": ["t", "y", "x"],
  "shape": [2, 5, 10],
  "tileSets": [{
    "tileShape": [null, null, null],
    "urlTemplate": "http://example.com/a/all.covjson"
  }, {
    "tileShape": [1, null, null],
    "urlTemplate": "http://example.com/b/{t}.covjson"
  }, {
    "tileShape": [null, 2, 3],
    "urlTemplate": "http://example.com/c/{y}-{x}.covjson"
  }]
}
```

Figure 25

http://example.com/a/all.covjson:

```
{
  "type": "NdArray",
  "dataType": "integer",
  "axisNames": ["t", "y", "x"],
  "shape": [2, 5, 10],
  "values": [
     1,  2,  3,  4,  5,  6,  7,  8,  9, 10,
    11, 12, 13, 14, 15, 16, 17, 18, 19, 20,
    21, 22, 23, 24, 25, 26, 27, 28, 29, 30,
    31, 32, 33, 34, 35, 36, 37, 38, 39, 40,
    41, 42, 43, 44, 45, 46, 47, 48, 49, 50,

    51, 52, 53, 54, 55, 56, 57, 58, 59, 60,
    61, 62, 63, 64, 65, 66, 67, 68, 69, 70,
    71, 72, 73, 74, 75, 76, 77, 78, 79, 80,
    81, 82, 83, 84, 85, 86, 87, 88, 89, 90,
    91, 92, 93, 94, 95, 96, 97, 98, 99, 100
  ]
}
```

Figure 26

http://example.com/b/0.covjson:

```
{
  "type": "NdArray",
  "dataType": "integer",
  "axisNames": ["t", "y", "x"],
  "shape": [1, 5, 10],
  "values": [
     1,  2,  3,  4,  5,  6,  7,  8,  9, 10,
    11, 12, 13, 14, 15, 16, 17, 18, 19, 20,
    21, 22, 23, 24, 25, 26, 27, 28, 29, 30,
    31, 32, 33, 34, 35, 36, 37, 38, 39, 40,
    41, 42, 43, 44, 45, 46, 47, 48, 49, 50
  ]
}
```

Figure 27

http://example.com/c/0-0.covjson:

```
{
  "type": "NdArray",
```

```
  "dataType": "integer",
  "axisNames": ["t", "y", "x"],
  "shape": [2, 2, 3],
  "values": [
     1,  2,  3,
    11, 12, 13,

    51, 52, 53,
    61, 62, 63
  ]
}
```

Figure 28

http://example.com/c/0-3.covjson:

```
{
  "type": "NdArray",
  "dataType": "integer",
  "axisNames": ["t", "y", "x"],
  "shape": [2, 2, 1],
  "values": [
    10,
    20,

