Open Geospatial Consortium, Inc.

Date: 2009-08-17

Reference number of this document: OGC 09-012

Version: 0.3.0

Category: Public Engineering Report

Editor: Craig Bruce

OGC® OWS-6 Symbology-Encoding Harmonization

Warning

This document is not an OGC Standard. This document is an OGC Public Engineering Report created as a deliverable in an OGC Interoperability Initiative and is not an official position of the OGC membership. It is distributed for review and comment. It is subject to change without notice and may not be referred to as an OGC Standard. Further, any OGC Engineering Report should not be referenced as required or mandatory technology in procurements.

Document type:	OpenGIS® Engineering Report
Document subtype:	NA
Document stage:	Approved for Public Release
Document language:	English

Preface

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

The changes made in this document version, relative to the previous version, are tracked by OpenOffice.org, and can be viewed if desired. If you choose to submit suggested changes by editing this document, please first accept all the current changes, and then make your suggested changes with change tracking on.

Forward

OWS-6 Testbed

OWS testbeds are part of OGC's Interoperability Program, a global, hands-on and collaborative prototyping program designed to rapidly develop, test and deliver Engineering Reports and Change Requests into the OGC Specification Program, where they are formalized for public release. In OGC's Interoperability Initiatives, international teams of technology providers work together to solve specific geoprocessing interoperability problems posed by the Initiative's sponsoring organizations. OGC Interoperability Initiatives include test beds, pilot projects, interoperability experiments and interoperability support services - all designed to encourage rapid development, testing, validation and adoption of OGC standards.

In April 2008, the OGC issued a call for sponsors for an OGC Web Services, Phase 6 (OWS-6) Testbed activity. The activity completed in June 2009. There is a series of on-line demonstrations available here: http://www.opengeospatial.org/pub/www/ows6/index.html The OWS-6 sponsors are organizations seeking open standards for their interoperability requirements. After analyzing their requirements, the OGC Interoperability Team recommended to the sponsors that the content of the OWS-6 initiative be organized around the following threads:

1. Sensor Web Enablement (SWE)
2. Geo Processing Workflow (GPW)
3. Aeronautical Information Management (AIM)
4. Decision Support Services (DSS)
5. Compliance Testing (CITE)

The OWS-6 sponsoring organizations were:

- U.S. National Geospatial-Intelligence Agency (NGA)
- Joint Program Executive Office for Chemical and Biological Defense (JPEO-CBD)
- GeoConnections - Natural Resources Canada
- U.S. Federal Aviation Agency (FAA)
- EUROCONTROL
- EADS Defence and Communications Systems
- US Geological Survey
- Lockheed Martin

- BAE Systems
- ERDAS, Inc.

The OWS-6 participating organizations were:
52North, AM Consult, Carbon Project, Charles Roswell, Compusult, con terra, CubeWerx, ESRI, FedEx, Galdos, Geomatys, GIS.FCU, Taiwan, GMU CSISS, Hitachi Ltd., Hitachi Advanced Systems Corp, Hitachi Software Engineering Co., Ltd., iGSI, GmbH, interactive instruments, lat/lon, GmbH, LISAsoft, Luciad, Lufthansa, NOAA MDL, Northrop Grumman TASC, OSS Nokalva, PCAvionics, Snowflake, Spot Image/ESA/Spacebel, STFC, UK, UAB CREAF, Univ Bonn Karto, Univ Bonn IGG, Univ Bunderswehr, Univ Muenster IfGI, Vightel, Yumetech.

Contents

Page

Figures

OGC® OWS-6 Symbology-Encoding Harmonization

1 Introduction

1.1 Scope

This OGC® document reports the results achieved in the Decision Support Services (DSS) subtask of the OWS-6 testbed initiative as it relates to the harmonization of OGC Styled Layer Descriptor (SLD) and Symbology Encoding (SE) symbology formats with ISO 19117 symbology format, International Hydrographic Organization S-52 symbology, USGS Topomap symbology, and Homeland Security Emergency Management symbology.

1.2 Document contributor contact points

All questions regarding this document should be directed to the editor or the contributors:

Name	Organization
Dr. Craig S. Bruce	CubeWerx Inc.

1.3 Revision history

Date	Release	Editor	Primary clauses modified	Description
2009-04-27	1.0.0	C. Bruce	Main body	OWS-6 project final release

1.4 Future work

Improvements in this document are desirable to further harmonize the OGC SLD/SE formats with ISO 19117 and other symbology standards.

2 References

The following documents are referenced in this document. For dated references, subsequent amendments to, or revisions of, any of these publications do not apply. For undated references, the latest edition of the normative document referred to applies.

OGC 09-015 (April 2009), *OWS-6 Styled Layer Descriptor (SLD) Changes (Engineering Report)*, Craig Bruce (ed.)

OGC 09-016 (April 2009), *OWS-6 Symbology Encoding (SE) Changes (Engineering Report)*, Craig Bruce (ed.)

OGC 05-078r4 (June 2007), *Styled Layer Descriptor profile of the Web Map Service Implementation Specification (version 1.1.0)*, Markus Lupp (ed.), <http://portal.opengeospatial.org/files/?artifact_id=22364>

OGC 05-077r4 (July 2006), *Symbology Encoding Implementation Specification (version 1.1.0)*, Markus Müller (ed.), <http://portal.opengeospatial.org/files/?artifact_id=16700>

ISO 19117:2005 (2005), *Geographic information — Portrayal*, <http://www.iso.org/iso/iso_catalogue/catalogue_tc/catalogue_detail.htm?csnumber=40395>

ISO 19117:Revision (Revision Draft, January 2009), *Geographic information — Portrayal*

IHO S-52 (1996), *Specifications for Chart Content and Display Aspects of ECDIS, 5th Edition* (amended 1999)

IHO S-52 C&S (March 2004), *IHO COLOUR & SYMBOL SPECIFICATIONS (C&S Specs) for ECDIS, S-52 Appendix 2 – Edition 4.2*, and *IHO PRESENTATION LIBRARY (PresLib) – Edition 3.3*

TENET Report (July 2008), Some Unresolved Issues With The OGC Symbology Encoding (SE), Neil Kirk, <http://portal.opengeospatial.org/files/?artifact_id=29160>

OGC 09-043 (April 2009), *OWS-6 DSS Enhancements to Symbology Encoding in Support of IHO S-52 and UKHO AML*, Alessandro Triglia, <http://portal.opengeospatial.org/files/?artifact_id=32917&version=1>

USGS, *National Mapping Program Technical Instructions — Part 5 — Publication symbols*, <http://rockyweb.cr.usgs.gov/nmpstds/acrodocs/qmaps/5psym202.pdf>

USGS, *National Mapping Program Technical Instructions — Part 6 — Publication symbols*, <http://rockyweb.cr.usgs.gov/nmpstds/acrodocs/qmaps/6psym403.pdf>

USGS, *Topographic Map Symbols*, <http://erg.usgs.gov/isb/pubs/booklets/symbols/topomapsymbols.pdf>

ANSI INCITS 415-2006 (July 2006), Homeland Security Mapping Standard — Point Symbology for Emergency Management, <http://webstore.ansi.org/RecordDetail.aspx?sku=ANSI+INCITS+415-2006>

3 Terms and definitions

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

3.1
graphic
Small icon picture drawn at a point or filling an area

3.2
layer
User-selectable content for a map

3.3
map
Pictorial representation of geographic data

3.4
style
Determines the appearance geographic data

4 Conventions

4.1 Abbreviated terms

CRS	Coordinate Reference System
CSS	Cascading Style Sheets
EMS	Emergency Management Symbology

GML	Geography Markup Language
HTTP	Hypertext Transfer Protocol
IHO	International Hydrographic Organization
ISO	International Organization for Standardization
OGC	Open Geospatial Consortium
SE	Symbology Encoding
SLD	Styled Layer Descriptor
SQL	Structured Query Language
SVG	Scalable Vector Graphics
UML	Unified Modeling Language
URI	Uniform Resource Identifier
USGS	United States Geological Survey
W3C	World Wide Web Consortium
WMS	Web Map Service
XML	Extensible Markup Language

4.2 UML notation

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

5 Harmonization overview

This OGC® document reports the results achieved in the Decision Support Services (DSS) subtask of the OWS-6 testbed initiative as it relates to the harmonization of OGC Styled Layer Descriptor (SLD) and Symbology Encoding (SE) symbology formats with revised ISO 19117 symbology format, International Hydrographic Organization S-52 symbology, USGS Topomap symbology, and Homeland Security Emergency Management symbology.

6 Harmonization between OGC SLD/SE and ISO 19117

6.1 Overview comparison

6.1.1 History

SLD was initially developed by OGC during the WMT-2 (Web-Mapping Testbed) project in 2000. WMT-2 was an Interoperability Program project, so the development was specifically focused on producing implementations quickly. ISO 19117 apparently predated SLD as a draft specification. It was considered during the WMT-2 project, but was dismissed as being practically empty.

By the end of the WMT-2 project, SLD 0.7.0 was defined and had multiple interoperable implementations. SLD had the capability to do simple to somewhat complex renderings for vector features and raster coverages. Most of the changes between version 0.7.0 and 1.1.0 have been syntactic in nature.

The official published version of ISO 19117:2005 was little different from the version dismissed in 2000. It included only feature styles and rules. All symbolization was provided by user-defined textual descriptions, so it is fair to say it had no symbolization capability at all.

