OGC® DOCUMENT: 18-067R4

External identifier of this OGC® document: http://www.opengis.net/doc/IS/symbology-1/2.0

Open Geospatial Consortium

OGC STYLES AND SYMBOLOGY MODEL AND ENCODINGS - PART 1: CORE

STANDARD
Implementation

CANDIDATE SWG DRAFT

Version: 2.0
Submission Date: 2029-01-01
Approval Date: 2029-01-01
Publication Date: 2029-01-01
Editor: Erwan Bocher, Olivier Ertz, Jerome Jacovella-St-Louis, Maxime Collombin

CONTENTS

LIST OF TABLES

LIST OF FIGURES

LIST OF RECOMMENDATIONS

CONTENTS

I ABSTRACT

This *OGC Styles and Symbology* Standard defines a Conceptual Model, a Logical Model and Encodings for describing symbology rules for the portrayal of geographical data. The targets of this Standard are symbology encodings and cartographic rendering engines.

The Standard is modularized into multiple requirements classes, with a minimal core describing an extensible framework, with clear extension points, for defining styles consisting of styling rules selected through expressions and applying symbolizers configured using properties.

The Standard defines a number of additional conformance classes covering a large number of essential portrayal use cases for symbolizing both vector data and coverage data.

Finally, the Standard defines two encodings for the logical model: one based on JSON which can be readily parsed by JSON parsers, as well as a more expressive encoding better suited for hand-editing inspired from Web Cartographic Styled Sheets (CSS) and related cartographic symbology encodings.

Additional encodings can be defined that can conform to the logical model, and existing encodings can be mapped fully or partially to the conceptual model defined in this Standard.

The granularity of the requirements classes facilitates conformance for encodings and rendering engines based on their supported capabilities.

Future parts may extend this Standard by defining additional symbolizer properties, system identifiers, or new types of expressions.

II KEYWORDS

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

ogcdoc, OGC document, style, symbology, conceptual, core, modular, neutral, portrayal

III PREFACE

Preface: This Standard is intended as a successor to the Symbology Encoding standard (SE 1.1) and the OGC Symbology Core Conceptual Model ("SymCore" 1.0). This Standard focuses primarily on describing symbology rules to be applied by a rendering engine to symbolize geospatial data. It also features a rich and extensible selector expression system which determines whether a particular symbology rule is to be applied or not by the rendering engine. In particular, it introduces a "system identifier" which can be mapped to concepts such as a particular layer, feature type or collection. Such concepts are used in OGC Web Services (including the Web Mapping Service (WMS) Style Layer Descriptor profile (SLD)), Web API Standards such as *OGC API — Maps* to render the data, and *OGC API — Styles* to provide styles, as well as data stores such as OGC GeoPackage (in which styles could also be stored using a portrayal extension). This Standard therefore follows the same motivation that split up SLD 1.0 (SLD 1.1 and SE 1.1) and defines requirements are not specific to any service (e.g., Web Map Service), are independent, and allow the concepts to be reused by other Standards willing to address aspects related to cartography. A general and portable symbology model is defined for use across the broad OGC Standards baseline, to be applied to geospatial datasets as well as online geospatial data and mapping services.
An important goal of this Standard is to enable richer symbology capabilities. This can be achieved through modularity which comes with extensibility. SE 1.1 is not modular per se, while this Standard is designed to be so by defining multiple requirements classes and the ability to conform with them at either the conceptual level or at the logical model. It also defines requirements classes for two encodings conforming to both the logical model and the conceptual model. The core coformance class covers the basic mechanics of defining a Style with StylingRules applying a Symbolizer, as well as how Expressions can be used to select whether a rule should be applied or not (Selectors). This provides two extension mechanisms that can be used by the additional requirements classes defined in this Standard as well by those introduced in future extension parts: new properties can be added to the Symbolizer class while defining their associated behavior, while new types of Expressions or System Identifiers can also be defined to be used within a Selector Expression or a ParameterValue Expression.

In contrast to the OGC Symbology Core Conceptual Model version 1.0, this version is clarified to not only be defining a conceptual model, but also a specific logical model. This newer version also defines a conceptual and logical model for expressions, which provides one specific mechanism for extension. This Standard clarifies that requirements classes extending this Standard can define additional properties for the Symbolizer class, providing the second extension mechanism. This Standard also includes symbology requirements classes covering the most common use cases for portraying both vector feature data as well as coverage data. Finally, this Standard defines two encodings conforming to the logical model. The more concrete nature of this standard better reflects the fact that it is an OGC Implementation Standard, and recognizes that for rendering engines to implement this Standard and for new or existing encodings to be mapped to the conceptual model of this Standard, the Standard needs to at least cover the essential portrayal use cases. This is demonstrated in a map gallery annex illustrating the various requirements classes coming into play for different use cases of varying complexity. The included encodings for the logical model readily allow to express all of the

symbology capabilities defined in this Standard and should greatly facilitate interoperability and encourage implementation.

In summary, this Standard offers a consistent approach to:

- provide the flexibility required to achieve adequate symbology rules for a variety of information communities; e.g., aviation symbols, weather symbols, thematic maps, etc.; and
- achieve high-level styling interoperability without encoding dependencies, by allowing to define multiple encodings of the same logical model, as well as providing a framework to map the logical models used by other encodings to the one defined in this Standard through the conceptual model that this Standard also defines.

Potential implementations of SymCore are expected to enhance OGC Standards such as the OGC API family of Standards, GeoPackage, the Web Map Service (WMS), Web Feature Service (WFS), and others. By sharing a common core, and using an extension mechanism, integration of these Standards for the purposes of cartographic representation can be greatly simplified.

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 SECURITY CONSIDERATIONS

No security considerations have been made for this document.

V SUBMITTING ORGANIZATIONS

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

- HEIG-VD (School of Management and Engineering Vaud)
- CNRS (National Center for Scientific Research)
- Strategic ACI
- IGN
- Ecere

VI SUBMITTERS

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

Name	Affiliation
Olivier Ertz (editor)	Haute École d'Ingénierie et de Gestion du Canton de Vaud (HEIG-VD)
Erwan Bocher (editor)	Laboratoire des Sciences et Techniques de l'information de la Communication et de la Connaissance / Centre national de la recherche scientifique (Lab-STICC CNRS)
Jérôme Jacovella-St-Louis (editor)	Ecere Corporation
Maxime Collombin (editor)	Haute École d'Ingénierie et de Gestion du Canton de Vaud (HEIG-VD)

1

SCOPE

1 SCOPE

This Standard presents the requirements that define version 2.0 of the *Styles and Symbology Model and Encodings – Part 1: Core* (*"SymCore 2.0"*). A Model for the portrayal of geographic data is defined at both the conceptual and logical levels, in a modular manner through the use of separate requirements classes and specific extension points. The requirements classes defined by this first part cover the most common use cases for portraying geospatial vector features and coverages. Several scenarios of varying complexity and the associated requirements classes enabling them are illustrated in an annex. Two encodings are defined by this Standard: one based on JSON which can be readily parsed with a JSON parser, and another more expressive encoding inspired from Web Cascading Style Sheets (CSS) better suited for hand-editing style sheets. Future parts to this Standard may further extend the portrayal capabilities by defining additional requirements classes.

The *Styles and Symbology Model* is a new approach (Bocher E., Ertz O., 2018, see bibliography):

- to provide the flexibility required to achieve adequate cartographic styling and fill the needs of a variety of information communities; e.g., aviation symbols, weather symbols, thematic maps, etc.; and
- to achieve high level styling interoperability without encoding dependencies.

2

CONFORMANCE

2 CONFORMANCE

The OGC Styles and Symbology Model and Encodings defines multiple requirements classes and their associated conformance classes.

The standardization target for all conformance classes except the conformance classes defining encodings are both Encodings and Rendering engines. The standardization target for encodings are Rendering Engines and other applications accepting styles as input, as well as applications generating styles (e.g., style editors or converters).

The conformance of all Rendering engines with all conformance classes except the conformance classes defining encodings is with the conceptual model that will result in the expected visual output. The conformance of all Encodings with the conformance classes targeting them can be either at a conceptual level or at a logical level. Conformance at the logical level implies conformance at the conceptual level as well. An Encoding conforms to the logical model if a Style can be encoded using the classes and property names defined in this Standard with only a simple consistent set of rules mapping them to the encoding. An Encoding can conform to the conceptual model of any conformance class either partially (if it can only preserve some of the information) or fully (if it losslessly preserves all information).

The goal is to allow different encodings to have equivalent content and semantics so that they can be interoperable, by establishing a clear mapping from an encoding to the logical and/or conceptual model defined in this Standard. The granularity of conformance classes provides a mechanism to clearly indicate which portrayal capabilities are interoperable with other encodings or rendering engines implementing this Standard. Most Encodings and Rendering engines are expected to easily conform to the requirements classes for basic portrayal.

This document establishes a core requirements class with a URI of http://www.opengis.net/spec/symbology-1/2.0/req/core.

Requirements and conformance test URIs defined in this document are relative to http://www.opengis.net/spec/symbology-1/2.0.

2.1. Requirements classes for basic portrayal

Requirements Class "Core" (http://www.opengis.net/spec/symbology-1/2.0/req/core)

The Core Requirements Class specifies requirements that all encodings and rendering engines must support if claiming conformance with this Standard. The Core class defines how a Style is defined as a list of StylingRules consisting of a Symbolizer and an optional Selector expression, as well as a minimal set Symbolizer properties applicable to all types of geospatial data.

Requirements Class "Basic Vector Features Styling" (http://www.opengis.net/spec/symbology-1/2.0/req/vector)

The Basic Vector Features Styling Requirements Class specifies requirements for styling vector features data, including referencing feature properties, specifying a fill and stroke to style vector geometry, as well as defining image, text or dot markers to symbolize points.

Requirements Class "Basic Coverage Styling" (http://www.opengis.net/spec/symbology-1/2.0/req/coverage)

The Basic Coverage Styling Requirements Class specifies requirements for styling coverage data, including referencing coverage range values (e.g., bands), mapping them to color channels as well as defining colormaps.

Requirements Class "Basic Labeling" (http://www.opengis.net/spec/symbology-1/2.0/req/labels)

The Basic Labeling Requirements Class specifies requirements for defining labels to be managed by a placement engine and rendered as a final pass on top of the map.

2.2. Requirements classes for common portrayal capabilities

Requirements Class "Hill Shading" (http://www.opengis.net/spec/symbology-1/2.0/req/hillshading)

The Hill Shading Requirements Class specifies requirements for styling elevation coverage data with a hill-shaded style.

Requirements Class "Font Outlines" (http://www.opengis.net/spec/symbology-1/2.0/req/fontoutlines)

The Font Outlines Requirements Class specifies requirements for displaying fonts with outlines so that text can be legible on any background.

Requirements Class "Hatches, Gradients and Stipples" (http://www.opengis.net/spec/symbology-1/2.0/req/hatchesgradientsstipples)

The Hatches, Gradients and Stipples Requirements Class specifies requirements for defining fils using hatches, gradients or stipples

Requirements Class "Shape Graphics" (http://www.opengis.net/spec/symbology-1/2.0/req/shapegraphics)

The Shape Graphics Requirements Class specifies requirements for defining geometric shapes which can be used for markers, labels (with Basic Labeling), or as patterns for stroke (with Pattern Strokes) or for fills (with Pattern Fills).

Requirements Class "Dashes" (http://www.opengis.net/spec/symbology-1/2.0/req/dashes)

The Dashes Requirements Class specifies requirements for defining dashed strokes.

Requirements Class "Casing and Centerline" (http://www.opengis.net/spec/symbology-1/2.0/req/casing)

The Casing and Centerline Requirements Class specifies requirements for defining strokes with built-in casing and centerline.

2.3. Requirements classes for additional expression capabilities

Requirements Class "Parameter Values" (http://www.opengis.net/spec/symbology-1/2.0/req/parametervalues)

The Parameter Values Requirements Class specifies requirements for using an Expression to define any property of a Symbolizer.

Requirements Class "Right-hand Identifiers" (http://www.opengis.net/spec/symbology-1/2.0/req/righthand)

The Right-hand Identifiers Requirements Class specifies requirements for defining operation expressions with identifiers on the right-hand side.

Requirements Class "Conditional Expressions" (http://www.opengis.net/spec/symbology-1/2.0/req/conditional)

The Conditional Expressions Requirements Class specifies requirements for defining conditional operation using a ternary `[if] ? [then] : [else]` structure.

Requirements Class "Variables" (http://www.opengis.net/spec/symbology-1/2.0/req/variables)

The Variables Requirements Class specifies requirements for defining variables which can be mapped to a dynamic value controlled by an application, or used to facilitate customizing a style.

Requirements Class "Arithmetic Operators" (http://www.opengis.net/spec/symbology-1/2.0/req/arithmetic)

The Arithmetic Operators Requirements Class specifies requirements for defining arithmetic operation expressions.

Requirements Class "Bitwise Operators" (http://www.opengis.net/spec/symbology-1/2.0/req/bitwise)

The Bitwise Operators Requirements Class specifies requirements for defining bitwise operation expressions.

Requirements Class "Text Relation Operators" (http://www.opengis.net/spec/symbology-1/2.0/req/textrelation)

The Text Relation Operators Requirements Class specifies requirements for defining text relation operation expressions.

Requirements Class "Function Expressions" (http://www.opengis.net/spec/symbology-1/2.0/req/functions)

The Functions Requirements Class specifies requirements for defining function call expressions.

Requirements Class "Spatial Relation Functions" (http://www.opengis.net/spec/symbology-1/2.0/req/spatial)

The Spatial Relation Requirements Class specifies requirements for standardized spatial relation functions to be used with feature geometry.

Requirements Class "Temporal Relation Functions" (http://www.opengis.net/spec/symbology-1/2.0/req/temporal)

The Temporal Relation Requirements Class specifies requirements for standardized temporal relation functions to be used with temporal properties.

Requirements Class "Array Relation Functions" (http://www.opengis.net/spec/symbology-1/2.0/req/arrayrelations)

The Array Relation Requirements Class specifies requirements for standardized array relation functions to be used with array properties.

Requirements Class "Text Manipulation Functions" (http://www.opengis.net/spec/symbology-1/2.0/req/textmanipulation)

The Text Manipulation Requirements Class specifies requirements for standardized text manipulation functions to be used with text properties.

Requirements Class "Geometry Manipulation Functions" (http://www.opengis.net/spec/symbology-1/2.0/req/geometrymanipulation)

The Geometry Manipulation Requirements Class specifies requirements for standardized geometry manipulation functions to be used with feature geometry.

2.4. Requirements classes for advanced stroke and fills

Requirements Class "Joins and Caps" (http://www.opengis.net/spec/symbology-1/2.0/req/joinscaps)

The Joins and Caps Requirements Class specifies requirements for defining a particular type of join and/or cap on strokes.

Requirements Class "Pattern Strokes" (http://www.opengis.net/spec/symbology-1/2.0/req/patternstrokes)

The Pattern Strokes Requirements Class specifies requirements for defining patern strokes using Graphics.

Requirements Class "Pattern Fills" (http://www.opengis.net/spec/symbology-1/2.0/req/patternfills)

The Pattern Strokes Requirements Class specifies requirements for defining patern fills using Graphics.

2.5. Requirements classes for additional portrayal capabilities

Requirements Class "Shape Outlines" (http://www.opengis.net/spec/symbology-1/2.0/req/shapeoutlines)

The Shape Outlines Requirements Class specifies requirements for displaying shapes with outlines so that they stand out on any background.

Requirements Class "MultiGraphics and Transforms" (http://www.opengis.net/spec/symbology-1/2.0/req/transforms)

The MultiGraphics and Transforms Requirements Class specifies requirements for building vector graphic hierarchy, including support for arbitrary transforms (translation, rotation and scaling).

Requirements Class "3D Models and Transforms" (http://www.opengis.net/spec/symbology-1/2.0/req/threedim)

The 3D Models and Transforms Requirements Class specifies requirements for defining 3D model Graphics and 3D transformations.

2.6. Requirements classes defining encodings

Requirements Class "JSON Styles and Symbology" (http://www.opengis.net/spec/symbology-1/2.0/req/json)

Requirements Class "Cascading Cartographic Symbology Style Sheets" (http://www.opengis.net/spec/symbology-1/2.0/req/ccsss)

2.7. Declaration of conformance

Conformance with this Standard shall be checked using all the relevant tests specified in Annex A (normative) of this document conformance to the respective conformance class is declared using the URIs listed in Table 1. A rendering engine accessible as an Web API can declare conformance to this Standard in its Conformance Declaration response.

The framework, concepts, and methodology for testing, and the criteria to be achieved to claim conformance are specified in the OGC Compliance Testing Policies and Procedures (OGC 08-134r11) and the OGC Compliance Testing website.

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

Full conformance at the conceptual level but not at the logical level, which can losslessly preserve all information of the conceptual model defined in a particular requirements class but necessitates defining a custom mapping and cannot be automated from a simple consistent set of encoding rules, is declared by appending `-concept` to the conformance URI.

Partial conformance at the conceptual level which can preserve some but not all information of the conceptual model defined in a particular requirements class is declared by appending `-partial-concept` to the conformance URI.

Table 1 — Conformance class URIs

CONFORMANCE CLASS	URI
Core	http://www.opengis.net/spec/symbology-1/2.0/conf/core
Basic Vector Features Styling	http://www.opengis.net/spec/symbology-1/2.0/conf/vector
Basic Coverage Styling	http://www.opengis.net/spec/symbology-1/2.0/conf/coverage
Basic Labeling	http://www.opengis.net/spec/symbology-1/2.0/conf/labels
Hill Shading	http://www.opengis.net/spec/symbology-1/2.0/conf/hillshading
Font Outlines	http://www.opengis.net/spec/symbology-1/2.0/conf/fontoutlines
Hatches, Gradients and Stipples	http://www.opengis.net/spec/symbology-1/2.0/conf/hatchesgradientsstipples
Shape Graphics	http://www.opengis.net/spec/symbology-1/2.0/conf/shapegraphics
Dashes	http://www.opengis.net/spec/symbology-1/2.0/conf/dashes

CONFORMANCE CLASS	URI
Casing and Centerline	http://www.opengis.net/spec/symbology-1/2.0/conf/casing
Parameter Values	http://www.opengis.net/spec/symbology-1/2.0/conf/parametervalues
Function Expressions	http://www.opengis.net/spec/symbology-1/2.0/conf/functions
Spatial Relation Functions	http://www.opengis.net/spec/symbology-1/2.0/conf/spatial
Temporal Relation Functions	http://www.opengis.net/spec/symbology-1/2.0/conf/temporal
Array Relation Functions	http://www.opengis.net/spec/symbology-1/2.0/conf/arrayrelations
Text Manipulation Functions	http://www.opengis.net/spec/symbology-1/2.0/conf/textmanipulation
Geometry Manipulation Functions	http://www.opengis.net/spec/symbology-1/2.0/conf/geometrymanipulation
Arithmetic Operators	http://www.opengis.net/spec/symbology-1/2.0/conf/arithmetic
Bitwise Operators	http://www.opengis.net/spec/symbology-1/2.0/conf/bitwise
Text Relation Operators	http://www.opengis.net/spec/symbology-1/2.0/conf/textrelation
Right-hand Identifiers	http://www.opengis.net/spec/symbology-1/2.0/conf/righthand
Conditional Expressions	http://www.opengis.net/spec/symbology-1/2.0/conf/conditional
Variables	http://www.opengis.net/spec/symbology-1/2.0/conf/variables
Joins and Caps	http://www.opengis.net/spec/symbology-1/2.0/conf/joinscaps
Pattern Strokes	http://www.opengis.net/spec/symbology-1/2.0/conf/patternstrokes
Pattern Fills	http://www.opengis.net/spec/symbology-1/2.0/conf/patternfills
Shape Outlines	http://www.opengis.net/spec/symbology-1/2.0/conf/shapeoutlines
MultiGraphics and Transforms	http://www.opengis.net/spec/symbology-1/2.0/conf/transforms
3D Models and Transforms	http://www.opengis.net/spec/symbology-1/2.0/conf/threedim
JSON Styles and Symbology	http://www.opengis.net/spec/symbology-1/2.0/conf/json

CONFORMANCE CLASS	URI
Cascading Cartographic Symbology	http://www.opengis.net/spec/symbology-1/2.0/conf/ccsss

3

NORMATIVE REFERENCES

3 NORMATIVE REFERENCES

The following documents are referred to in the text in such a way that some or all of their content constitutes requirements of this document. For dated references, only the edition cited applies. For undated references, the latest edition of the referenced document (including any amendments) applies.

OpenGIS Web Map Service (WMS) Implementation Specification. Open Geospatial Consortium (2006). https://portal.opengeospatial.org/files/14416

OpenGIS Styled Layer Descriptor Profile of the Web Map Service Implementation Specification. Open Geospatial Consortium (2007). https://portal.opengeospatial.org/files/22364

OpenGIS Symbology Encoding Implementation Specification. Open Geospatial Consortium (2007). https://portal.opengeospatial.org/files/16700

OGC Filter Encoding 2.0 Encoding Standard – With Corrigendum. Open Geospatial Consortium (2014). http://docs.opengeospatial.org/is/09-026r2/09-026r2.html

The Specification Model – Standard for Modular specifications. Open Geospatial Consortium (2009). https://portal.opengeospatial.org/files/?artifact_id=34762&version=2

A. Phillips, M. Davis: IETF RFC 4646, *Tags for Identifying Languages*. RFC Publisher (2006). https://www.rfc-editor.org/info/rfc4646.

ISO: ISO 19117:2012, *Geographic information – Portrayal*. International Organization for Standardization, Geneva (2012). https://www.iso.org/standard/46226.html.

The Unified Code for Units of Measure (UCUM), 2017 http://unitsofmeasure.org/ucum.html

W3C css-fonts-3, *CSS Fonts Module Level 3*. https://www.w3.org/TR/css-fonts-3/.

Panagiotis (Peter) A. Vretanos: OGC Common Query Language (CQL2) (Draft). OGC 21-065, Open Geospatial Consortium, http://docs.ogc.org/DRAFTS/21-065.html

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.

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

This document used the terms defined in Policy Directive 492, which is based on the ISO/IEC Directives, Part 2, Rules for the structure and drafting of International Standards. In particular, the word “shall” (not “must”) is the verb form used to indicate a requirement to be strictly followed to conform to this standard 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.

4.1. Portrayal

Presentation of information to humans (Note 1 to entry: Within the scope of this International Standard, portrayal is restricted to the portrayal of geographic information). [SOURCE: ISO 19117:2012, 4.20]

4.2. Layer

Abstraction of reality specified by a geographic data model (feature, coverage…). A layer may be represented using a set of symbols (Style). A layer contributes to a single geographic subject and may be a theme.

4.3. Style

A sequence of rules of symbolizing instructions to be applied by a rendering engine on one or more features and/or coverages.

4.4. Rendering engine

Automated process that produces graphics using a pipeline of layers and styles as inputs.

4.5. Render

Conversion of digital graphics data into visual form (EXAMPLE Generation of an image on a video display) [SOURCE: ISO 19117:2012, 4.27]

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. Abbreviated terms

The abbreviated terms clause gives a list of the abbreviated terms necessary for understanding this document.

IETF	Internet Engineering Task Force, https://ietf.org/
ISO	International Organization for Standardization, https://www.iso.org
OGC	Open Geospatial Consortium, www.opengeospatial.org
UML	Unified Modeling Language, http://www.uml.org/

6

OVERVIEW

6 OVERVIEW

This Standard defines conformance classes for:

- a core symbology conceptual and logical model based on Styles as a list of StylingRules consisting of a Symbolizer and optional Selector Expression (Section 7),
- additional conformance classes extending this core model with:
  - basic portrayal capabilities of vector features (Section 8),
  - portrayal capabilities of coverage data (Section 9),
  - label placement and font outlines capabilities (Section 10),
  - shape graphics capabilities including circles, ellipses, arcs, rectangles (including rounded rectangles) and paths (polylines and polygons), with the ability to use outlines (Section 11),
  - vector graphic hierarchy capabilities, including full 2D transforms (Section 12),
  - advanced strokes capabilities, including specific joins and caps, dashed lines, casing and centerlines, and graphic patterns (Section 13),
  - advanced fill capabilities, including graphic patterns, hatches, gradients and stipples (Section 14),
  - using expressions as parameter values for any symbolizer properties (Section 15),
  - using identifiers in the right side of operation expressions, conditional expressions and variables (Section 16),
  - arithmetic, bitwise and text relation operators (Section 17),
  - function call expressions and standardized functions for spatial, temporal and array relations, as well as text and geometry manipulation (Section 18), and
  - using 3D model graphics and 3D transforms (Section 19),
- two encodings for the logical and conceptual model covering all of these conformance classes:
  - an encoding based on JSON that can be readily parsed by JSON parser (Section 20), and
  - a more expressive encoding based on Web CSS which is better suited to hand-edit styles (Section 21).

In addition, the following annexes are included:

- a normative Abstract Test Suite for the conformance classes (Annex A),
- an informative mapping of SLD/SE and notable vendor extensions to the conceptual model and requirements classes (Annex B),
- an informative map gallery of practical use cases alongside the styles used to generate the maps encoded in the *Cascading Cartographic and Symbology Style Sheets* encoding (Annex C), and
- an informative revision history (Annex D).

7

REQUIREMENTS CLASS "CORE"

7 REQUIREMENTS CLASS "CORE"

7.1. Overview

The requirements described in this section define the symbology core requirement class. The following UML diagram (Figure 1) shows the fundamental concepts of the Symbology Conceptual and Logical Model:

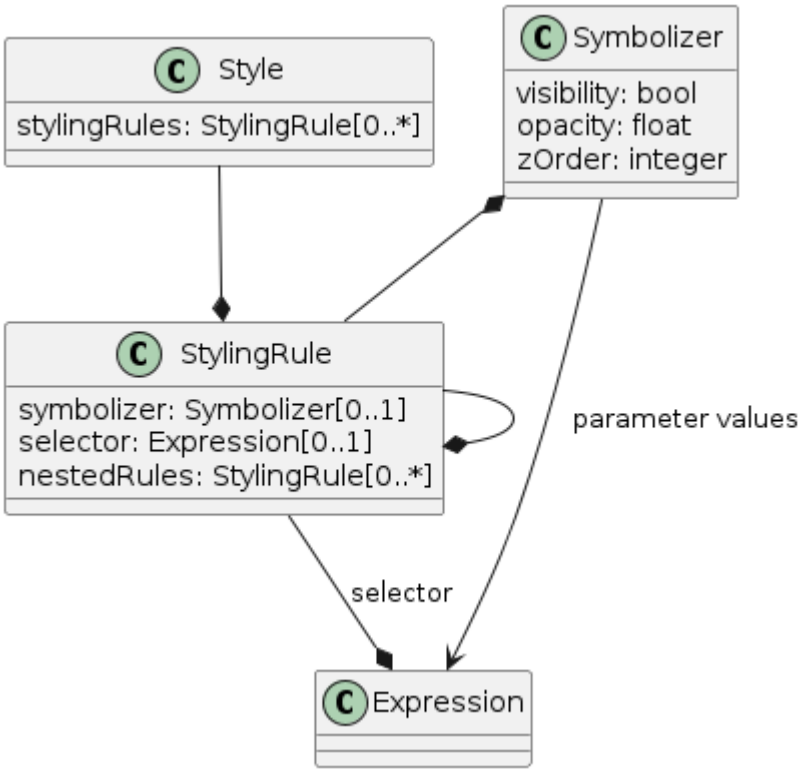

Figure 1 — Symbology Core Classes UML Diagram

7.1.1. Styles

A Style is the root concept of the Styles and Symbology Conceptual Model and consists of an ordered list of StylingRules. The StylingRules are to be applied in order so that rules selected later override earlier ones. This allows to compose styles in a cascading manner by inheriting from a base style then overriding some specific portrayal behaviors.

Table 2 — Style class

NAME	DEFINITION	DATA TYPE AND VALUE	MULTIPLICITY
stylingRules	Styling rules defining the style	StylingRule	0..*

7.1.2. Styling Rules

A StylingRule specifies that a particular Symbolizer (defining how to portray data) should be applied if the rule is selected based on a Selector expression (e.g., feature-property conditions or map scales). A StylingRule can contain nested rules, which will only be applied if the the parent rule's Selector is evaluated to true, and whose Symbolizers will inherit from and override the parent StylingRule's Symbolizer.

Table 3 — StylingRule

NAME	DEFINITION	DATA TYPE AND VALUE	MULTIPLICITY
name	A name for the rule (useful for generating legends)	string	0..1
selector	Selector defining the condition to apply the rule's symbolizer	Expression	0..1
symbolizer	Symbolizer to apply by the rendering engine	Symbolizer	0..1
nested Rules	Additional styling rules to consider if this rule is selected partially overriding this rule's symbolizer	StylingRule	0..*

The selector (also sometimes called a *filter*), for the rule is defined as a predicate expression evaluating to either true or false.

Example — Example showcasing a Style made of StylingRules using CCSSS encoding

```
#Landuse    // Selector for this first rule
{
   // Symbolizer properties to be set if this rule is selected
   visibility: false;
}

#Roads    // Selector for this second rule
{
   // Symbolizer properties to be set if this rule is selected
   visibility: false;

   // Nested rule only considered if this parent rule is selected
   [viz.sd < 200000] // Selector for this nested rule
   {
      // Symbolizer properties to be set if this nested rule is selected
      visibility: true;
   }
}
```

7.1.3. Symbolizers

A Symbolizer describes how to portray geographic data (e.g., vector features or gridded coverage data) based on a set of properties. This core requirements class only defines three basic properties relevant to most portrayal use cases. Separate requirements classes and later parts will extend this Symbolizer class with additional properties to provide more advanced portrayal capabilities.

Table 4 — Symbolizer class

NAME	DEFINITION	DATA TYPE AND VALUE	MULTIPLICITY
visibility	Whether to display the data	bool	0..1
opacity	Degree of opacity	float	0..1
zOrder	Specify a rendering order	int	0..1

Example — CCSSS Example showcasing the limited symbolization possible with this Core requirements class

```
{
   visibility: false;

   #Landuse
   {
      opacity = 0.5;
      zOrder = 1;
      visibility = true;
   }

   #Buildings
   {
      opacity = 0.8;
      zOrder = 2;
      visibility = true;
   }
}
```

7.1.4. Expressions

Expressions are a syntactic entity that may be evaluated to determine its value (from Wikipedia). The concept has similar meaning to Expressions as defined in the OGC Filter Encoding 2.0 standard, and can also be implemented with the OGC Common Query Language (CQL2), with the caveat that the Expressions defined here are not restricted to evaluate to a boolean value as is the case for the first version of CQL2.

Expressions can be used in two places according to this Styles & Symbology conceptual model: as a Selector defining the condition whether to apply a StylingRule or not, as well as to define the values of Symbolizer properties.