    60,
    70
  ]
}
```

Figure 29

12.4. 6.4. Coverage Objects

A CoverageJSON object with the type `"Coverage"` is a coverage object.

- If a coverage has a commonly used identifier, that identifier SHOULD be included as a member of the coverage object with the name `"id"`.
- A coverage object MUST have a member with the name `"domain"` where the value is either a domain object or a URL.
- If the value of `"domain"` is a URL and the referenced domain has a `"domainType"` member, then the coverage object SHOULD have the member `"domainType"` where the value MUST equal that of the referenced domain.
- If the coverage object is part of a coverage collection which has a `"domainType"` member then that member SHOULD be omitted in the coverage object.
- A coverage object MAY have a member with the name `"parameters"` where the value is an object where each member has as name a short identifier and as value a parameter object. The identifier corresponds to the commonly known concept of “variable name” and is merely used in clients for conveniently accessing the corresponding range object.

- A coverage object MUST have a `"parameters"` member if the coverage object is not part of a coverage collection or if the coverage collection does not have a `"parameters"` member.
- A coverage object MAY have a member with the name `"parameterGroups"` where the value is an array of ParameterGroup objects.
- A coverage object MUST have a member with the name `"ranges"` where the value is a range set object. Any member of a range set object has as name any of the names in a `"parameters"` object in scope and as value either an NdArray or TiledNdArray object or a URL resolving to a CoverageJSON document of such object. A `"parameters"` member in scope is either within the enclosing coverage object or, if part of a coverage collection, in the parent coverage collection object. The shape and axis names of each NdArray or TiledNdArray object MUST correspond to the domain axes defined by `"domain"`, while single-valued axes MAY be omitted. If the referenced parameter object has a `"categoryEncoding"` member, then each non-null array element of the `"values"` member of the NdArray object, or the linked NdArray objects within a TiledNdArray object, MUST be equal to one of the values defined in the `"categoryEncoding"` object and be interpreted as the matching category.

Example:

See the Vertical Profile Coverage Example

12.5. 6.5. Coverage Collection Objects

A CoverageJSON object with the type `"CoverageCollection"` is a coverage collection object.

- A coverage collection object MAY have the member `"domainType"` with a string value to indicate that the coverage collection only contains coverages of the given domain type. See the [“Common CoverageJSON Domain Types Specification”][domain-types], which forms part of this specification, for details. Custom domain types may be used as recommended in the section “Extensions”.
- If a coverage collection object has the member `"domainType"`, then this member is inherited to all included coverages.
- A coverage collection object MUST have a member with the name `"coverages"`. The value corresponding to `"coverages"` is an array. Each element in the array is a coverage object as defined above.
- A coverage collection object MAY have a member with the name `"parameters"` where the value is an object where each member has as name a short identifier and as value a parameter object.
- A coverage collection object MAY have a member with the name `"parameterGroups"` where the value is an array of ParameterGroup objects.

- A coverage collection object MAY have a member with the name `"referencing"` where the value is an array of reference system connection objects.

Example:

See the Coverage Collection Example

13

7. EXTENSIONS

13 7. EXTENSIONS

A CoverageJSON document can be extended with custom members and types in a robust and interoperable way. For that, it makes use of absolute URIs and compact URIs (prefix:suffix) in order to avoid conflicts with other extensions and future versions of the format. A central registry of compact URI prefixes is provided which anyone can extend and which is a simple mapping from compact URI prefix to namespace URI in order to avoid collisions with other extensions that are based on compact URIs as well. Extensions that do not follow this approach MAY use simple names instead of absolute or compact URIs but have to accept the consequence of the document being less interoperable and future-proof. In certain use cases this is not an issue and may be a preferred solution for simplicity reasons, for example, if such CoverageJSON documents are only used internally and are not meant to be shared to a wider audience.

13.1. 7.1. Custom members

If a custom member is added to a CoverageJSON document, its name SHOULD be a compact URIs of the form `"prefix:suffix"`.

Example:

```
{
  "type" : "Coverage",
  "dct:license": "https://creativecommons.org/licenses/by/4.0/",
  ...
}
```

Figure 30

The prefix SHOULD be registered at https://covjson.org/prefixes/ which in the example above would be `dct = http://purl.org/dc/terms/`.

If the value of a custom member can have multiple structures, for example a string or an object, then a client should ignore the member if it does not understand the structure that is used.

Example of a different value structure:

```
{
  "type" : "Coverage",
  "dct:license": {
    "id": "https://creativecommons.org/licenses/by/4.0/",
    "label": {
      "en": "Creative Commons Attribution 4.0 International License"
    }
  },
  ...
}
```

Figure 31

13.2. 7.2. Custom types

Custom types MAY be used with the following members:

- `"domainType"` in domain objects
- `"dataType"` in axis objects
- `"type"` in reference system objects
- `"type"` in unit symbol objects
- `"type"` within custom members that have an object as value

The custom value of those members SHOULD be either an absolute URI or a compact URI. If a compact URI is used, then the prefix SHOULD be registered at https://covjson.org/prefixes/.

Example of a custom unit symbol type using an absolute URI:

```
{
  "type" : "Parameter",
  "unit" : {
    "symbol": {
      "value": "degreeC",
      "type": "http://www.opengis.net/def/uom/UDUNITS/"
    }
  },
  "observedProperty" : {
    "label" : {
      "en": "Air temperature"
    }
  }
}
```

Figure 32

Example of a custom reference system type using a compact URI:

```
{
  "type": "uor:HEALPixRS",
  "uor:h": 3,
  "uor:k": 3,
  "uor:ordering": "nested"
}
```

Figure 33

14

8. JSON-LD

8. JSON-LD

If no JSON-LD context is given, then the default context `https://covjson.org/context.jsonld` SHALL be assumed. Note that this context includes registered namespace prefixes and MAY be updated in a backwards-compatible way as the format evolves.

Additional semantics not provided by the default context MAY be provided by specifying an explicit `"@context"` member in the root of a CoverageJSON document. The value of that member MUST be an array where the first element is the default context URL. Any additional context definitions SHALL NOT override definitions of the default context, except when the definition is identical.

Providing an explicit context is especially useful for extensions. A recommended practice is to include any used namespace prefixes, even if registered, in the explicit context. This provides additional clarity and helps humans understand the document more quickly.

It is NOT RECOMMENDED to use the explicit JSON-LD context to map simple names, for example, `"license": "dct:license"`. On one side, this would hinder interoperability for generic non-JSON-LD clients, as they generally rely on absolute URIs or registered prefixes() of compact URIs. On the other side, it would make documents less future-proof as there may be name collisions with future versions of the format where semantics of that name may be defined differently. It is therefore RECOMMENDED to use compact or absolute URIs if an explicit JSON-LD context is included.

Note that domain axis values and range values SHOULD NOT be exposed as linked data via the JSON-LD context since they are not suitable for such representation.

Example:

```
{
  "@context": [
    "https://covjson.org/context.jsonld",
    {
      "dct": "http://purl.org/dc/terms/",
      "dct:license": { "@type": "@id" }
    }
  ],
  "type" : "Coverage",
  "dct:license": "https://creativecommons.org/licenses/by/4.0/",
   ...
}
```

Figure 34

In this example, additional semantics for the registered `dct` prefix are provided by stating that the `"dct:license"` member value in this document is an identifier and not just an unstructured string.

15

9. RESOLVING DOMAIN AND RANGE URLS

15

9. RESOLVING DOMAIN AND RANGE URLS

If a domain or range is referenced by a URL in a CoverageJSON document, then the client should, whenever is appropriate, load the data from the given URL and treat the loaded data as if it was directly embedded in place of the URL. When sending HTTP requests, the `Accept` header SHOULD be set appropriately to the CoverageJSON media type.

16

10. COMMON DOMAIN TYPES

16

10. COMMON DOMAIN TYPES

This specification defines the following domain types: Grid, VerticalProfile, PointSeries, Point, MultiPointSeries, MultiPoint, PolygonSeries, Polygon, MultiPolygonSeries, MultiPolygon, Trajectory, Section.

Requirements for all domain types defined in this specification:

- The axis and coordinate identifiers `"x"` and `"y"` MUST refer to horizontal spatial coordinates, `"z"` to vertical spatial coordinates, and all of `"x"`, `"y"`, and `"z"` MUST be referenced by a spatial coordinate reference system.
- The axis and coordinate identifier `"t"` MUST refer to temporal coordinates and be referenced by a temporal reference system.
- If a spatial CRS is used that has the axes longitude and latitude, or easting and northing, then the axis and coordinate identifier `"x"` MUST refer to longitude / easting, and `"y"` to latitude / northing.
- A domain that states conformance to one of the domain types in this specification MUST only contain axes defined by the domain type: additional axes are not allowed.
- In a Coverage object, the axis ordering in `"axisNames"` of NdArray objects SHOULD follow the order "t", "z", "y, "x", "composite", leaving out all axes that do not exist or are single-valued.

Table 1 — Domain Types table

DOMAIN TYPE	X	Y	Z	T	COMPOSITE
Grid	+	+	[+]	[+]	
VerticalProfile	1	1	+	[1]	
PointSeries	1	1	[1]	+	
Point	1	1	[1]	[1]	
MultiPointSeries				+	+
MultiPoint				[1]	+
PolygonSeries			[1]	+	1
Polygon			[1]	[1]	1

DOMAIN TYPE	X	Y	Z	T	COMPOSITE
MultiPolygonSeries			[1]	+	+
MultiPolygon			[1]	[1]	+
Trajectory			[1]		+
Section			+		+

Table 2 — Table Key

SYMBOL	DESCRIPTION
1	Axis with one coordinate
[1]	Optional axis with one coordinate
+	Axis with one or more coordinates
[+]	Optional axis with one or more coordinates

16.1. 10.1. Grid

- A domain with Grid domain type MUST have the axes `"x"` and `"y"` and MAY have the axes `"z"` and `"t"`.

Domain example:

```
{
  "type": "Domain",
  "domainType": "Grid",
  "axes": {
    "x": { "values": [1,2,3] },
    "y": { "values": [20,21] },
    "z": { "values": [1] },
    "t": { "values": ["2008-01-01T04:00:00Z"] }
  },
  "referencing": [...]
}
```

Figure 35

Coverage example:

```
{
```

```
  "type" : "Coverage",
  "domain" : {
    "type" : "Domain",
    "domainType" : "Grid",
    "axes": {
      "x": { "values": [1,2,3] },
      "y": { "values": [20,21] },
      "z": { "values": [1] },
      "t": { "values": ["2008-01-01T04:00:00Z"] }
    },
    "referencing": [...]
  },
  "parameters" : {
    "temperature": {...}
  },
  "ranges" : {
    "temperature" : {
      "type" : "NdArray",
      "dataType": "float",
      "axisNames": ["t", "z", "y", "x"],
      "shape": [1, 1, 2, 3],
      "values" : [...]
    }
  }
}
```

Figure 36

16.2. 10.2. VerticalProfile

- A domain with VerticalProfile domain type MUST have the axes `"x"`, `"y"`, and `"z"`, where `"x"` and `"y"` MUST have a single coordinate value only.
- A domain with VerticalProfile domain type MAY have the axis `"t"` which MUST have a single coordinate value only.

Domain example:

```
{
  "type": "Domain",
  "domainType": "VerticalProfile",
  "axes": {
    "x": { "values": [1] },
    "y": { "values": [21] },
    "z": { "values": [1,5,20] },
    "t": { "values": ["2008-01-01T04:00:00Z"] }
  },
  "referencing": [...]
}
```

Figure 37

Coverage example:

```
{
  "type" : "Coverage",
```

```
  "domain" : {
    "type": "Domain",
    "domainType": "VerticalProfile",
    "axes": {
      "x": { "values": [1] },
      "y": { "values": [21] },
      "z": { "values": [1,5,20] },
      "t": { "values": ["2008-01-01T04:00:00Z"] }
    },
    "referencing": [...]
  },
  "parameters" : {
    "temperature": {...}
  },
  "ranges" : {
    "temperature" : {
      "type" : "NdArray",
      "dataType": "float",
      "axisNames": ["z"],
      "shape": [3],
      "values" : [...]
    }
  }
}
```

Figure 38

16.3. 10.3. PointSeries

- A domain with PointSeries domain type MUST have the axes `"x"`, `"y"`, and `"t"` where `"x"` and `"y"` MUST have a single coordinate value only.
- A domain with PointSeries domain type MAY have the axis `"z"` which MUST have a single coordinate value only.

Domain example:

```
{
  "type": "Domain",
  "domainType": "PointSeries",
  "axes": {
    "x": { "values": [1] },
    "y": { "values": [20] },
    "z": { "values": [1] },
    "t": { "values": ["2008-01-01T04:00:00Z","2008-01-01T05:00:00Z"] }
  },
  "referencing": [...]
}
```

Figure 39

Coverage example:

```
{
  "type" : "Coverage",
  "domain" : {
```

```
    "type": "Domain",
    "domainType": "PointSeries",
    "axes": {
      "x": { "values": [1] },
      "y": { "values": [20] },
      "z": { "values": [1] },
      "t": { "values": ["2008-01-01T04:00:00Z","2008-01-01T05:00:00Z"] }
    },
    "referencing": [...]
  },
  "parameters" : {
    "temperature": {...}
  },
  "ranges" : {
    "temperature" : {
      "type" : "NdArray",
      "dataType": "float",
      "axisNames": ["t"],
      "shape": [2],
      "values" : [...]
    }
  }
}
```

Figure 40

16.4. 10.4. Point

- A domain with Point domain type MUST have the axes "x" and "y" and MAY have the axes "z" and "t" where all MUST have a single coordinate value only.

Domain example:

```
{
  "type": "Domain",
  "domainType": "Point",
  "axes": {
    "x": { "values": [1] },
    "y": { "values": [20] },
    "z": { "values": [1] },
    "t": { "values": ["2008-01-01T04:00:00Z"] }
  },
  "referencing": [...]
}
```

Figure 41

Coverage example:

```
{
  "type" : "Coverage",
  "domain" : {
    "type": "Domain",
    "domainType": "Point",
    "axes": {
      "x": { "values": [1] },
```