ISO 19117:Revision is being revised in 2009 and has considerably more detailed content, especially in the area of graphic parameterization. In fact, it has more graphical capability than SLD 1.1.0.

SLD/SE are also currently being revised as part of the OWS-6 project, to refine the encoding and increase its capability. The harmonization comparison is made mostly between ISO 19117:Revision and the OWS-6 change requests for SLD and SE.

6.1.2 Terminology differences

Terminology differences between SLD and ISO 19117 pose significant challenges for direct harmonization. Field names are not going to match if the concepts these names encode use different terms. Some examples are that SLD refers to “styles” whereas ISO refers to “schema remapping” and “portrayal”.

SE refers to “symbolizers” whereas ISO refers to “symbols”. In the SE design, “symbol” is considered a vague and overloaded term, sometimes meaning a style and sometimes meaning an icon or portrayed feature, so “symbolizer” was chosen instead, avoiding direct use of the term “symbol”. In ISO, “symbol” means a (low-level) style.

SE refers to “graphics” whereas ISO appears to refer to “icons”. Perhaps “icon” is a better term, as “graphic” is rather overloaded.

6.1.3 Scope differences

ISO includes a feature schema remapping mechanism in its definition whereas this is considered out-of-scope for SE. A significant amount of the ISO specification is dedicated to the definition of this mechanism. However, this ability to remap feature data from one application schema to another is a generally useful mechanism that really does not belong specifically in a portrayal system. SE assumes that whatever transformations are needed have already been applied to the feature data before rendering is applied.

SLD includes a concept of map layers but ISO does not include this concept.

6.1.4 XML Schema vs. UML

The design of SLD/SE is defined using XML Schema and the design of ISO 19117 is defined using UML. XML Schema contains some capabilities and concepts that are not portable to other schema languages such as UML. The most notable non-portable concept is XML attributes. In general, XML attributes should be avoided in OGC specifications because no other structuring language can represent this concept.

During the initial development of SLD/SE, it was decided that it would be preferable to make all fields of of an element be elements for improved portability. This principle is evident in the SLD 1.1.0 definition of a **NamedLayer**, since **Name** is a sub-element where names are normally given in attributes in XML. This principle was forgotten, however, with the addition of **version** attributes to SE and SLD.

This principle was also not followed with the SE 1.1.0 **SvgParameter** definition, though there it is more out of convenience, since the XML encoding of graphic properties would be significantly more verbose using sub-elements. However, an alternative proposal to **SvgParameter** is provided in the revised SLD/SE designs.

XML attributes should be replaced by sub-elements where feasible in the revised SLD and SE designs.

6.1.5 Property- and class-name capitalization

The standard practice for property and class names in many programming languages is to use “lowerCamelCase” for property names and “UpperCamelCase” for class names. In “CamelCase”, spaces are removed between words of a phrase and the initial letter of each word is capitalized instead. In lowerCamelCase, the initial letter of the first word is made into lowercase. Letter case is significant in many programming environments, so a class name of **LineString** and a property name of **lineString** could be used in the same declaration without ambiguity.

In ISO UML-design practice, the programming-language paradigm is utilized but an additional two-letter capitalized package identifier (namespace) with a trailing underscore character is placed before class names. For example, a class name might be **SY_LinePointSymbol** and a property of this class might be **lineIcon**.

Unfortunately, OGC does not seem to have a standard practice for naming classes and properties, so a variety of conventions are used. In SLD/SE and various other specifications, UpperCamelCase is used for both properties and classes, where classes normally have the same base name as the archetypal property (element) of the class, with the additional word "**Type**" appended to the name. Properties that are represented as XML attributes use lowerCamelCase names. GML uses a mix of UpperCamelCase and lowerCamelCase for element names, essentially at random.

The general recommendation here is that OGC adopt the ISO practice, but with arbitrary-length namespaces. In XML realizations of the designs, the XML namespace mechanism with lowerCamelCase names should be used rather than the ISO underscore mechanism. For instance, a design class name of **SE_FeatureTypeStyle** would be realized in XML as **se:FeatureTypeStyle** (or with an implicit namespace).

However, this recommendation will be an awkward and contentious issue, involving renaming every identifier in most existing specifications. Therefore, it is recommended that no action be taken on this issue at this time.

6.1.6 Harmonization objective

It is not practical to make SLD/SE have an identical physical representation to ISO 19117 for numerous reasons, including differing OGC and ISO practices and conventions, differing terminology, and the practical need for backward compatibility with existing SLD/SE designs and implementations. Instead, the objective is to provide all of the important functionality offered by the revised ISO design, with a particular focus on the graphical parameters, while retaining the basic character and purpose of the existing SLD/SE designs.

6.2 Map handling

ISO does not address map handling but SLD does with its root **StyledLayerDescriptor** element. The map handing is realized mostly as just a list of map layers (which are discussed in clause 0). This allows an SLD document to contain all of the content of a map (minus specific rendering-environment parameters like image type and map bounding box). In fact, the XML encoding for an OGC WMS GetMap request uses an SLD body to specify the map content. An SLD document can also be used as a simple style library, defining many different styles for many different map layers. An SLD document is also a convenient bundle to use to save, load, and edit all of the styling information for a data store. An SLD-enabled WMS provides public operations for this purpose. An SLD file can also be used to supply the symbology for various feature-file formats. For instance, a Shapefile with ***.shp**, ***.dbf**, etc. files could include a companion ***.sld** file to supply the styling information. (CubeWerx implements this and it is rather handy.)

The revised SLD element hierarchy has a **StyledLayerDescriptor** that references multiple **Layer**s which reference multiple **Style**s which reference multiple **FeatureTypeStyle**s or **CoverageStyle**s. The ISO design starts at handling feature types.

6.3 Layer handling

A “layer” in SLD provides a means of grouping many different but related feature/coverage types in potentially different styles into a single convenient user entity. This reflects the OGC WMS design where the relevant user-selectable content objects are layer and style. Layers often have a one-to-one correspondence with feature types, but sometimes they are more complicated. For example, a layer called “Context” could be defined which includes boundaries, roads, land usage, built-up areas, waterways, buildings, etc. where the content is filtered so that only some feature types are displayed when zoomed out and all are displayed when zoomed in. Many systems such as Google Maps work in this way, altering the feature types selected based on zoom level.

ISO 19117 does not include a layer concept. An intermediate ISO draft from 2007 (ISO/TC 211 N 2167) contained a class called **PF_PortrayalMapping** which is logically equivalent to a layer/style pair (grouping multiple feature-type portrayals and a style description into a single entity), but this class has been dropped in ISO 19117:Revision.

Without layers, a user will always work at the level of feature types, which may be an inconveniently fine granularity for some purposes.

6.4 Schema mapping

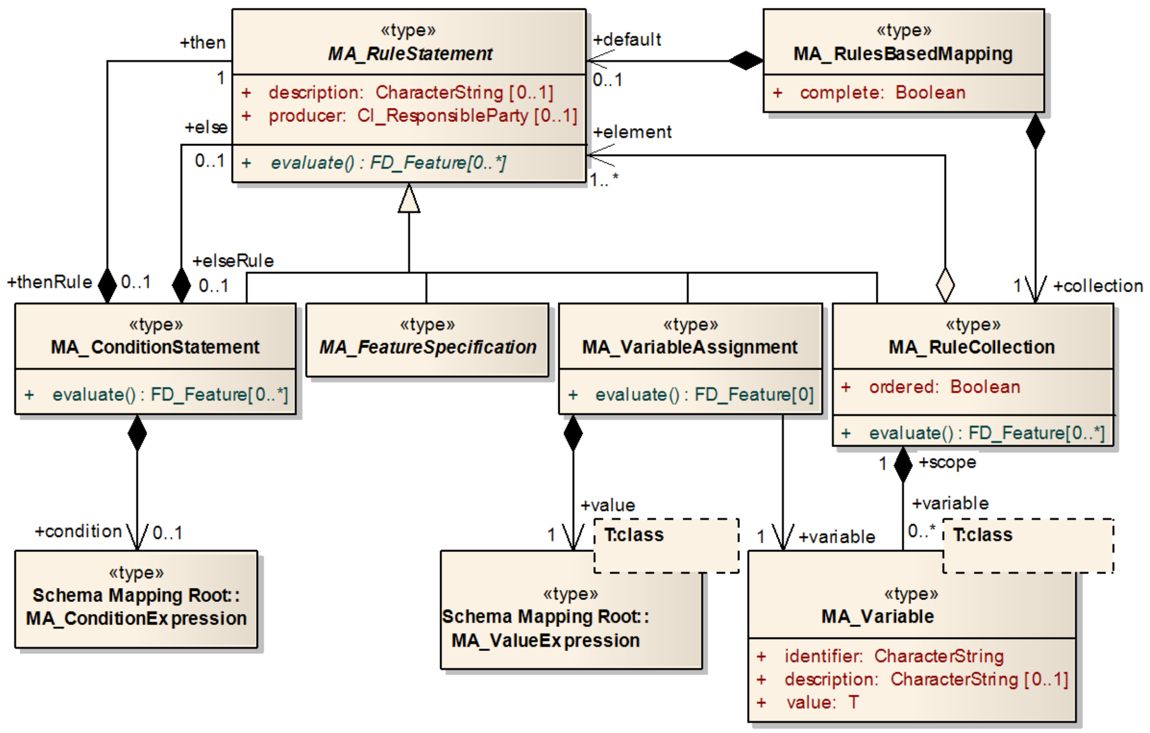
6.4.1 ISO Overview

ISO 19117:Revision defines a complex mechanism for converting features from one schema (type) to another. The concept is that the portrayal mechanism does not have any rule selection built into it but instead relies on the schema-mapping mechanism to take each input user feature and transform it into a (essentially virtual) portrayal feature (or features) that corresponds to one (or more) particular type of symbol to be drawn. I.e., the portrayal features will have a one-to-one correspondence to the legend of the map.