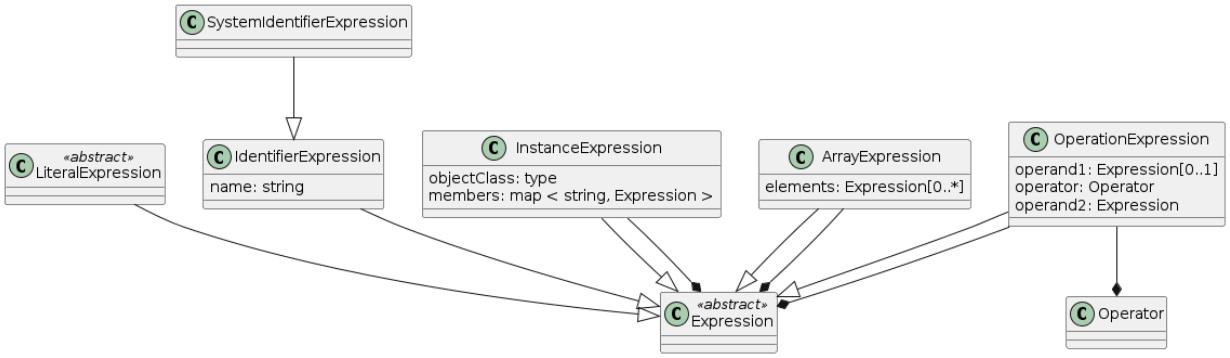

Figure 2 — Expression Classes UML Diagram

7.1.4.1. Parameter Values

The use of an expression as a Symbolizer property is called a *Parameter Value*, where an Expression can be substituted where a particular data type is expected, in which case the Expression will be resolved to that expected data type. Conceptually, all Symbolizer properties can be specified using any type of expression *Parameter Value*, but this is not required by this core conformance class, and specific requirements classes such as the "Symbolizer Parameter Value Expresssions" requirements class enable this behavior. Without a requirements class specifying otherwise, only Literal Expressions compatible with the expected data type are allowed as *Parameter Values*.

7.1.4.2. Identifier Expressions

An Identifier Expression is represented by a textual property that gets resolved at runtime.

Typical example of identifier expressions include feature properties (e.g., `population`) and coverage range values (e.g., a particular imagery band `B8`).

7.1.4.2.1. System Identifiers

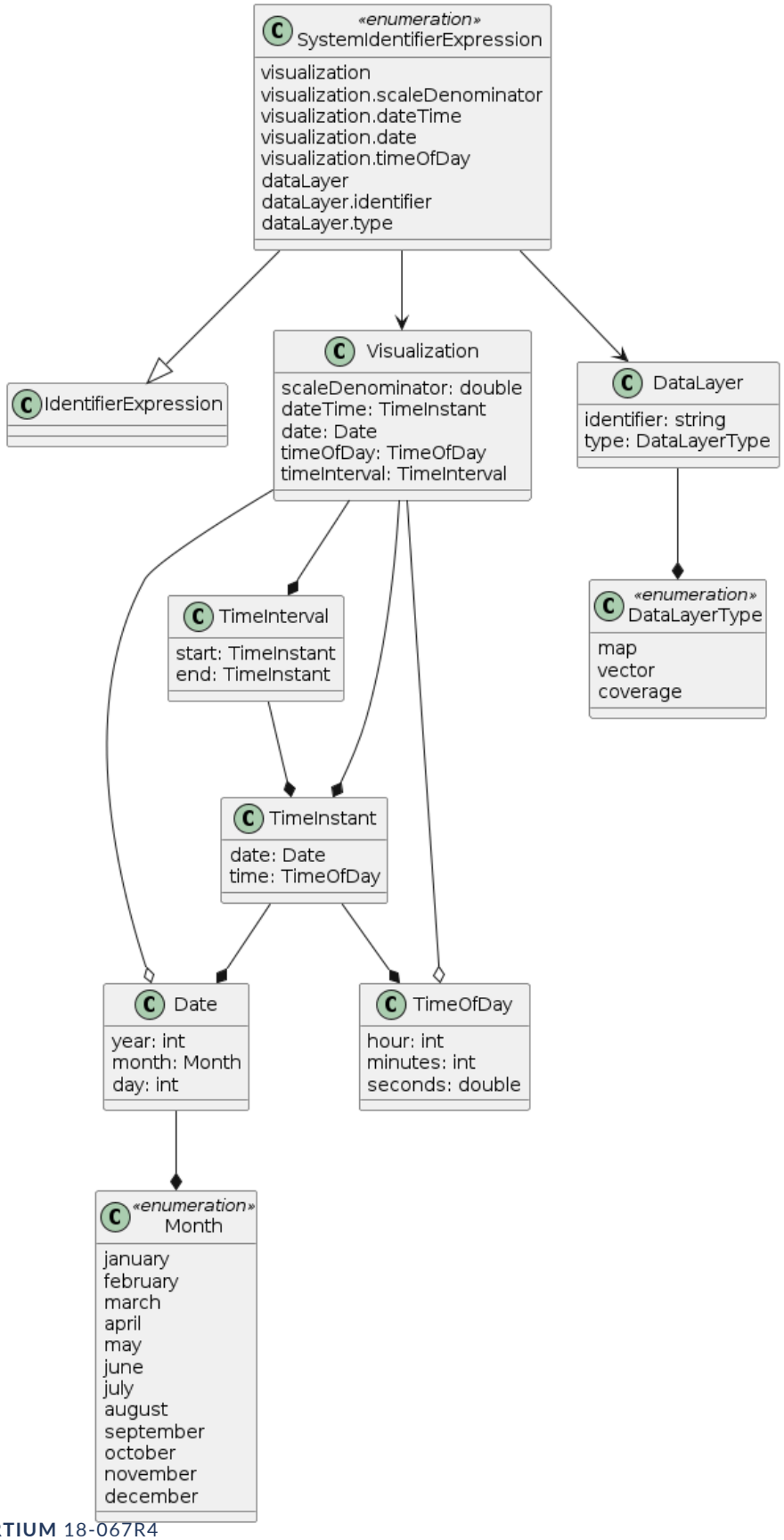

Figure 3 — System Identifiers UML Diagram

One specific type of identifier expressions use System Identifiers. A System Identifier refers to an internal state of the system, or a particular aspect of the data being portrayed.

The following System Identifiers are defined by this conformance class:

Table 5 — Visualization System Identifiers

NAME	DEFINITION	DATA TYPE AND VALUE
visualization.scaleDenominator	Current visualization scale denominator	float
visualization.dateTime	Current visualization date and time	DateTime
visualization.date	Current visualization date	Date
visualization.timeOfDay	Current visualization time of day	Time
visualization.timeInterval	Current visualization time of day	TimeInterval

Example 1 — CCSSS Example showcasing scale-dependent selector using system identifiers

```
#Roads
{
   visibility: false;

   [viz.sd < 200000] // Roads will only be visible at scale 1:200,000 and
closer
   {
      visibility: true;
   }
}
```

Table 6 — Month enumeration

NAME
january
february
march
april
may
june

NAME
july
august
september
october
november
december

Table 7 — Date class

NAME	DEFINITION	DATA TYPE AND VALUE	MULTIPLICITY
year		int	1
month		Month	1
day		int	1

Table 8 — TimeOfDay class

NAME	DEFINITION	DATA TYPE AND VALUE	MULTIPLICITY
hour		int	1
minutes		Month	1
seconds		int	1

Table 9 — Class TimeInstant

NAME	DEFINITION	DATA TYPE AND VALUE	MULTIPLICITY
date		Date	1
time		TimeOfDay	1

Table 10 — TimeInterval class

NAME	DEFINITION	DATA TYPE	MULTIPLICITY
start	start of the interval	TimeInstant	1
end	end of the interval	TimeInstant	1

Example 2 — CCSSS Example showcasing visualization time-dependent selector using system identifiers

```
#Landuse
{
   visibility: false;

   // Landuse will only be visible when visualization date is later than
January 1, 2020
   [viz.date > DATE('2020-01-01')]
   {
      visibility: true;
   }
}
```

Table 11 — Layer System Identifiers

NAME	DEFINITION	DATA TYPE AND VALUE
dataLayer.identifier	Identifier of data layers	string
dataLayer.type	Data type of data layers	enumeration: vector, coverage, ...

Example 3 — CCSSS Example showcasing selectors depending on layer identifiers and data type using system identifiers

```
[dataLayer.type = vector]
{
   // Display vector layers at 60% opacity by default
   opacity: 0.6;

   [dataLayer.identifier = 'Landuse']
   {
      // Except for Landuse that should show at 80% opacity
      opacity: 0.8;
   }

   // In CCSSS, the #<layerid> syntax is equivalent
   // to [dataLayer.identifier = '<layerid>']
   #Roads
   {
      opacity = 1.0; // and roads at 100% opacity
   }
}
```

7.1.4.3. Literal Expressions

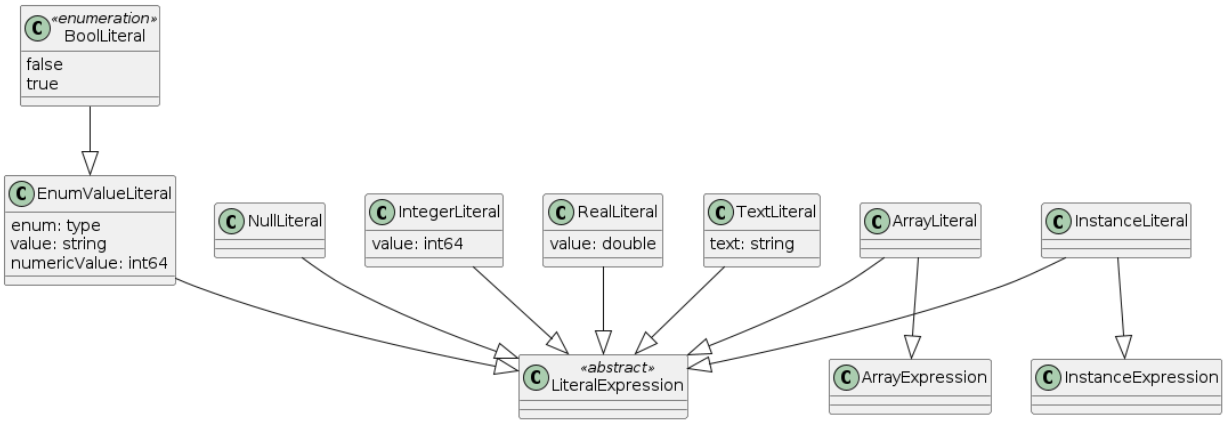

Figure 4 — Literal Classes UML Diagram

A literal expression is an expression that has a fixed value in the encoding of the style. A similar literal class concept was originally defined in the OGC Filter Encoding 2.0 standard section 7.5.1.

The following types of literals are defined:

- An integer literal represents a whole numeric value (CCSSS example: `123`)
- A real literal represents a real number (integer, fractional or irrational number) (CCSSS example: `0.85`)
- Text literals are Unicode character strings (CCSSS example: `'København'`)
- A null literal represents an unset value (CCSSS example: `null`)
- Enumeration value literals allow to refer by name to a set of pre-defined values associated with the context where they are used (CCSSS example: `vector`)
- Boolean literals are a specialized type of enumeration literals which are either true or false (CCSSS example: `false`)
- Array literals (see Array Expressions) (CCSSS example: `(1, 2, 3)`)
- Instance literals (see Instance Expressions) (CCSSS examples: `Color(20, 100, 80)` or `Color(r: 20, g: 100, b: 80)`)

7.1.4.4. Operation Expressions

An Operation Expression performs an operation specified by an Operator on one, two or three operand expressions, depending on whether they unary, binary or ternary operators. This core requirements class defines only Logical and Relational operators. Implementations are also required to support prioritizing operations (typically encoded using parentheses: `( )`). Other requirements classes introduce support for additional operators for arithmetic, text and bitwise

operations. Without support for the *Any right-hand operand* requirements class, the right-hand operands of RelationExpressions are limited to LiteralExpressions.

Table 12 — OperationExpression class

NAME	DEFINITION	DATA TYPE AND VALUE	MULTIPLICITY
operand1	First operand	Expression	0..1
operator	Operator	Operator	1
operand2	Second operand	Expression	1

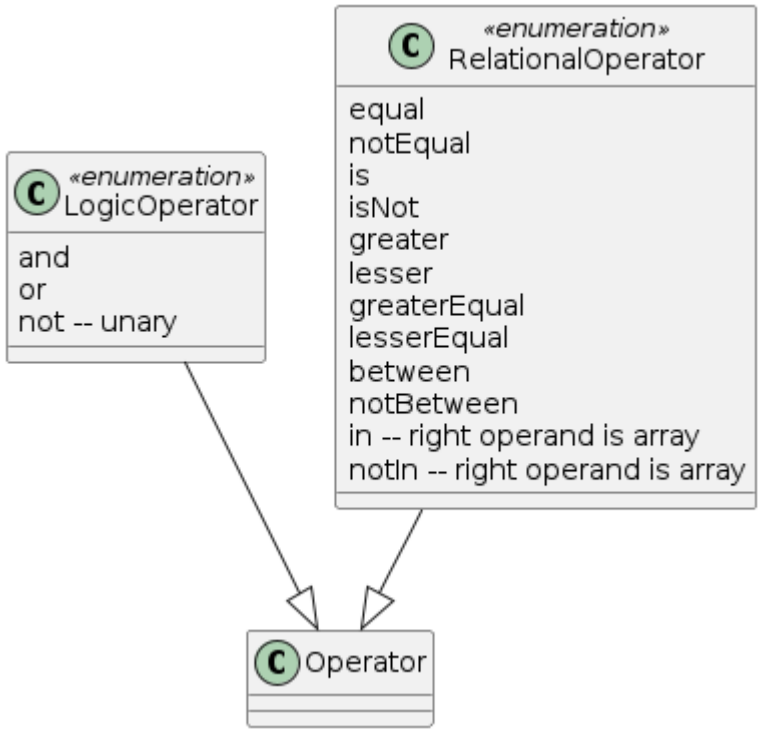

Figure 5 — Core Logic and Relational Operators UML Diagram

This requirements class defines the following Logical Operators :

Table 13 — Logical Operators

NAME	CCSSS EXAMPLE	TYPE	DEFINITION
not	`not`	unary	True if and only if the operand is false
and	`and`	binary	True if and only if both operands are true

NAME	CCSSS EXAMPLE	TYPE	DEFINITION
or	`or`	binary	True if and only if either or both of the operands is true

This requirements class defines the following Relational Operators:

Table 14 — Relational Operators

NAME	CCSSS EXAMPLE	TYPE	DEFINITION
equal	`=`	binary	The first operand is equal to the second one
notEqual	`<>`	binary	The first operand is not equal to the second one
is (null)	`is`	binary	Only used with `null` literal for second operand (the first operand is null)
isNot (null)	`is not`	binary	Only used with `null` literal for second operand (the first operand is not null)
lesser	`<`	binary	The first operand is smaller than the second one
greater	`>`	binary	The first operand is greater than the second one
lesserEqual	`<=`	binary	The first operand is smaller than or equal to the second one
greater Equal	`>=`	binary	The first operand is greater than or equal to the second one
in (array)	`in`	binary	Only used with an array for second operand (the first operand is one of the elements of the array)
notIn (array)	`not in`	binary	Only used with an array for second operand (the first operand is not one of the elements of the array)
between	`between … and`	ternary	The first operand is between the second and third operands (inclusively)
notBetween	`not between … and`	ternary	The first operand is not between the second and third operands (inclusively)

Example — CCSSS Example showcasing the use of logical and relational operators in a selector

```
#Roads
{
   visibility: false;

   // Roads will be visible between scales 1:20,000 and 1:200,000
   [viz.sd > 20000 and viz.sd < 200000]
   {
      visibility: true;
   }
}
```

7.1.4.5. Array Expressions

An array expressions defines zero, one, or more element expressions forming an ordered set.

CCSSS example: `(123.0, viz.sd)`.

An *array literal* is a special type of array expression whose elements are all literals.

7.1.4.6. Instance Expressions

An instance expressions instantiates an object of a class while optionally specifying values for member components. The default values for any member if not explicitly overridden can is specified by the class member definitions.

CCSSS examples:

- `DateTime(year: 2022, month: april, day: 6)`
- `Color(r: 20, g: 100, b: 80))`

An *instance literal* is a special type of instance expression assigning only literal expressions to its member components.

A Symbolizer may either set an object property with a complete new instance, or specifically override a particular member of that object property. This capability is particularly useful when using nested StylingRules, where a nested rule can inherit the Symbolizer from the parent StylingRule.

Example — CCSSS Example showcasing the use of logical and relational operators in a selector

```
#Roads
{
   // Default to a gray stroke (completely replace stroke property)
   stroke: { gray };
   // For scales closer than 1:200,000 ...
   [viz.sd < 200000]
   {
      stroke.width: 5px; // inherit gray color, but modify the width to be 5
pixels.

      // For scales closer than 1:10,000, replace the stroke completely with a
5 meters blue stroke
      [viz.sd < 10000] { stroke: { blue, width: 5m } }
   }
}
```

7.2. Requirements

7.2.1. Core

REQUIREMENTS CLASS 1: CORE	
IDENTIFIER	`http://www.opengis.net/spec/symbology-1/2.0/req/core`
TARGET TYPE	Style encodings and Renderers

7.2.1.1. Style `stylingRules` property

REQUIREMENT 1	
IDENTIFIER	`/req/core/rules`
A	An Encoding SHALL support individually storing and preserving the order of each StylingRule constituting a Style.
B	A Renderer SHALL support considering in order each individual StylingRule constituting a style.
C	The Symbolizer properties specified in a StylingRule appearing later in a Style's list of rules and whose Selector evaluates to true SHALL be interpreted as overriding all Symbolizer properties set by earlier rules.

7.2.1.2. StylingRule `nestedRules` property

REQUIREMENT 2	
IDENTIFIER	`/req/core/nested`
A	An Encoding SHALL support …
B	A Renderer SHALL support …

7.2.1.3. Identifier Expressions

REQUIREMENT 3	
IDENTIFIER	`/req/core/expressions`
A	An Encoding SHALL support …
B	A Renderer SHALL support …

7.2.1.4. StylingRule `selector` property

REQUIREMENT 4	
IDENTIFIER	`/req/core/selector`
A	An Encoding SHALL support …
B	A Renderer SHALL support …

7.2.1.5. StylingRule `symbolizer` property

REQUIREMENT 5	
IDENTIFIER	`/req/core/symbolizer`
A	An Encoding SHALL support …
B	A Renderer SHALL support …

7.2.1.6. Symbolizer properties Parameter Values

REQUIREMENT 6	
IDENTIFIER	`/req/core/parameters`
A	An Encoding SHALL support …
B	A Renderer SHALL support …

7.2.1.7. Symbolizer `visibility` property

REQUIREMENT 7	
IDENTIFIER	`/req/core/visibility`
A	An Encoding SHALL support …
B	A Renderer SHALL support …

7.2.1.8. Symbolizer `opacity` property

REQUIREMENT 8	
IDENTIFIER	`/req/core/opacity`
A	An Encoding SHALL support …
B	A Renderer SHALL support …

7.2.1.9. Symbolizer `zOrder` property

REQUIREMENT 9	
IDENTIFIER	`/req/core/zorder`
A	An Encoding SHALL support …

REQUIREMENT 9	
B	A Renderer SHALL support …

7.2.1.10. Literal Expressions

REQUIREMENT 10	
IDENTIFIER	`/req/core/literals`
A	An Encoding SHALL support …
B	A Renderer SHALL support …

7.2.1.11. Identifier Expressions

REQUIREMENT 11	
IDENTIFIER	`/req/core/identifiers`
A	An Encoding SHALL support …
B	A Renderer SHALL support …

7.2.1.12. Visualization Scale Denominator System Identifiers

REQUIREMENT 12	
IDENTIFIER	`/req/core/scale`
A	An Encoding SHALL support …
B	A Renderer SHALL support …

7.2.1.13. Visualization Time System Identifiers

REQUIREMENT 13	
IDENTIFIER	`/req/core/time`
A	An Encoding SHALL support …
B	A Renderer SHALL support …

7.2.1.14. Data Layer System Identifiers

REQUIREMENT 14	
IDENTIFIER	`/req/core/layers`
A	An Encoding SHALL support …
B	A Renderer SHALL support …

7.2.1.15. Operation Expressions

REQUIREMENT 15	
IDENTIFIER	`/req/core/operations`
A	An Encoding SHALL support defining operation expressions consisting of operands and an operator for unary, binary and ternary operators.
B	An Encoding SHALL support defining the priority of expressions using multiple operators (e.g., using parentheses: ( )).
C	A Renderer SHALL support evaluating operation expressions while respecting the priority of operations.

7.2.1.16. Logical Operation Expressions

REQUIREMENT 16	
IDENTIFIER	`/req/core/logical`
A	An Encoding SHALL support …
B	A Renderer SHALL support …

7.2.1.17. Relational Operation Expressions

REQUIREMENT 17	
IDENTIFIER	`/req/core/relational`
A	An Encoding SHALL support …
B	A Renderer SHALL support …

7.2.1.18. Array Expressions

REQUIREMENT 18	
IDENTIFIER	`/req/core/arrays`
A	An Encoding SHALL support …
B	A Renderer SHALL support …

7.2.1.19. Instance Expressions

REQUIREMENT 19	
IDENTIFIER	`/req/core/instances`
A	An Encoding SHALL support …

REQUIREMENT 19	
B	A Renderer SHALL support …

8

REQUIREMENTS CLASS "BASIC VECTOR FEATURES STYLING"

8 REQUIREMENTS CLASS "BASIC VECTOR FEATURES STYLING"

8.1. Overview

This requirements class adds support for defining how to Fill and render the Stroke of vector geometry. In addition, it defines the ability to specify a Marker consisting of one or more Graphic, to be rendered at the exact location of point geometry, at the centroid of polygon geometry, or at each point of line geometry.

8.1.1. Symbolizer extension for vector features

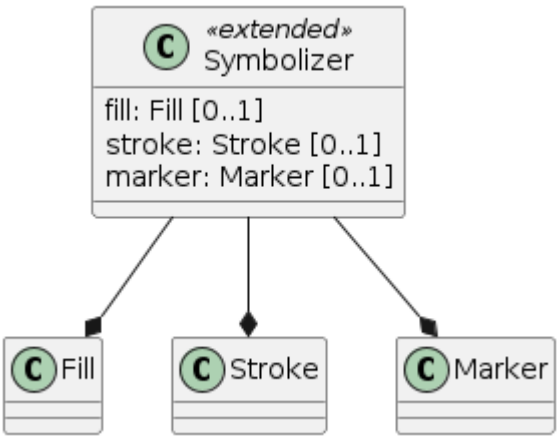

Figure 6 — Basic Vector Features Extended Symbolizer Class UML Diagram

The following three new Symbolizer properties are defined by this requirements class:

Table 15 — Symbolizer class (vector features extension)

NAME	DEFINITION	DATA TYPE AND VALUE	MULTIPLICITY
fill	Fill to fill inside of geometries	Fill	0..1
stroke	Stroke to render geometry outlines	Stroke	0..1
marker	Marker to render at geometries points or centroid (for polygons)	Marker	0..1

8.1.2. Extended Identifiers for vector features

This requirements class defines as valid identifiers all of the accessible properties for a feature being portrayed with the name of those properties being the text of the identifier. Those identifiers are to be resolved with the value of that property. An encoding must support an escaping mechanism in cases where the name of a feature property might clash with a syntactic element.

8.1.2.1. Extended System Identifiers

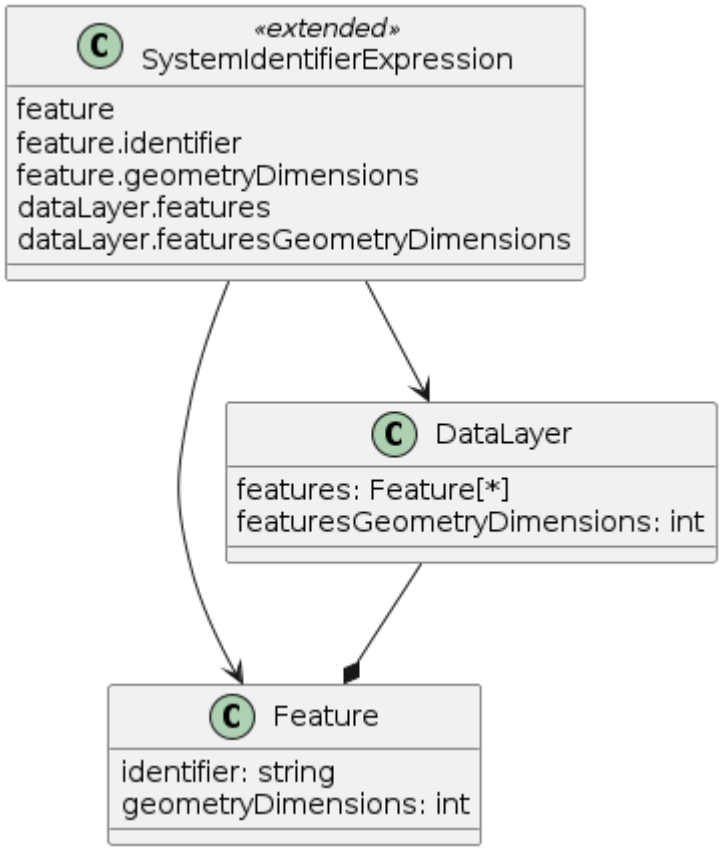

Figure 7 — Vector Features System Identifiers UML Class Diagram

In addition, the following new System Identifiers are defined by this requirements class:

Table 16 — System identifiers for vector features

NAME	DEFINITION	DATA TYPE AND VALUE
feature.identifier	Identifier of a particular feature	string or number

NAME	DEFINITION	DATA TYPE AND VALUE
feature.geometryDimensions	Dimension of geometry for a particular feature	0 (points), 1 (lines), 2 (polygons), 3 (polyhedrons)
dataLayer.featuresGeometryDimension	Dimension of feature geometry (if same for all features)	0 (points), 1 (lines), 2 (polygons), 3 (polyhedrons), `null` (mixed)

8.1.3. Colors

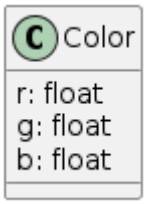

Figure 8 — Color class UML Diagram

Both the Fill and Stroke allow to define a Color, specified using red, green and blue components. The ability to define Colors in other spaces (e.g., Lab, CMYK) is defined in separate requirements classes and applies wherever this basic RGB Color class is used.

Table 17 — Color class

NAME	DEFINITION	TYPE	MULTIPLICITY
r	Red component	float	1
g	Green component	float	1
b	Blue component	float	1

8.1.4. Fills

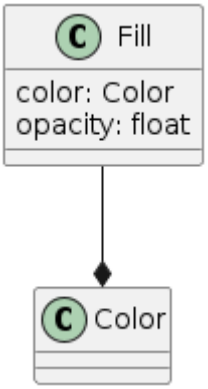

Figure 9 — UML Class Diagram of the Symbology Fill

A Fill defines the graphical symbolizing parameters required to draw the filling of a two-dimensional shape such as a polygon. Separate conformance classes can extend this Fill class with additional properties.

Table 18 — Fill class

NAME	DEFINITION	TYPE	MULTIPLICITY
color	Color of the fill	Color	0..1
opacity	Opacity of the fill	float	0..1

8.1.5. Strokes

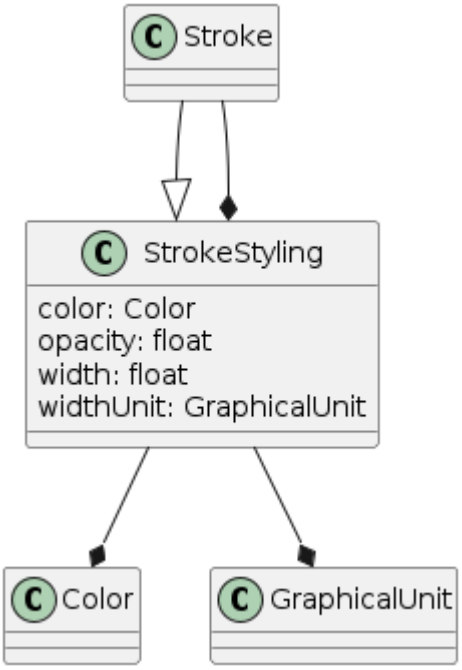

Figure 10 — Stroke Class UML Diagram

A Stroke defines the graphical symbolizing parameters for drawing an outline (e.g., for linear geometries or the exterior of a polygon geometry). As an abstract class and part of the base of the core graphical concepts, StrokeClass is a global point of extension to specify concrete ways to draw outlines (e.g., the PenStroke and GraphicStroke extensions). The StrokeClass properties are documented in the following table.

A Stroke object inherits from a base *StrokeStyling* class, which is re-used in a separate requirements class for defining more complex strokes:

Table 19 — StrokeStyling class

NAME	DEFINITION	DATA TYPE AND VALUE	MULTIPLICITY
color	Color of the stroke	Color	0..1
width	Width of the stroke	float (uom-qualified)	0..1

8.1.5.1. Graphical Units of Measure

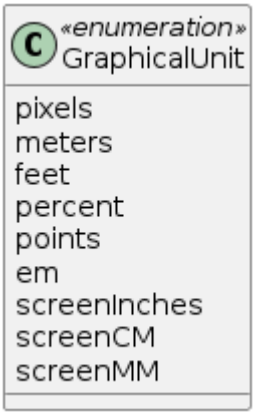

Figure 11 — Graphical Units Enumeration UML Diagram

For styling parameters that define sizing and positioning of graphical objects (width, displacement, etc.), a specific unit of measure needs to be provided for the rendering engine. Therefore, properties where a unit of measure is relevant can be qualified with a particular uom codes.

Below is the list of allowed units of measure as per UCUM (except for pixel):

- portrayal units: pixel, millimeter, inch, percentage; and
- ground units: meter, foot.

The portrayal unit “pixel” is the default unit of measure. If available, the pixel size depends on the viewer client resolution, otherwise it is equal to 0.28mm x 0.28mm (~ 90 DPI).

8.1.6. Markers

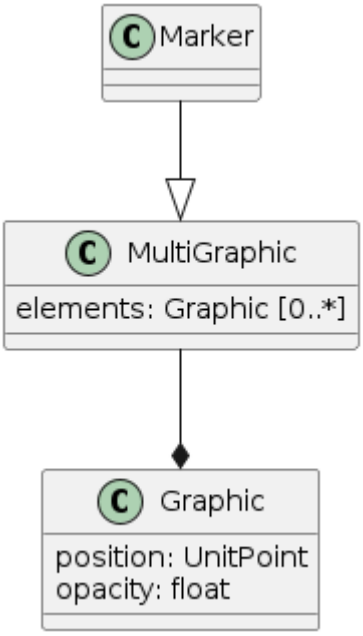

Figure 12 — Marker Class UML Diagram

A Marker inherits from a MultiGraphic which can contain multiple graphical elements:

Table 20 — MultiGraphic class

NAME	DEFINITION	DATA TYPE AND VALUE	MULTIPLICITY
elements	Children elements	Graphic	1..*

Without support for the *Multi Graphics and Transforms* requirements class, a MultiGraphic cannot contain another MultiGraphic.