```
      "y": { "values": [20] },
      "z": { "values": [1] },
      "t": { "values": ["2008-01-01T04:00:00Z"] }
    },
    "referencing": [...]
  },
  "parameters" : {
    "temperature": {...}
  },
  "ranges" : {
    "temperature" : {
      "type" : "NdArray",
      "dataType": "float",
      "values" : [...]
    }
  }
}
```

Figure 42

16.5. 10.5. MultiPointSeries

- A domain with MultiPointSeries domain type MUST have the axes `"composite"` and `"t"`.
- The axis `"composite"` MUST have the data type `"tuple"` and the coordinate identifiers `"x","y","z"` or `"x","y"`, in that order.

Domain example:

```
{
  "type": "Domain",
  "domainType": "MultiPointSeries",
  "axes": {
    "t": { "values": ["2008-01-01T04:00:00Z", "2008-01-01T05:00:00Z"] },
    "composite": {
      "dataType": "tuple",
      "coordinates": ["x","y","z"],
      "values": [
        [1, 20, 1],
        [2, 21, 3]
      ]
    }
  },
  "referencing": [...]
}
```

Figure 43

Domain example without z:

```
{
  "type": "Domain",
  "domainType": "MultiPointSeries",
  "axes": {
    "t": { "values": ["2008-01-01T04:00:00Z", "2008-01-01T05:00:00Z"] },
    "composite": {
```

```
      "dataType": "tuple",
      "coordinates": ["x","y"],
      "values": [
        [1, 20],
        [2, 21]
      ]
    }
  },
  "referencing": [...]
}
```

Figure 44

Coverage example:

```
{
  "type" : "Coverage",
  "domain" : {
    "type": "Domain",
    "domainType": "MultiPointSeries",
    "axes": {
      "t": { "values": ["2008-01-01T04:00:00Z", "2008-01-01T05:00:00Z"] },
      "composite": {
        "dataType": "tuple",
        "coordinates": ["x","y","z"],
        "values": [
          [1, 20, 1],
          [2, 21, 3],
          [2, 20, 4]
        ]
      }
    }
  },
  "parameters" : {
    "temperature": {...}
  },
  "ranges" : {
    "temperature" : {
      "type" : "NdArray",
      "dataType": "float",
      "axisNames": ["t", "composite"],
      "shape": [2, 3],
      "values" : [...]
    }
  }
}
```

Figure 45

16.6. 10.6. MultiPoint

- A domain with MultiPoint domain type MUST have the axis `"composite"` and MAY have the axis `"t"` where `"t"` MUST have a single coordinate value only.
- The axis `"composite"` MUST have the data type `"tuple"` and the coordinate identifiers `"x","y","z"` or `"x","y"`, in that order.

Domain example:

```
{
  "type": "Domain",
  "domainType": "MultiPoint",
  "axes": {
    "t": { "values": ["2008-01-01T04:00:00Z"] },
    "composite": {
      "dataType": "tuple",
      "coordinates": ["x","y","z"],
      "values": [
        [1, 20, 1],
        [2, 21, 3]
      ]
    }
  },
  "referencing": [...]
}
```

Figure 46

Domain example without z and t:

```
{
  "type": "Domain",
  "domainType": "MultiPoint",
  "axes": {
    "composite": {
      "dataType": "tuple",
      "coordinates": ["x","y"],
      "values": [
        [1, 20],
        [2, 21]
      ]
    }
  },
  "referencing": [...]
}
```

Figure 47

Coverage example:

```
{
  "type" : "Coverage",
  "domain" : {
    "type": "Domain",
    "domainType": "MultiPoint",
    "axes": {
      "t": { "values": ["2008-01-01T04:00:00Z"] },
      "composite": {
        "dataType": "tuple",
        "coordinates": ["x","y","z"],
        "values": [
          [1, 20, 1],
          [2, 21, 3]
        ]
      }
    }
  },
  "parameters" : {
    "temperature": {...}
```

```
  },
  "ranges" : {
    "temperature" : {
      "type" : "NdArray",
      "dataType": "float",
      "axisNames": ["composite"],
      "shape": [2],
      "values" : [...]
    }
  }
}
```

Figure 48

16.7. 10.7. Trajectory

- A domain with Trajectory domain type MUST have the axis `"composite"` and MAY have the axis `"z"` where `"z"` MUST have a single coordinate value only.
- The axis `"composite"` MUST have the data type `"tuple"` and the coordinate identifiers `"t","x","y","z"` or `"t","x","y"`, in that order.
- The value ordering of the axis `"composite"` MUST follow the ordering of its `"t"` coordinate as defined in the corresponding reference system.

Domain example:

```
{
  "type": "Domain",
  "domainType": "Trajectory",
  "axes": {
    "composite": {
      "dataType": "tuple",
      "coordinates": ["t","x","y","z"],
      "values": [
        ["2008-01-01T04:00:00Z", 1, 20, 1],
        ["2008-01-01T04:30:00Z", 2, 21, 3]
      ]
    }
  },
  "referencing": [...]
}
```

Figure 49

Domain example without z:

```
{
  "type": "Domain",
  "domainType": "Trajectory",
  "axes": {
    "composite": {
      "dataType": "tuple",
      "coordinates": ["t","x","y"],
      "values": [
```

```
        ["2008-01-01T04:00:00Z", 1, 20],
        ["2008-01-01T04:30:00Z", 2, 21]
      ]
    }
  },
  "referencing": [...]
}
```

Figure 50

Domain example with z defined as constant value:

```
{
  "type": "Domain",
  "domainType": "Trajectory",
  "axes": {
    "composite": {
      "dataType": "tuple",
      "coordinates": ["t","x","y"],
      "values": [
        ["2008-01-01T04:00:00Z", 1, 20],
        ["2008-01-01T04:30:00Z", 2, 21]
      ]
    },
    "z": { "values": [5] }
  },
  "referencing": [...]
}
```

Figure 51

Coverage example:

```
{
  "type" : "Coverage",
  "domain" : {
    "type": "Domain",
    "domainType": "Trajectory",
    "axes": {
      "composite": {
        "dataType": "tuple",
        "coordinates": ["t","x","y","z"],
        "values": [
          ["2008-01-01T04:00:00Z", 1, 20, 1],
          ["2008-01-01T04:30:00Z", 2, 21, 3]
        ]
      }
    },
    "referencing": [...]
  },
  "parameters" : {
    "temperature": {...}
  },
  "ranges" : {
    "temperature" : {
      "type" : "NdArray",
      "dataType": "float",
      "axisNames": ["composite"],
      "shape": [2],
      "values" : [...]
    }
  }
```

```
}
```

Figure 52

16.8. 10.8. Section

- A domain with Section domain type MUST have the axes `"composite"` and `"z"`.
- The axis `"composite"` MUST have the data type `"tuple"` and the coordinate identifiers `"t"`,`"x"`,`"y"`, in that order.
- The value ordering of the axis `"composite"` MUST follow the ordering of its `"t"` coordinate as defined in the corresponding reference system.

Domain example:

```
{
  "type": "Domain",
  "domainType": "Section",
  "axes": {
    "z": { "values": [10,20,30] },
    "composite": {
      "dataType": "tuple",
      "coordinates": ["t","x","y"],
      "values": [
        ["2008-01-01T04:00:00Z", 1, 20],
        ["2008-01-01T04:30:00Z", 2, 21]
      ]
    }
  },
  "referencing": [...]
}
```

Figure 53

Coverage example:

```
{
  "type" : "Coverage",
  "domain" : {
    "type": "Domain",
    "domainType": "Section",
    "axes": {
      "z": { "values": [10,20,30] },
      "composite": {
        "dataType": "tuple",
        "coordinates": ["t","x","y"],
        "values": [
          ["2008-01-01T04:00:00Z", 1, 20],
          ["2008-01-01T04:30:00Z", 2, 21]
        ]
      }
    },
    "referencing": [...]
  },
  "parameters" : {
```

```
    "temperature": {...}
  },
  "ranges" : {
    "temperature" : {
      "type" : "NdArray",
      "dataType": "float",
      "axisNames": ["z", "composite"],
      "shape": [3, 2],
      "values" : [...]
    }
  }
}
```

Figure 54

16.9. 10.9. Polygon

Polygons in this domain domain type are defined equally to GeoJSON, except that they can only contain `[x,y]` positions (and not `z` or additional coordinates): - A LinearRing is an array of 4 or more `[x,y]` arrays where each of `x` and `y` is a coordinate value. The first and last `[x,y]` elements are identical. - A Polygon is an array of LinearRing arrays. For Polygons with multiple rings, the first MUST be the exterior ring and any others MUST be interior rings or holes.

- A domain with Polygon domain type MUST have the axis `"composite"` which has a single Polygon value.
- The axis `"composite"` MUST have the data type `"polygon"` and the coordinate identifiers `"x","y"`, in that order.
- A Polygon domain MAY have the axes `"z"` and `"t"` which both MUST have a single coordinate value only.

Domain example:

```
{
  "type": "Domain",
  "domainType": "Polygon",
  "axes": {
    "composite": {
      "dataType": "polygon",
      "coordinates": ["x","y"],
      "values": [
        [ [ [100.0, 0.0], [101.0, 0.0], [101.0, 1.0], [100.0, 1.0], [100.0, 0.
0] ]  ]
      ]
    },
    "z": { "values": [2] },
    "t": { "values": ["2008-01-01T04:00:00Z"] }
  },
  "referencing": [...]
}
```

Figure 55

Coverage example:

```
{
  "type" : "Coverage",
  "domain" : {
    "type": "Domain",
    "domainType": "Polygon",
    "axes": {
      "composite": {
        "dataType": "polygon",
        "coordinates": ["x","y"],
        "values": [
          [ [ [100.0, 0.0], [101.0, 0.0], [101.0, 1.0], [100.0, 1.0], [100.0,
0.0] ]  ]
        ]
      },
      "z": { "values": [2] },
      "t": { "values": ["2008-01-01T04:00:00Z"] }
    },
    "referencing": [...]
  },
  "parameters" : {
    "temperature": {...}
  },
  "ranges" : {
    "temperature" : {
      "type" : "NdArray",
      "dataType": "float",
      "values" : [...]
    }
  }
}
```

Figure 56

16.10. 10.10. PolygonSeries

- A domain with PolygonSeries domain type MUST have the axes `"composite"` and `"t"` where `"composite"` MUST have a single Polygon value. Polygons are defined in the Polygon domain type.
- A domain with PolygonSeries domain type MAY have the axis `"z"` which MUST have a single coordinate value only.
- The axis `"composite"` MUST have the data type `"polygon"` and the coordinate identifiers `"x","y"`, in that order.

Domain example:

```
{
  "type": "Domain",
  "domainType": "PolygonSeries",
  "axes": {
    "composite": {
      "dataType": "polygon",
```

```
        "coordinates": ["x","y"],
        "values": [
          [ [ [100.0, 0.0], [101.0, 0.0], [101.0, 1.0], [100.0, 1.0], [100.0, 0.
0] ]  ]
        ]
      },
      "z": { "values": [2] },
      "t": { "values": ["2008-01-01T04:00:00Z","2008-01-01T05:00:00Z"] }
    },
    "referencing": [...]
}
```

Figure 57

Coverage example:

```
{
  "type" : "Coverage",
  "domain" : {
    "type": "Domain",
    "domainType": "PolygonSeries",
    "axes": {
      "composite": {
        "dataType": "polygon",
        "coordinates": ["x","y"],
        "values": [
          [ [ [100.0, 0.0], [101.0, 0.0], [101.0, 1.0], [100.0, 1.0], [100.0,
0.0] ]  ]
        ]
      },
      "z": { "values": [2] },
      "t": { "values": ["2008-01-01T04:00:00Z","2008-01-01T05:00:00Z"] }
    },
    "referencing": [...]
  },
  "parameters" : {
    "temperature": {...}
  },
  "ranges" : {
    "temperature" : {
      "type" : "NdArray",
      "dataType": "float",
      "axisNames": ["t"],
      "shape": [2],
      "values" : [...]
    }
  }
}
```

Figure 58

16.11. 10.11. MultiPolygon

- A domain with MultiPolygon domain type MUST have the axis `"composite"` where the values are Polygons. Polygons are defined in the Polygon domain type.

- The axis `"composite"` MUST have the data type `"polygon"` and the coordinate identifiers `"x"`, `"y"`, in that order.
- A MultiPolygon domain MAY have the axes `"z"` and `"t"` which both MUST have a single coordinate value only.

Domain example:

```
{
  "type": "Domain",
  "domainType": "MultiPolygon",
  "axes": {
    "composite": {
      "dataType": "polygon",
      "coordinates": ["x","y"],
      "values": [
        [ [ [100.0, 0.0], [101.0, 0.0], [101.0, 1.0], [100.0, 1.0], [100.0, 0.
0] ]  ],
        [ [ [200.0, 10.0], [201.0, 10.0], [201.0, 11.0], [200.0, 11.0], [200.0,
 10.0] ] ]
      ]
    },
    "z": { "values": [2] },
    "t": { "values": ["2008-01-01T04:00:00Z"] }
  },
  "referencing": [...]
}
```

Figure 59

Coverage example:

```
{
  "type" : "Coverage",
  "domain" : {
    "type": "Domain",
    "domainType": "MultiPolygon",
    "axes": {
      "composite": {
        "dataType": "polygon",
        "coordinates": ["x","y"],
        "values": [
          [ [ [100.0, 0.0], [101.0, 0.0], [101.0, 1.0], [100.0, 1.0], [100.0,
0.0] ]  ],
          [ [ [200.0, 10.0], [201.0, 10.0], [201.0, 11.0], [200.0, 11.0], [200.
0, 10.0] ] ]
        ]
      },
      "z": { "values": [2] },
      "t": { "values": ["2008-01-01T04:00:00Z"] }
    },
    "referencing": [...]
  },
  "parameters" : {
    "temperature": {...}
  },
  "ranges" : {
    "temperature" : {
      "type" : "NdArray",
      "dataType": "float",
      "axisNames": ["composite"],
```

```
        "shape": [2],
        "values" : [...]
      }
    }
}
```

Figure 60

16.12. 10.12. MultiPolygonSeries

- A domain with MultiPolygonSeries domain type MUST have the axes `"composite"` and `"t"` where the values of `"composite"` are Polygons. Polygons are defined in the Polygon domain type.
- The axis `"composite"` MUST have the data type `"polygon"` and the coordinate identifiers `"x"`,`"y"`, in that order.
- A MultiPolygon domain MAY have the axis `"z"` which MUST have a single coordinate value only.

Domain example:

```
{
  "type": "Domain",
  "domainType": "MultiPolygonSeries",
  "axes": {
    "composite": {
      "dataType": "polygon",
      "coordinates": ["x","y"],
      "values": [
        [ [ [100.0, 0.0], [101.0, 0.0], [101.0, 1.0], [100.0, 1.0], [100.0, 0.
0] ]  ],
        [ [ [200.0, 10.0], [201.0, 10.0], [201.0, 11.0], [200.0, 11.0], [200.0,
 10.0] ] ]
      ]
    },
    "z": { "values": [2] },
    "t": { "values": ["2008-01-01T04:00:00Z", "2010-01-01T00:00:00Z"] }
  },
  "referencing": [...]
}
```

Figure 61

Coverage example:

```
{
  "type" : "Coverage",
  "domain" : {
    "type": "Domain",
    "domainType": "MultiPolygonSeries",
    "axes": {
      "composite": {
        "dataType": "polygon",
        "coordinates": ["x","y"],
```

```
        "values": [
          [ [ [100.0, 0.0], [101.0, 0.0], [101.0, 1.0], [100.0, 1.0], [100.0,
0.0] ]  ],
          [ [ [200.0, 10.0], [201.0, 10.0], [201.0, 11.0], [200.0, 11.0], [200.
0, 10.0] ] ]
        ]
      },
      "z": { "values": [2] },
      "t": { "values": ["2008-01-01T04:00:00Z", "2010-01-01T00:00:00Z", "2012-
01-01T00:00:00Z"] }
    },
    "referencing": [...]
  },
  "parameters" : {
    "temperature": {...}
  },
  "ranges" : {
    "temperature" : {
      "type" : "NdArray",
      "dataType": "float",
      "axisNames": ["t", "composite"],
      "shape": [3, 2],
      "values" : [...]
    }
  }
}
```

Figure 62

17

MEDIA TYPE AND FILE EXTENSION

17 MEDIA TYPE AND FILE EXTENSION

The CoverageJSON media type SHALL be `application/prs.coverage+json` with an optional parameter `profile` which is a non-empty list of space-separated URIs identifying specific constraints or conventions that apply to a CoverageJSON document according to RFC6906.

The only profile URI defined in this document is `https://covjson.org/def/core#standalone` which asserts that all domain and range objects are directly embedded in a CoverageJSON document and not referenced by URLs. There is no `charset` parameter and CoverageJSON documents MUST be serialized using the UTF-8 character encoding.

The file extension SHALL be `covjson`.