For example, consider a user feature type for roads that have a line-geometry property, a name property, and a road-type property with possible values “minor street”, “collector road”, and “highway”, which the user wishes to portray differently. The schema mapping mechanism would take each input feature and convert it into one of the three portrayal feature types which correspond to the road types. These portrayal features would only have the geometry property and name because the road-type property is no longer relevant after the portrayal-feature type has been discerned.

6.4.2 ISO Rule-based schema mapping

The rule-based schema-mapping mechanism represents a “program” of statements that are executed sequentially by the runtime system. The crux of the mapping mechanism is the **MA_RuleStatement** class and its derived classes defined in the UML diagram in Figure 1 [ISO 19117:Revision].



«type» MA_RuleStatement
+ description: CharacterString [0..1]
+ producer: CI_ResponsibleParty [0..1]
+ evaluate() : FD_Feature[0..*]
«type» MA_RulesBasedMapping
+ complete: Boolean
+then
1
+else
0..1
+default
0..1
+element
1..*
+thenRule
0..1
+elseRule
0..1
1
+collection
«type» MA_ConditionStatement
+ evaluate() : FD_Feature[0..*]
«type» MA_FeatureSpecification
«type» MA_VariableAssignment
+ evaluate() : FD_Feature[0]
«type» MA_RuleCollection
+ ordered: Boolean
+ evaluate() : FD_Feature[0..*]
1
+scope
+variable
0..*
+condition
0..1
+value
1
1
+variable
T:class
T:class
«type» Schema Mapping Root::MA_ConditionExpression
«type» Schema Mapping Root::MA_ValueExpression
«type» MA_Variable
+ identifier: CharacterString
+ description: CharacterString [0..1]
+ value: T

Figure 1: ISO Rule Statement

The **MA_ConditionStatement** allows an if/then/else statement to be represented. It references an **MA_ConditionExpression**. Expressions are represented with two string values, one representing the language of the expression and the other representing the expression content. This is certainly better than defining an ad-hoc expression language, though leaving the language open is also problematic. Interoperability requires people to choose the same expression language.

The **MA_FeatureSpecification** statement constructs an output feature by assigning expression values to a list of the properties of the output feature. The data type of the expression result is open, so it can handle integers, geometries, etc. An **MA_RulesBasedMapping** could construct any number of output features, including zero, and gives the option to produce a default feature(s) if none are produced by the normal program.

The **MA_VariableAssignment** statement assigns a value to a runtime variable by evaluating an expression. A variable is scoped to a specific **MA_RuleCollection** and its subordinate statements.

The **MA_RuleCollection** class allows a block of statements to be treated as a single statement, like the curly braces ("{" and "}") in C-derived languages. Since **MA_RuleCollection** has an open diamond UML notation on its 1..* **element** link to **MA_RuleStatement**, it does not "contain" or "own" the statements; it merely "references" them. This means that statements can be referenced (executed) more than once, which allows the semantic equivalent of **goto** statements in other programming languages and allows loops.

ISO 19117:Revision says, "These classes define a grammar that, lacking a looping capability, is not Turing complete," but this appears to be false because of the open diamond. In the simplest case, a **MA_RuleCollection** could reference itself with its **element** property, which would be valid since **MA_RuleCollection** is derived from **MA_RuleStatement**, assuming that this is an allowed interpretation of the UML open diamond with a class that has such a link that points back to itself. If an object can refer to any object of the same class, then it can refer to itself. Loops could be more indirect as well; rule collection A could refer to rule collection B as an **element** and rule collection B could refer back to rule collection A. If the open diamond remains, then an explicit "no-loops" requirement needs to be added to the specification to disallow loops. This appears to be the intention of the authors, so the subject of the computational complexity of Turing completeness will not be pursued here.

The semantics of rule-based mapping indicate that it is a many-to-many feature mapping, i.e., it converts one or more input features to zero or more portrayals (portrayal features). It is clear how the zero or more output features are generated (by zero or more **MA_FeatureSpecification** instructions being executed in the program); however it is unclear how more than one feature is selected to be fused together. Also, the practical benefit of many-to-many feature mapping is unclear and perhaps dubious. What crucially important portrayal work requires a many-to-many mapping and cannot adequately be simulated by a much simpler and more efficient one-to-many mechanism?

The variable-assignment mechanism and the rule-statement-reuse capability do no appear to provide any additional expressive power to the mapping mechanism since loops appear to be disallowed. The practical value of reuse is to reduce the total number and redundancy of rule statements in a symbology collection by reusing statements between portrayal specifications, which also reduces the management effort, and the practical value of variable assignment appears to be to allow a greater degree of reuse by allowing statement blocks to be parameterized. For example, if one feature type specified the number of road lanes with a property called **nlanes** and another feature type with **width_per_side**, then the total number of lanes could be computed assigned to a variable and a common statement block could be used to generate the portrayal feature. On the down side, variable assignment will complicate and in most practical cases eliminate runtime optimization.

The runtime model implied by the rule-based mechanism is quite inefficient. The runtime system supplies the program with one feature or overlapping groups of features at a time and it executes the statements of the program sequentially and builds temporary features one property at a time to be used for portrayal. The portrayal mechanism then examines these temporary features and disposes of them.

Performing these micro-operations for each input feature (or worse, each overlapping group of input features) will be slow to execute. Analyzing these mapping programs at runtime and transforming them into some more efficient mechanism will be the only means to make execution efficient. This kind of optimization will likely be practical only with certain simple patterns of mapping statements. Without program transformation, spatial and attribute indexing will likely be unusable, and these are very powerful query optimization mechanisms. A literally-executed rule-mapping program may fetch every feature in a feature table or multiple feature tables where only a few features would have been fetched with query optimization. A lack of index utilization will most likely be a much larger practical problem than the relative inefficiency of the execution of the micro-operations of rule-based programs.

An author of ISO 19117:Revision informally indicated that a recommended practice for encoding symbology will be to maximize the reuse of statement blocks by substituting AND, OR, and NOT logical operators with conditional branches to reused code. This will complicate query optimization or make it infeasible. For example, if a portrayal system is implemented on top of a relational database system, an objective for query optimization may be to reduce the program to a minimal number of independent SQL query conditions that can be executed to retrieve the necessary features for each different style of portrayal. To do this, the AND, OR, and NOT logical operations will need to be extracted from the structure of the program and placed back into the query conditions.

In general, either rule-based mapping will be slow to execute or implementations will need sophisticated rule-based-program analysis and query-optimization capabilities to be efficient. All implementations will need the internal mechanisms to execute the programs verbatim since query optimization will not always be feasible or possible. In other words, practical implementations will need two different rule-based runtime systems.

6.4.3 Other ISO schema-mapping methods

Two other schema-mapping methods known as transformation mapping and population mapping are also defined for use with ISO 19117:Revision.

Transformation mapping models relational-algebra operations of Select (filter those features selected), and Project (select only certain properties for retrieval), and Join (combine multiple tables together into one based on a matching property value). The ISO revision draft does not have this section fleshed out at the time of writing this report, but one can imagine how it works. Combinations in sequence of the given basic operations can extract any information from a set of database tables (features in this case). One could consult any number of sources to learn more about the well-known subject of "relational algebra" ([3] Wikipedia)

A practical problem with the transformation-mapping approach is that it imposes a burden on implementors to supply a relational-algebra execution system in their portrayal system. A portrayal system that is implemented on top of relational database system will provide a relational-algebra execution for free, but one implemented on top of a directory of Shapefiles will not. Select and Project are relatively easy to implement in any feature-retrieval system, but Join is much more complicated, especially to do efficiently (which needs indexing), even though the need for Joins should be rather rare in most styling.

The execution of a transformation mapping will be much more efficient than the rule-based approach, since large macro-operations are involved and decades of research have gone into optimizing these macro-operations and many practical (database) systems provide these optimizations for use.

Population mapping is completely undefined in the ISO 19117:Revision draft. Apparently, it is the baby of one person involved in the design process. One could argue that international standards are not a suitable place to conduct experimental research.

An overall problem with the ISO approach to feature mapping is that the multiplicity of equivalent methods either imposes the burden on all implementations to support all three (really, four) methods, or the practical interoperability of the styling encoding will be reduced. If some systems support some methods and different symbology encodings use different methods, then some ISO symbology encodings will not work on some systems. Also, the people who manage and exchange symbology encodings will need to be proficient in all of the methods and so will the editing tools they use. Automatically translating between the different methods and between external symbology encodings (such as OGC SE) may be difficult or impossible.