A MultiGraphic itself inherits from a Graphic allowing to specify an opacity and a 2D position:

Table 21 — Graphic class

NAME	DEFINITION	DATA TYPE AND VALUE	MULTIPLICITY
opacity	Opacity	float	0..1
position	Position	UnitPoint	0..1

Table 22 — UnitPoint class

NAME	DEFINITION	DATA TYPE AND VALUE	MULTIPLICITY
x	Horizontal position	float (uom-qualified)	1
y	Vertical position	float (uom-qualified)	1

8.1.6.1. Dot Graphics

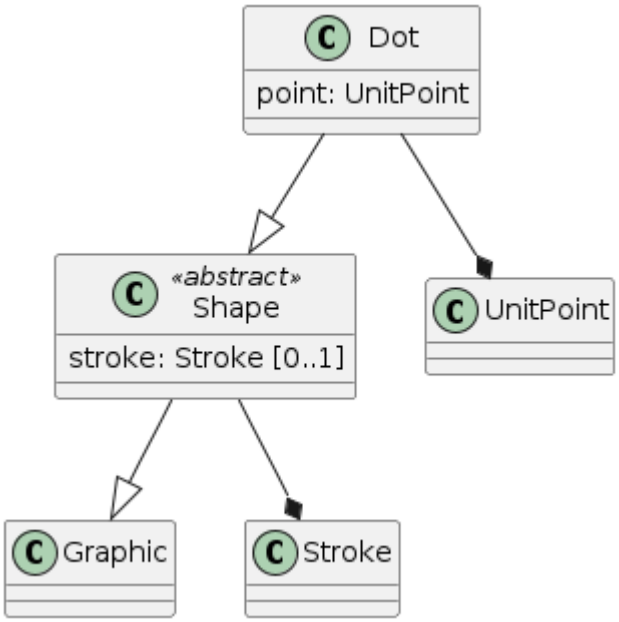

Figure 13 — Dot Class UML Diagram

A round Dot is one type of Graphic defined by this requirements class that can be used as part of a Marker.

A Dot inherits from a Shape class allowing to define a `stroke` which controls the color and size of the dot (with the stroke's `width` property).

Table 23 — Shape

NAME	DEFINITION	TYPE	MULTIPLICITY
stroke	Stroke with which to render the shape outline	Stroke	0..1

8.1.6.2. Image Graphics

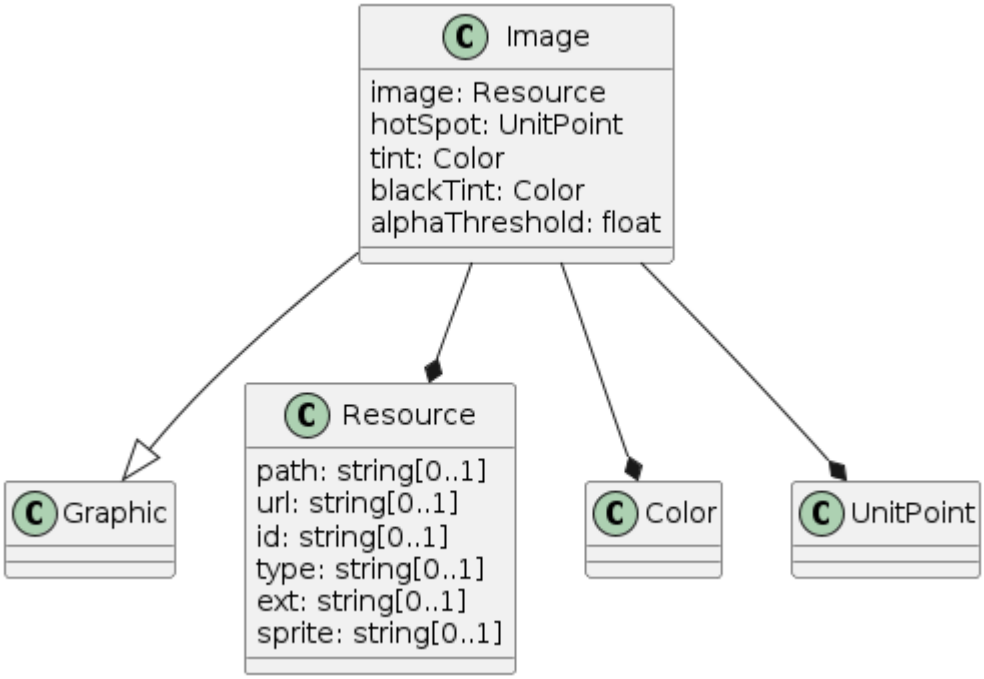

Figure 14 — Image Class UML Diagram

An Image is one type of graphic defined by this requirements class that can be used as part of a Marker.

Table 24 — Image

NAME	DEFINITION	TYPE	MULTIPLICITY
image	The image to use	Resource	1
hotSpot	0,0 position in the image	UnitPoint (uom-qualified)	1
tint	Tint (multiplying white)	Color	1
blackTint	Black tint (used for black	Color	1
alphaThreshold	Alpha value considered pickable	float	1

The actual image content is defined by a Resource, which can be specified by one or multiple properties:

Table 25 — Resource class

NAME	DEFINITION	TYPE	MULTIPLICITY
url	URL for online resource	string	0..1
path	(preferably relative) path to local file	string	0..1
id	ID within a database table or a sprite	string	0..1
type	Selected media type for the resource	string	0..1
ext	File extension to build paths/URL with id	string	0..1
sprite	Name of source sprite file	string	0..1

8.1.6.3. Text Graphics

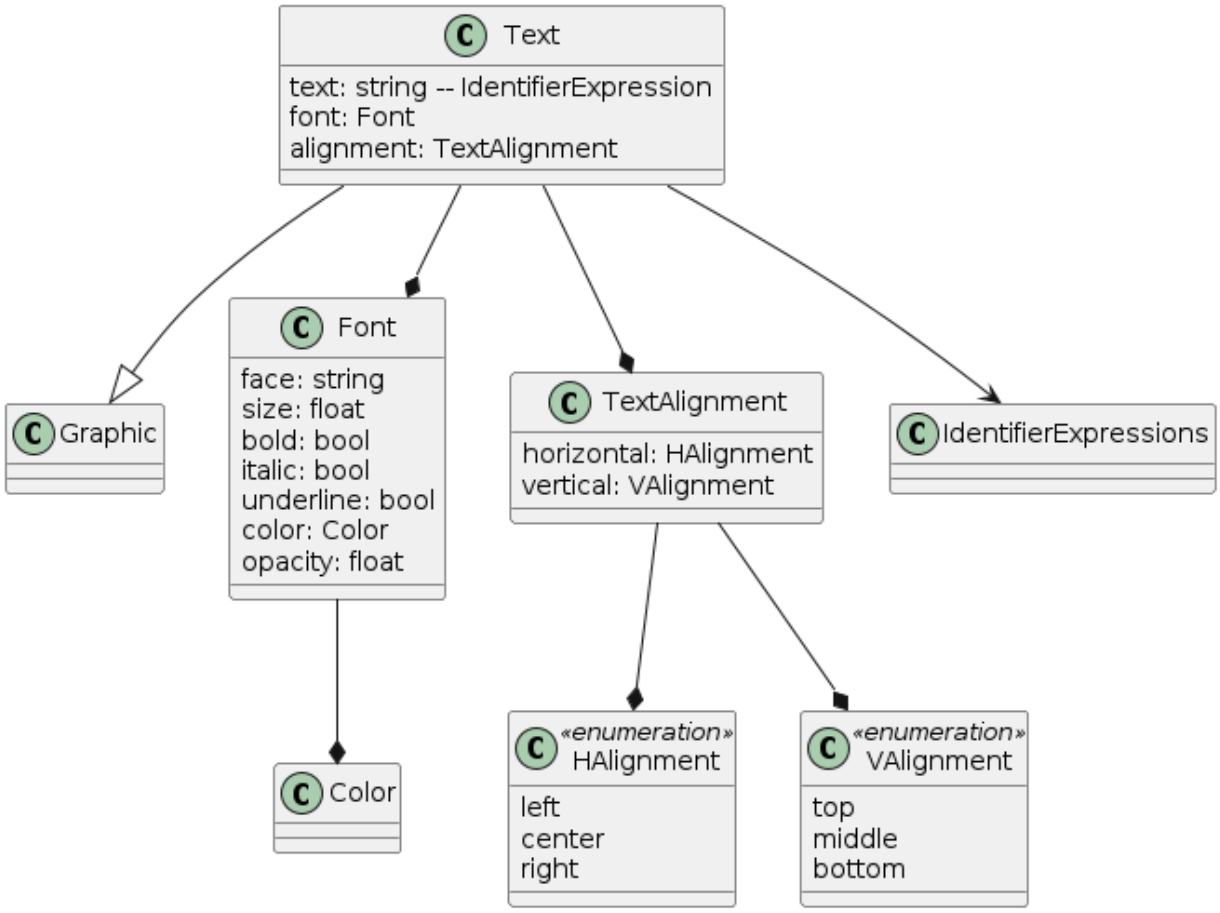

Figure 15 — Text Class UML Diagram

A Text is one type of Graphic defined by this requirements class that can be used as part of a Marker.

Table 26 — Text

NAME	DEFINITION	DATA TYPE AND VALUE	MULTIPLICITY
text	The character string for the text	string (unicode)*	1
font	The font to use for the text	Font	0..1
alignment	The alignment to use for the text	TextAlignment	0..1

This requirements class requires support for identifier parameter value expression for the `text` property without any additional requirements class. This allows to display the value of feature properties with a Text Marker.

Table 27 — TextAlignment

NAME	DEFINITION	DATA TYPE AND VALUE	MULTIPLICITY
horzAlignment	Horizontal alignment	HAlignment	0..1
vertAlignment	Vertical alignment	VAlignment	0..1

Table 28 — HAlignment enumeration

NAME	DEFINITION
left	Left alignment
center	Center alignment
right	Right alignment

Table 29 — VAlignment enumeration

NAME	DEFINITION
top	Top alignment
middle	Middle alignment

NAME	DEFINITION
bottom	Bottom alignment

The font property of a Text Graphic is defined uses the following *Font* class:

Table 30 — Font class

NAME	DEFINITION	DATA TYPE AND VALUE	MULTIPLICITY
face	Face name	string	0..1
size	Size of the font (pt)	float	0..1
bold	Use bold style	bool	0..1
italic	Use italic style	bool	0..1

8.2. Requirements

8.2.1. Basic Vector Features Styling

REQUIREMENTS CLASS 2: BASIC VECTOR FEATURES STYLING	
IDENTIFIER	`http://www.opengis.net/spec/symbology-1/2.0/req/vector`
TARGET TYPE	Style encodings and Renderers
PREREQUISITE	Requirements class 1: `http://www.opengis.net/spec/symbology-1/2.0/req/core`

8.2.1.1. Symbolizer `fill` property

REQUIREMENT 20	
IDENTIFIER	`/req/vector/fill`

REQUIREMENT 20	
A	An Encoding SHALL support defining a `fill` Symbolizer property as an object including a `color` and an `opacity`.
B	A Renderer SHALL support filling vector geometry based on a `fill` property specifying a `color` and an `opacity`.

8.2.1.2. Symbolizer `stroke` property

REQUIREMENT 21	
IDENTIFIER	`/req/vector/stroke`
A	An Encoding SHALL support defining a `stroke` Symbolizer property as an object including a `color` and a units of measure-qualified ("uom-qualified") `width`.
B	A Renderer SHALL support drawing the stroke of vector geometry based on a `stroke` property specifying a `color` and a uom-qualified `width`.

8.2.1.3. Graphical Units of Measure

REQUIREMENT 22	
IDENTIFIER	`/req/vector/units`
A	An Encoding SHALL support qualifying distance properties identified as "uom-qualified" with a unit of measure (pixel, millimeter, inch, percentage, meter, foot).
B	A Renderer SHALL support applying the appropriate unit transformation when rendering elements specifying a uom-qualified distances.

8.2.1.4. Symbolizer `marker` property

REQUIREMENT 23	
IDENTIFIER	`/req/vector/marker`
A	An Encoding SHALL support defining a `marker` Symbolizer property as an array of graphical elements (Graphics).

REQUIREMENT 23	
B	A Renderer SHALL support drawing the grahical elements of a marker at the points of vector geometry (for points and line strings) and at the centroid of polygons.

8.2.1.5. Graphics

REQUIREMENT 24	
IDENTIFIER	`/req/vector/graphics`
A	An Encoding SHALL support defining graphical elements (graphics) specifying a uom-qualified 2D real position offsetting the graphic from its original position.
B	A Renderer SHALL support drawing the Graphics at the specified uom-qualified 2D position offset relative to its original position.

8.2.1.6. Dot Graphics

REQUIREMENT 25	
IDENTIFIER	`/req/vector/dot`
A	An Encoding SHALL support defining a Dot Graphic inheriting from a base Shape class specifying a stroke, from which the color and the size of the point is inferred.
B	A Renderer SHALL support drawing the Graphics as a round dot using the width of its Stroke as the point size and in the color of the Stroke.

8.2.1.7. Image Graphics

REQUIREMENT 26	
IDENTIFIER	`/req/vector/image`
A	An Encoding SHALL support defining an Image Graphic inheriting from a base Shape class specifying a stroke, from which the color and the size of the point is inferred.
B	A Renderer SHALL support drawing the Graphics as a round dot using the width of its Stroke as the point size and in the color of the Stroke.

8.2.1.8. Text Graphics

REQUIREMENT 27	
IDENTIFIER	`/req/vector/text`
A	An Encoding SHALL support defining a Text Graphic with a `text` unicode string content, a `font`, and an `alignment` with a horizontal (`horzAlignment`: `left`, `right` or `center`) and vertical component (`vertAlignment`: `top`, `middle`, `bottom`).
B	A Renderer SHALL support drawing the `text` of a Text Graphic using the `font` and the `alignment` specified.

8.2.1.9. Fonts

REQUIREMENT 28	
IDENTIFIER	`/req/vector/fonts`
A	An Encoding SHALL support defining Fonts with a `face` property indicating the face name (also known as the font *family*), a `size` specified in *points*, a `bold` flag indicating to use a bold weight if true, and an `italic` flag indicating to use an italic style if true.
B	A Renderer SHALL support drawing text using the fonts specified.

9

REQUIREMENTS CLASSES FOR COVERAGE STYLING

9 REQUIREMENTS CLASSES FOR COVERAGE STYLING

9.1. Requirements Class “Basic Coverage Styling”

9.1.1. Overview

This requirements class adds support for defining how to portray coverage data by mapping values from the range (e.g., individual imagery bands) to color channels or to a single channel, as well as providing an option to apply a color and/or opacity map based on the values.

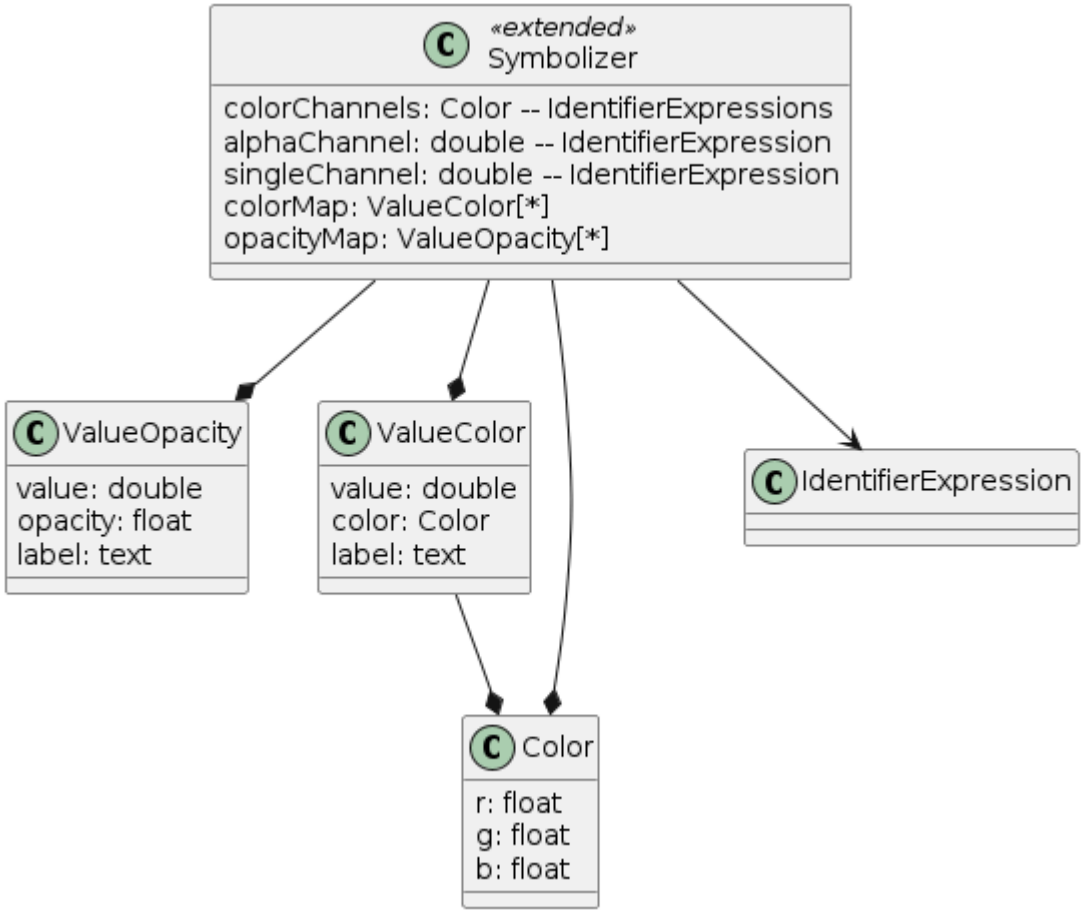

Figure 16 — Extended Symbolizer Class UML Diagram

9.1.1.1. Extended Identifiers

This requirements class defines as valid identifiers all of the accessible properties (the values of its range, as defined by the range type) for a coverage being portrayed with the name of those properties being the text of the identifier. Those identifiers are to be resolved with the value of that property. An encoding must support an escaping mechanism in cases where the name of a feature property might clash with a syntactic element.

9.1.1.2. Extended Symbolizer

Table 31 — Extended Symbolizer properties

NAME	DEFINITION	DATA TYPE AND VALUE	MULTIPLICITY
color Channels	Value of the Red, Green and Blue color channels	Color	0..1
alphaChannel	Value of the alpha (opacity) channel	float	0..1
singleChannel	Value of a single (gray-scale) output channel	float	0..1
colorMap	Color map to be applied based on single Cannel	ValueColor	0..*
opacityMap	Opacity map to be applied based on single Cannel	ValueOpacity	0..*

This requirements class requires support for identifier parameter value expression for the red (`r`), green (`g`) and blue (`b`) components of the `colorChannels`, `alphaChannel` and `singleChannel` properties without any additional requirements class.

This allows to map the range values of a coverage to the output color channels.

The color map is defined with a Value/Color pair object:

Table 32 — ValueColor properties

NAME	DEFINITION	DATA TYPE AND VALUE	MULTIPLICITY
value	Key value for selecting this color	float	1
color	Color associated with this value	Color	1
name	Label to use for this value when generating a legend	string	0..1

Similarly, the opacity map is defined with a Value/Opacity pair object:

Table 33 — ValueOpacity properties

NAME	DEFINITION	DATA TYPE AND VALUE	MULTIPLICITY
value	Key value for selecting this color	float	1
color	Color associated with this value	Color	1
name	Label to use for this value when generating a legend	string	0..1

9.1.1.3. Colors

Both the `colorChannels` and the `colorMap` properties use a Color class, specified using red, green and blue components. The ability to define Colors in other spaces (e.g., Lab, CMYK) is defined in separate requirements classes and applies wherever this basic RGB Color class is used.

Table 34 — Color

NAME	DEFINITION	DATA TYPE AND VALUE	MULTIPLICITY
r	Red	float	1
g	Green	float	1
b	Blue	float	1

9.1.2. Requirements

REQUIREMENTS CLASS 3: BASIC COVERAGE STYLING	
IDENTIFIER	`http://www.opengis.net/spec/symbology-1/2.0/req/coverage`
TARGET TYPE	Style encodings and Renderers
PREREQUISITE	Requirements class 1: `http://www.opengis.net/spec/symbology-1/2.0/req/core`

9.1.2.1. Symbolizer `colorChannels` property

REQUIREMENT 29	
IDENTIFIER	`/req/coverage/color-channels`
A	An Encoding SHALL support defining a `colorChannels` Symbolizer property as a Color object.
B	A Renderer SHALL support …

9.1.2.2. Symbolizer `alphaChannel` property

REQUIREMENT 30	
IDENTIFIER	`/req/coverage/alpha-channel`
A	An Encoding SHALL support defining a `alphaChannel` Symbolizer property as a float.
B	A Renderer SHALL support …

9.1.2.3. Symbolizer `singleChannel` property

REQUIREMENT 31	
IDENTIFIER	`/req/coverage/single-channel`
A	An Encoding SHALL support defining a `singleChannel` Symbolizer property as a float.
B	A Renderer SHALL support …

9.1.2.4. Symbolizer `colorMap` property

REQUIREMENT 32	
IDENTIFIER	`/req/coverage/color-map`
A	An Encoding SHALL support defining a `colorMap` Symbolizer property as an array of `value` (float) and `color` (Color) pairs which can be tagged with a `name`.
B	A Renderer SHALL support …

9.1.2.5. Symbolizer `opacityMap` property

REQUIREMENT 33	
IDENTIFIER	`/req/coverage/opacity-map`
A	An Encoding SHALL support defining a `opacityMap` Symbolizer property as an array of `value` (float) and `opacity` (float) pairs which can be tagged with a `name`.
B	A Renderer SHALL support …

9.2. Requirements Class “Hill Shading”

9.2.1. Overview

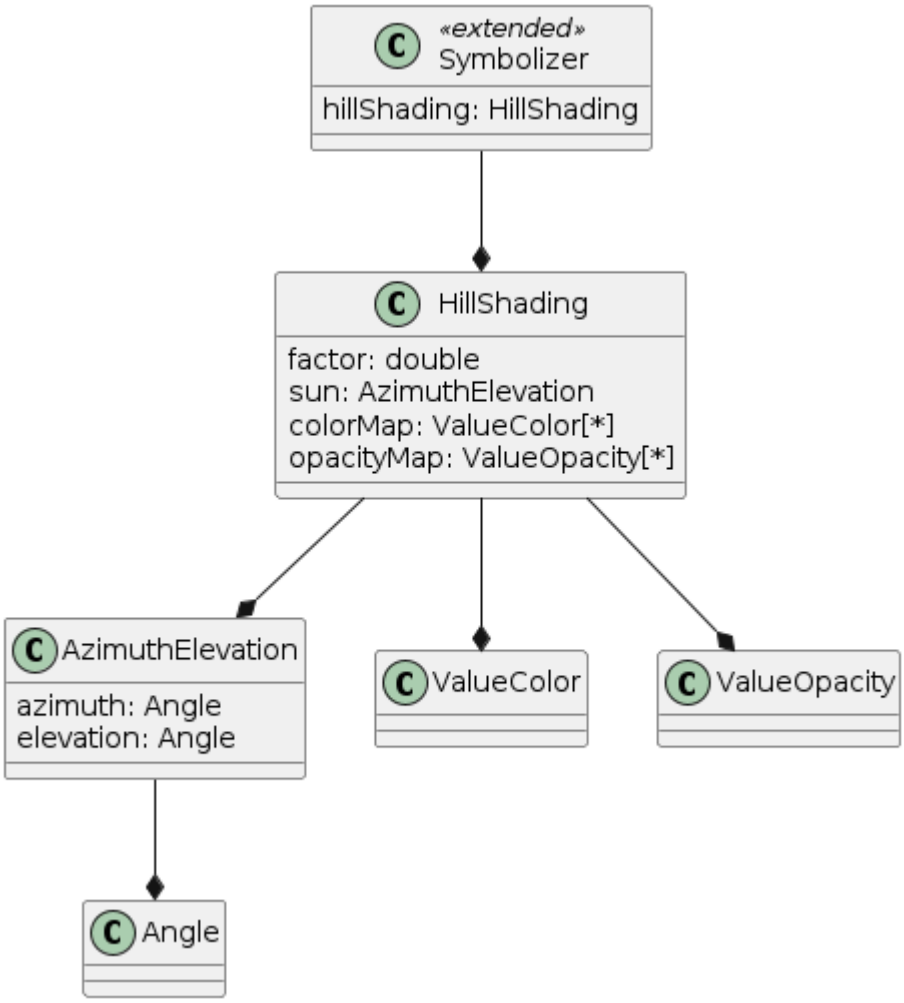

Figure 17 — Extended Symbolizer Class UML Diagram

This requirements classand adds the capability to portray elevation coverages with a hill shading style. It depends on the *Basic Coverage Styling* requirements class.

9.2.1.1. Extended Symbolizer

Table 35 — Extended Symbolizer properties

NAME	DEFINITION	DATA TYPE AND VALUE	MULTIPLICITY
hillShading	Hill shading parameters	HillShading	0..1

Table 36 — HillShading properties

NAME	DEFINITION	DATA TYPE AND VALUE	MULTIPLICITY
factor	Factor controlling the intensity of the shading	float	0..1
sun	Orientation of the sun	AzimuthElevation	0..1
colorMap	Color map to be applied based on calculated shade	ValueColor	0..*
opacity Map	Opacity map to be applied based on calculated shade	ValueOpacity	0..*

The `sun` property is specified using this AzimuthElevation class:

Table 37 — AzimuthElevation properties

NAME	DEFINITION	DATA TYPE AND VALUE	MULTIPLICITY
azimuth	Azimuth angle relative to North	Angle	0..1
elevation	Elevation angle relative to the ground	Angle	0..1

9.2.2. Requirements

REQUIREMENTS CLASS 4: HILL SHADING	
IDENTIFIER	`http://www.opengis.net/spec/symbology-1/2.0/req/hillshading`
TARGET TYPE	Style encodings and Renderers
PREREQUISITE	Requirements class 3: `http://www.opengis.net/spec/symbology-1/2.0/req/coverage`

9.2.2.1. Symbolizer `hillShading` property

REQUIREMENT 34	
IDENTIFIER	`/req/hillshading/hillshading`
A	An Encoding SHALL support defining a `hillshading` Symbolizer property as an object including `factor`, `sun`, `colorMap` and `opacityMap` properties.
B	A Renderer SHALL support …

9.2.2.2. HillShading `factor` property

REQUIREMENT 35	
IDENTIFIER	`/req/hillshading/factor`
A	An Encoding SHALL support defining a `factor` HillShading property as a float.
B	A Renderer SHALL support …

9.2.2.3. HillShading `sun` property

REQUIREMENT 36	
IDENTIFIER	`/req/hillshading/sun`
A	An Encoding SHALL support defining a `sun` HillShading property as an object consisting of `azimuth` and `elevation` angle members representing the orientation of the sun relative to the terrain.
B	A Renderer SHALL support …

9.2.2.4. HillShading `colorMap` property

REQUIREMENT 37	
IDENTIFIER	`/req/hillshading/color-map`
A	An Encoding SHALL support defining a `colorMap` HillShading property as an array of `value` (float) and `color` (Color) pairs.
B	A Renderer SHALL support …

9.2.2.5. HillShading `opacityMap` property

REQUIREMENT 38	
IDENTIFIER	`/req/hillshading/opacity-map`
A	An Encoding SHALL support defining a `opacityMap` HillShading property as an array of `value` (float) and `opacity` (float) pairs.
B	A Renderer SHALL support …

10

REQUIREMENTS CLASSES FOR LABELING

10 REQUIREMENTS CLASSES FOR LABELING

10.1. Requirements Class “Basic Labeling”

10.1.1. Overview

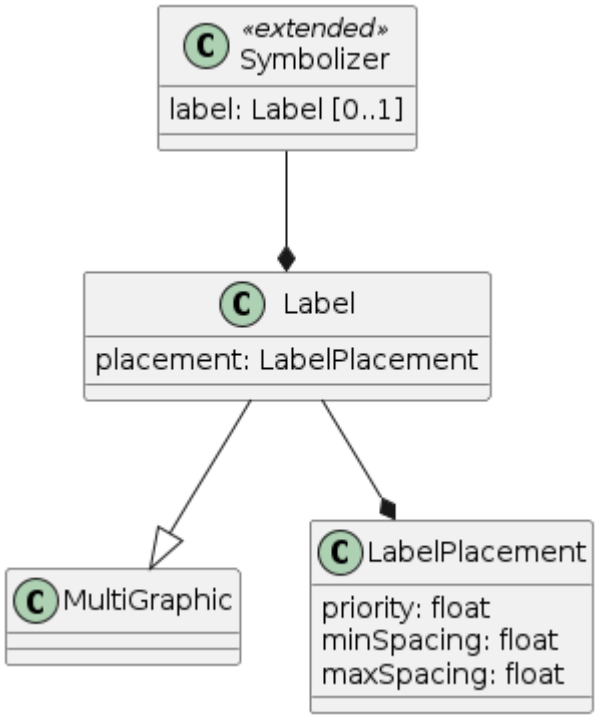

Figure 18 — Label Class UML Diagram

This requirements class adds support for labels that are placed using using a placement algorithm (and may not always be placed) drawn as a separate pass on top of styled vector features and coverages. Like Markers, Labels inherit from the MultiGraphic class and may contain multiple graphical elements.

Table 38 — Extended Symbolizer Class

NAME	DEFINITION	TYPE	MULTIPLICITY
label	Graphic placed and drawns	Label	0..1

10.1.2. Requirements

10.2. Requirements Class “Font Outlines”

10.2.1. Overview

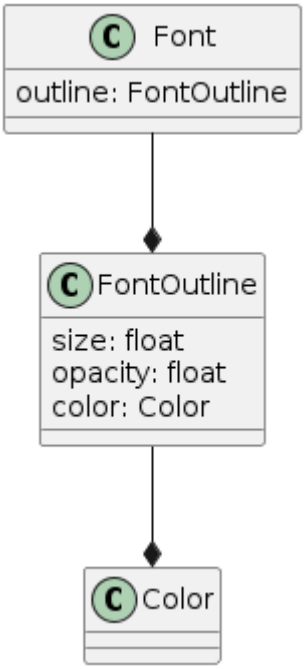

Figure 19 — Font outlines UML Diagram

This requirements class adds support for Text font outlines to improve readability on any background.