Another overall problem is in the complexity and verbosity of the design. The design spans dozens of classes, not even considering all of the unspecified classes of the transformation and population mapping mechanisms and the parallel Portrayal Feature mechanism. The description goes on for 43 pages and is still quite terse. By comparison, the description of the equivalent material in SLD 1.1.0 goes on for 8 pages describing two classes and includes several examples and an extended discussion of scale handling. With "implementation-free" designs, there is always the temptation to add more and more layers of abstractions. There will be no push-back from implementors because there are none. If mechanisms like this were added to SE, the implementors would revolt, and perhaps the style designers along with them.

6.4.4 SE feature selection

OGC SE, along with ISO 19117:2005, uses a simple declarative rule-based mechanism for selecting the portrayal to apply to features. The **FeatureTypeStyle** in SE contains a **FeatureTypeName** to select (identify) the feature type and a **Rule** which contains an OGC **Filter** expression and a **MinScaleDenominator** and a **MaxScaleDenominator**. A **Filter** expression contains an (unfortunately verbose) XML encoding of common expressions like "**a > 5 and b = 11**". The scale elements identify the fixed range of scales for which the rule is applicable. Special dimensional constraints can be applied using a **DomainConstraints** sub-element and (raster) coverages are handled with a **CoverageStyle** element that is parallel to **FeatureTypeStyle**.

The SE runtime model is simple. Features of one feature type are passed as input to the rendering engine which uses an SE document plus instance-specific parameters (e.g., map extent and resolution, background color) as control information and the rendering engine produces a rendered map as output (e.g., a PNG image).

The SE design is easy to use and straightforward to implement and optimize. A **FeatureTypeStyle** is limited to a single feature type and the the rules are simple filters that can be implemented in numerous different ways and are easy to optimize. An implementation could retrieve all relevant features in one query or make a separate query for each rule (or each symbolizer or each rule). The queries are simple conditions for which indexing may be available. The implementors can choose between execution efficiency and implementation simplicity.

The fixed scale filtering means that out-of-scope rules can simply be discarded before query-execution begins (thereby eliminating all effort in executing the other constraints of these rules). ISO 19117 uses an external function (say, **Scale()**) as an embedded part of a rule expression. This makes optimization more difficult since a static analysis of a rule condition is needed to determine if it contains a **Scale()** invocation and if it is used in a simple way (such as "**a == 11 and Scale() >= 20000**") versus a complex way (such as "**Scale() >= b**" or "**a == 11 or Scale() >= 20000**"). Combined with the difficulty in optimizing ISO 19117:Revision schema mappings, ISO implementations may waste a significant or even an enormous amount of time executing schema-mapping programs that always produce zero features as output because the scale condition is buried inside of the mapping program and the query optimizer could not determine that the entire program or branches of the program need not be executed. An enormous amount of time could be wasted since portrayal at finer scales tends to access a much greater density of data than at coarser scales. For example, a road portrayal may include all city streets at a fine scale but only major highways at a coarse scale, so the mapping program may be run for every city street in America even though the user is viewing the whole country and only seeing the major highways. Really, scale conditions **must** be optimized for efficient execution. A symbology-encoding design that impedes this optimization does so at its own peril.

On the other hand, SE could easily be extended to include a runtime scale function in its **Filter** language if this should ever be required. However, the use of the static scales would be strongly recommended for all cases where it is sufficient (almost all) because of the ease of optimization.

SE provides a one-to-many feature-to-portrayal mapping with no feature transformations. However, an optional symbolizer parameterization mechanism has been devised for SE and is described in Clause 6.7.1. This mechanism allows formal parameters to be identified by name for a symbolizer and for argument values to be passed into the symbolizer from outside, normally through a remote symbolizer reference in a **FeatureTypeStyle**. This mechanism can be implemented in a straightforward way by essentially substituting the argument content in every place a feature-property reference is made within the symbolizer. This mechanism allows symbolizers to be reused to the same extent as the ISO schema-mapping approach. It is made optional for backward compatibility and for simplicity when symbolizers are included inline with feature type styles (where there is no opportunity to reference them remotely).

On the other hand, a general schema-remapping mechanism could be useful with SE portrayal to fuse multiple source features together to achieve a many-to-many mapping should this ever be necessary. Such a mechanism would generally useful to OGC for numerous purposes and so should be designed outside of SE and could be utilized in a way that is orthogonal to SE: a feature-processing pipeline could transform the features into the portrayal schema before the SE processing is performed. The SE design itself does not need to be complicated by this capability. Feature transformation should be approached in a “pay more, get more” fashion. With the ISO 19117:Revision design, you “pay more” even when most of the time you only want to use simple conditional styling. Also keep in mind that an individual remapping program must be supplied for each different source feature type that is to be mapped to a particular portrayal. There is no free lunch.

The ISO 19117:Revision schema-mapping concept sounds interesting, but the mechanism described is convoluted and over-complex, and the benefits over the relatively simple mechanism in SE are somewhat dubious. It would be difficult to implement, difficult to optimize, and difficult to design and transform styles for ISO 19117:Revision. The SE feature-selection approach should remain as-is.

6.5 Portrayal feature & portrayal specification

ISO 19117:Revision contains a great deal of verbosity and redundancy in specializing feature catalogues and all descendent classes, feature schema mapping and rule-statement classes, and feature instances and all descendent classes for portrayal features. A portrayal feature is a feature that is suitable for drawing. This approach may step over the line into the dark side of object-oriented design methodology — overstructuring. SE has none of this redundancy. Even if a schema-remapping facility were added directly into SE, there would be no need for specialized type of feature for portrayal. You would remap from an ordinary feature to a different ordinary feature with the schema and semantics that a style was designed for and draw that.

The specialized portrayal feature catalog has the differences of having an **SY_Presentation** be part of a feature type and has default values for feature attributes and associations. In SE, the **Symbolizer** is identified in a **Rule** of a **FeatureTypeStyle** and there is no need for default property values since the input feature is styled as-is.

The specialized portrayal-feature instance includes a display-priority value, portrayal sub-features, and portrayal-feature attribute values. It is unclear what a sub-feature is or why it is indispensable and the attributes seem to be handled in an odd way; they are a list of key/value pairs rather than objects with direct attributes of the appropriate types, which seems rather wasteful. Is this how features are modeled in the ISO general feature model? Perhaps sub-feature is another name for feature association.

The **PF_DisplayPriority** class has an odd circular definition. Its sole member is a **compare** function that takes a **PF_DisplayPriority** as an argument and returns a **Boolean**. There appears to be no means by which a style designer may assign a display priority; it seems to be inherent. The semantic of the priority value is defined as being "used to determine the order in which symbols and symbol elements are displayed. A realization of this type will allow a total ordering of portrayal features for display." The total ordering explains why there need only be two comparison-result values (less-than or greater-than), though the semantic of the return value is not defined (i.e., does **false** mean less-than?). The circular definition and the use of an internal function which is not feasible to realize in a heterogeneous web environment hint that this is really a virtual mechanism that is handled automatically and opaquely by the runtime system, perhaps in coordination with an ordered list of feature types to portray in the user interface of a viewing program.

In SE, display priority is implied by using the painter's model for processing and SLD/SE from start to finish. The lists of layers, styles, feature-type styles, and symbolizers are ordered and succeeding elements are drawn over top of preceding elements. Since ISO 19117:Revision does not have map or layer concepts, the display priority mechanism appears to be a roundabout way to determine the order of the map from some external source.

The portrayal specification package in ISO 19117:Revision appears to be redundant retelling of the portrayal-feature-instance story, which itself is a redundant retelling of the feature-instance story. It adds an optional fallback portrayal feature should the construction of a primary portrayal feature fail. It is unclear how this fallback would be populated with a sensible geometry for the map being drawn. The purpose of **PF_PortrayalSpecification** appears to be supply feature properties to the symbolization mechanism. This explains the presence of the ability to include static features inline; these can be used to manufacture graphic icons. SE just uses the input-data features as-is and SE graphic icons can be built up from “mark” graphics which can include internal geometries.

6.6 Feature & style catalogs

ISO 19117:Revision defines or references catalogs for many classes, including the feature catalog, portrayal-feature catalog, portrayal catalog, and portrayal-rule catalog. In fact, ISO does not appear to be able to function without catalogs that are populated for the data one wishes to use. What is one wishes to use a standalone Shapefile?

OGC and SLD/SE approach catalogs and repositories in a generic way. A catalog supplies generic metadata for any type of object and repositories can store any type of object. SLD and SE classes can be realized as small individual XML documents that can reference subordinate objects through hyperlinks, so repository services are not required for operation (ordinary web-accessible files can be used). Also, rather than just being abstract designs, OGC catalogs and repositories exist as deployed interoperable web services.

6.7 Symbolization

ISO 19117:2005 was practically empty in terms of defining symbols, supplying only the means to specify generic textual metadata for the graphical parameterization you would like to have exist. This approach is non-interoperable to the point of being useless, so one of the primary (and perhaps the only important) focus of ISO 19117:Revision is to flesh out the graphical parameterization of symbols.

6.7.1 Root classes

The root class for symbolization in ISO 19117:Revision is **SY_Presentation**. Its UML diagram is shown in Figure 2 [ISO 19117:Revision].