10.2.1.1. Font Outlines

Table 39 — Extended Text Class

NAME	DEFINITION	TYPE	MULTIPLICITY
outline	Outline for the text	FontOutline	0..1

Table 40 — FontOutline class

NAME	DEFINITION	DATA TYPE AND VALUE	MULTIPLICITY
size	Size of the outline	float	0..1
opacity	Opacity of the outline	float	0..1
color	Color of the outline	Color	0..1

10.2.2. Requirements

11

REQUIREMENTS CLASSES FOR SHAPES

11 REQUIREMENTS CLASSES FOR SHAPES

11.1. Requirements Class “Shape Graphics”

11.1.1. Overview

This requirements class adds support for vector shape graphics.

11.1.1.1. Rectangles, Circles, and Ellipses

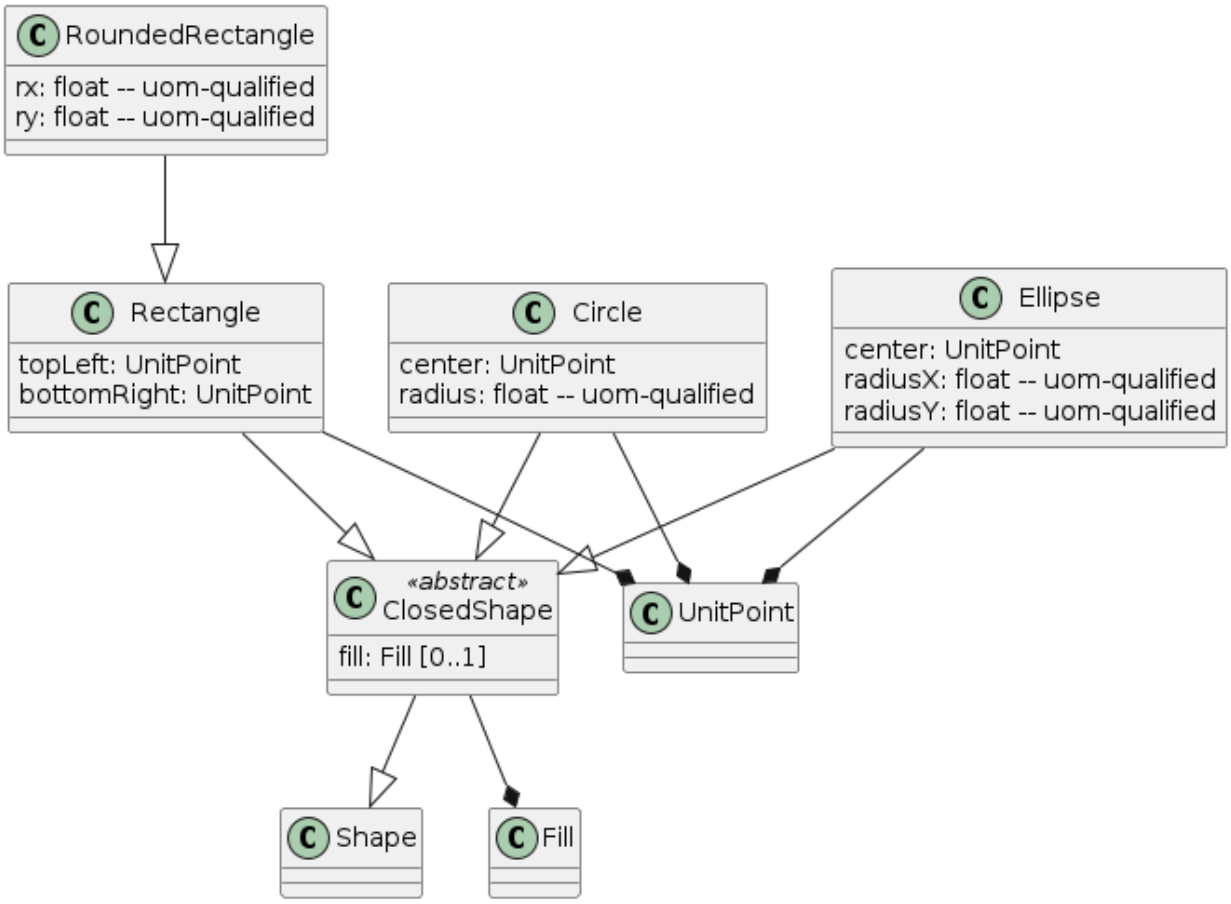

Figure 20 — Rectangles, Circles and Ellipses Classes UML Diagram

Table 41 — Class ClosedShape

NAME	DEFINITION	DATA TYPE AND VALUE	MULTIPLICITY
fill	Fill of the shape	Fill	1

11.1.1.1.1. Rectangles

Table 42 — Rectangle

NAME	DEFINITION	DATA TYPE AND VALUE	MULTIPLICITY
topLeft	Top left corner of the rectangle	UnitPoint	1
bottomRight	Bottom right corner of the rectangle	UnitPoint	1

11.1.1.1.2. Rounded Rectangles

Table 43 — Class RoundRectangle

NAME	DEFINITION	DATA TYPE AND VALUE	MULTIPLICITY
rx	Horizontal radius of the rounded corners	float (uom-qualified)	1
ry	Vertical radius of the rounded corners	float (uom-qualified)	1

11.1.1.1.3. Circles

Table 44 — Class Circle

NAME	DEFINITION	DATA TYPE AND VALUE	MULTIPLICITY
center	Center of the circle	UnitPoint	1
radius	Radius of the circle	float (uom-qualified)	1

11.1.1.1.4. Ellipses

Table 45 — Class Ellipse

NAME	DEFINITION	DATA TYPE AND VALUE	MULTIPLICITY
center	Center of the ellipse	UnitPoint	1
radiusX	Horizontal radius	float (uom-qualified)	1
radiusY	Vertical radius	float (uom-qualified)	1

11.1.1.2. Arcs

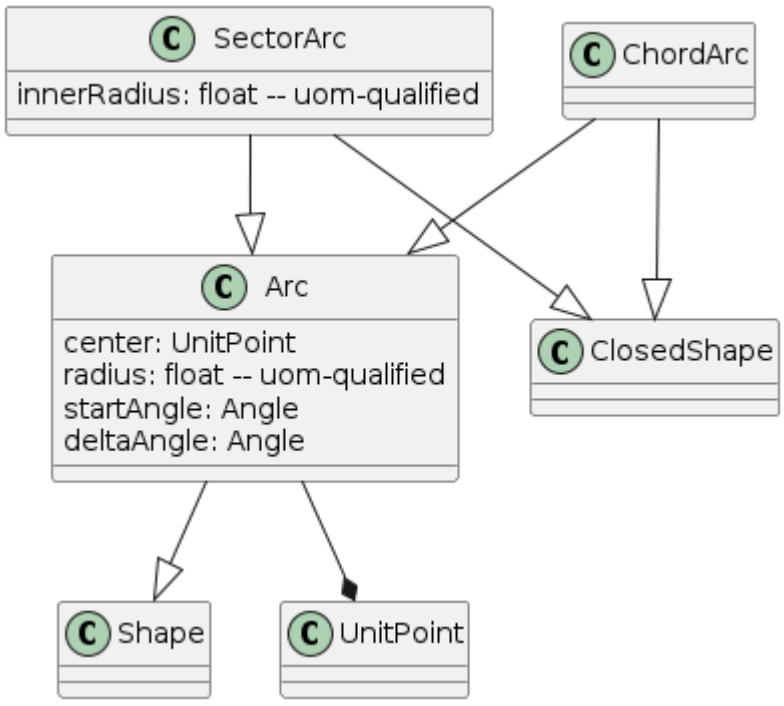

Figure 21 — Arc Classes UML Diagram

Table 46 — Arc class

NAME	DEFINITION	TYPE	MULTIPLICITY
center	Center	UnitPoint	1

NAME	DEFINITION	TYPE	MULTIPLICITY
radius	Radius	float (uom-qualified)	1
startAngle	Start angle	Angle	1
deltaAngle	Delta angle	Angle	1

Table 47 — SectorArc class

NAME	DEFINITION	TYPE	MULTIPLICITY
innerRadius	Inner radius	float (uom-qualified)	1

Table 48 — ChordArc class

NAME	DEFINITION	TYPE	MULTIPLICITY
(no additonal members)			

11.1.1.3. Paths

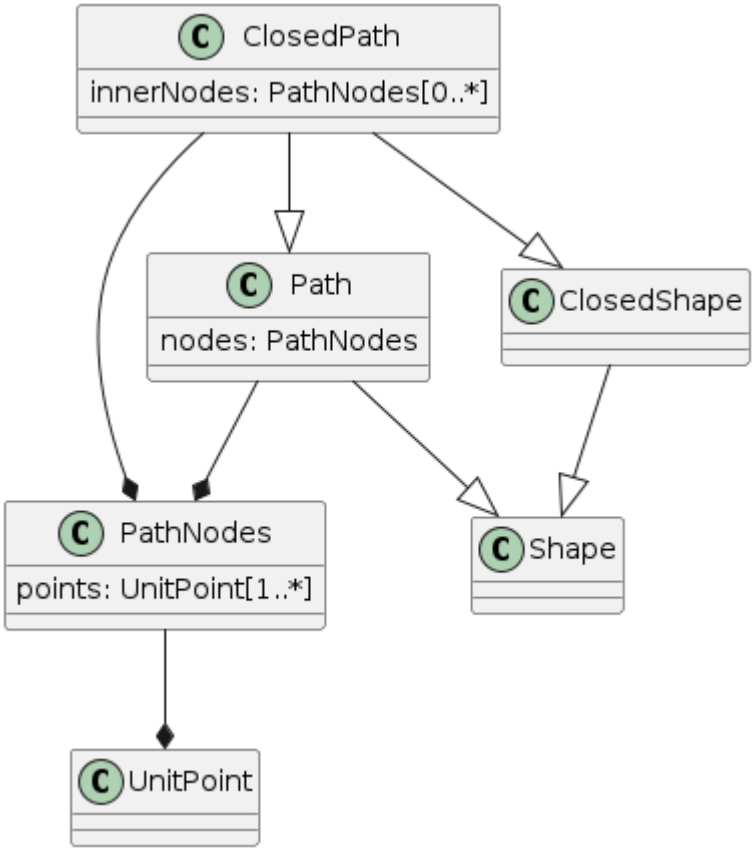

Figure 22 — Path Classe UML Diagram

Table 49 — Path class

NAME	DEFINITION	TYPE	MULTIPLICITY
nodes	Nodes of the path	PathNodes	1

11.1.1.3.1. Closed Paths

A closed path is a ClosedShape that can be filled, and can optionally contain inner nodes to create holes, like polygon interior rings.

Table 50 — ClosedPath class

NAME	DEFINITION	TYPE	MULTIPLICITY
innerNodes	Inner nodes of the path	PathNodes	0..*

11.1.2. Requirements

11.2. Requirements Class “Shapes Outlines”

11.2.1. Overview

Table 51 — ShapeOutline class

NAME	DEFINITION	TYPE	MULTIPLICITY
color	Color of the outline	Color	0..1
opacity	Opacity of the outline	float (0..1)	0..1
thickness	Thickness of the outline	float (uom-qualified)	0..1

Table 52 — Extended Shape class

NAME	DEFINITION	TYPE	MULTIPLICITY
outline	Shape outline	ShapeOutline	0..1

11.2.2. Requirements

12

REQUIREMENTS CLASS "MULTI GRAPHICS AND TRANSFORMS"

12 REQUIREMENTS CLASS "MULTI GRAPHICS AND TRANSFORMS"

12.1. Overview

This requirements class adds support for vector graphic hierarchies and transformations.

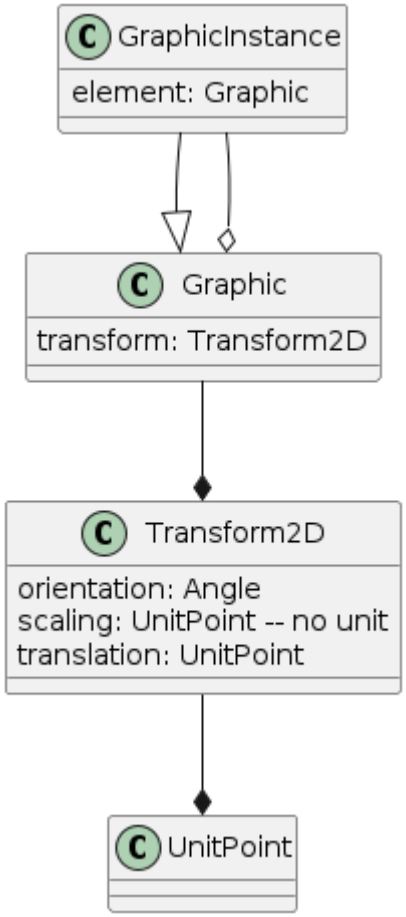

Figure 23 — MultiGraphic and Transforms UML Diagram

12.1.1. Graphic and MultiGraphic Extensions

This requirements class removes the restriction on MultiGraphics containing other MultiGraphics.

Table 53 — Extended Graphic class

NAME	DEFINITION	TYPE	MULTIPLICITY
transform	2D transformation	Transform2D	1

The `transform.translation` is equivalent to the Graphic's `position` property.

Table 54 — Transform2D class

NAME	DEFINITION	TYPE	MULTIPLICITY
orientation	Orientation	Angle	1
scaling	Scaling	UnitPoint (no unit)	1
translation	Translation	UnitPoint	1

12.1.2. Graphic Instances

The GraphicInstance class allows instantiating multiple instances of a single Graphic definition object, with each instance having a different tranformation applied.

Table 55 — GraphicInstance class

NAME	DEFINITION	TYPE	MULTIPLICITY
element	Instantiated graphic	Graphic	1

12.2. Requirements

13

REQUIREMENTS CLASSES FOR ADVANCED STROKES

13 REQUIREMENTS CLASSES FOR ADVANCED STROKES

13.1. Requirements Class “Joins and Caps”

13.1.1. Overview

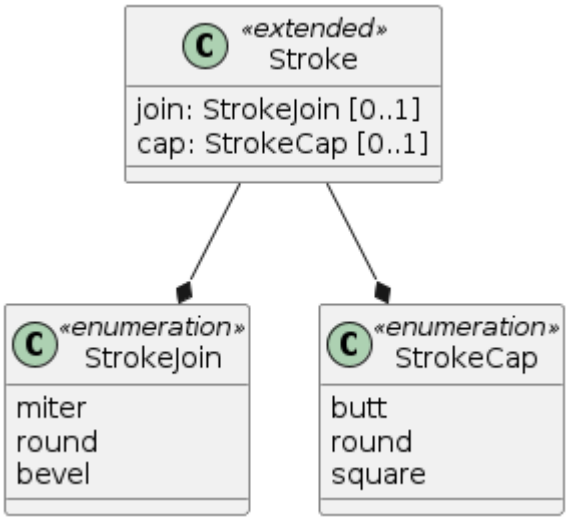

Figure 24 — Joins and Caps classes UML diagram

This requirements class adds support for Strokes rendered with additional join and cap types.

Table 56 — Extended Stroke class

NAME	DEFINITION	TYPE	MULTIPLICITY
join	Join	StrokeJoin	0..1
cap	Cap	StrokeCap	0..1

Table 57 — StrokeJoin enumeration

NAME	DEFINITION
miter	Miter join
round	Round join
bevel	Bevel join

Table 58 — StrokeCap enumeration

NAME	DEFINITION
butt	Butt cap
round	Round cap
square	Square cap

13.1.2. Requirements

13.2. Requirements Class “Dashes”

13.2.1. Overview

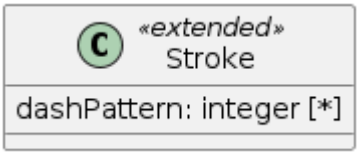

Figure 25 — Dashed strokes class UML diagram

This requirements class adds support for dashed Strokes.

Table 59 — Extended Stroke class

NAME	DEFINITION	TYPE	MULTIPLICITY
dashPattern	Dash pattern	int	0..*

13.2.2. Requirements

13.3. Requirements Class “Casing and Centerline”

13.3.1. Overview

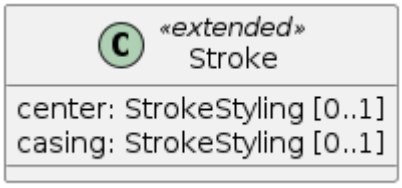

Figure 26 — Casing and centerline strokes class UML diagram

This requirements class adds support for Strokes with a casing and centerline.

Table 60 — Extended Stroke class

NAME	DEFINITION	TYPE	MULTIPLICITY
casing	Style of stroke casing	StrokeStyling	0..1
center	Style of stroke center line	StrokeStyling	0..1

13.3.2. Requirements

13.4. Requirements Class “Pattern Strokes”

13.4.1. Overview

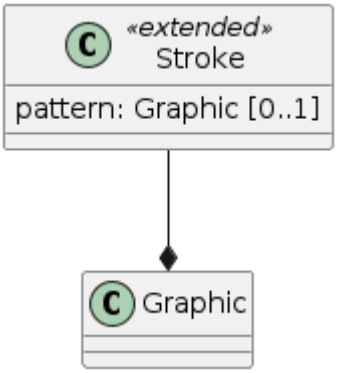

Figure 27 — Pattern strokes class UML Diagram

This requirements class adds support for Strokes with a pattern defined by a Graphic.

Table 61 — Extended Stroke class

NAME	DEFINITION	TYPE	MULTIPLICITY
pattern	Pattern graphic	Graphic	0..1

13.4.2. Requirements

14

REQUIREMENTS CLASSES FOR ADVANCED FILLS

14 REQUIREMENTS CLASSES FOR ADVANCED FILLS

14.1. Requirements Class “Pattern Fills”

14.1.1. Overview

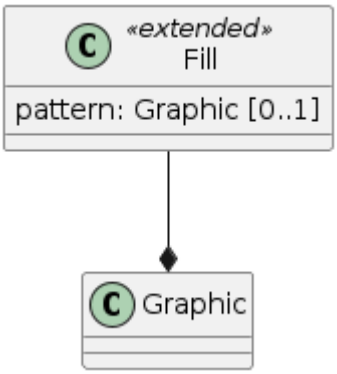

Figure 28 — Pattern Fill UML Diagram

This requirements class adds support for defining how to fill a shape with a pattern.

Table 62 — Extended Fill class

NAME	DEFINITION	TYPE	MULTIPLICITY
pattern	Graphic to repeat in pattern	Graphic	0..1

14.1.2. Requirements

14.2. Requirements Class “Hatches and Gradients, Stipples”

14.2.1. Overview

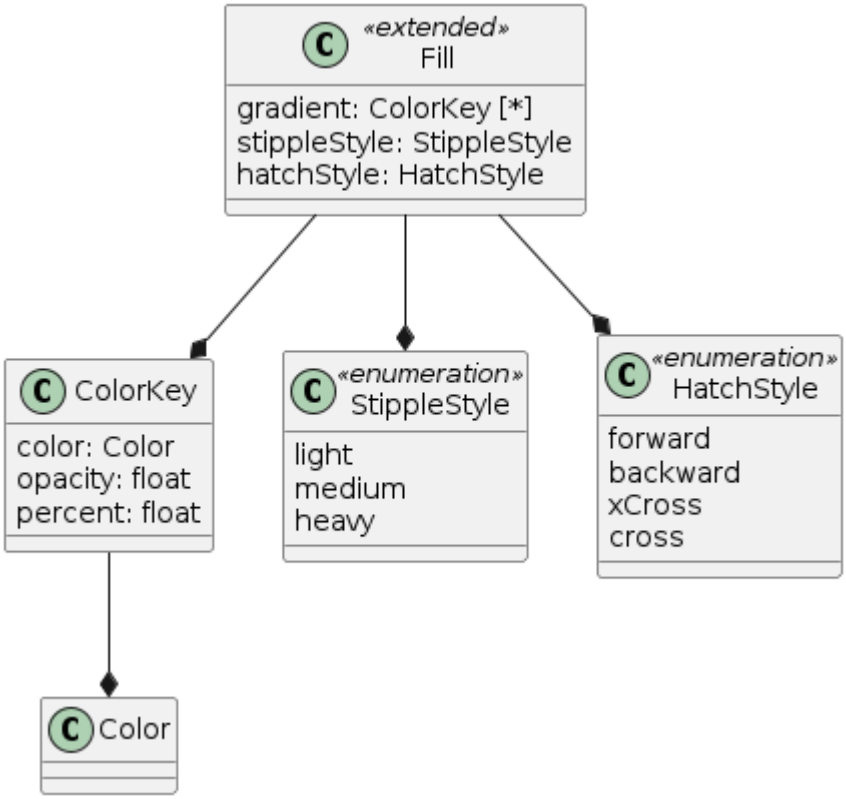

Figure 29 — Hatches, Gradients and Stipple Fills UML Diagram

This requirements class adds support for hatch, stipple and gradient fills.

Table 63 — Extended Fill class

NAME	DEFINITION	TYPE	MULTIPLICITY
hatchStyle	Hatch styl	HatchStyle	0..1

NAME	DEFINITION	TYPE	MULTIPLICITY
stippleStyle	Opacity	StippleStyle	0..1
gradient	Percent	ColorKey	0..*

14.2.1.1. Gradients

Table 64 — ColorKey class

NAME	DEFINITION	DATA TYPE AND VALUE	MULTIPLICITY
color	Color	Color	1
opacity	Opacity	float	0..1
percent	Percent	float	0..1

14.2.1.2. Stipples

Table 65 — StippleStyle class

NAME	DEFINITION	DATA TYPE AND VALUE	MULTIPLICITY
light	Light	float	1
medium	Medium	float	1
heavy	Heavy	float	1

14.2.1.3. Hatches

Table 66 — HatchStyle class

NAME	DEFINITION	DATA TYPE AND VALUE	MULTIPLICITY
forward	Forward	float	1

NAME	DEFINITION	DATA TYPE AND VALUE	MULTIPLICITY
backward	Backward	float	1
xCross	X-Cross	float	1
cross	Cross	float	1

14.2.2. Requirements

15

REQUIREMENTS CLASS "SYMBOLIZER PARAMETER VALUE EXPRESSIONS"

15 REQUIREMENTS CLASS "SYMBOLIZER PARAMETER VALUE EXPRESSIONS"

15.1. Overview

This requirements class adds support for using **ParameterValues** i.e., arbitrary **Expressions** (not only literals), for all **Symbolizer** properties.

15.2. Requirements

16

REQUIREMENTS CLASSES FOR ADDITIONAL EXPRESSIONS

16 REQUIREMENTS CLASSES FOR ADDITIONAL EXPRESSIONS

16.1. Requirements Class “Any right-hand operands”

16.1.1. Overview

This requirements class removes the restriction that right-hand side operands of RelationExpressions are limited to LiteralExpressions. For expressions encoded using CQL2, it is analogous to the *CQL2 Property-Property comparison* requirements class.

16.1.2. Requirements

16.2. Requirements Class “Conditional Expressions”

16.2.1. Overview

16.2.1.1. Conditional expressions

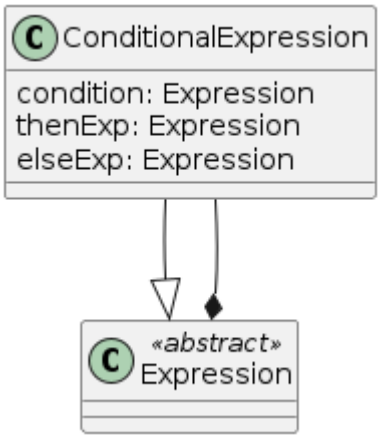

Figure 30 — Conditional Expressions UML Class Diagram

Table 67 — ConditionalExpression class

NAME	DEFINITION	TYPE	MULTIPLICITY
condition	Condition	Expression	1
thenExp	Expression resolving to when condition is true	Expession	1
elseExp	Expression resolving to when condition is false	Expession	1

16.3. Requirements Class “Variables”

16.3.1. Overview

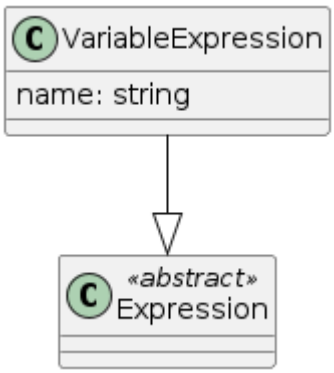

Figure 31 — Variable Expressions UML Class Diagram

This requirements class adds support for defining variables that can be used to facilitate re-use of definitions in encodings, as well as to provide configurable elements that can be associated with application control such as a slider or drop-down control.

16.3.1.1. Variable expressions

Table 68 — VariableExpression class

NAME	DEFINITION	TYPE	MULTIPLICITY
name	Name of the variable	string	1

16.3.2. Requirements

17

REQUIREMENTS CLASSES FOR ADDITIONAL OPERATORS

17 REQUIREMENTS CLASSES FOR ADDITIONAL OPERATORS

17.1. Requirements Class “Arithmetic Operators”

17.1.1. Overview

This requirements class adds support for arithmetic operations.

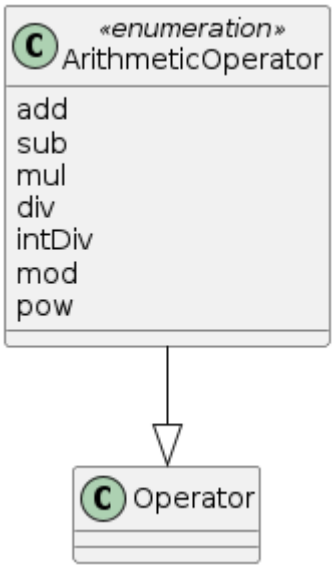

Figure 32 — Arithmetic Operators UML Diagram

Table 69 — arithmetic

NAME	DEFINITION	DATA TYPE AND VALUE	MULTIPLICITY
add	Addition	Numeric	1..1
sub	Subtraction	Numeric	1..1
mul	Multiplication	Numeric	1..1
div	Division	Numeric	1..1

NAME	DEFINITION	DATA TYPE AND VALUE	MULTIPLICITY
intDiv	Integer Division	Numeric	1..1
mod	Modulus	Numeric	1..1
pow	Power	Numeric	1..1

17.1.2. Requirements

17.2. Requirements Class “Bitwise Operators”

17.2.1. Overview

This requirements class adds support for operations on individual bits.

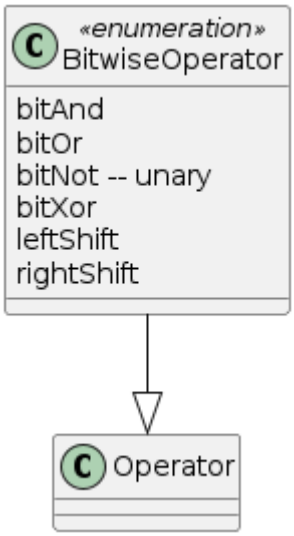

Figure 33 — Bitwise Operators UML Diagram

Table 70 — bitwise

NAME	DEFINITION	DATA TYPE AND VALUE	MULTIPLICITY
bitAnd	Bitwise AND	Integer	1..1

NAME	DEFINITION	DATA TYPE AND VALUE	MULTIPLICITY
bitOr	Bitwise OR	Integer	1..1
bitNot	Bitwise NOT	Integer	1..1
bitXor	Bitwise XOR	Integer	1..1
leftShift	Left Shift	Integer	1..1
rightShift	Right Shift	Integer	1..1

17.2.2. Requirements

17.3. Requirements Class “Text Relation Operators”

17.3.1. Overview

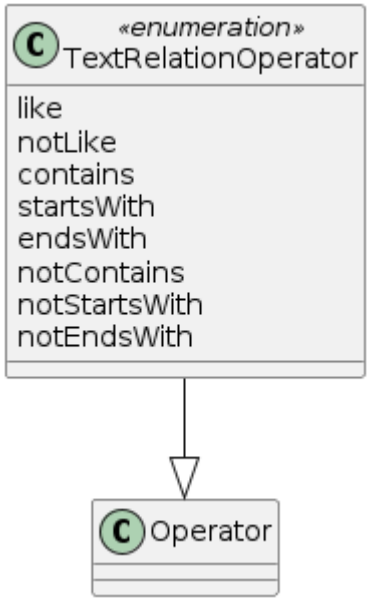

Figure 34 — Bitwise Operators UML Diagram

This requirements class adds more advanced support for evaluating the relation between text values, including the *like* operator, *starts with*, *contains*, *ends with*, as well as *not* variants for all these.

Table 71 — comparison

NAME	DEFINITION	DATA TYPE AND VALUE	MULTIPLICITY
like	Like	Text	1..1
notLike	Not Like	Text	1..1
contains	Contains	Text	1..1
startsWith	Starts With	Text	1..1
endsWith	Ends With	Text	1..1
notContains	Does Not Contain	Text	1..1
notStartsWith	Does Not Start With	Text	1..1
notEndsWith	Does Not End With	Text	1..1

17.3.2. Requirements

18

REQUIREMENTS CLASSES FOR FUNCTIONS

18 REQUIREMENTS CLASSES FOR FUNCTIONS

18.1. Requirements Class “Function Expressions”

18.1.1. Overview

This requirements class adds support for function call expressions.

18.1.1.1. Function call expressions

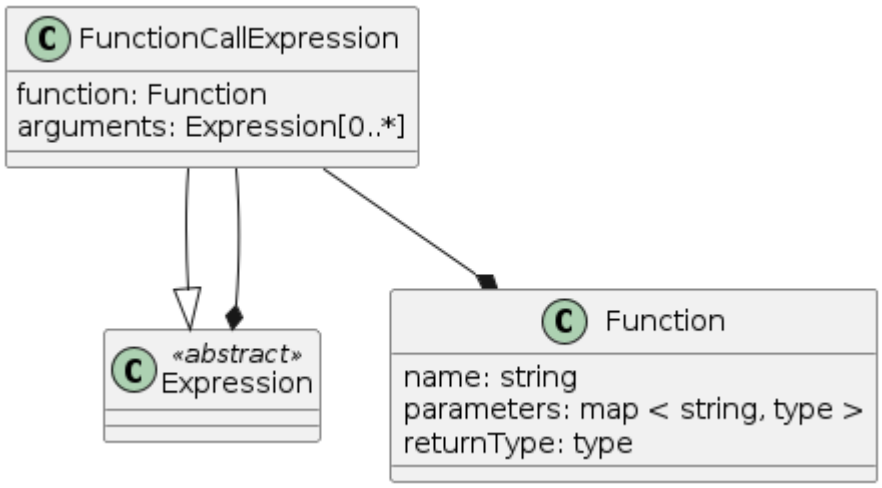

Figure 35 — Functions UML Class Diagram

Table 72 — FunctionCallExpression class

NAME	TYPE	DEFINITION	MULTIPLICITY
function	Function	Function to be called	1
arguments	Expression	Arguments to the function	0..*

Table 73 — Function class

NAME	TYPE	DEFINITION	MULTIPLICITY
name	string	Name	1
parameters	map < string, type >	Parameters	1
returnType	type	Return type	1

18.1.1.2. Standard functions

A standard function is a Function with a pre-determined set of parameters, return value and behavior that may be registered with a URI. Requirements class in this section defines standard functions.

Table 74 — Standard functions URI mapping

KEY	VALUE	DEFINITION
uri	Function	A URI allows to identify standard functiona, even if identical function names are used in different encoding, extension or implementations for different behaviors.

18.1.2. Requirements

18.2. Requirements Class “Spatial Relation Functions”

18.2.1. Overview

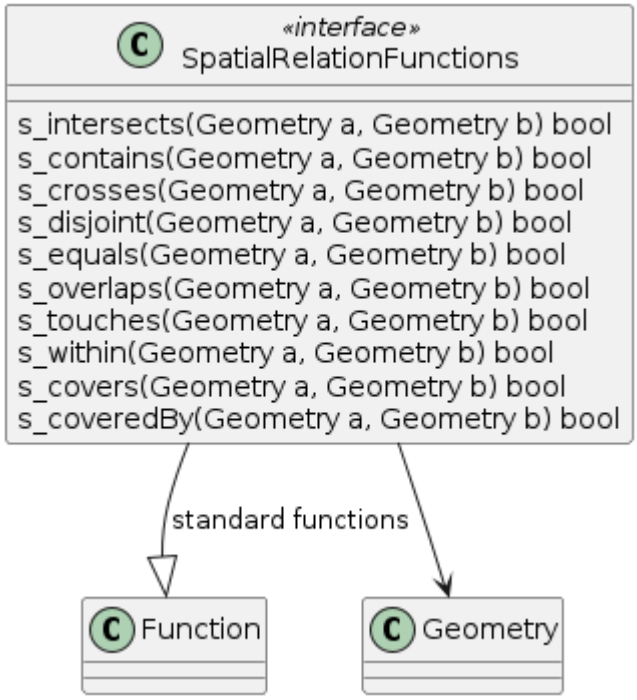

Figure 36 — Spatial Relation Functions UML Diagram

Table 75 — Spatial relation functions

NAME	PARAMETERS	RETURN TYPE	DEFINITION
s_intersects	s_intersects(Geometry a, Geometry b)	bool	
s_contains	s_contains(Geometry a, Geometry b)	bool	
s_crosses	s_crosses(Geometry a, Geometry b)	bool	
s_disjoint	s_disjoint(Geometry a, Geometry b)	bool	

NAME	PARAMETERS	RETURN TYPE	DEFINITION
s_equals	s_equals(Geometry a, Geometry b)	bool	
s_overlaps	s_overlaps(Geometry a, Geometry b)	bool	
s_touches	s_touches(Geometry a, Geometry b)	bool	
s_within	s_within(Geometry a, Geometry b)	bool	
s_covers	s_covers(Geometry a, Geometry b)	bool	
s_coveredBy	s_coveredBy(Geometry a, Geometry b)	bool	

NOTE: Although *s_covers()* and *s_coveredBy()* are not defined in *Simple Features Access (SFA)*, they are part of the *Dimensionally Extended-9 Intersection Model (DE-9IM)*. *covers()* is a more inclusive relation than *contains()* that does not distinguish between points in the boundary vs. in the interior of geometries, and for most situations is the correct relation to use. *coveredBy()* is a more inclusive relation than *within()*, and similarly is the correct relation to use for most situations. These relations are already implemented in several backends, but they can also be implemented using the *relate()* method defined in SFA using the following masks:

```
covers(a, b):
   relate(a, b, T*****FF*) or
   relate(a, b, *T****FF*) or
   relate(a, b, ***T**FF*) or
   relate(a, b, ****T*FF*)

coveredBy(a, b):
   relate(a, b, T*F**F***) or
   relate(a, b, *TF**F***) or
   relate(a, b, **FT*F***) or
   relate(a, b, **F*TF***)
```

Figure 37

18.2.1.1. Geometry System Identifiers

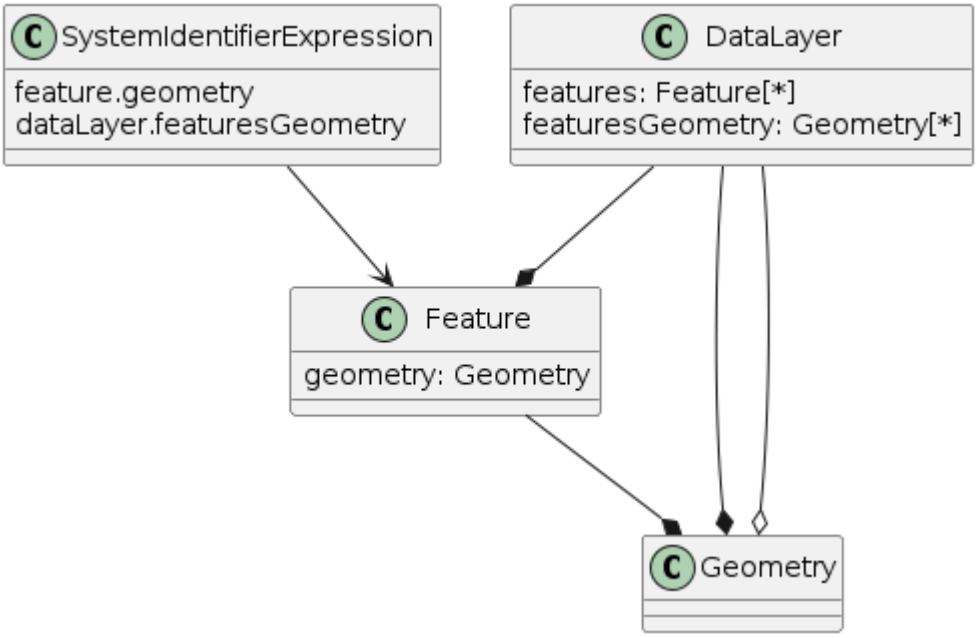

Figure 38 — Geometry System Identifiers UML Class Diagram

SpatialRelationFunctions are defined in the OGC Simple Feature Access — Part 1: Common Architecture specification and listed in the following table. They involve operations on geometries.

18.2.1.2. Geometry

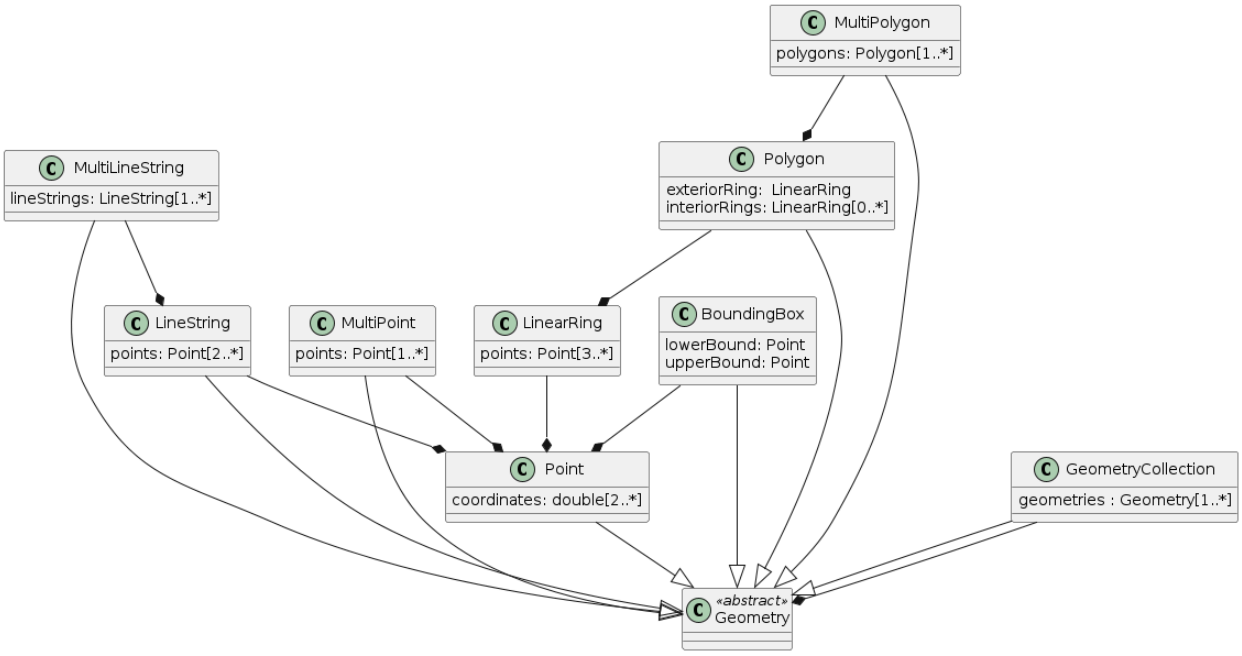

Figure 39 — Geometry Classes UML Diagram

18.2.1.2.1. Bounding boxes

Although not a feature geometry per se, a bounding box is a valid Geometry argument for the spatial relation and geometry manipulation functions, and can describe the envelope of a Geometry.

Table 76 — BoundingBox class

NAME	DEFINITION	TYPE	MULTIPLICITY
lowerBound	Point with smallest value for each coordinate	Point	1
upperBound	Point with greatest value for each coordinate	Point	1

18.2.1.2.2. Points

A Point is a 0-dimensional geometric object and represents a single location in coordinate space. A Point has an x-coordinate value, a y-coordinate value. If called for by the associated Spatial Reference System, it may also have coordinate values for z and m.

Table 77 — Point class

NAME	DEFINITION	TYPE	MULTIPLICITY
coordinates	coordinates	double	2..*

The boundary of a Point is the empty set.

A MultiPoint is a geometry made of one or more Points.

The elements of a MultiPoint are restricted to Points. The Points are not connected or ordered in any semantically important way (see the discussion at GeometryCollection). A MultiPoint is simple if no two Points in the MultiPoint are equal (have identical coordinate values in X and Y). Every MultiPoint is spatially equal to a simple Multipoint. The boundary of a MultiPoint is the empty set.

Table 78 — MultiPoint class

NAME	DEFINITION	TYPE	MULTIPLICITY
points	points	Point	2..*

18.2.1.2.3. Lines

A LineString is a curve with linear interpolation between Points. Each consecutive pair of Points defines a line segment.

Table 79 — LineString class

NAME	DEFINITION	TYPE	MULTIPLICITY
points	Points in this LineString	Point	2..*

A MultiLineString is a geometry made up of one or more LineStrings.

Table 80 — MultiLineString class

NAME	DEFINITION	TYPE	MULTIPLICITY
lineStrings	Line strings in this MultiLineString	LineString	1..*

18.2.1.2.4. Polygons

A Polygon is a planar surface defined by 1 exterior boundary and 0 or more interior boundaries.

Each interior boundary defines a hole in the Polygon.

Table 81 — Polygon class

NAME	DEFINITION	TYPE	MULTIPLICITY
exteriorRing	Exterior ring of the polygon	LinearRing	1
interiorRings	Interior rings of the polygon	LinearRing	0..*

The exterior boundary LinearRing defines the "top" of the surface which is the side of the surface from which the exterior boundary appears to traverse the boundary in a counter clockwise direction. The interior LinearRings will have the opposite orientation, and appear as clockwise when viewed from the "top".

The assertions for Polygons (the rules that define valid Polygons) are as follows:

- **a)** Polygons are topologically closed;
- **b)** The boundary of a Polygon consists of a set of LinearRings that make up its exterior and interior boundaries;
- **c)** No two Rings in the boundary cross and the Rings in the boundary of a Polygon may intersect at a Point but only as a tangent i.e.,

```
∀ P ∈ Polygon, ∀ c1,c2 ∈ P.Boundary(), c1≠c2,
∀ p, q ∈ Point, p, q ∈ c1, p ≠ q ,
[p ∈ c2] ⇒ [ ∃ δ > 0 ∋ [|p-q|<δ] ⇒ [q ∉ c2] ];
```

Figure 40

NOTE: This last condition says that at a point common to the two curves, nearby points cannot be common. This forces each common point to be a point of tangency.

- **d)** A Polygon may not have cut lines, spikes or punctures i.e.,

```
∀ P ∈ Polygon, P = P.Interior.Closure;
```

Figure 41

- **e)** The interior of every Polygon is a connected point set;
- **f)** The exterior of a Polygon with 1 or more holes is not connected. Each hole defines a connected component of the exterior.

In the above assertions, interior, closure and exterior have the standard topological definitions. The combination of (a) and © makes a Polygon a regular closed Point set. Polygons are simple geometric objects.

A Polygon is made of an exterior ring and zero or more interior rings.

A MultiPolygon is a Geometry made up of one or more Polygons.

Table 82 — MultiPolygon class

NAME	DEFINITION	TYPE	MULTIPLICITY
polygons	Polygons of the multipolygon	Polygon	1..*

The assertions for MultiPolygons are as follows.

- **a)** The interiors of 2 Polygons that are elements of a MultiPolygon may not intersect.

```
∀ M ∈ MultiPolygon, ∀ Pi, Pj ∈ M.Geometries(),i ≠ j,
Interior(Pi) ∩ Interior(Pj) = ∅;
```

Figure 42

- **b)** The boundaries of any 2 Polygons that are elements of a MultiPolygon may not "cross" and may touch at only a finite number of Points.

```
∀ M ∈ MultiPolygon, ∀ Pi, Pj ∈ M.Geometries(),
∀ ci, cj ∈ Curve ci ∈ Pi.Boundaries(), c j ∈ Pj.Boundaries()
∃ k ∈ Integer ∋ ci ∩ cj = { p1, ..., pk | pm ∈ Point, 0 < m < k };
```

Figure 43

Note: Crossing is prevented by assertion (a) above.

- **c)** A MultiPolygon is defined as topologically closed.
- **d)** A MultiPolygon may not have cut lines, spikes or punctures, a MultiPolygon is a regular closed Point set:

```
∀ M ∈ MultiPolygon, M = Closure(Interior(M))
```

Figure 44

- **e)** The interior of a MultiPolygon with more than 1 Polygon is not connected; the number of connected components of the interior of a MultiPolygon is equal to the number of Polygons in the MultiPolygon.

The boundary of a MultiPolygon is a set of closed curves corresponding to the boundaries of its element Polygons. Each curve in the boundary of the MultiPolygon is in the boundary of exactly

1 element Polygon, and every curve in the boundary of an element Polygon is in the boundary of the MultiPolygon.

18.2.1.2.5. Geometry collections

A GeometryCollection is a geometric object that is a collection of some number of geometric objects. All the elements in a GeometryCollection shall be in the same Spatial Reference System (the Spatial Reference System for the GeometryCollection).

Table 83 — GeometryCollection class

NAME	DEFINITION	TYPE	MULTIPLICITY
geometries	geometries in the collection	Geometry	1..*

18.2.2. Requirements

18.3. Requirements Class “Temporal Relation Functions”

18.3.1. Overview

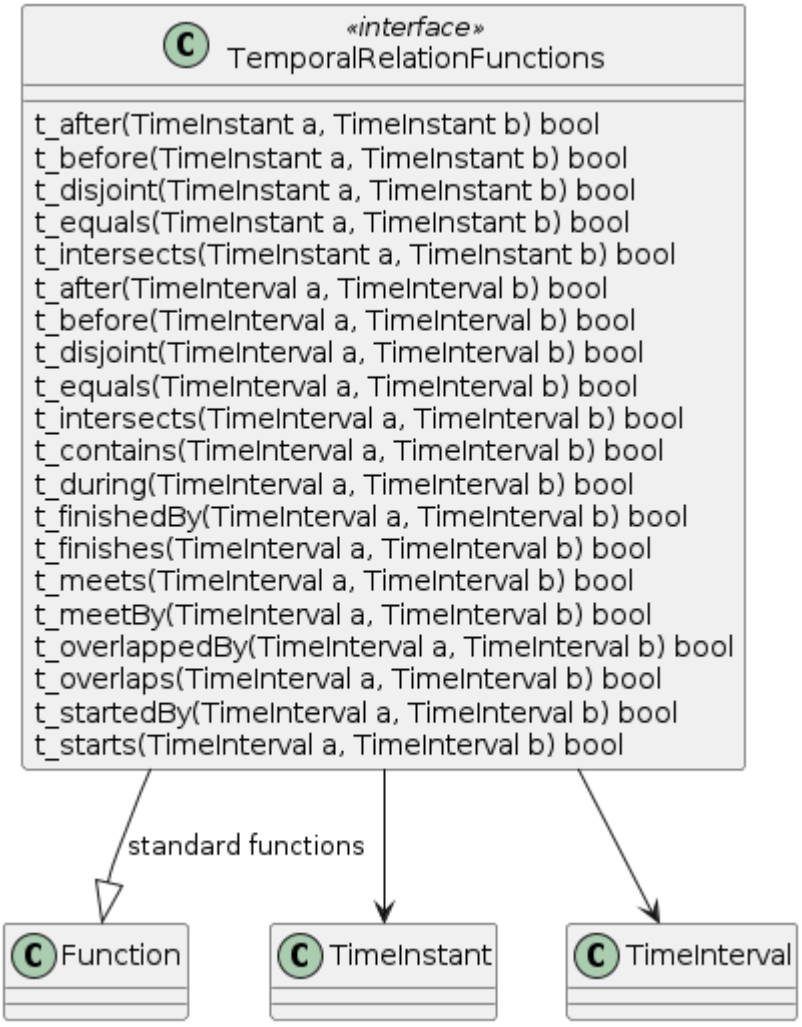

Figure 45 — Temporal Relation Functions UML Diagram

Table 84 — Temporal relation functions acting on time instants

NAME	PARAMETERS	RETURN TYPE	DEFINITION
t_after	(TimeInstant a, TimeInstant b)	bool	
t_before	(TimeInstant a, TimeInstant b)	bool	
t_disjoint	(TimeInstant a, TimeInstant b)	bool	
t_equals	(TimeInstant a, TimeInstant b)	bool	
t_intersects	(TimeInstant a, TimeInstant b)	bool	

Table 85 — Temporal relation functions acting on time intervals

NAME	PARAMETERS	RETURN TYPE	DEFINITION
t_after	(TimeInterval a, TimeInterval b)	bool	
t_before	(TimeInterval a, TimeInterval b)	bool	
t_disjoint	(TimeInterval a, TimeInterval b)	bool	
t_equals	(TimeInterval a, TimeInterval b)	bool	
t_intersects	(TimeInterval a, TimeInterval b)	bool	
t_contains	(TimeInterval a, TimeInterval b)	bool	
t_during	(TimeInterval a, TimeInterval b)	bool	
t_finishedBy	(TimeInterval a, TimeInterval b)	bool	
t_finishes	(TimeInterval a, TimeInterval b)	bool	
t_meets	(TimeInterval a, TimeInterval b)	bool	
t_meetBy	(TimeInterval a, TimeInterval b)	bool	
t_overlappedBy	(TimeInterval a, TimeInterval b)	bool	
t_overlaps	(TimeInterval a, TimeInterval b)	bool	

NAME	PARAMETERS	RETURN TYPE	DEFINITION
t_startedBy	(TimeInterval a, TimeInterval b)	bool	
t_starts	(TimeInterval a, TimeInterval b)	bool	

18.3.2. Requirements

18.4. Requirements Class “Array Relation Functions”

18.4.1. Overview

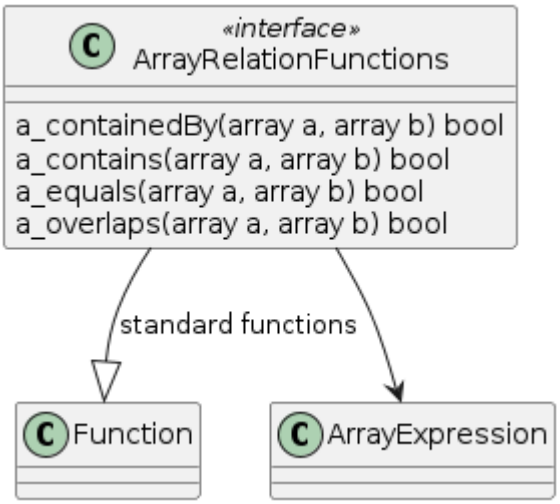

Figure 46 — Array Relation Functions UML Diagram

Table 86 — Array relation functions

NAME	PARAMETERS	RETURN TYPE	DEFINITION
a_containedBy	(array a, array b)	bool	
a_contains	(array a, array b)	bool	
a_equals	(array a, array b)	bool	
a_overlaps	(array a, array b)	bool	

18.4.2. Requirements

18.5. Requirements Class “Text Manipulation Functions”

18.5.1. Overview

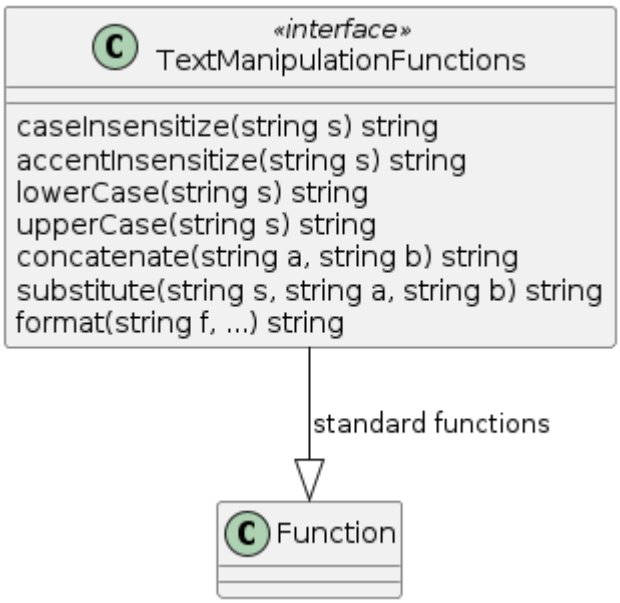

Figure 47 — Text Manipulation Functions UML Diagram

Table 87 — Text manipulation functions

NAME	PARAMETERS	RETURN TYPE	DEFINITION
caseInsensitize	(string s)	string	
accentInsensitize	(string s)	string	
lowerCase	(string s)	string	
upperCase	(string s)	string	
concatenate	(string a, string b)	string	

NAME	PARAMETERS	RETURN TYPE	DEFINITION
substitute	(string s, string a, string b)	string	
format	(string f, ...)	string	

18.5.2. Requirements

18.6. Requirements Class “Geometry Manipulation Functions”

18.6.1. Overview

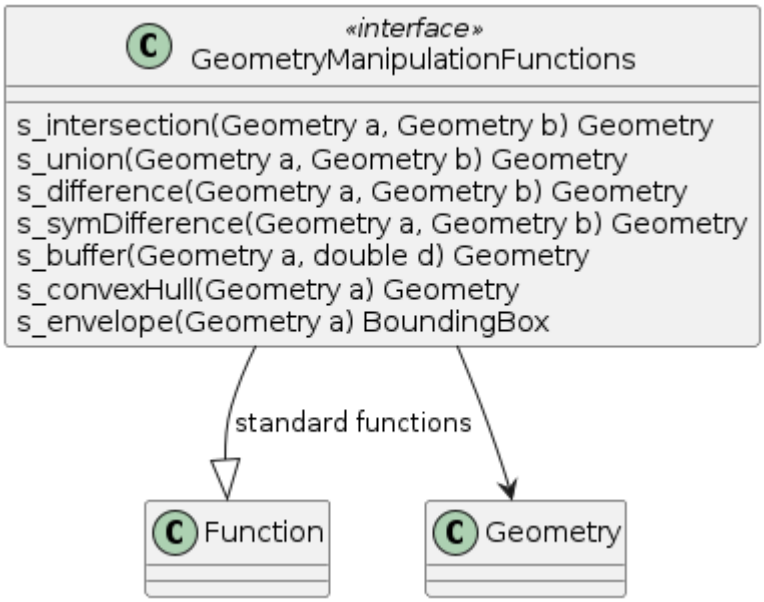

Figure 48 — Geometry Manipulation Functions UML Diagram

Table 88 — Geometry manipulation functions

NAME	PARAMETERS	RETURN TYPE	DEFINITION
s_intersection	(Geometry a, Geometry b)	Geometry	
s_union	(Geometry a, Geometry b)	Geometry	

NAME	PARAMETERS	RETURN TYPE	DEFINITION
s_difference	(Geometry a, Geometry b)	Geometry	
s_symDifference	(Geometry a, Geometry b)	Geometry	
s_convexHull	(Geometry a)	Geometry	
s_buffer	(Geometry a, double d)	Geometry	
s_envelope	(Geometry a)	BoundingBox	

18.6.2. Requirements

19

REQUIREMENTS CLASS "3D MODELS AND TRANSFORMS"

19 REQUIREMENTS CLASS “3D MODELS AND TRANSFORMS”

19.1. Overview

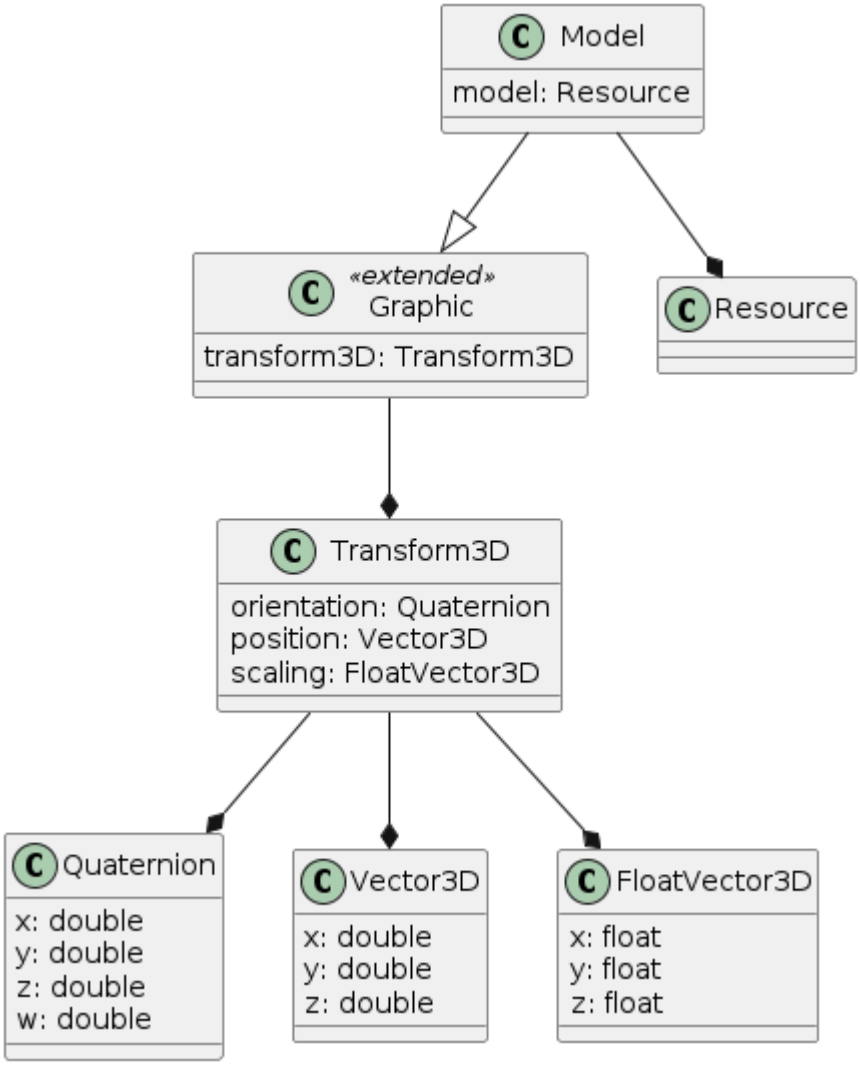

Figure 49

This requirements class adds support for 3D model Graphics as well as 3D transformations (position, orientation, and scaling) for Graphics.

Table 89 — Extended Graphic class

NAME	DEFINITION	TYPE	MULTIPLICITY
transform3D	3D Transformation	Transform3D	0..1

NOTE: The 2D position, orientation and scaling properties defined in the *MultiGraphics and Transform* requirements class correspond to their equivalent components in the 3D tranformation.

Table 90 — Model class

NAME	DEFINITION	TYPE	MULTIPLICITY
model	Resource for 3D model	Resource	1

Table 91 — Transform3D class

NAME	DEFINITION	DATA TYPE AND VALUE	MULTIPLICITY
orientation	orientation	Quaternion	1
position	position	Vector3D	1
scaling	scaling	FloatVector3D	1

Table 92 — Vector3D class

NAME	DEFINITION	DATA TYPE AND VALUE	MULTIPLICITY
x	x coordinate	double	1
y	y coordinate	double	1
z	z coordinate	double	1

Table 93 — FloatVector3D class

NAME	DEFINITION	DATA TYPE AND VALUE	MULTIPLICITY
x	x coordinate	float	1

NAME	DEFINITION	DATA TYPE AND VALUE	MULTIPLICITY
y	y coordinate	float	1
z	z coordinate	float	1

Table 94 — Quaternion class

NAME	DEFINITION	DATA TYPE AND VALUE	MULTIPLICITY
x	x coordinate	double	1
y	y coordinate	double	1
z	z coordinate	double	1
w	w coordinate	double	1

19.2. Requirements

20

REQUIREMENTS CLASS "JSON STYLES AND SYMBOLOGY"

20 REQUIREMENTS CLASS "JSON STYLES AND SYMBOLOGY"

20.1. Overview

This requirements class defines a JSON encoding for the Styles and Symbology Logical Model. The primary objective of this encoding is to be readily parsable and writable by machines using a standard JSON parser. Styles encoded with this encoding are not easily hand-edited by cartographers.

The encoding of expressions for both selectors and symbolizer parameter values in this encoding is based on the CQL2-JSON encoding.

In order to represent a partial update to an object property of a symbolizer, an object with all the modified properties is specified, as well as setting the `"alter"` property to `true`. For example, the following is equivalent to the *CartoSym CSS* representation of `fill.color.r: 100`, and will preserve the values of the other fill properties as well as the green and blue component of the fill color set by earlier rules or specified by defaults:

```
"fill": {
   "alter": true,
   "color": {
      "alter": true,
      "r": 100
   }
}
```

Figure 50

A similar mechanism is also defined to modify only some elements of an array by using an object with `"index"` set to the zero-based array index, and `"value"` set to new value with which to update that array element. This mechanism can be combined with the partial update of objects to only modify some properties of an object array element, or to partially modify an array inside of an object property. For example, the following is equivalent to the *CartoSym CSS* representation of `marker.elements[1]: Text { text: "Marker" }`:

```
"marker": {
   "alter": true,
   "elements": {
      "index": 1,
      "value": { "type": "Text", "text": "Marker" }
   }
}
```

Figure 51

Variables are encoded as JSON Pointer to a top-level `"$variables"` section in the same source file, or in an included file e.g., `{ "$ref": "#/$variables/myVariable" }`. Base style sheets

can be included using an “$include” property in the top-level object followed by either a single relative file path string or an array of relative file path strings to include. The styling rules from the included files will be included before any styling rules defined by the current file, allowing to override the symbolization fully or partially.