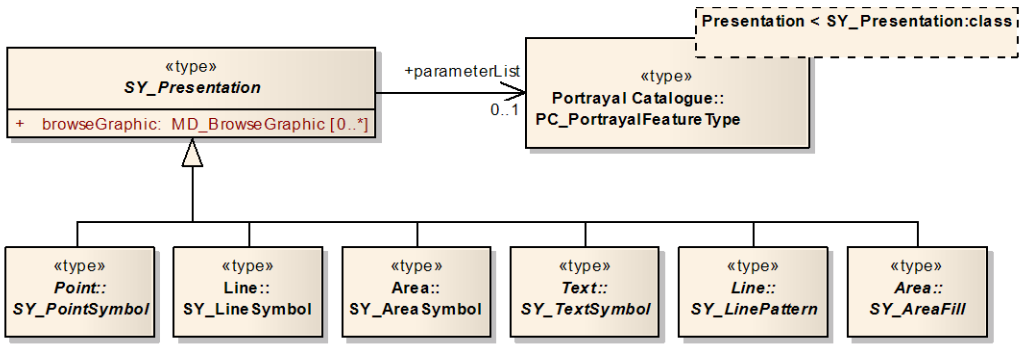


Figure 2: ISO Presentation

for browsing or generating legends.

The root symbolizer element in SE is **Symbolizer** and the OWS-6 change proposal for SE (referred to as SE throughout this section) includes numerous changes over SE 1.1.0 which were inspired by the ISO 19117 revised design and W3C SVG format. The name "symbolizer" is used in SE because it is felt that the term "symbol" is too overloaded.

Symbolizer is an abstract element which has **Version**, **Name**, **Description**, **LegendGraphic**, **ArgumentList**, and **FormalParameters** properties. **Version** is an important practical concept for SE since XML-encoded SE fragments may be strewn about the Web and users may build up their styling from numerous fragments of different versions of SE. Portrayal systems need to recognize the different versions and parse them appropriately. ISO uses the concept that each dataset has a complete and coherent symbology and schema specifications integrated into a set of catalogs, which is likely not realistic. **Description** gives multi-lingual text metadata, including titles, abstracts, etc. making use of the OWS-Common mechanism. The ISO equivalent merely uses a single character string. ISO should take a more general approach to descriptions. **LegendGraphic** has the same purpose as **browseGraphic** in ISO.

ArgumentList and **FormalParameters** provide an optional mechanism to parameterize symbolizers to enable them to be reused among incompatible feature types. **ArgumentList** is optional and gives a list of named arguments and values which may be either constants or **Filter** expressions. **FormalParameters** is optional and gives a list of formal parameters and descriptions for the symbolizer. If the **FormalParameters** element is not present, then the symbolizer uses the formal parameters of its parent or the feature properties directly if no parent takes formal parameters. If **FormalParameters** are present, then the current symbolizer or a nearest parent symbolizer must provide an **ArgumentList** that matches the formal parameters exactly, including argument order and the symbolizer shall not make use of any variable names that are not included in the formal parameters (to avoid defeating the purpose of symbolizer parameterization). The arguments are named to increase the chance of detecting any drift between the SE fragments where the arguments are generated and the symbolizers where they are consumed. The normal use case will be for the argument list to be given in a **SymbolizerReference** and the formal parameters to be declared in the referenced remote symbolizer.

Symbolizer has derived elements **PointSymbolizer**, **LineSymbolizer**, **AreaSymbolizer**, **TextSymbolizer**, and **RasterSymbolizer**, **CompositeSymbolizer**, **SymbolizerReference**. The first four correspond to the obvious ISO symbols, but ISO lacks a specific means to symbolize raster data. **CompositeSymbolizer** allows multiple sub-symbolizers to be combined into one, giving SE a one-to-many feature-to-portrayal mapping. The term "composite" is used instead of "compound" since the sub-symbolizer contents may over-plot each other. It is unclear if ISO can achieve the x-to-many mapping in this same simple way. Suppose that you wish to portray a city as a small circle plus a text label. In SE, this would be accomplished using a **CompositeSymbolizer** with **PointSymbolizer** and **TextSymbolizer** components. In

ISO, it appears that you would need to write a mapping program that builds two different portrayal features, which seems like significantly more style-design effort. Mind you, there are compounding sub-classes for the point, line, and area symbols, but none that compound different symbol types. **SymbolizerReference** allows an external symbolizer to be referenced using a hyperlink.

ISO includes **SY_LinePattern** and **SY_AreaFill** as top-level symbols to enable reuse. This is a little conceptually odd considering that they are not actually symbols like the rest of the subclasses of **SY_Presentation**. Perhaps this is why the name was changed from **SY_Symbol** in earlier ISO drafts. SE provides this capability using the **StrokeReference** subclass of **Stroke** and a **FillReference** subclass of **Fill**. One could also imagine a **LabelReference** subclass of text label. Putting the reuse mechanism at the graphic level rather than the symbol level makes more conceptual sense.

The **CompositeSymbolizer** was added to SE to realize the conceptually cleaner ISO-based design of having a **Rule** reference exactly one **Symbolizer**. SE 1.1.0 achieved the same functionality by allowing a **Rule** to directly reference multiple **Symbolizer**s, but this means that a group of symbolizers that are intended to be used together are awkward to reuse in different **Rule**s. **SymbolizerReference** was added to SE to clean up the previously less-explicit referencing mechanism. **PolygonSymbolizer** of SE 1.1.0 was changed to **AreaSymbolizer** because the style can be used with more geometry types than just polygons.

6.7.2 Coordinate reference systems

In the ISO symbol design, most classes include at least one function that returns the coordinate reference system in use for instances. For example, **SY_LinePattern** includes the public functions **lineCRS() : SC_CRS** and **localCRS(measure: Real) : SC_CRS**. The **localCRS** function takes a parameter which is the linear distance along a line. This mechanism is very abstract, and given that functions are used, it cannot be encoded into an interoperable concrete form. It is equivalent to saying that a coordinate reference system exists for every object, but the style designer cannot know what it is or redefine it. The **SC_CRS** class appears to be defined in ISO 19115 which costs US$189.00 to look at.

ISO 19117:Revision does not even directly include a unit-of-measure concept; it is implied by the CRS that the style designer cannot know or change. The user needs to know or explicitly specify what units are meant when a stroke width of 0.8 is given or a geometry translation of -4,6. This issue needs to be addressed in the ISO design. How does the user make the stroke width 0.8 mm?

SE uses a much more pragmatic coordinate-reference-system model, based on the SVG approach. All symbolizers can include a **UnitOfMeasure** element. Within a symbolizer, the view box for plotting the map is defined as a rectangle with coordinates 0,0 being the upper left corner and coordinate values advancing leftward (X) and downward (Y) in the active unit of measure. Geometries to be plotted are logically converted into the symbolizer CRS. The source-data and map geographic CRSes are irrelevant in this context. Many sub-elements can also include a **UnitOfMeasure**, which overrides the global scope within the local scope. Graphical parameters such as stroke width are in the local unit of measure. A similar CRS concept is used within graphic icons.

Defining the Y coordinate to advance downward may be awkward to deal with, but this is how SVG is defined, which is where SE borrows many of its graphical semantics. SE 1.1.0 had a simpler CRS model (pixels only) and limited geometry-transformation capabilities. The pixels-only approach poses problems when plotting at different resolutions.

6.7.3 Parameterization

Parameterization provides the means by which graphic parameters may access feature properties to give them variable values. The UML diagram for the ISO parameterization is shown in Figure 3 [ISO 19117:Revision].

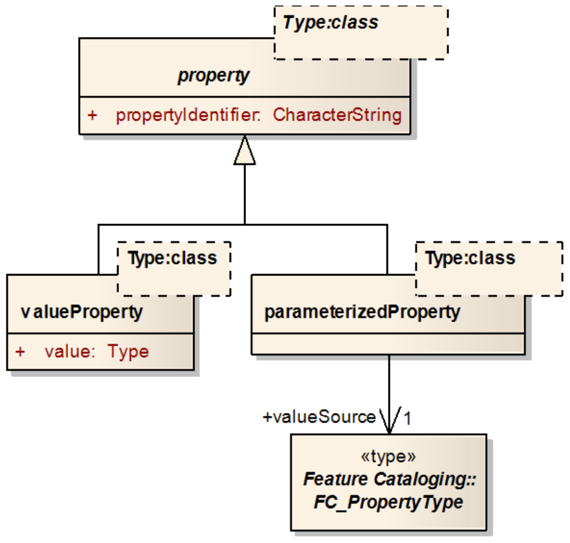


Figure 3: ISO Parameterization

SE provides the corresponding functionality with the hidden class **ParameterValueType**. It is defined to take either a literal value or use an OGC **Filter** expression. In SE 1.1.0, it could also mix filters and literal values, but this is awkward and redundant. The variable names used in the expression refer to source-feature properties unless formal parameters are in scope, in which case they refer to the formal parameters. SE parameters may include arbitrarily complex expressions where ISO parameters may only reference a portrayal-feature property. However, an ISO portrayal feature may have arbitrary properties with values computed from arbitrary expressions. The SE approach would seem to be more convenient, since the expression is placed directly where it is needed rather than being pushed back into a conceptually different part of the process. The ISO mechanism is hampered by remaining within the direct expressive capabilities of UML.

6.7.4 Line symbolization

The ISO **SY_LineSymbol** is defined by the UML diagram in Figure 4 [ISO 19117:Revision].