20.1.1. Example

The following example demonstrates the CartoSym JSON encoding for the same example Basic Vector Styling for polygons equivalent to the one seen in the following CartoSym CSS section (Example 1 in 21.1.8.1. Basic Vector Styling).

```
{
   "metadata":
   {
      "title": "Styling polygon vector features",
      "abstract": "Basic vector features styling (polygons)"
   },
   "stylingRules": [
      {
         "selector":
         {
            "op": "and",
            "args": [
               { "op": "=", "args": [ { "sysId": "dataLayer.id" }, "Landuse" ]
},
               { "op": "=", "args": [ { "sysId": "dataLayer.type" }, "vector" ]
 },
               { "op": "=", "args": [ { "sysId": "dataLayer.
featuresGeometryDimensions" }, 2 ] }
            ]
         },
         "symbolizer":
         {
            "$comment": "Do not show Landuse layer by default",
            "visibility": false
         },
         "nestedRules": [
            {
               "$comment": "Show land use if zoomed in more than 1:200,000 for
data valid within visualization's selected time range",
               "selector":
               {
                  "op": "and",
                  "args": [
                     { "op": "<", "args": [ { "sysId": "vis.id" }, 200000 ] },
                     { "op": ">=", "args": [ { "property": "validDate" }, {
"sysId": "vis.timeInterval.start.date" } ] },
                     { "op": "<=", "args": [ { "property": "validDate" }, {
"sysId": "vis.timeInterval.end.date" } ] }
                  ]
               },
               "symbolizer":
               {
                  "visibility": true,
                  "opacity": 0.8,
                  "zOrder": 1,
                  "fill": { "color": "gray", "opacity": 0.5 },
                  "stroke": { "color": "gray", "width": { "px": 2.0 },
"opacity": 1.0 }
```

```
                },
                "nestedRules": [
                   {
                      "$comment": "Select different fill and stroke color based
on FunctionCode property",
                      "selector": { "op": "=", "args": [ { "property":
"FuntionCode" }, "parking" ] },
                      "symbolizer":
                      {
                         "fill": { "alter": true, "color": "darkGray" },
                         "stroke": { "alter": true, "color": [ 32, 32, 32 ] }
                      }
                   },
                   {
                      "selector": { "op": "=", "args": [ { "property":
"FuntionCode" }, "park" ] },
                      "symbolizer":
                      {
                         "fill": { "alter": true, "color": "darkGreen" },
                         "stroke": { "alter": true, "color": "green" }
                      }
                   },
                   {
                      "selector": { "op": "=", "args": [ { "property":
"FuntionCode" }, "commercial" ] },
                      "symbolizer":
                      {
                         "fill": { "alter": true, "color": "lightGray" },
                         "stroke": { "alter": true, "color": "lightGray" }
                      }
                   },
                   {
                      "$comment": "If zoomed in more than 1:10,000",
                      "selector": { "op": "<", "args": [ { "sysId": "vis.sd" },
10000 ] },
                      "symbolizer":
                      {
                         "$comment": "Change stroke width to 4 pixels and add a
text marker (positioned at centroid + horizontal offset) showing FunctionTitle
property",
                         "stroke": { "alter": true, "width": { "px": 4.0 } },
                         "marker": {
                            "elements": [
                               {
                                  "type": "Text",
                                  "position": [20, 0],
                                  "$comment": "Offset 20 pixels to the right",
                                  "text": { "property": "FunctionTitle" },
                                  "alignment": [ "left", "top" ],
                                  "font": {
                                     "face": "Arial",
                                     "size": 14,
                                     "bold": true,
                                     "italic": true,
                                     "opacity": 1.0,
                                     "color": [ 0, 0, 0 ]
                                  }
                               }
                            ]
                         }
                      },
                      "nestedRules": [
                         {
```

```
                        "$comment": "Add icons at centroid based on land
use function code property",
                        "selector": { "op": "=", "args": [ { "property":
"FuntionCode" }, "parking" ] },
                        "symbolizer":
                        {
                           "marker": {
                              "alter": true,
                              "elements":
                              {
                                 "index": 1,
                                 "value": {
                                    "type": "Image",
                                    "image": {
                                       "uri": "http://example.com/
parkingIcon",
                                       "path": "parkingIcon.png",
                                       "id": "parking",
                                       "type": "image/png",
                                       "ext": "png"
                                    },
                                    "hotSpot": [ { "pc": 50 }, { "pc": 50 }
],
                                    "tint": "white",
                                    "blackTint": "blue",
                                    "alphaThreshold": 0.1
                                 }
                              }
                           }
                        },
                        {
                        "$comment": "Add icons at centroid based on land
use function code property",
                        "selector": { "op": "=", "args": [ { "property":
"FuntionCode" }, "park" ] },
                        "symbolizer":
                        {
                           "marker": {
                              "alter": true,
                              "elements": {
                                 "index": 1,
                                 "value": {
                                    "type": "Image",
                                    "image": {
                                       "uri": "http://example.com/park",
                                       "path": "park.png",
                                       "id": "park",
                                       "type": "image/png",
                                       "ext": "png"
                                    },
                                    "hotSpot": [ { "pc": 50 }, { "pc": 50 }
],
                                    "tint": "white",
                                    "blackTint": "blue",
                                    "alphaThreshold": 0.1
                                 }
                              }
                           }
                        },
                        {
```

```
                        "$comment": "Add icons at centroid based on land
use function code property",
                        "selector": { "op": "=", "args": [ { "property":
"FuntionCode" }, "commercial" ] },
                        "symbolizer":
                        {
                           "marker": {
                              "alter": true,
                              "elements": {
                                 "index": 1,
                                 "value": {
                                    "type": "Image",
                                    "image": {
                                       "uri": "http://example.com/
commercial",
                                       "path": "commercial.png",
                                       "id": "commercial",
                                       "type": "image/png",
                                       "ext": "png"
                                    },
                                    "hotSpot": [ { "pc": 50 }, { "pc": 50 }
],
                                    "tint": "white",
                                    "blackTint": "blue",
                                    "alphaThreshold": 0.1
                                 }
                              }
                           }
                        }
                     }
                  ]
               }
            ]
         }
      ]
}
```

20.1.2. JSON Schema

The following JSON Schema describes the encoding.

```
{
  "$schema": "https://json-schema.org/draft/2019-09/schema",
  "$ref": "#/$defs/style",
  "$defs": {
    "style": {
      "type": "object",
      "required": [ "stylingRules" ],
      "properties": {
         "$comment": { "type": "string" },
         "$include": {
            "oneOf": [
               { "type": "string" },
               { "type": "array", "items": { "type": "string" }, "minItems": 1
}
            ]
         },
         "metadata": {
            "type": "object",
```

```
            "properties":
            {
               "$comment": { "type": "string" },
               "title": { "type": "string" },
               "abstract": { "type": "string" },
               "authors": { "type": "array", "items": { "type": "string" } },
               "geoDataClasses": { "type": "array", "items": { "type":
"string", "format": "uri" } }
            }
         },
         "stylingRules": {
            "type": "array",
            "items": { "$ref": "#/$defs/stylingRule" }
         },
         "$variables": { "type": "object" }
      }
    },
    "stylingRule": {
      "type": "object",
      "properties":
      {
         "$comment": { "type": "string" },
         "selector": { "$ref": "#/$defs/boolExpression" },
         "symbolizer": { "$ref": "#/$defs/symbolizer" },
         "nestedRules":
         {
            "type": "array",
            "items": { "$ref": "#/$defs/stylingRule" }
         }
      }
    },
    "symbolizer": {
      "type": "object",
      "properties":
      {
         "$comment": { "type": "string" },

         "visibility": { "$ref": "#/$defs/boolExpression" },
         "opacity": { "$ref": "#/$defs/opacity" },
         "zOrder": { "$ref": "#/$defs/numericExpression" },

         "fill": { "$ref": "#/$defs/fill" },
         "stroke": { "$ref": "#/$defs/stroke" },
         "marker": { "$ref": "#/$defs/marker" },
         "label": { "$ref": "#/$defs/label" }
      }
    },
    "fill":
    {
       "oneOf": [
         { "$ref": "#/$defs/idOrFnExpression" },
         {
            "type": "object",
            "properties":
            {
               "alter": { "type": "boolean" },
               "color": { "$ref": "#/$defs/color" },
               "opacity": { "$ref": "#/$defs/opacity" }
            }
         }
       ]
    },
    "stroke":
```

```
{
   "oneOf": [
     { "$ref": "#/$defs/idOrFnExpression" },
     {
        "allOf": [
           { "$ref": "#/$defs/strokeStyling" },
           {
              "type": "object",
              "properties":
              {
                 "casing": { "$ref": "#/$defs/strokeStyling" },
                 "centerLine": { "$ref": "#/$defs/strokeStyling" }
              }
           }
        ]
     }
   ]
},
"marker": { "$ref": "#/$defs/multiGraphic" },
"label": {
   "allOf": [
      { "$ref": "#/$defs/multiGraphic" },
      {
         "type": "object",
         "properties":
         {
            "placement": { "$ref": "#/$defs/labelPlacement" }
         }
      }
   ]
},
"multiGraphic": {
   "oneOf": [
     { "$ref": "#/$defs/idOrFnExpression" },
     {
        "allOf": [
           { "$ref": "#/$defs/abstractGraphic" },
           {
              "type": "object",
              "required": [ "elements" ],
              "properties":
              {
                 "elements": { "$ref": "#/$defs/graphicArray" }
              }
           }
        ]
     }
   ]
},
"graphicArray": {
   "oneOf": [
     { "$ref": "#/$defs/arrayExpression" },
     {
        "type": "array",
        "items": { "$ref": "#/$defs/graphic" }
     },
     {
        "type": "object",
        "required": [ "index", "value" ],
        "properties": {
           "index": { "type": "integer", "minimum": 0 },
           "value": { "$ref": "#/$defs/graphic" }
        }
```

```
      }
    ]
  },
  "abstractGraphic": {
    "type": "object",
    "properties": {
       "alter": { "type": "boolean" },
       "position": { "$ref": "#/$defs/unitPoint" },
       "opacity": { "$ref": "#/$defs/opacity" }
    }
  },
  "graphic": {
    "oneOf": [
      { "$ref": "#/$defs/idOrFnExpression" },
      { "$ref": "#/$defs/image" },
      { "$ref": "#/$defs/shape" },
      { "$ref": "#/$defs/text" },
      { "$ref": "#/$defs/multiGraphic" }
    ]
  },
  "image": {
    "allOf": [
      { "$ref": "#/$defs/abstractGraphic" },
      {
          "type": "object",
          "required": [ "type", "image" ],
          "properties":
          {
             "type": { "type": "string", "enum": [ "Image" ] },
             "image": { "$ref": "#/$defs/resource" },
             "hotSpot": { "$ref": "#/$defs/unitPoint" },
             "tint": { "$ref": "#/$defs/color" },
             "blackTint": { "$ref": "#/$defs/color" },
             "alphaThreshold": { "$ref": "#/$defs/opacity" }
          }
      }
    ]
  },
  "text": {
    "allOf": [
      { "$ref": "#/$defs/abstractGraphic" },
      {
          "type": "object",
          "required": [ "type", "text" ],
          "properties":
          {
             "type": { "type": "string", "enum": [ "Text" ] },
             "text": { "$ref": "#/$defs/characterExpression" },
             "font": { "$ref": "#/$defs/font" },
             "alignment": { "$ref": "#/$defs/textAlignment" }
          }
      }
    ]
  },
  "textAlignment": {
    "oneOf": [
       { "$ref": "#/$defs/idOrFnExpression" },
       {
          "type": "array",
          "minItems": 2,
          "maxItems": 2,
          "prefixItems": [
             { "$ref": "#/$defs/hAlignment" },
```

```
            { "$ref": "#/$defs/vAlignment" }
          ]
      },
      {
         "type": "object",
         "properties":
         {
           "hAlignment": { "$ref": "#/$defs/hAlignment" },
           "vAlignment": { "$ref": "#/$defs/vAlignment" },
           "alter": { "type": "boolean" }
         }
      }
   ]
},
"hAlignment": {
   "oneOf": [
      { "$ref": "#/$defs/idOrFnExpression" },
      {
         "type": "string",
         "enum": [ "left", "center", "right" ]
      }
   ]
},
"vAlignment": {
   "oneOf": [
      { "$ref": "#/$defs/idOrFnExpression" },
      {
        "type": "string",
        "enum": [ "top", "middle", "bottom" ]
      }
   ]
},
"font": {
   "oneOf": [
      { "$ref": "#/$defs/idOrFnExpression" },
      {
         "type": "object",
         "properties": {
            "alter": { "type": "boolean" },
            "face": { "$ref": "#/$defs/characterExpression" },
            "size": { "$ref": "#/$defs/numericExpression" },
            "bold": { "$ref": "#/$defs/boolExpression" },
            "italic": { "$ref": "#/$defs/boolExpression" },
            "underline": { "$ref": "#/$defs/boolExpression" },
            "color": { "$ref": "#/$defs/color" },
            "opacity": { "$ref": "#/$defs/opacity" }
         }
      }
   ]
},
"abstractShape": {
  "allOf": [
     { "$ref": "#/$defs/abstractGraphic" },
     {
        "type": "object",
        "required": [ "type" ],
        "properties": {
           "type": { "type": "string" },
           "stroke": { "$ref": "#/$defs/stroke" }
        }
     }
  ]
},
```

```
"shape": {
  "oneOf": [
     { "$ref": "#/$defs/dot" },
     { "$ref": "#/$defs/arc" },
     { "$ref": "#/$defs/path" },
     { "$ref": "#/$defs/rectangle" },
     { "$ref": "#/$defs/circle" },
     { "$ref": "#/$defs/ellipse" },
     { "$ref": "#/$defs/sectorArc" },
     { "$ref": "#/$defs/chordArc" },
     { "$ref": "#/$defs/closedPath" }
  ]
},
"closedShape": {
  "allOf": [
     { "$ref": "#/$defs/abstractShape" },
     {
        "type": "object",
        "properties": {
           "fill": { "$ref": "#/$defs/fill" }
        }
     }
  ]
},
"dot": {
  "allOf": [
     { "$ref": "#/$defs/abstractShape" },
     {
      "type": "object",
      "required": [ "type" ],
      "properties":
      {
         "type": { "type": "string", "enum": [ "Dot" ] }
      }
     }
   ]
},
"abstractArc": {
  "allOf": [
     { "$ref": "#/$defs/abstractShape" },
     {
      "type": "object",
      "required": [ "startAngle", "deltaAngle", "radius" ],
      "properties":
      {
         "center": { "$ref": "#/$defs/unitPoint" },
         "radius": { "$ref": "#/$defs/unitValue" },
         "startAngle": { "$ref": "#/$defs/angle" },
         "deltaAngle": { "$ref": "#/$defs/angle" }
      }
     }
   ]
},
"arc": {
   "allOf" : [
     { "$ref": "#/$defs/abstractArc" },
     {
         "type": "object",
         "required": [ "type" ],
         "properties":
         {
            "type": { "type": "string", "enum": [ "Arc" ] }
         }
```

```
      }
    ]
  },
  "sectorArc": {
    "allOf" : [
      { "$ref": "#/$defs/closedShape" },
      { "$ref": "#/$defs/abstractArc" },
      {
          "type": "object",
          "required": [ "type" ],
          "properties":
          {
             "type": { "type": "string", "enum": [ "SectorArc" ] }
          }
      }
    ]
  },
  "chordArc": {
    "allOf" : [
      { "$ref": "#/$defs/closedShape" },
      { "$ref": "#/$defs/abstractArc" },
      {
          "type": "object",
          "required": [ "type" ],
          "properties":
          {
             "type": { "type": "string", "enum": [ "ChordArc" ] }
          }
      }
    ]
  },
  "circle": {
    "allOf" : [
      { "$ref": "#/$defs/closedShape" },
      {
       "type": "object",
       "required": [ "type" ],
       "properties":
       {
          "type": { "type": "string", "enum": [ "Circle" ] },
          "center": { "$ref": "#/$defs/unitPoint" },
          "radius": { "$ref": "#/$defs/unitValue" }
       }
      }
    ]
  },
  "ellipse": {
    "allOf" : [
      { "$ref": "#/$defs/closedShape" },
      {
       "type": "object",
       "required": [ "type", "radii" ],
       "properties":
       {
          "type": { "type": "string", "enum": [ "Ellipse" ] },
          "center": { "$ref": "#/$defs/unitPoint" },
          "radii": {
             "oneOf": [
                {
                   "type": "array",
                   "minItems": 2,
                   "maxItems": 2,
                   "items": { "$ref": "#/$defs/unitValue" }
```

```
                    },
                    {
                       "type": "object",
                       "properties":
                       {
                         "x": { "$ref": "#/$defs/unitValue" },
                         "y": { "$ref": "#/$defs/unitValue" },
                         "alter": { "type": "boolean" }
                       }
                    }
                 ]
              }
           }
        }
     ]
  },
  "rectangle": {
     "allOf" : [
       { "$ref": "#/$defs/closedShape" },
       {
        "type": "object",
        "required": [ "type", "topLeft", "bottomRight" ],
        "properties":
        {
           "type": { "type": "string", "enum": [ "Rectangle" ] },
           "topLeft": { "$ref": "#/$defs/unitPoint" },
           "bottomRight": { "$ref": "#/$defs/unitPoint" }
        }
       }
     ]
  },
  "roundedRectangle": {
     "allOf" : [
       { "$ref": "#/$defs/closedShape" },
       {
        "type": "object",
        "required": [ "type", "topLeft", "bottomRight" ],
        "properties":
        {
           "type": { "type": "string", "enum": [ "RoundedRectangle" ] },
           "topLeft": { "$ref": "#/$defs/unitPoint" },
           "bottomRight": { "$ref": "#/$defs/unitPoint" },
           "radii": {
              "oneOf": [
                 {
                    "type": "array",
                    "minItems": 2,
                    "maxItems": 2,
                    "items": { "$ref": "#/$defs/unitValue" }
                 },
                 {
                    "type": "object",
                    "properties":
                    {
                      "x": { "$ref": "#/$defs/unitValue" },
                      "y": { "$ref": "#/$defs/unitValue" },
                      "alter": { "type": "boolean" }
                    }
                 }
              ]
           }
        }
       }
```

```
    ]
  },
  "abstractPath": {
    "allOf": [
      { "$ref": "#/$defs/abstractShape" },
      {
       "type": "object",
       "required": [ "nodes" ],
       "properties":
       {
          "nodes": { "$ref": "#/$defs/pathNodes" }
       }
      }
    ]
  },
  "path": {
    "allOf": [
      { "$ref": "#/$defs/abstractPath" },
      {
         "type": "object",
         "required": [ "type" ],
         "properties":
         {
            "type": { "type": "string", "enum": [ "Path" ] }
         }
      }
    ]
  },
  "closedPath": {
    "allOf": [
      { "$ref": "#/$defs/abstractPath" },
      { "$ref": "#/$defs/closedShape" },
      {
       "type": "object",
       "properties":
       {
          "type": { "type": "string", "enum": [ "ClosedPath" ] },
          "innerNodes":
          {
             "anyOf": [
               { "$ref": "#/$defs/arrayExpression" },
               {
                  "type": "array",
                  "items": { "$ref": "#/$defs/pathNodes" }
               },
               {
                  "type": "object",
                  "required": [ "index", "value" ],
                  "properties": {
                     "index": { "type": "integer", "minimum": 0 },
                     "value": { "$ref": "#/$defs/pathNodes" }
                  }
               }
             ]
          }
       }
      }
    ]
  },
  "pathNodes": {
    "oneOf": [
      { "$ref": "#/$defs/arrayExpression" },
      {
```

```
            "type": "array",
            "minItems": 1,
            "items": { "$ref": "#/$defs/unitPoint" }
         },
         {
            "type": "object",
            "required": [ "index", "value" ],
            "properties": {
               "index": { "type": "integer", "minimum": 0 },
               "value": { "$ref": "#/$defs/unitPoint" }
            }
         }
      ]
   },
   "resource": {
      "type": "object",
      "properties":
      {
         "alter": { "type": "boolean" },
         "uri": { "$ref": "#/$defs/characterExpression" },
         "path": { "$ref": "#/$defs/characterExpression" },
         "id": { "$ref": "#/$defs/characterExpression" },
         "type": { "$ref": "#/$defs/characterExpression" },
         "ext": { "$ref": "#/$defs/characterExpression" },
         "sprite": { "$ref": "#/$defs/characterExpression" }
      }
   },
   "labelPlacement": {
      "oneOf": [
        { "$ref": "#/$defs/idOrFnExpression" },
        {
            "type": "object",
            "properties":
            {
               "alter": { "type": "boolean" },
               "priority": { "$ref": "#/$defs/numericExpression" },
               "minSpacing": { "$ref": "#/$defs/numericExpression" },
               "maxSpacing": { "$ref": "#/$defs/numericExpression" }
            }
        }
      ]
   },
   "strokeStyling":
   {
      "oneOf": [
        { "$ref": "#/$defs/idOrFnExpression" },
        {
            "type": "object",
            "properties":
            {
               "alter": { "type": "boolean" },
               "color": { "$ref": "#/$defs/color" },
               "opacity": { "$ref": "#/$defs/opacity" },
               "width": { "$ref": "#/$defs/unitValue" }
            }
        }
      ]
   },
   "angle":
   {
      "oneOf": [
         { "$ref": "#/$defs/numericExpression" },
```

```
            { "type": "object", "properties": { "deg": { "$ref": "#/$defs/
numericExpression" } }, "required": [ "deg" ] },
            { "type": "object", "properties": { "rad": { "$ref": "#/$defs/
numericExpression" } }, "required": [ "rad" ] }
         ]
      },
      "unitValue":
      {
         "oneOf": [
            { "$ref": "#/$defs/numericExpression" },
            { "type": "object", "properties": { "px": { "$ref": "#/$defs/
numericExpression" } }, "required": [ "px" ] },
            { "type": "object", "properties": { "mm": { "$ref": "#/$defs/
numericExpression" } }, "required": [ "mm" ] },
            { "type": "object", "properties": { "cm": { "$ref": "#/$defs/
numericExpression" } }, "required": [ "cm" ] },
            { "type": "object", "properties": { "in": { "$ref": "#/$defs/
numericExpression" } }, "required": [ "in" ] },
            { "type": "object", "properties": { "pt": { "$ref": "#/$defs/
numericExpression" } }, "required": [ "pt" ] },
            { "type": "object", "properties": { "em": { "$ref": "#/$defs/
numericExpression" } }, "required": [ "em" ] },
            { "type": "object", "properties": { "pc": { "$ref": "#/$defs/
numericExpression" } }, "required": [ "pc" ] },
            { "type": "object", "properties": { "m" : { "$ref": "#/$defs/
numericExpression" } }, "required": [ "m"  ] },
            { "type": "object", "properties": { "ft": { "$ref": "#/$defs/
numericExpression" } }, "required": [ "ft" ] }
         ]
      },
      "unitPoint":
      {
         "oneOf": [
           { "$ref": "#/$defs/idOrFnExpression" },
           {
              "type": "array",
              "minItems": 2,
              "maxItems": 3,
              "items": { "$ref": "#/$defs/unitValue" }
           },
           {
              "type": "object",
              "properties":
              {
                 "x": { "$ref": "#/$defs/unitValue" },
                 "y": { "$ref": "#/$defs/unitValue" },
                 "z": { "$ref": "#/$defs/unitValue" },
                 "alter": { "type": "boolean" }
              }
           }
         ]
      },
      "opacity":
      {
         "anyOf": [
            { "type": "number", "minimum": 0, "maximum": 1 },
            { "$ref": "#/$defs/numericExpression" }
         ]
      },
      "color":
      {
         "oneOf": [
           { "$ref": "#/$defs/idOrFnExpression" },
```

```
{
   "type": "object",
   "properties":
   {
      "r": { "type": "integer", "mininum": 0, "maximum": 255 },
      "g": { "type": "integer", "mininum": 0, "maximum": 255 },
      "b": { "type": "integer", "mininum": 0, "maximum": 255 },
      "alter": { "type": "boolean" }
   }
},
{
   "type": "array",
   "items": { "type": "integer", "mininum": 0, "maximum": 255 },
   "minItems": 3,
   "maxItems": 3
},
{
   "type": "string",
   "enum": [
      "black",
      "dimGray",
      "dimGrey",
      "gray",
      "grey",
      "darkGray",
      "darkGrey",
      "silver",
      "lightGray",
      "lightGrey",
      "gainsboro",
      "whiteSmoke",
      "white",
      "rosyBrown",
      "indianRed",
      "brown",
      "fireBrick",
      "lightCoral",
      "maroon",
      "darkRed",
      "red",
      "snow",
      "mistyRose",
      "salmon",
      "tomato",
      "darkSalmon",
      "coral",
      "orangeRed",
      "lightSalmon",
      "sienna",
      "seaShell",
      "chocolate",
      "saddleBrown",
      "sandyBrown",
      "peachPuff",
      "peru",
      "linen",
      "bisque",
      "darkOrange",
      "burlyWood",
      "tan",
      "antiqueWhite",
      "navajoWhite",
      "blanchedAlmond",
```

```
"papayaWhip",
"moccasin",
"orange",
"wheat",
"oldLace",
"floralWhite",
"darkGoldenrod",
"goldenrod",
"cornsilk",
"gold",
"khaki",
"lemonChiffon",
"paleGoldenrod",
"darkKhaki",
"beige",
"lightGoldenRodYellow",
"olive",
"yellow",
"lightYellow",
"ivory",
"oliveDrab",
"yellowGreen",
"darkOliveGreen",
"greenYellow",
"chartreuse",
"lawnGreen",
"darkSeaGreen",
"forestGreen",
"limeGreen",
"lightGreen",
"paleGreen",
"darkGreen",
"green",
"lime",
"honeyDew",
"seaGreen",
"mediumSeaGreen",
"springGreen",
"mintCream",
"mediumSpringGreen",
"mediumAquaMarine",
"aquamarine",
"turquoise",
"lightSeaGreen",
"mediumTurquoise",
"darkSlateGray",
"darkSlateGrey",
"paleTurquoise",
"teal",
"darkCyan",
"aqua",
"cyan",
"lightCyan",
"azure",
"darkTurquoise",
"cadetBlue",
"powderBlue",
"lightBlue",
"deepSkyBlue",
"skyBlue",
"lightSkyBlue",
"steelBlue",
"aliceBlue",
```

```
                "dodgerBlue",
                "slateGray",
                "slateGrey",
                "lightSlateGray",
                "lightSlateGrey",
                "lightSteelBlue",
                "cornflowerBlue",
                "royalBlue",
                "midnightBlue",
                "lavender",
                "navy",
                "darkBlue",
                "mediumBlue",
                "blue",
                "ghostWhite",
                "slateBlue",
                "darkSlateBlue",
                "mediumSlateBlue",
                "mediumPurple",
                "blueViolet",
                "indigo",
                "darkOrchid",
                "darkViolet",
                "mediumOrchid",
                "thistle",
                "plum",
                "violet",
                "purple",
                "darkMagenta",
                "magenta",
                "fuschia",
                "orchid",
                "mediumVioletRed",
                "deepPink",
                "hotPink",
                "lavenderBlush",
                "paleVioletRed",
                "crimson",
                "pink",
                "lightPink"
            ]
        }
    ]
},
"idOrFnExpression": {
    "anyOf": [
        { "$ref": "#/$defs/systemIdentifier" },
        { "$ref": "#/$defs/propertyRef" },
        { "$ref": "#/$defs/functionRef" }
    ]
},
"anyExpression": {
    "anyOf": [
        { "$ref": "#/$defs/systemIdentifier" },
        { "$ref": "#/$defs/scalarExpression" },
        { "$ref": "#/$defs/geomExpression" },
        { "$ref": "#/$defs/temporalExpression" },
        { "$ref": "#/$defs/array" },
        { "type": null }
    ]
},
"systemIdentifier": {
    "type": "object",
```

```
    "required": [ "sysId" ],
    "properties":
    {
       "sysId": {
          "type": "string",
          "enum": [
            "dataLayer",
            "dataLayer.id",
            "dataLayer.type",
            "dataLayer.featuresGeometryDimensions",
            "vis",
            "vis.sd",
            "vis.timeInterval",
            "vis.timeInterval.start",
            "vis.timeInterval.start.date",
            "vis.timeInterval.end",
            "vis.timeInterval.date"
          ]
       }
    }
},
"boolExpression": {
    "$dynamicAnchor": "boolExpression",
    "oneOf": [
      { "$ref": "#/$defs/andOrExpression" },
      { "$ref": "#/$defs/notExpression" },
      { "$ref": "#/$defs/comparisonPredicate" },
      { "$ref": "#/$defs/spatialPredicate" },
      { "$ref": "#/$defs/temporalPredicate" },
      { "$ref": "#/$defs/arrayPredicate" },
      { "$ref": "#/$defs/functionRef" },
      { "type": "boolean" }
    ]
},
"andOrExpression": {
  "type": "object",
  "required": [ "op", "args" ],
  "properties": {
    "op": {
      "type": "string",
      "enum": [ "and", "or" ]
    },
    "args": {
      "type": "array",
      "minItems": 2,
      "items": {
        "$dynamicRef": "#boolExpression"
      }
    }
  }
},

"notExpression": {
  "type": "object",
  "required": [ "op", "args" ],
  "properties": {
    "op": {
      "type": "string",
      "enum": [ "not" ]
    },
    "args": {
      "type": "array",
      "minItems": 1,
```

```
      "maxItems": 1,
      "items": {
        "$dynamicRef": "#boolExpression"
      }
    }
  }
},

"comparisonPredicate": {
  "oneOf": [
    { "$ref": "#/$defs/binaryComparisonPredicate" },
    { "$ref": "#/$defs/isLikePredicate" },
    { "$ref": "#/$defs/isBetweenPredicate" },
    { "$ref": "#/$defs/isInListPredicate" },
    { "$ref": "#/$defs/isNullPredicate" }
  ]
},

"binaryComparisonPredicate": {
  "type": "object",
  "required": [ "op", "args" ],
  "properties": {
    "op": {
      "type": "string",
      "enum": [ "=", "<>", "<", ">", "<=", ">=" ]
    },
    "args": {
      "$ref": "#/$defs/scalarOperands"
    }
  }
},
"scalarOperands": {
  "type": "array",
  "minItems": 2,
  "maxItems": 2,
  "prefixItems": [
    { "$ref": "#/$defs/scalarExpression" },
    { "$ref": "#/$defs/scalarExpression" }
  ]
},

"scalarExpression": {
  "oneOf": [
    { "$ref": "#/$defs/characterExpression" },
    { "$ref": "#/$defs/numericExpression" },
    { "$dynamicRef": "#boolExpression" },
    { "$ref": "#/$defs/temporalInstantExpression" }
  ]
},

"temporalInstantExpression": {
  "oneOf": [
    { "$ref": "#/$defs/instantInstance" },
    { "$ref": "#/$defs/propertyRef" },
    { "$ref": "#/$defs/systemIdentifier" },
    { "$ref": "#/$defs/functionRef" }
  ]
},

"isLikePredicate": {
  "type": "object",
  "required": [ "op", "args" ],
  "properties": {
```

```
      "op": {
        "type": "string",
        "enum": [ "like" ]
      },
      "args": {
        "$ref": "#/$defs/isLikeOperands"
      }
    }
  },
  "isLikeOperands": {
        "type": "array",
        "prefixItems": [
          { "$ref": "#/$defs/characterExpression" },
          { "$ref": "#/$defs/patternExpression" }
        ],
        "additionalItems": false
  },
  "patternExpression": {
    "oneOf": [
      {
        "type": "object",
        "required": [ "casei" ],
        "properties": {
          "casei": {
            "$ref": "#/$defs/patternExpression"
          }
        }
      },
      {
        "type": "object",
        "required": [ "accenti" ],
        "properties": {
          "accenti": {
            "$ref": "#/$defs/patternExpression"
          }
        }
      },
      { "type": "string" }
    ]
  },
  "isBetweenPredicate": {
    "type": "object",
    "required": [ "op", "args" ],
    "properties": {
      "op": { "type": "string", "enum": [ "between" ] },
      "args": { "$ref": "#/$defs/isBetweenOperands" }
    }
  },
  "isBetweenOperands": {
    "type": "array",
    "minItems": 3,
    "maxItems": 3,
    "items": {
      "$ref": "#/$defs/numericExpression"
    },
    "additionalItems": false
  },
  "numericExpression": {
    "oneOf": [
      { "$ref": "#/$defs/arithmeticExpression" },
      { "type": "number" },
      { "$ref": "#/$defs/propertyRef" },
      { "$ref": "#/$defs/systemIdentifier" },
```

```
      { "$ref": "#/$defs/functionRef" }
    ]
  },

  "isInListPredicate": {
    "type": "object",
    "required": [ "op", "args" ],
    "properties": {
      "op": {
        "type": "string",
        "enum": [ "in" ]
      },
      "args": {
        "$ref": "#/$defs/inListOperands"
      }
    }
  },
  "inListOperands": {
    "type": "array",
    "prefixItems": [
      {
        "$ref": "#/$defs/scalarExpression"
      },
      {
        "type": "array",
        "items": {
          "$ref": "#/$defs/scalarExpression"
        }
      }
    ],
    "additionalItems": false
  },

  "isNullPredicate": {
    "type": "object",
    "required": [ "op", "args" ],
    "properties": {
      "op": {
        "type": "string",
        "enum": [ "isNull" ]
      },
      "args": {
        "$ref": "#/$defs/isNullOperand"
      }
    }
  },
  "isNullOperand": {
    "oneOf": [
      { "$ref": "#/$defs/characterExpression" },
      { "$ref": "#/$defs/numericExpression" },
      { "$dynamicRef": "#boolExpression" },
      { "$ref": "#/$defs/geomExpression" },
      { "$ref": "#/$defs/temporalExpression" }
    ]
  },

  "spatialPredicate": {
    "type": "object",
    "required": [ "op", "args" ],
    "properties": {
      "op": {
        "type": "string",
        "enum": [
```

```
            "s_contains",
            "s_crosses",
            "s_disjoint",
            "s_equals",
            "s_intersects",
            "s_overlaps",
            "s_touches",
            "s_within",
            "s_covers",
            "s_coveredBy"
          ]
        },
        "args": { "$ref": "#/$defs/spatialOperands" }
      }
    },
    "spatialOperands": {
      "type": "array",
      "prefixItems": [
        { "$ref": "#/$defs/geomExpression" },
        { "$ref": "#/$defs/geomExpression" }
      ],
      "additionalItems": false
    },
    "geomExpression": {
      "oneOf": [
        { "$ref": "#/$defs/spatialInstance" },
        { "$ref": "#/$defs/propertyRef" },
        { "$ref": "#/$defs/systemIdentifier" },
        { "$ref": "#/$defs/functionRef" }
      ]
    },

    "temporalPredicate": {
      "type": "object",
      "required": [ "op", "args" ],
      "properties": {
        "op": {
          "type": "string",
          "enum": [
            "t_after",
            "t_before",
            "t_contains",
            "t_disjoint",
            "t_during",
            "t_equals",
            "t_finishedBy",
            "t_finishes",
            "t_intersects",
            "t_meets",
            "t_metBy",
            "t_overlappedBy",
            "t_overlaps",
            "t_startedBy",
            "t_starts"
          ]
        },
        "args": { "$ref": "#/$defs/temporalOperands" }
      }
    },
    "temporalOperands": {
      "type": "array",
      "prefixItems": [
        { "$ref": "#/$defs/temporalExpression" },
```

```
      { "$ref": "#/$defs/temporalExpression" }
    ],
    "additionalItems": false
  },
  "temporalExpression": {
    "oneOf": [
      { "$ref": "#/$defs/temporalInstance" },
      { "$ref": "#/$defs/propertyRef" },
      { "$ref": "#/$defs/systemIdentifier" },
      { "$ref": "#/$defs/functionRef" }
    ]
  },

  "arrayPredicate": {
    "type": "object",
    "required": [ "op", "args" ],
    "properties": {
      "op": {
        "type": "string",
        "enum": [
          "a_containedBy",
          "a_contains",
          "a_equals",
          "a_overlaps"
        ]
      },
      "args": {
          "type": "array",
          "minItems": 2,
          "maxItems": 2,
          "items": {
            "oneOf": [
              { "$ref": "#/$defs/array" },
              { "$ref": "#/$defs/propertyRef" },
              { "$ref": "#/$defs/systemIdentifier" },
              { "$ref": "#/$defs/functionRef" }
            ]
          }
      }
    }
  },
  "arrayExpression": {
    "type": "array",
    "items": {
      "oneOf": [
        { "$ref": "#/$defs/array" },
        { "$ref": "#/$defs/propertyRef" },
        { "$ref": "#/$defs/systemIdentifier" },
        { "$ref": "#/$defs/functionRef" }
      ]
    }
  },
  "array": {
    "type": "array",
    "items": {
      "oneOf": [
        { "$ref": "#/$defs/characterExpression" },
        { "$ref": "#/$defs/numericExpression" },
        { "$dynamicRef": "#boolExpression" },
        { "$ref": "#/$defs/geomExpression" },
        { "$ref": "#/$defs/temporalExpression" },
        { "$ref": "#/$defs/array" }
      ]
```

```
    }
  },

  "arithmeticExpression": {
    "type": "object",
    "required": [ "op", "args" ],
    "properties": {
      "op": {
        "type": "string",
        "enum": [ "+", "-", "*", "/", "^" ]
      },
      "args": {
        "$ref": "#/$defs/arithmeticOperands"
      }
    }
  },
  "arithmeticOperands": {
    "type": "array",
    "minItems": 2,
    "maxItems": 2,
    "items": {
      "oneOf": [
        { "$ref": "#/$defs/arithmeticExpression" },
        { "$ref": "#/$defs/propertyRef" },
        { "$ref": "#/$defs/systemIdentifier" },
        { "$ref": "#/$defs/functionRef" },
        { "type": "number" }
      ]
    }
  },

  "propertyRef": {
    "type": "object",
    "required": [ "property" ],
    "properties": {
      "property": { "type": "string" }
    }
  },

  "casei": {
    "type": "object",
    "required": [ "casei" ],
    "properties": {
      "casei": {
        "$ref": "#/$defs/characterExpression"
      }
    }
  },

  "accenti": {
    "type": "object",
    "required": [ "accenti" ],
    "properties": {
      "accenti": {
        "$ref": "#/$defs/characterExpression"
      }
    }
  },

  "characterExpression": {
    "oneOf": [
      { "$ref": "#/$defs/casei" },
      { "$ref": "#/$defs/accenti" },
```

```
      { "type": "string" },
      { "$ref": "#/$defs/propertyRef" },
      { "$ref": "#/$defs/systemIdentifier" },
      { "$ref": "#/$defs/functionRef" }
    ]
  },

  "functionRef": {
    "type": "object",
    "required": [ "function" ],
    "properties": {
      "function": {
        "$ref": "#/$defs/function"
      }
    }
  },

  "function": {
    "type": "object",
    "required": [ "name" ],
    "properties": {
      "name": {
        "type": "string"
      },
      "args": {
        "type": "array",
        "items": {
          "oneOf": [
            { "$ref": "#/$defs/characterExpression" },
            { "$ref": "#/$defs/numericExpression" },
            { "$dynamicRef": "#boolExpression" },
            { "$ref": "#/$defs/geomExpression" },
            { "$ref": "#/$defs/temporalExpression" },
            { "$ref": "#/$defs/array" }
          ]
        }
      }
    }
  },

  "scalarLiteral": {
    "oneOf": [
      { "type": "string" },
      { "type": "number" },
      { "type": "boolean" },
      { "$ref": "#/$defs/instantInstance" }
    ]
  },

  "spatialInstance": {
    "oneOf": [
      { "$ref": "#/$defs/geometryLiteral" },
      { "$ref": "#/$defs/bboxLiteral" }
    ]
  },
  "geometryLiteral": {
    "oneOf": [
      { "$ref": "#/$defs/point" },
      { "$ref": "#/$defs/linestring" },
      { "$ref": "#/$defs/polygon" },
      { "$ref": "#/$defs/multipoint" },
      { "$ref": "#/$defs/multilinestring" },
      { "$ref": "#/$defs/multipolygon" }
```

```
    ]
  },
  "point": {
    "title": "GeoJSON Point",
    "type": "object",
    "required": [ "type", "coordinates" ],
    "properties": {
      "type": {
        "type": "string",
        "enum": [ "Point" ]
      },
      "coordinates": {
        "type": "array",
        "minItems": 2,
        "items": {
          "type": "number"
        }
      },
      "bbox": {
        "type": "array",
        "minItems": 4,
        "items": {
          "type": "number"
        }
      }
    }
  },
  "linestring": {
    "title": "GeoJSON LineString",
    "type": "object",
    "required": [ "type", "coordinates" ],
    "properties": {
      "type": {
        "type": "string",
        "enum": [ "LineString" ]
      },
      "coordinates": {
        "type": "array",
        "minItems": 2,
        "items": {
          "type": "array",
          "minItems": 2,
          "items": {
            "type": "number"
          }
        }
      },
      "bbox": {
        "type": "array",
        "minItems": 4,
        "items": {
          "type": "number"
        }
      }
    }
  },
  "polygon": {
    "title": "GeoJSON Polygon",
    "type": "object",
    "required": [ "type", "coordinates" ],
    "properties": {
      "type": {
        "type": "string",
```

```
        "enum": [ "Polygon" ]
      },
      "coordinates": {
        "type": "array",
        "items": {
          "type": "array",
          "minItems": 4,
          "items": {
            "type": "array",
            "minItems": 2,
            "items": {
              "type": "number"
            }
          }
        }
      },
      "bbox": {
        "type": "array",
        "minItems": 4,
        "items": {
          "type": "number"
        }
      }
    }
  },
  "multipoint": {
    "title": "GeoJSON MultiPoint",
    "type": "object",
    "required": [ "type", "coordinates" ],
    "properties": {
      "type": {
        "type": "string",
        "enum": [ "MultiPoint" ]
      },
      "coordinates": {
        "type": "array",
        "items": {
          "type": "array",
          "minItems": 2,
          "items": {
            "type": "number"
          }
        }
      },
      "bbox": {
        "type": "array",
        "minItems": 4,
        "items": {
          "type": "number"
        }
      }
    }
  },
  "multilinestring": {
    "title": "GeoJSON MultiLineString",
    "type": "object",
    "required": [ "type", "coordinates" ],
    "properties": {
      "type": {
        "type": "string",
        "enum": [ "MultiLineString" ]
      },
      "coordinates": {
```

```
        "type": "array",
        "items": {
          "type": "array",
          "minItems": 2,
          "items": {
            "type": "array",
            "minItems": 2,
            "items": {
              "type": "number"
            }
          }
        }
      },
      "bbox": {
        "type": "array",
        "minItems": 4,
        "items": {
          "type": "number"
        }
      }
    }
  },
  "multipolygon": {
    "title": "GeoJSON MultiPolygon",
    "type": "object",
    "required": [ "type", "coordinates" ],
    "properties": {
      "type": {
        "type": "string",
        "enum": [ "MultiPolygon" ]
      },
      "coordinates": {
        "type": "array",
        "items": {
          "type": "array",
          "items": {
            "type": "array",
            "minItems": 4,
            "items": {
              "type": "array",
              "minItems": 2,
              "items": {
                "type": "number"
              }
            }
          }
        }
      },
      "bbox": {
        "type": "array",
        "minItems": 4,
        "items": {
          "type": "number"
        }
      }
    }
  },
  "bboxLiteral": {
    "type": "object",
    "required": [
      "bbox"
    ],
    "properties": {
```

```
      "bbox": {
        "$ref": "#/$defs/bbox"
      }
    }
  },
  "bbox": {
    "type": "array",
    "oneOf": [
      { "minItems": 4, "maxItems": 4 },
      { "minItems": 6, "maxItems": 6 }
    ],
    "items": { "type": "number" }
  },

  "temporalInstance": {
    "oneOf": [
      { "$ref": "#/$defs/instantInstance" },
      { "$ref": "#/$defs/intervalInstance" }
    ]
  },
  "instantInstance": {
    "oneOf": [
      { "$ref": "#/$defs/dateInstant" },
      { "$ref": "#/$defs/timestampInstant" }
    ]
  },
  "dateInstant": {
    "type": "object",
    "required": [ "date" ],
    "properties": {
      "date": { "$ref": "#/$defs/dateString" }
    }
  },
  "timestampInstant": {
    "type": "object",
    "required": [ "timestamp" ],
    "properties": {
      "timestamp": { "$ref": "#/$defs/timestampString" }
    }
  },
  "instantString": {
    "oneOf": [
      { "$ref": "#/$defs/dateString" },
      { "$ref": "#/$defs/timestampString" }
    ]
  },
  "dateString": {
    "type": "string",
    "format": "date"
  },
  "timestampString": {
    "type": "string",
    "format": "date-time"
  },
  "intervalInstance": {
    "type": "object",
    "required": [ "interval" ],
    "properties": {
      "interval": { "$ref": "#/$defs/intervalArray" }
    }
  },
  "intervalArray": {
    "type": "array",
```

```
      "minItems": 2,
      "maxItems": 2,
      "items": {
        "oneOf": [
          { "$ref": "#/$defs/instantString" },
          { "type": "string", "enum": [ ".." ] },
          { "$ref": "#/$defs/propertyRef" },
          { "$ref": "#/$defs/systemIdentifier" },
          { "$ref": "#/$defs/functionRef" }
        ]
      }
    }
  }
}
```