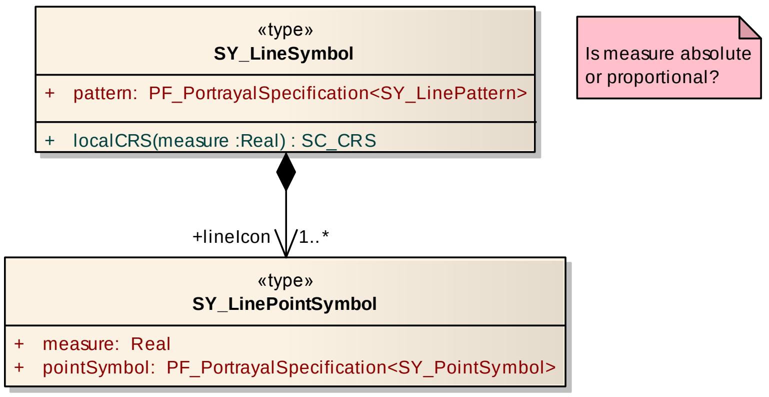

«type»
SY_LineSymbol
+ pattern: PF_PortrayalSpecification<SY_LinePattern>
+ localCRS(measure :Real) : SC_CRS
Is measure absolute or proportional?
+lineIcon
1..*
«type»
SY_LinePointSymbol
+ measure: Real
+ pointSymbol: PF_PortrayalSpecification<SY_PointSymbol>

Figure 4: ISO Line Symbol

It causes a line to be styled with an **SY_LinePattern** plus optional **SY_LinePointSymbols** which are over-plotted along the line. Since the **lineIcon**s are supplied manually, one would presume this would generally be used for arrowheads or a graphic annotation in the middle of a line, in which case, the **measure** would need to be proportional or else the capability would be unusable in practice. How long is a line?

The SE **LineSymbolizer** element extends the **Symbolizer** abstract element and adds sub-elements **Geometry**, **UnitOfMeasure**, **PerpendicularOffset**, **Transform**, and **Stroke**. Symbolizer sub-elements are placed in the order of the flow of processing. Most sub-elements are optional. Symbolization is carried out in the context of the current source-data feature.

The **Geometry** element extracts the geometry to use from the source-data feature (there can be more than one). **UnitOfMeasure** gives a well-known identifier for the unit of measure to use and is discussed further in Clause 6.7.2. The geometry is transformed into the map view-box CRS with the indicated or default unit of measure. The **PerpendicularOffset** and **Transform** elements manipulate the geometry and the **Stroke** element supplies the style to use to draw the line, as discussed below.

ISO **SY_LinePattern** is defined in Figure 5 [ISO 19117:Revision]. It has subclasses **SY_CompoundLinePattern**, **SY_PointSymbolLinePattern**, **SY_GraphicsLinePattern**, and **SY_TransformedLinePattern**.

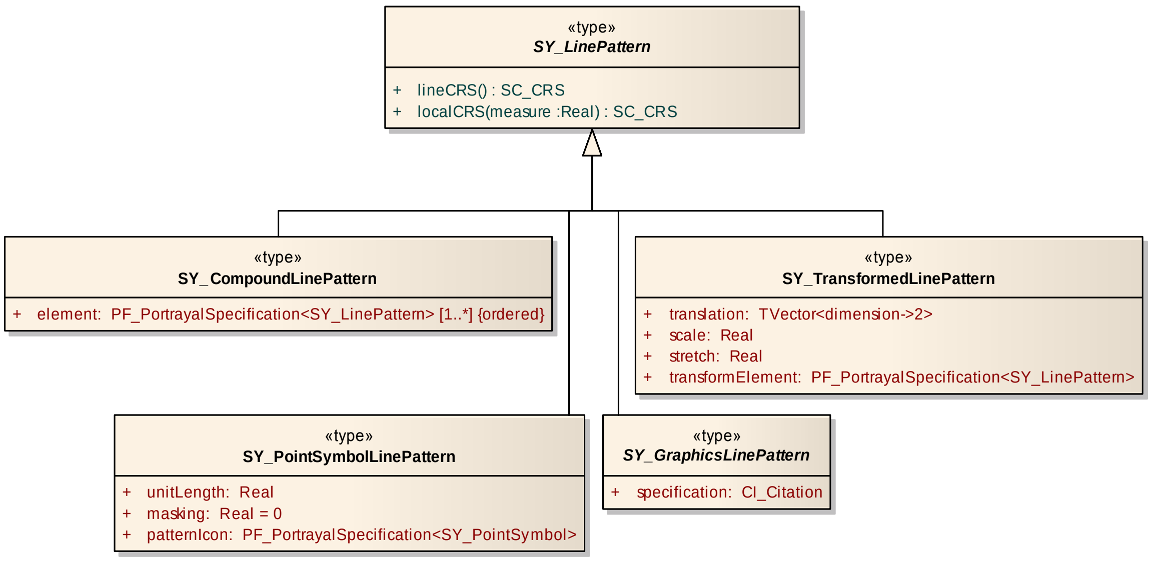

«type»
SY_LinePattern
+ lineCRS() : SC_CRS
+ localCRS(measure :Real) : SC_CRS
«type»
SY_CompoundLinePattern
+ element: PF_PortrayalSpecification<SY_LinePattern> [1..*] {ordered}
«type»
SY_TransformedLinePattern
+ translation: TVector<dimension->2>
+ scale: Real
+ stretch: Real
+ transformElement: PF_PortrayalSpecification<SY_LinePattern>
«type»
SY_PointSymbolLinePattern
+ unitLength: Real
+ masking: Real = 0
+ patternIcon: PF_PortrayalSpecification<SY_PointSymbol>
«type»
SY_GraphicsLinePattern
+ specification: CI_Citation

Figure 5: ISO Line Pattern

SE supplies all stroke styling with the abstract **Stroke** element which has derived elements **PenStroke**, **GraphicStroke**, **TextStroke**, **CompoundStroke**, and **StrokeReference**. **StrokeReference** references a remote stroke using a hyperlink.

ISO **SY_CompoundLinePattern** appears to supply an ordered list of **SY_LinePattern** objects, allowing a composite stroke pattern to be built from simpler stroke patterns. The true meaning of **PF_PortrayalSpecification<SY_LinePattern>** is a unclear. It templatizes the **presentation** properties of **PF_InlinePortrayal** and **PC_PortrayalFeatureType** and perhaps other related classes. The real meaning of this is very convoluted and poorly explained. Does each **element** instance cause a portrayal feature to be constructed? If so, then how does one make the constructed geometry have the right coordinate values?

SE **CompoundStroke** allows a stroke pattern to be built from simple stroke types. It includes sub-elements **PreGap**, list of **StrokeElement**s and/or **AlternativeStrokeElements**, **PostGap**, and an optional list of **StrokeAnnotationGraphic**s. The **PreGap** gives the distance to advance along the line before plotting anything and is in the unit of measure in context and **PostGap** gives the distance from the end of the line to stop plotting (to clear the way for arrowheads, for example). **StrokeAnnotationGraphic**s allow a stroke to include arrowheads at either end or other annotations, same as ISO **SY_LinePointSymbol**. **StrokeAnnotationGraphic** includes a **RelativePosition**, which is a decimal number between 0 and 1 where 0 means the starting point of the line, and **RelativeOrientation**, which is described below with **GraphicStroke**.

SE **StrokeElement** includes a **PreGap**, a simple sub-stroke element, a **Length**, and a **PostGap**. **PostGap** gives the distance to advance after rendering the sub-stroke. Both

PreGap and **PostGap** are allowed, though they are normally redundant, to give more flexibility in **AlternativeStrokeElements** (below). Gaps can be supplied in ISO using **SY_TransformedLinePattern**. **Length** gives the distance to plot using the sub-stroke style. SE **Length** is needed with simple strokes because they are inherently infinitely long. With a **GraphicStroke** sub-stroke, **Length** overrides the previous value. ISO does not appear to have the capability to limit compound-stroke element lengths in all cases, which is a crucial requirement.

SE **AlternativeStrokeElements** supplies a list of alternative **StrokeElements** in order of preference. Normally, the first **StrokeElement** in the list will be used for styling, but if using it would produce an undesirable appearance, the rendering system can consider the alternatives in turn, choosing the first one that can be used successfully. Normally, this capability will only be used with a **GraphicStroke StrokeElement** to supply an alternative **PenStroke StrokeElement** to use instead of the **GraphicStroke** on sharp corners when the graphic would overshoot the line segment or over-plot previously plotted graphics, as discussed further in Clause 7.2. This mechanism does not add fundamentally more implementation complexity to SE, and simple implementations can always choose the first alternative.

ISO **SY_PointSymbolLinePattern** allows a graphic icon to be repeated along the length of a line. The **unitLength** property gives the repetition length. ISO does not specify where in the repetition length the point is plotted (e.g., the start or the middle). The **masking** property gives the distance around the point symbol to erase from of the super-pattern. This erasing may be difficult to implement.

SE **GraphicStroke** repeatedly plots a graphic along a line and includes sub-elements **Graphic**, **Length**, and **RelativeOrientation**. **Graphic** specifies the graphic icon to plot. **Length** gives the linear length to reserve for the graphic, which is plotted at the midpoint. The default length is the width of the view box of the graphic. **RelativeOrientation** tells how to orient the graphic with respect to the line and is an enumerated type which allows the values **normal**, **line**, **portrayal**, and **normalUp**. The value **normal** means perpendicular to the line; **line** means in the direction of the line; **portrayal** means straight up with respect to the parent portrayal environment, which will normally be the map; and **normalUp** means perpendicular to the line but to rotate an additional 180° to avoid the graphic facing downward with respect to the portrayal environment. The ISO relative orientation is unclear, as discussed in Clause 6.7.6.1.