20.2. Requirements

21

REQUIREMENTS CLASS "CASCADING CARTOGRAPHIC SYMBOLOGY STYLE SHEETS"

21 REQUIREMENTS CLASS "CASCADING CARTOGRAPHIC SYMBOLOGY STYLE SHEETS"

21.1. Overview

This requirements class defines an encoding for the Styles and Symbology Logical Model inspired from Web Cascading Style Sheets (CSS) and other CSS-like cartographic symbology encodings. The primary objective of this encoding is expressiveness so as to facilitate hand-edition by cartographers. Parsing this encoding requires a custom parser.

21.1.1. Cascading Style Sheets

A style is encoded as a series of styling rules into a *style sheet* document. Symbolizer properties set further down in rules override those set by rules at the same level apearing earlier (higher up).

It is also possible to include another style sheet using the `.include` directive followed by a relative path enclosed within single quotes (`' '`), separated by a space e.g., `.include '. /baseStyle.ccsss'`. The include directive is replaced by the content of that file in a pre-processing step in a recursive manner, so that style sheets can be *cascaded*.

Because later rules allow to override earlier ones, this cascading allows to apply for example:

- run-time styling preferences over default user styling,
- user styling preferences over default application styling,
- application styling preferences over default data provider styling.

21.1.2. Comments

A style sheet can also contain comments either using C-style multi-line comments (between `/*` and `*/`) or C++-style single-line comments (starting with `//`).

21.1.3. Metadata elements

In addition to styling rules, the style sheet can define metadata elements prefixed by a dot (`.`) as the first character on a line. The following metadata elements are defined:

Table 95 — Metadata elements

NAME	DEFINITION
title	Short human-readable title for the style
abstract	Detailed description of the style
authors	Authors of the style
geoData Classes	Comma-separated URIs of GeoDataClasses allowing to identify data sources for which the style is suitable

NOTE: We should probably include these metadata elements in Core conceptual model.

21.1.4. Styling Rules and Symbolizers

A styling rule applies zero or more symbolizer properties if its Selector is evaluated to *true*. A styling rule may also include nested rules that are only evaluated when the parent rule itself is selected. Symbolizer properties set within a nested rule override symbolizer properties set in the parent rule. It is also possible to override only a portion of a property by assigning a value to member of an object, or an element of an array.

The grammar for specifying a rule in this encoding is:

[<selector>]* '{' [<symbolizer property assignment>]* [<nested rule>]* '}'

The grammar for a symbolizer property assignment is:

<symbolizer property name> ['.' <member> | '[' <index> ']']* ':' <expression> ';'

21.1.5. Selectors

A selector is either:

- an expression as described in the section below,
- or a short-hand form for selecting a data layer:

‘#’<data layer name>

21.1.6. Expressions

21.1.6.1. CQL2-Text encoding

Whether in Selectors or values of property symbolizers, Expressions are encoded using CQL2-Text. However, a number of extensions are introduced to cover the full range of capabilities defined by the styling and symbology conceptual model.

The first such extension is that in the context of a symbolizer property value, expressions are not limited to boolean predicates being evaluated as true or false, but can return any type of values such as numerical, text, or complex values, including geometry.

21.1.6.2. Instance expressions

The concept of an object instance is introduced, with the grammar:

<Class Name> ‘(‘ <member initializer list> ‘)’.

This corresponds somewhat to CQL2 constructs for temporal intervals and WKT geometry for example,

- `POLYGON( (0 40, 10 40, 10 50, 0 50, 0 40) )`
- `INTERVAL('2021-01-01T00:00:00Z','2021-12-31T23:59:59Z')`

In the first example, POLYGON is the class name, and the array of coordinates is a list of a single member initializer. In the second second example, INTERVAL is the class name, and the start and end dates are a list of two member initializers.

In both of these examples, the members being initialized are implied from the order. Due to the heavy use of object instances in the symbology model, the ability to identify specific member to initialize is introduced. The syntax for doing so tries to be consistent with the overall CSS inspiration, and therefore the <member> ‘:’ value is used, and the ‘;’ is added as an alternative way to separate members in the initializer list. As examples, the above instantiations could also be written:

- `POLYGON(rings: (0 40, 10 40, 10 50, 0 50, 0 40))`
- `INTERVAL(start: '2021-01-01T00:00:00Z'; end: '2021-12-31T23:59:59Z')`

Another capability introduced is the ability to infer the class from the expected type where the instance expression is used. For example, if a POLYGON object is assigned to a `poly` property expecting that object type, then the expression can be written simply as:

- `(rings: (0 40, 10 40, 10 50, 0 50, 0 40))`

In both of these examples, the members being initialized are implied from the order.

When not all members of a class are initialized within an instantiation, a default value is inherited. Those default values are specified in a dedicated section below

NOTE: Should we diverge from CQL2 and allow the use of `{ }` as an alternative to `( )` to define instances, and of `[ ]` as an alternative to `( )` to define arrays in this encoding for clarity?

21.1.6.3. Tuples and related implications

The concept of a tuple is introduced, corresponding to the CQL2-Text Well-Known Text (WKT) coordinates in the above polygon examples. Based on the logical model, each of those tuples actually correspond to a Point instance. In order to maintain consistency with the CQL2-Text WKT encoding, the logical model as well as the object instance encodings, a number of alternative encoding rules need to be established. A tuple must be able to replace either:

- an array of coordinate values,
- an instance of a class defining a single property being such an array (e.g., Point), or
- a class identified as being replaceable by a tuple.

Therefore, the following equivalent definitions are all valid where a POINT is expected:

- `POINT(30 10)` — The standard WKT form where the tuple replaces the array of coordinates, implying the *coordinates* member
- `POINT( (30, 10) )` — Using an array of coordinates, implying the *coordinates* member
- `POINT(coordinates: 30 10)` — The tuple replacing the array of coordinates, explicit coordinates *member*
- `POINT(coordinates: (30, 10))` — Using an array of coordinates, explicit coordinates *member*
- `(30 10)` — Implied class, tuple replacing array of coordinates, implied *coordinates* member
- `( (30, 10) )` — Implied class, using array of coordinates, implied *coordinates* member
- `(coordinates: 30 10)` — Implied class, tuple replacing array of coordinates, explicit *coordinates* member
- `(coordinates: (30, 10))` — Implied class, using an array of coordinates, explicit *coordinates* member
- `30 10` — Implied class, tuple replacing Point instance

The following are some equivalent examples of valid definitions where a polygon is expected:

- `POLYGON( (0 40, 10 40, 10 50, 0 50, 0 40) )`

- `POLYGON( ( POINT(0 40), POINT(10 40), POINT(10 50), POINT(0 50), POINT(0 40) ) )`
- `POLYGON( ( POINT( (0, 40) ), POINT( (10, 40) ), POINT( (10, 50) ), POINT( (0, 50) ), POINT( (0, 40) ) ) )`
- `POLYGON( ( POINT(coordinates: (0, 40)), POINT(coordinates: (10, 40)), POINT(coordinates: (10, 50)), POINT(coordinates: (0, 50)), POINT(coordinates: (0, 40)) ) )`
- `POLYGON( ( POINT(coordinates: 0 40), POINT(coordinates: 10 40), POINT(coordinates: 10 50), POINT(coordinates: 0 50), POINT(coordinates: 0 40) ) )`

The classes defined in this Standard that can be encoded as tuples are:

- *Point* (from WKT)
- *UnitPoint* (defining only coordinate values),
- *TextAlignment* (defining a horizontal and vertical alignment),
- *Color* (defining red, green and blue components),
- *ValueColor* (defining a value and color — note that the color itself can be a tuple),
- *ValueOpacity* (defining a value and an opacity,
- *Quaternion* (defining w, x, y, z components),
- *Vector3D* and *FloatVector3D* (defining x, y, z components) — NOTE: Change to UnitPoint3D?.

The tuple syntax can also be used to specify the *dashPattern* array of dashed strokes.

The following are equivalent example definitions where a *UnitPoint* is expected allowing each coordinate to be qualified with a unit of measure:

- `50 pc 50 pc`
- `UnitPoint(x: 50 pc; y: 50 pc;)`
- `UnitPoint(x: 50 pc, y: 50 pc)`
- `(50 pc, 50 pc)`

The following are equivalent example definitions where a *TextAligment* is expected:

- `left top`
- `TextAlignment(left, top)`

- `TextAlignment(horizontal: left; vertical: top)`
- `(left, top)`

21.1.6.4. Units of measure

The following abbreviations are used, following a value, to qualify it with a unit of measure. These are mainly used for *UnitPoint* coordinates, as well as with distance value such as those specified for Shape graphics e.g., the radius of a circle.

Table 96 — Units of measure abbreviations

UNIT	ABBREVIATION	NOTES
pixels	px	Display unit; not dependent on display resolution
meters	m	Real-world unit (SI unit of length)
feet	ft	Real-world unit (0.3048 m)
percent	pc	Relative value whose meaning depends on context e.g., Image's *hot Spot*, *scaling*.
points	pt	One point is equal to 1/72 inch (0.3528 mm).
em	em	One em is equal to the target's selected or default font point size.
screen Inches	in	Display unit (25.4 mm); see OGC *API – Maps* "Display Resolution" requirements class (`mm-per-pixel`) which defaults to 0.28 mm/pixel (~90.7142857 pixels/inch).
screenCM	cm	Display unit (10 mm); see OGC *API – Maps* "Display Resolution" requirements class (`mm-per-pixel`) which defaults to 0.28 mm/pixel (0.028 cm/pixel).
screenMM	mm	Display unit; see OGC *API – Maps* "Display Resolution" requirements class (`mm-per-pixel`) which defaults to 0.28 mm/pixel.
(none)		For relative values like scaling, 100 times less than if *pc* is specified. For screen-space objects, defaults to pixels.

21.1.6.5. Conditional expressions

As an extension to CQL2-Text, this encoding adds support for conditional expressions using the grammar:

<if-expression> '?' <then-expression> ':' <else-expression>

21.1.6.6. Enumeration values

As an extension to CQL2-Text, this encoding supports the concept of enumeration values, which do not need to be enclosed in single quotes. For compatibility with CQL2, the same identifier enclosed within single quotes is considered equivalent.

Enumeration values must conform to the CQL2 *identifier* grammar production rule, but may not be enclosed within double-quotes.

In a context where an enumeration is expected, the enumeration value takes precedence over an identifier that may be resolved by the same name. In order to explicitly refer to the identifier, the identifier can be enclosed in double-quotes.

21.1.6.7. Colors

Color values can be expressed using any of the forms allowed for instantiations, or using a tuple:

- `Color(255, 100, 50)`
- `Color(r: 255, g: 100, b: 50)`
- `(255, 100, 50)`
- `(r: 255; g: 100; b: 50;)`
- `255 100 50`

Colors can also be expressed as a hexadecimal value prefixed by a # symbol, as in Web CSS:

- `#FF6432`

Finally, colors can also be expressed as an enumeration value for named web colors:

- `red`
- `'red'`

Table 97 — Named web color enumeration

NAME	HEXADECIMAL	VALUE (R, G, B)
black	`0x000000`	0, 0, 0
dimGray	`0x696969`	105, 105, 105

NAME	HEXADECIMAL	VALUE (R, G, B)
dimGrey	`0x696969`	105, 105, 105
gray	`0x808080`	128, 128, 128
grey	`0x808080`	128, 128, 128
darkGray	`0xa9a9a9`	169, 169, 165
darkGrey	`0xa9a9a9`	169, 169, 165
silver	`0xc0c0c0`	192, 192, 192
lightGray	`0xd3d3d3`	211, 211, 211
lightGrey	`0xd3d3d3`	211, 211, 211
gainsboro	`0xdcdcdc`	220, 220, 220
whiteSmoke	`0xf5f5f5`	245, 245, 245
white	`0xffffff`	255, 255, 255
rosyBrown	`0xbc8f8f`	188, 143, 143
indianRed	`0xcd5c5c`	205, 92, 92
brown	`0xa52a2a`	165, 42, 42
fireBrick	`0xb22222`	178, 34, 34
lightCoral	`0xf08080`	240, 128, 128
maroon	`0x800000`	128, 0, 0
darkRed	`0x8b0000`	139, 0, 0
red	`0xff0000`	255, 0, 0
snow	`0xfffafa`	255, 250, 250
mistyRose	`0xffe4e1`	255, 228, 225

NAME	HEXADECIMAL	VALUE (R, G, B)
salmon	`0xfa8072`	250, 128, 114
tomato	`0xff6347`	255, 99, 71
darkSalmon	`0xe9967a`	233, 150, 122
coral	`0xff7f50`	255, 127, 80
orangeRed	`0xff4500`	255, 69, 0
lightSalmon	`0xffa07a`	255, 160, 122
sienna	`0xa0522d`	160, 82, 45
seaShell	`0xfff5ee`	255, 245, 238
chocolate	`0xd2691e`	210, 105, 30
saddleBrown	`0x8b4513`	139, 69, 19
sandyBrown	`0xf4a460`	244, 164, 96
peachPuff	`0xffdab9`	255, 218, 185
peru	`0xcd853f`	205, 133, 63
linen	`0xfaf0e6`	250, 240, 230
bisque	`0xffe4c4`	255, 228, 196
darkOrange	`0xff8c00`	255, 140, 0
burlyWood	`0xdeb887`	222, 184, 135
tan	`0xd2b48c`	210, 180, 140
antiqueWhite	`0xfaebd7`	250, 235, 215
navajoWhite	`0xffdead`	255, 222, 173
blanchedAlmond	`0xffebcd`	255, 235, 205

NAME	HEXADECIMAL	VALUE (R, G, B)
papayaWhip	0xffefd5	255, 239, 213
moccasin	0xffe4b5	255, 228, 181
orange	0xffa500	255, 165, 0
wheat	0xf5deb3	245, 222, 179
oldLace	0xfdf5e6	253, 245, 230
floralWhite	0xfffaf0	255, 250, 240
darkGoldenrod	0xb8860b	184, 134, 11
goldenrod	0xdaa520	218, 165, 32
cornsilk	0xfff8dc	255, 248, 220
gold	0xffd700	255, 215, 0
khaki	0xf0e68c	240, 230, 140
lemonChiffon	0xfffacd	255, 250, 205
paleGoldenrod	0xeee8aa	238, 232, 170
darkKhaki	0xbdb76b	189, 183, 107
beige	0xf5f5dc	245, 245, 220
lightGoldenRodYellow	0xfafad2	250, 250, 210
olive	0x808000	128, 128, 0
yellow	0xffff00	255, 255, 0
lightYellow	0xffffe0	255, 255, 224
ivory	0xfffff0	255, 255, 240
oliveDrab	0x6b8e23	107, 142, 35

NAME	HEXADECIMAL	VALUE (R, G, B)
yellowGreen	0x9acd32	154, 205, 50
darkOliveGreen	0x556b2f	85, 107, 47
greenYellow	0xadff2f	173, 255, 47
chartreuse	0x7fff00	127, 255, 0
lawnGreen	0x7cfc00	124, 252, 0
darkSeaGreen	0x8fbc8f	143, 188, 139
forestGreen	0x228b22	34, 139, 34
limeGreen	0x32cd32	50, 205, 50
lightGreen	0x90ee90	144, 238, 144
paleGreen	0x98fb98	152, 251, 152
darkGreen	0x006400	0, 100, 0
green	0x008000	0, 128, 0
lime	0x00ff00	0, 255, 0
honeyDew	0xf0fff0	240, 255, 240
seaGreen	0x2e8b57	46, 139, 87
mediumSeaGreen	0x3cb371	60, 179, 113
springGreen	0x00ff7f	0, 255, 127
mintCream	0xf5fffa	245, 255, 250
mediumSpringGreen	0x00fa9a	0, 250, 154
mediumAquaMarine	0x66cdaa	102, 205, 170
aquamarine	0x7fffd4	127, 255, 212

NAME	HEXADECIMAL	VALUE (R, G, B)
turquoise	`0x40e0d0`	64, 224, 208
lightSeaGreen	`0x20b2aa`	32, 178, 170
mediumTurquoise	`0x48d1cc`	72, 209, 204
darkSlateGray	`0x2f4f4f`	47, 79, 79
darkSlateGrey	`0x2f4f4f`	47, 79, 79
paleTurquoise	`0xafeeee`	175, 238, 238
teal	`0x008080`	0, 128, 128
darkCyan	`0x008b8b`	0, 139, 139
aqua	`0x00ffff`	0, 255, 255
cyan	`0x00ffff`	0, 255, 255
lightCyan	`0xe0ffff`	224, 255, 255
azure	`0xf0ffff`	240, 255, 255
darkTurquoise	`0x00ced1`	0, 206, 209
cadetBlue	`0x5f9ea0`	95, 158, 160
powderBlue	`0xb0e0e6`	176, 224, 230
lightBlue	`0xadd8e6`	173, 216, 230
deepSkyBlue	`0x00bfff`	0, 191, 255
skyBlue	`0x87ceeb`	135, 206, 235
lightSkyBlue	`0x87cefa`	135, 206, 250
steelBlue	`0x4682b4`	70, 130, 180
aliceBlue	`0xf0f8ff`	240, 248, 255

NAME	HEXADECIMAL	VALUE (R, G, B)
dodgerBlue	`0x1e90ff`	30, 144, 255
slateGray	`0x708090`	112, 128, 144
slateGrey	`0x708090`	112, 128, 144
lightSlateGray	`0x778899`	119, 136, 153
lightSlateGrey	`0x778899`	119, 136, 153
lightSteelBlue	`0xb0c4de`	176, 196, 222
cornflowerBlue	`0x6495ed`	100, 149, 237
royalBlue	`0x4169e1`	65, 105, 225
midnightBlue	`0x191970`	25, 25, 112
lavender	`0xe6e6fa`	230, 230, 250
navy	`0x000080`	0, 0, 128
darkBlue	`0x00008b`	0, 0, 139
mediumBlue	`0x0000cd`	0, 0, 205
blue	`0x0000ff`	0, 0, 255
ghostWhite	`0xf8f8ff`	248, 248, 255
slateBlue	`0x6a5acd`	106, 90, 205
darkSlateBlue	`0x483d8b`	72, 61, 139
mediumSlateBlue	`0x7b68ee`	123, 104, 238
mediumPurple	`0x9370db`	147, 112, 219
blueViolet	`0x8a2be2`	138, 43, 226
indigo	`0x4b0082`	75, 0, 130

NAME	HEXADECIMAL	VALUE (R, G, B)
darkOrchid	`0x9932cc`	153, 50, 204
darkViolet	`0x9400d3`	148, 0, 211
mediumOrchid	`0xba55d3`	186, 85, 211
thistle	`0xd8bfd8`	216, 191, 216
plum	`0xdda0dd`	221, 160, 221
violet	`0x40e0d0`	238, 130, 238
purple	`0x800080`	128, 0, 128
darkMagenta	`0x8b008b`	139, 0, 139
magenta	`0xff00ff`	255, 0, 255
fuschia	`0xff00ff`	255, 0, 255
orchid	`0xda70d6`	218, 112, 214
mediumVioletRed	`0xc71585`	199, 21, 133
deepPink	`0xff1493`	255, 20, 147
hotPink	`0xff69b4`	255, 155, 180
lavenderBlush	`0xfff0f5`	255, 240, 245
paleVioletRed	`0xdb7093`	219, 112, 147
crimson	`0xdc143c`	220, 20, 60
pink	`0xffc0cb`	255, 192, 203
lightPink	`0xffb6c1`	255, 182, 193

21.1.7. Default class member values

The following default values are defined by this encoding.