ISO **SY_TransformedLinePattern** allows a subordinate line pattern to be transformed with respect to the line coordinate system. It has properties **translation**, **scale**, and **stretch**. They are all unparameterized. The **translation** property specifies a translation in two dimensions, along the line and perpendicular to the line. The new line pattern may be longer than or shorter than the original pattern because of corners in the line. ISO does not specify how to handle corners, which have a discontinuity in the translated position. The **scale** property specifies scaling perpendicular to the line and the **stretch** property stretches a pattern out along the line. Use of the dimensions along and perpendicular to the line is rather clever. However, the generality of this mechanism may make it difficult to implement.

SE **PerpendicularOffset** of the **LineSymbolizer** transforms the line geometry in the same way that perpendicular translation in ISO transforms the line pattern. The SE method may be easier to implement because it is a very specific mechanism; it transforms one complete geometry and not fragments of styles. Translation along the line is supplied by gaps in **CompoundStroke**. Scaling a pattern perpendicular to the line and stretching a pattern along the line are not specifically needed since the same effect can be realized by using wider stroke styles or scaling graphic-stroke icons vertically and/or horizontally in their own coordinate spaces.

SE also includes a **Transform** element in every symbolizer. This element allows a 2D affine transformation to be specified with respect to the map coordinate space using translation, rotation, scaling, and/or a full 3×3 affine-transformation matrix. This capability could be used with a composite symbolizer to produce a shadow effect, for instance. ISO does not appear to offer this capability for line styles.

ISO **GR_Stroke** is defined in Figure 6 [ISO 19117:Revision]. It gives a stroking style for use with **SY_GraphicsLinePattern**, though the binding is considered to be loose, allowing ISO to use any number of different graphic languages.

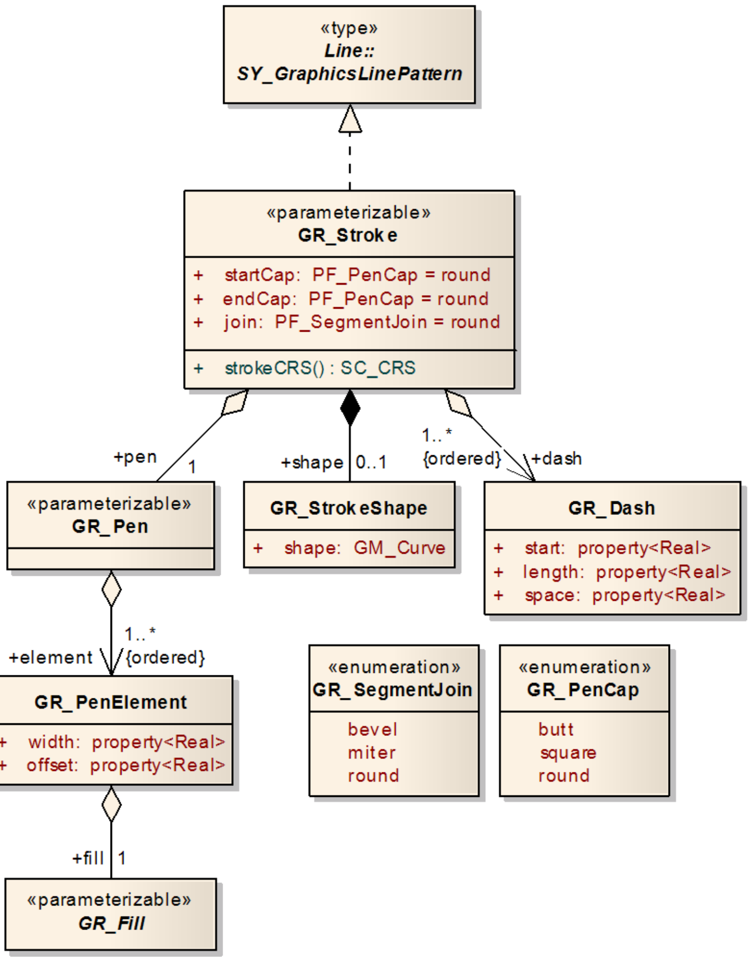


Figure 6: ISO Stroke

GR_Stroke and descendent classes supply a number of graphical parameters for strokes, including **startCap**, **endCap**, **join**, **width**, **offset**, **fill**, **shape**, and **dash start**, **length**, and **space**. The properties **startCap** and **endCap** give the pen shape with which to start and end lines. The **join** property gives the shape to use to join line segments together. The properties **startCap**, **endCap**, and **join** are not parameterized even though similar enumerated properties in **GR_TextStyle** are. The properties **width** and **offset** are used to give the drawing width of the pen and the purpose of offset is unspecified. Perhaps it is a linear or perpendicular offset, similar to the functionality provided by **SY_TransformedLinePattern**. The **fill** property gives the filling pattern for the pen plotting. The **shape** property gives an optional linear geometry to stroke. This capability seems a little out of place since stroke should be a style and not a graphic element. The repeated **start**, **length**, and **space** elements give a dash pattern.

SE **PenStroke** defines a stroke style in a very similar way to ISO. **PenStroke** includes sub-elements **Color**, **Stipple**, **Opacity**, **Width**, **LineJoin**, **LineCap**, **DashArray**, and **DashOffset**. All but **Stipple** are derived from SVG. A **Stroke** must have a **Color** or a **Stipple** but not both. **Color** gives a solid color encoded in the simple SVG/HTML

"*#RRGGBB*" form. **Stipple** refers to a graphic fill pattern. **Opacity** is self-explanatory and ISO does not provide it. **Width** gives the pen width. **LineJoin** and **LineCap** give the pen shapes at line corners and line ends. SE does not include specific caps for each end of the line and neither does SVG, so this capability may not be very important. **DashArray** gives a list of pen-down/pen-up drawing lengths and **DashOffset** gives the distance before the first pen-down. This is the similar dashing information to ISO though organized in the SVG way. The ISO dash **start** and **space** are redundant in all but the first dash pattern.

SE provides a **TextStroke** which ISO does not have. This allows a line style to include an embedded text label along the line. It includes a lone **LineLabel** sub-element which specifies the label content and style and is described in Clause 0. This capability is necessary, for example, to draw some USGS contour lines which have a solid line that is interrupted by numeric elevation values.

The SE design for OWS-6 is greatly improved over SE 1.1.0. SE 1.1.0 had very poor support for complex line styles.

6.7.5 Area symbolization

ISO area symbolization is provided by **SY_AreaSymbol** which is defined in Figure 7 [ISO 19117:Revision].

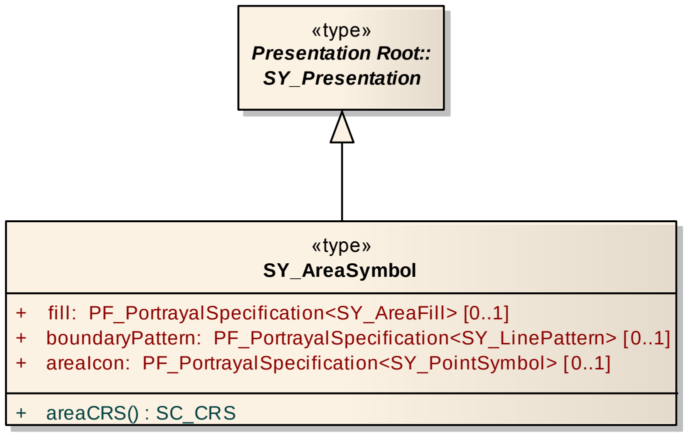

«type»
Presentation Root::
SY_Presentation
«type»
SY_AreaSymbol
+ fill: PF_PortrayalSpecification<SY_AreaFill> [0..1]
+ boundaryPattern: PF_PortrayalSpecification<SY_LinePattern> [0..1]
+ areaIcon: PF_PortrayalSpecification<SY_PointSymbol> [0..1]
+ areaCRS() : SC_CRS

Figure 7: ISO Area Symbol

The **fill** property gives an optional area-fill pattern; the **boundaryPattern** property gives an optional boundary-line stroking pattern; and the **areaIcon** property gives an optional point symbol to plot at a specific location within the area.

SE **AreaSymbolizer** extends abstract element **Symbolizer** and has sub-elements **Geometry**, **UnitOfMeasure, PerpendicularOffset**, **Transform**, **Fill**, and **Stroke**. They are all optional. **Geometry** to **Transform** and **Stroke** are described with the **LineSymbolizer** in Clause 6.7.4. The **PerpendicularOffset** transformation applies only to the boundary outline of an area geometry, not its internal area. **Fill** specifies a fill pattern. SE does not have an **areaIcon** equivalent, but an area geometry can be rendered with a **PointSymbolizer** if necessary.

ISO **SY_AreaFill** and subordinate classes are defined in Figure 8 [ISO 19117:Revision].