NOTE: Should this be encoding-specific, or defined in the conceptual model?

Table 98 — Symbolizer

NAME	DEFAULT VALUE
visibility	true
opacity	1
zOrder	1
fill	default Fill
stroke	default Stroke
marker	default Marker
alphaChannel	1
colorChannels	For multi-field coverage: fields[0], fields[1], fields[2]
singleChannel	For single-field coverage: fields[0]

Table 99 — Stroke class default values

NAME	DEFAULT VALUE
opacity	1
width	1 px
color	black

Table 100 — Fill class default values

NAME	DEFAULT VALUE
opacity	1
color	white

Table 101 — Marker class default values

NAME	DEFAULT VALUE
elements	Single default Dot

Table 102 — Dot class default values

NAME	DEFAULT VALUE
size	10 px
color	white

21.1.8. Examples

Example — CCSSS Example encoding style using “Core” requirements class

```
.title 'Styling a land use layer'
.abstract 'Styling land use data with Style & Symbology Core'

#Landuse[dataLayer.type = vector]
{
   visibility: false;

   [viz.sd < 200000 and viz.date > DATE('2020-01-01')]
   {
      visibility: true;
      opacity: 0.5;
      zOrder: 1;
   }
}
```

21.1.8.1. Basic Vector Styling

Example 1 — CCSSS Example encoding style for polygon features using “Basic Vector Styling” requirements class

```
.title 'Styling polygon vector features'
.abstract 'Basic vector features styling (polygons)'
#Landuse[dataLayer.type = vector and dataLayer.featuresGeometryDimensions = 2]
{
   // Do not show Landuse layer by default
   visibility: false;

   // Show land use if zoomed in more than 1:200,000 for data valid within
visualization's selected time range
   [viz.sd < 200000 and validDate >= viz.timeInterval.start.date and validDate
<= viz.timeInterval.end.date]
   {
```

```
      visibility: true;
      opacity: 0.8;
      zOrder: 1;
      fill: (color: gray; opacity: 0.5);
      stroke: (color: gray; width: 2.0 px; opacity: 1.0);

      // Select different fill and stroke color based on FunctionCode property
      [FunctionCode = 'parking']
      {
         fill.color: darkGray;
         stroke.color: #202020;
      }
      [FunctionCode = 'park']
      {
         fill.color: darkGreen;
         stroke.color: green;
      }
      [FunctionCode = 'commercial']
      {
         fill.color: lightGray;
         stroke.color: lightGray;
      }

      // If zoomed in more than 1:10,000
      [viz.sd < 10000]
      {
         // Change stroke width to 4 pixels
         stroke.width: 4.0 px;
         // Add a text marker (positioned at centroid + horizontal offset)
showing FunctionTitle property
         marker: (elements: (
            Text(
               position: 20 0; // Offset 20 pixels to the right
               text: FunctionTitle;
               alignment: left top;
               font: (
                  face: 'Arial';
                  size: 14;
                  bold: true;
                  italic: true;
                  opacity: 1.0;
                  color: black;
               )
            )
         ));
         // Add icons at centroid based on land use function code property
         [FunctionCode = 'parking']
         {
            marker.elements[1]: (
               Image(
                  image: (uri: 'http://example.com/parkingIcon'; path:
'parkingIcon.png'; id: 'parking'; type: 'image/png'; ext: 'png');
                  hotSpot: 50 pc 50 pc; tint: white; blackTint: blue;
alphaThreshold: 0.1;
               )
            )
         }
         [FunctionCode = 'park']
         {
            marker.elements[1]: (
               Image(
                  image: (uri: 'http://example.com/park'; path: 'park.png'; id:
 'park'; type: 'image/png'; ext: 'png');
```

```
                  hotSpot: 50 pc 50 pc; tint: white; blackTint: blue;
alphaThreshold: 0.1;
               )
            )
         }
         [FunctionCode = 'commercial']
         {
            marker.elements[1]: (
               Image(
                  image: (uri: 'http://example.com/commercial'; path:
'commercial.png'; id: 'commercial'; type: 'image/png'; ext: 'png');
                  hotSpot: 50 pc 50 pc; tint: white; blackTint: blue;
alphaThreshold: 0.1;
               )
            )
         }
      }
   }
}
```

Example 2 — CCSSS Example encoding style for line features using "Basic Vector Styling" requirements class

```
.title 'Styling line vector features'
.abstract 'Basic vector features styling (lines)'

#Roads[dataLayer.type = vector and dataLayer.featuresGeometryDimensions = 1]
{
   visibility: false;

   [viz.sd < 200000 and validDate >= viz.timeInterval.start.date and validDate
<= viz.timeInterval.end.date]
   {
      visibility: true
      opacity: 0.8;
      zOrder: 2;
      stroke: (color: gray; width: 5.0 px; opacity: 1.0);

      // If zoomed in more than 1:10,000
      [viz.sd < 10000]
      {
         // Change stroke width to 8 meters
         stroke.width: 8.0 m;
         // Add Dot at each segment end-point
         marker: (elements: (
            Dot(
               size: 10 m; // Alias for base Shape class stroke width
               color: white;
            )
         ));
      }
   }
}
```

Example 3 — CCSSS Example encoding style for point features using "Basic Vector Styling" requirements class

```
.title 'Styling point vector features'
.abstract 'Basic vector features styling (points)'
#Amenities[dataLayer.type = 'vector' and dataLayer.featuresGeometryDimensions =
 0]
{
```

```
   visibility: false;

   [viz.sd < 200000 and validDate >= viz.timeInterval.start.date and validDate
<= viz.timeInterval.end.date]
   {
      visibility: true
      opacity: 0.5
      zOrder: 3

      // If zoomed in more than 1:10,000
      [viz.sd < 10000]
      {
         // Add a double-Dot marker (positioned at centroid)
         marker: (elements: (
            Dot(  // larger white dot underneath
               size: 10 px; // Alias for base Shape class stroke width
               alignment: center middle;
               color: white;
            ),
            Dot(  // smaller orange dot in center
               size: 8 px; // Alias for base Shape class stroke width
               alignment: center middle;
               color: orange;
            )
         ));
         [viz.sd < 5000]
         {
            // Add a Text graphic to Marker at 1:5000
            marker.elements[1]: (
               Text(
                  position: 20 0; // Offset 20 pixels to the right
                  text: Name; // Name of Amenity
                  alignment: left middle;
                  font: (
                     face: 'Arial';
                     size: 12;
                     bold: false;
                     italic: false;
                     opacity: 1.0;
                     color: darkGray;
                  )
               )
            )
         }
      }
   }
}
```

21.1.8.2. Basic Coverage Styling

Example 1 — CCSSS Example encoding style rendering DEM with a color map using "Basic Coverage Styling" requirements class

```
.title 'DEM with color map'
.abstract 'Applying a color map to a Digital Elevation Model with Style &
Symbology Basic Coverage Styling'
#Elevation[dataLayer.type = coverage]
{
   visibility: false;
```

```
   [viz.sd < 200000]
   {
      visibility: true;
      opacity: 0.8;
      zOrder: 1;

      singleChannel: elevation; // Use elevation coverage field
      colorMap: (0 96 136 73, 900 226 219 167, 1300 252 197 117, 1900 254 168
134, 2500 250 250 250);
   }
}
```

Example 2 — CCSSS Example encoding style rendering sentinel-2 in natural color from 3 bands using "Basic Coverage Styling" requirements class

```
.title 'sentinel-2 natural color'
.abstract 'Styling a Sentinel-2A coverage with Style & Symbology Basic
Coverage Styling'
#Sentinel2L2A[dataLayer.type = coverage]
{
   visibility: false;

   [viz.sd < 200000]
   {
      visibility: true;
      opacity: 0.8;
      zOrder: 1;

      colorChannels: B04 B03 B02;
      alphaChannel: 1.0;
      [SCL=8] { alphaChannel: 0.5; }   // Reduce alpha channel value for
medium cloud proability
      [SCL=9] { alphaChannel: 0.0; }   // Zero alpha channel value for high
cloud proability
   }
}
```

21.1.8.3. Arithmetic Operations (with Basic Coverage Styling)

Example — CCSSS Example encoding style rendering NDVI with color map using "Basic Coverage Styling" requirements class

```
.title 'sentinel-2 NDVI'
.abstract 'Styling a Sentinel-2A coverage for NDVI with Style & Symbology
Basic Coverage Styling'
#Sentinel2L2A[dataLayer.type = coverage]
{
   visibility: false;

   [viz.sd < 200000]
   {
      visibility: true;
      opacity: 0.8;
      zOrder: 1;

      singleChannel: (B08 - B04)/(B08 + B04);
      alphaChannel: 1.0;
      [SCL=8] { alphaChannel: 0.5; }   // Reduce alpha channel value for
medium cloud proability
```

```
      [SCL=9] { alphaChannel: 0.0; }   // Zero alpha channel value for high
cloud proability
      colorMap: (-1.0 saddleBrown,0.0 peru,0.2 goldenrod,0.5 olive,0.6
yellowGreen,0.8 greenYellow,1.0 lime);
   }
}
```

21.1.8.4. Hill Shading

Example 1 — CCSSS Example encoding style rendering DEM with hill-shading and a color map using "Basic Coverage Styling" and "Hill Shading" requirements classes

```
.title 'hill-shaded DEM with color map'
.abstract 'Applying hill-shading and a color map to a Digital Elevation Model
with Style & Symbology Basic Coverage Styling'
#Elevation[dataLayer.type = coverage]
{
   visibility: false;

   [viz.sd < 200000]
   {
      visibility: true;
      opacity: 0.8;
      zOrder: 1;

      singleChannel: elevation; // Use elevation coverage field
      // This color map is mapped from elevation values
      colorMap: (0 96 136 73, 900 226 219 167, 1300 252 197 117, 1900 254 168
134, 2500 250 250 250);
      hillShading: (factor: 56; sun: (azimuth: 45.0; elevation: 60.0));
   }
}
```

Example 2 — CCSSS Example encoding style rendering DEM with hill-shading using an opacity and shading intensity color map using "Basic Coverage Styling" and "Hill Shading" requirements classes

```
.title 'hill-shaded DEM with opacity and color map for shading intensity'
.abstract 'Applying hill-shading a color map to a Digital Elevation Model with
Style & Symbology Basic Coverage Styling'

// This type of intensity-based hill shading can be overlaid on top of imagery
or vector maps

#Elevation[dataLayer.type = coverage]
{
   visibility: false;

   [viz.sd < 200000]
   {
      visibility: true;
      opacity: 0.8;
      zOrder: 1;

      singleChannel: elevation; // Use elevation coverage field
      hillShading: (
         factor: 56;
         sun: (azimuth: 45.0; elevation: 60.0)
         // These colors and opacity are mapped from the 0..1 shading
intensity rather than elevation values
```

```
            colorMap : (0 black, 0.15 gray, 0.35 silver, 0.55 white);
            opacityMap: (0 0.75, 0.15 0.50, 0.35 0.25, 0.55 0.00);
        );
    }
}
```

21.2. Requirements

A

ANNEX A (NORMATIVE) ABSTRACT TEST SUITE

A

ANNEX A (NORMATIVE) ABSTRACT TEST SUITE

A Symbology Encoding or Rendering Engine implementation shall satisfy at minimum the abstract tests of the “Core” conformance class to be conformant with this specification, and optionally the abstract tests of any number of additional conformance classes.

The root OGC URI identifier for the conformance classes of this Abstract Tests Suite is:

http://www.opengis.net/spec/symbology-1/2.0/conf/

A.1. Conformance Class “Core”

CONFORMANCE CLASS A.1	
IDENTIFIER	`http://www.opengis.net/spec/symbology-1/2.0/conf/core`
SUBJECT	Requirements Class “Core”
TARGET TYPE	Encoding Rendering engine

ABSTRACT TEST A.1	
IDENTIFIER	`/conf/core/rules`
REQUIREMENT	Requirement 1: `/req/core/rules`
TEST PURPOSE	Validate the encoding of styling rules
TEST METHOD	Given: When: Then:

A.2. Conformance Class “Basic Vector Features Styling”

CONFORMANCE CLASS A.2	
IDENTIFIER	`http://www.opengis.net/spec/symbology-1/2.0/conf/vector`
SUBJECT	Requirements Class “Basic Vector Features Styling”
TARGET TYPE	Encoding Rendering engine

ABSTRACT TEST A.2	
IDENTIFIER	`/conf/vector/stroke`
REQUIREMENT	Requirement 21: `/req/vector/stroke`
TEST PURPOSE	Validate the encoding of a symbolizer stroke
TEST METHOD	Given: When: Then:

A.3. Conformance Class “Basic Coverage Styling”

CONFORMANCE CLASS A.3	
IDENTIFIER	`http://www.opengis.net/spec/symbology-1/2.0/conf/coverage`
SUBJECT	Requirements Class “Basic Coverage Styling”
TARGET TYPE	Encoding Rendering engine

ABSTRACT TEST A.3	
IDENTIFIER	`/conf/coverage/color-channels`
REQUIREMENT	Requirement 29: `/req/coverage/color-channels`
TEST PURPOSE	Validate the encoding of a symbolizer `colorChannels`
TEST METHOD	Given: When: Then:

B

ANNEX B (INFORMATIVE) MAPPING OF SLD/SE AND NOTABLE VENDOR EXTENSIONS TO THE CONCEPTUAL MODEL

B

ANNEX B (INFORMATIVE) MAPPING OF SLD/SE AND NOTABLE VENDOR EXTENSIONS TO THE CONCEPTUAL MODEL

This annex maps constructs of the OGC Styled Layer Descriptor (SLD) and Symbology Encoding (SE) 1.1 to the conceptual model and requirements classes defined in this Standard, with the goal of facilitating conformance to this Standard for Rendering engines based on SLD/SE and writing conversion tools that can convert in either or both direction between SLD/SE and the two encodings defined in this Standard.

C

ANNEX C (INFORMATIVE) PORTRAYAL USE CASES GALLERY

C

ANNEX C
(INFORMATIVE)
PORTRAYAL USE CASES GALLERY

This annex illustrates the capabilities of the different requirements classes defined in this standard with a map gallery of practical use cases alongside the styles used to generate the maps encoded in the *Cascading Styles and Symbology Stylesheets* encoding defined in this Standard.

See supplementary materials for same style sheets encoded using the JSON Styles & Symbology encoding.

C.1. Choropleth (graduated map)

Requirements class used:

- Core
- Basic Vector Features Styling

Data sources used

- A rectangular grid of the fraction of high vegetation.

Example — Style encoded using CCSSS encoding

```
{
  // Note: The following line is optional as 1 pixel black stroke is the
default
  stroke: { black; width: 1px }
  [highVegetationFract between 0   and 0.2] { fill: { yellow }; };
  [highVegetationFract between 0.2 and 0.5] { fill: { orange }; };
  [highVegetationFract > 0.5]               { fill: { red } };
}
```

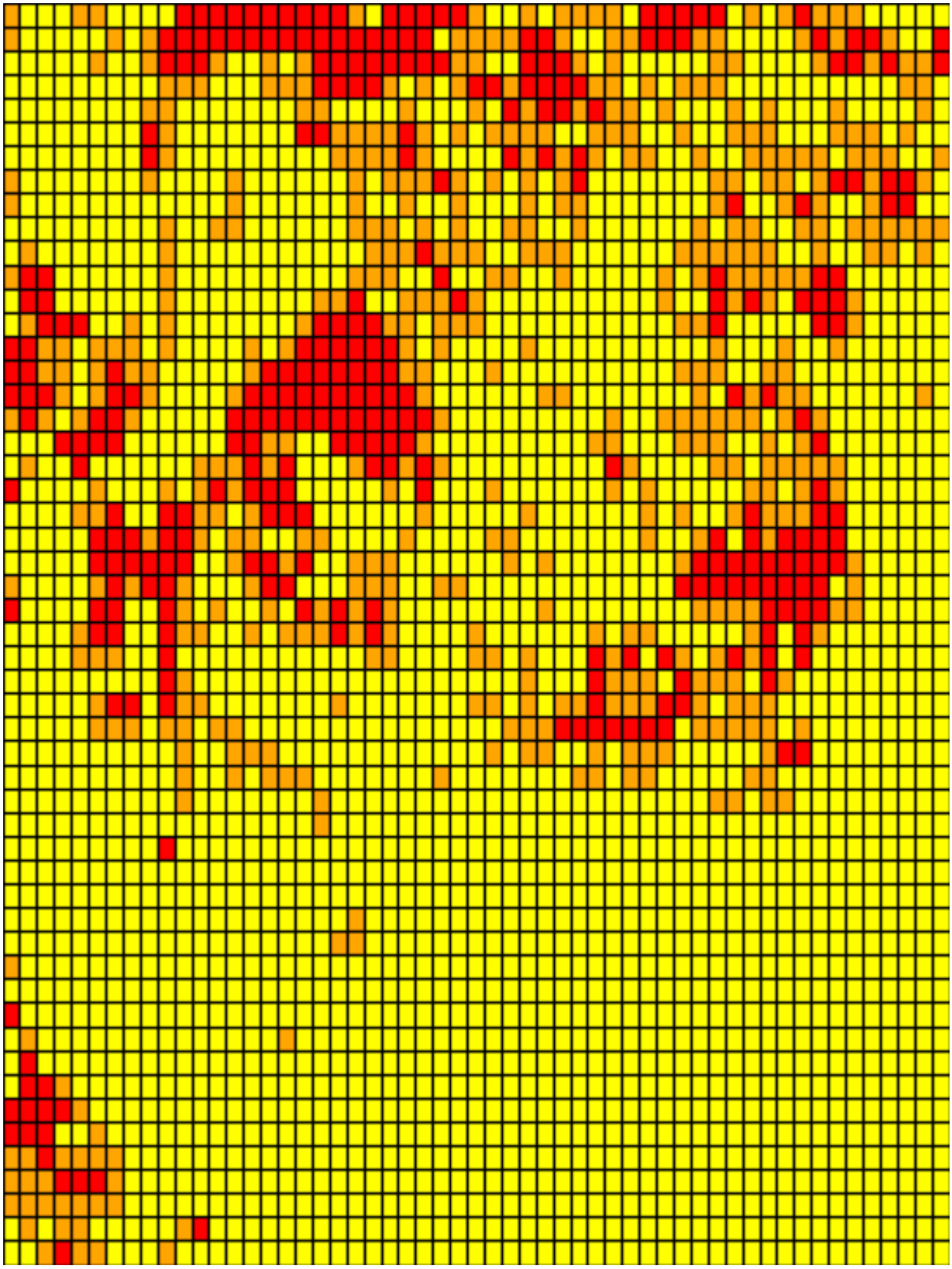

Figure C.1 — Example rendering of the Choropleth map

C.2. Overriding rules

Requirements class used:

- Core
- Basic Vector Features Styling

Data sources used

- Circumpolar distribution and carbon storage of thermokarst landscapes

Citation: Olefeldt, D., Goswami, S., Grosse, G., Hayes, D., Hugelius, G., Kuhry, P., McGuire, A.D, Romanovsky, V.E, Sannel, A.B.K., Schuur, E.A.G., Turetsky, M.R: Circumpolar distribution and carbon storage of thermokarst landscapes. Nature Communications, 7, 13043. Available at: https://www.nature.com/articles/ncomms13043#f3 (2016)

Example — Style encoded using CCSSS encoding

```
#Thermokarst
{
   stroke: { width: 0 };
   [TKWP = 'Low'                                        ]{ fill: { #a1ff74 }; }
   [TKThLP = 'Low'                                      ]{ fill: { #74b2ff }; }
   [TKHP = 'Low'                                        ]{ fill: { #fd846d }; }
   [TKWP = 'Low' AND TKHP = 'Low'                       ]{ fill: { #f7ff7c }; }
   [TKWP = 'Low' AND TKThLP = 'Low'                     ]{ fill: { #beffe9 }; }
   [TKWP = 'Low' AND TKThLP = 'Low' AND TKHP = 'Low']{ fill: { #9d9d9d }; }
   [TKWP = 'Moderate'                                   ]{ fill: { #4de600 }; }
   [TKThLP = 'Moderate'                                 ]{ fill: { #0070ff }; }
   [TKHP = 'Moderate'                                   ]{ fill: { #fe0001 }; }
   [TKWP = 'Moderate' AND TKHP = 'Moderate'             ]{ fill: { #eae600 }; }
   [TKThLP = 'Moderate' AND TKHP = 'Moderate'           ]{ fill: { #ff00c4 }; }
   [TKWP = 'Moderate' AND TKThLP = 'Moderate'           ]{ fill: { #00e7a9 }; }
   [TKWP = 'High'                                       ]{ fill: { #39a60d }; }
   [TKThLP = 'High'                                     ]{ fill: { #014da9 }; }
   [TKHP = 'High'                                       ]{ fill: { #9a0602 }; }
   [TKWP = 'High' AND TKThLP = 'High'                   ]{ fill: { #00a882 }; }
   [TKWP = 'Very High'                                  ]{ fill: { #334d27 }; }
   [TKThLP = 'Very High'                                ]{ fill: { #002674 }; }
   [TKWP = 'Very High' AND TKThLP = 'Very High'         ]{ fill: { #007f7e }; }
}
```

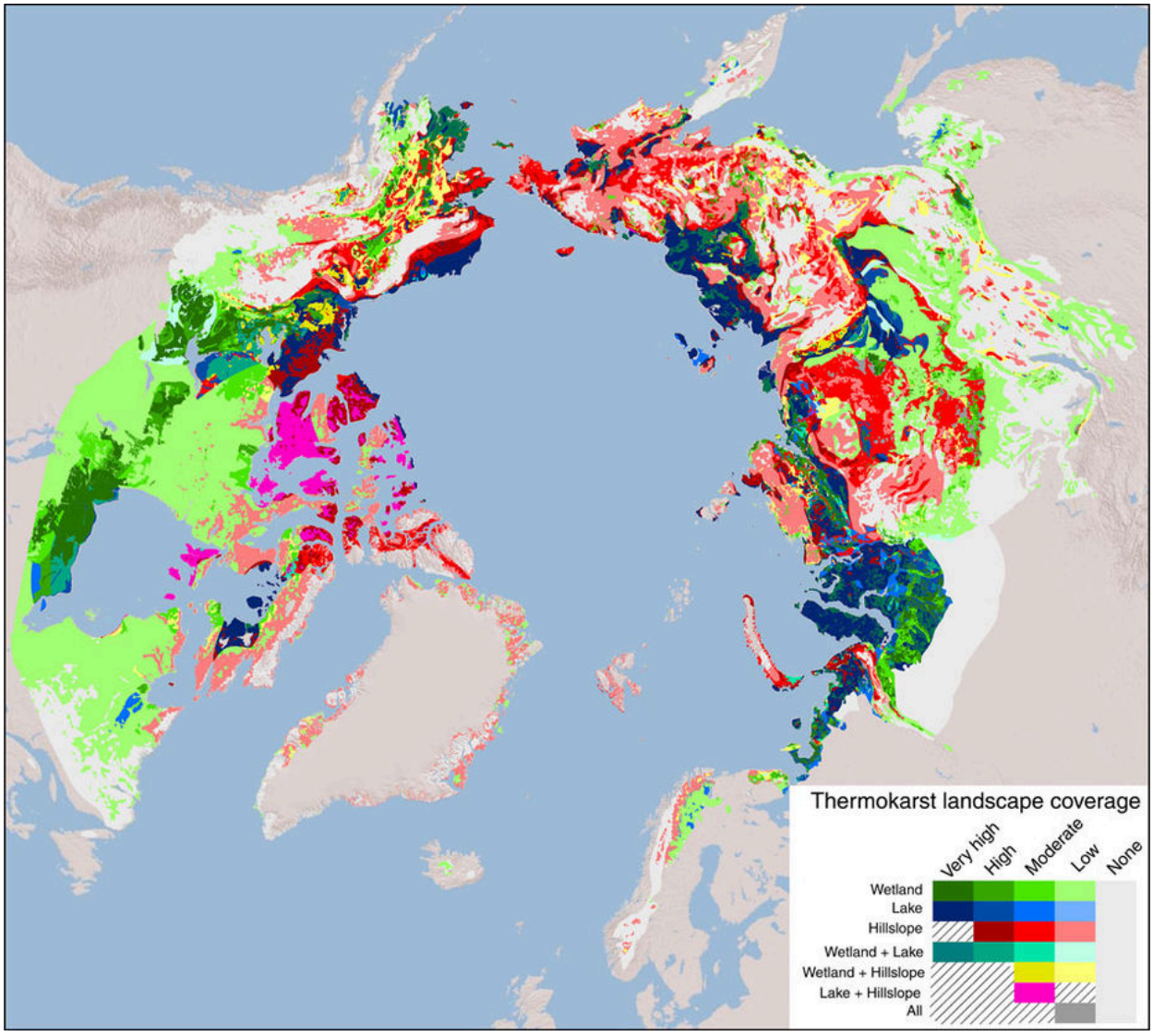

Figure C.2 — Example rendering of the thermokarst landscapes

D

ANNEX D (INFORMATIVE) REVISION HISTORY

ANNEX D (INFORMATIVE) REVISION HISTORY

Table D.1 — Revision history

DATE	RELEASE	AUTHOR	PARAGRAPH MODIFIED	DESCRIPTION
2020-08-27		E. Bocher (CNRS) O. Ertz (HEIG-VD)		Edits considering received and answered comments from TC vote
2019-09-18		E. Bocher (CNRS) O. Ertz (HEIG-VD)		Edits considering received and answered comments (18-067r2)
2019-08-21		E. Bocher (CNRS) O. Ertz (HEIG-VD)		Update abstract to be more comprehensive
2019-03-29		E. Bocher (CNRS) O. Ertz (HEIG-VD)		Edits considering received and answered comments
2018-09-07		E. Bocher (CNRS) O. Ertz (HEIG-VD)		Release for early public comment (18-067)
2018-06-05		E. Bocher (CNRS) O. Ertz (HEIG-VD)		Initial document
2022-12-01		E. Bocher (CNRS) O. Ertz & M. Collombin (HEIG-VD), J. St-Louis (Ecere)		Initial work towards version 2.0
2023-02-23		E. Bocher (CNRS) O. Ertz & M. Collombin (HEIG-VD), J. St-Louis (Ecere)		Progress towards 2.0 in preparation for 125th MM in Frascati

BIBLIOGRAPHY

BIBLIOGRAPHY

This Standard is deeply inspired by the work from the following documents that preceded it:

[1] Testbed-12 Semantic Portrayal, Registry and Mediation Engineering Report

[2] OGC Portrayal Concept Development Study

[3] OGC Testbed-13: Portrayal Engineering Report

[4] OGC Testbed-14: CityGML and AR Engineering Report

[5] OGC Testbed-14: Symbology Engineering Report

[6] OGC Testbed-15: Open Portrayal Framework Engineering Report

[7] OGC Testbed-15: Portrayal Summary ER

[8] Bocher E, Ertz O. 2018. A redesign of OGC Symbology Encoding standard for sharing cartography. PeerJ Computer Science 4:e143, https://doi.org/10.7717/peerj-cs.143