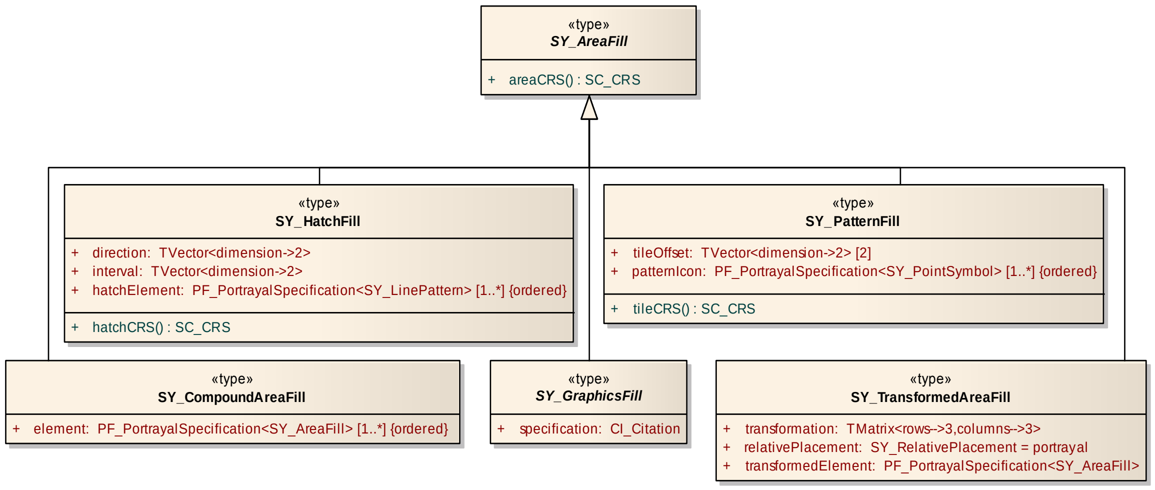

«type»
SY_AreaFill
+ areaCRS() : SC_CRS
«type»
SY_HatchFill
+ direction: TVector<dimension->2>
+ interval: TVector<dimension->2>
+ hatchElement: PF_PortrayalSpecification<SY_LinePattern> [1..*] {ordered}
+ hatchCRS() : SC_CRS
«type»
SY_PatternFill
+ tileOffset: TVector<dimension->2> [2]
+ patternIcon: PF_PortrayalSpecification<SY_PointSymbol> [1..*] {ordered}
+ tileCRS() : SC_CRS
«type»
SY_CompoundAreaFill
+ element: PF_PortrayalSpecification<SY_AreaFill> [1..*] {ordered}
«type»
SY_GraphicsFill
+ specification: CI_Citation
«type»
SY_TransformedAreaFill
+ transformation: TMatrix<rows->3,columns->3>
+ relativePlacement: SY_RelativePlacement = portrayal
+ transformedElement: PF_PortrayalSpecification<SY_AreaFill>

Figure 8: ISO Area Fill

ISO **SY_CompoundAreaFill** includes a list of sub-fill elements. The semantics of how the compounding works are not spelled out. Unlike with stroke styles, fill styles are inherently two-dimensional, so one would expect compounding to be defined in two dimensions. SE does not include this mechanism as it seems unnecessary since external graphic icons (for example, in SVG or PNG format) can be used which contain any tiling pattern desired or many can be built using the internal graphic-icon-styling mechanism.

ISO **SY_TransformedAreaFill** allows a fill pattern to be transformed with a 3×3 affine-transformation matrix with a relative placement. SE does not include this functionality since tiling graphics can include any content.

ISO **SY_HatchFill** defines a hatching pattern for fills using a line style, direction, and interval. It cannot achieve cross-hatching since only a single line can be specified. Perhaps this is what **SY_CompoundAreaFill** is for, composing fill patterns (making one pattern be plotted over top of another). In SE, graphic composition is handled inside the **Graphic** mechanism. SE does not include hatching functionality since tiling graphics can include any content.

ISO **SY_PatternFill** produces a repeated-icon fill and has properties **tileOffset** and **patternIcon**. The **tileOffset** property provides two two-dimensional vectors giving the intervals between successive icons in two directions. The **patternIcon** property gives a list of graphic icons.

SE **GraphicFill** produces a repeated-icon fill and has sub-elements **UnitOfMeasure**, **Graphic**, and **TileGap**. **Graphic** supplies a single graphic icon and **TileGap** gives X and Y gaps between successive tiles. In SE, graphic icons have an inherent "view box" (sometimes a bounding box) and this rectangle is what determines how they are tiled together. Formats like SVG and PNG have very explicit view boxes. With no **TileGap**, the view boxes are plotted to abut each other, with the pattern repeating in rows and columns. This makes it feasible to produce, say, a PNG of the desired pattern with some elements straddling the edge between two repetitions and be confident that they will be plotted with no interruption. They are true rectangular "tiles". The **TileGap** can be used of the inherent view box of the graphic icon is not satisfactory.

SE relies on the rectangular-tile semantic to produce all of the complicated fill patterns. The problem of producing these tiles is pushed into the **Graphic** element or external formats. External formats can supply the most professional-looking patterns. In ISO, graphic icons appear to be plotted in a strictly point-oriented way and it offers more control in producing fill patterns.

ISO **SY_GraphicsFill** is used in conjunction with **GR_Fill** (or some other graphic language) to produce simple fill patterns. **GR_Fill** is defined in Figure 9 [ISO 19117:Revision].

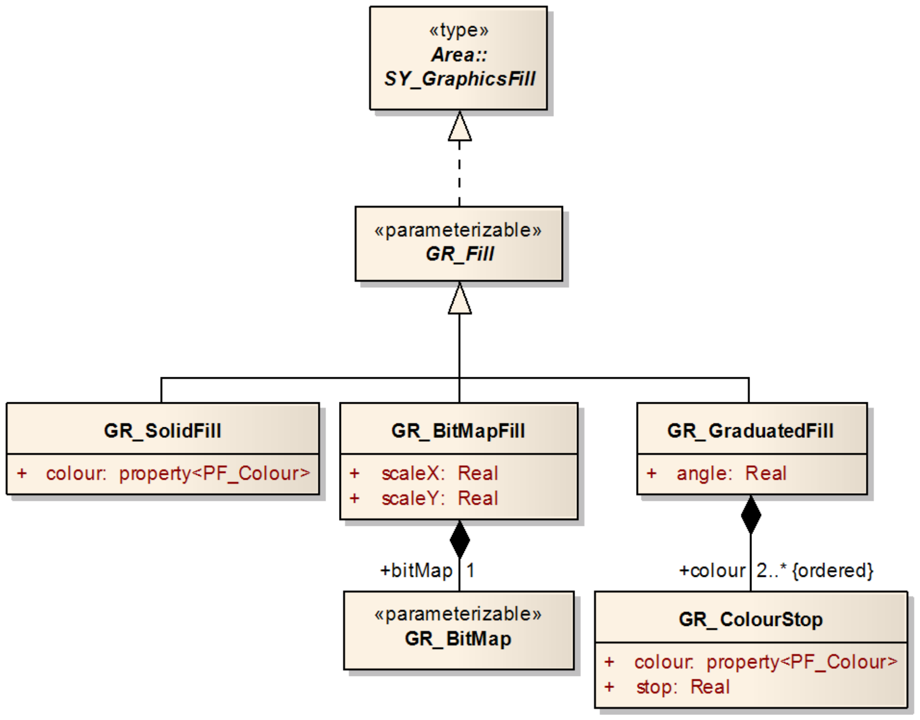


Figure 9: ISO Fill

ISO **GR_SolidFill** provides fills of a solid color. SE supplies this functionality using the **SolidFill** type of **Fill**. It has sub-elements **Color** and **Opacity**.

ISO **GR_BitMapFill** references a bitmap image (matrix of colored pixels) and allows it to be scaled in the X and Y directions. The exact meaning of the scaling factors is unclear, since neither the bitmap nor the fill have an inherent CRS. **GR_BitMapFill** is the closest ISO match to how SE **GraphicFill** works. The ISO bitmap image can include any complex pattern, though only in a raster format. SE can use vector formats like SVG in the same way just as easily.

ISO **GR_GraduatedFill** provides a graduated fill at an angle between multiple colors. In a graduated fill, the fill color smoothly changes from one value to another perpendicular to the angle. The specifics of the mechanism are not described at all. SE does not provide this capability directly. However, in cases where graduated fills are used within graphic icons and not specifically for filling feature geometries, an external format such as SVG which includes graduated fills can be used for these icons rather than relying on the internal graphic-icon-building mechanism. There probably are not many symbology standards that use graduated fills for feature geometries.

The splitting of the fill mechanism between **SY_AreaFill**, et al. and **GR_Fill**, et al. seems a bit odd, considering that they both define graphical patterns. In SE, the symbolizers operate at the feature-presentation level and the **Stroke**, **Fill**, etc. elements operate at the graphic-pattern level, and **Fill** is integrated to supply both solid fills and repeated-graphic fills. The odd split causes a problem in ISO in that that the more complex stroke and fill patterns are unavailable for use in building **GR_Graphic** icons. In SE, internally built graphic icons can make full use of all of the **Stroke** functionality. In ISO, a **GR_Graphic** stroke cannot have a repeated graphic pattern, even though implementations will have this functionality available.

6.7.6 Point symbolization

6.7.6.1 ISO point symbolization

ISO point symbolization is provided by **SY_PointSymbol** which is defined in Figure 10 [ISO 19117:Revision].

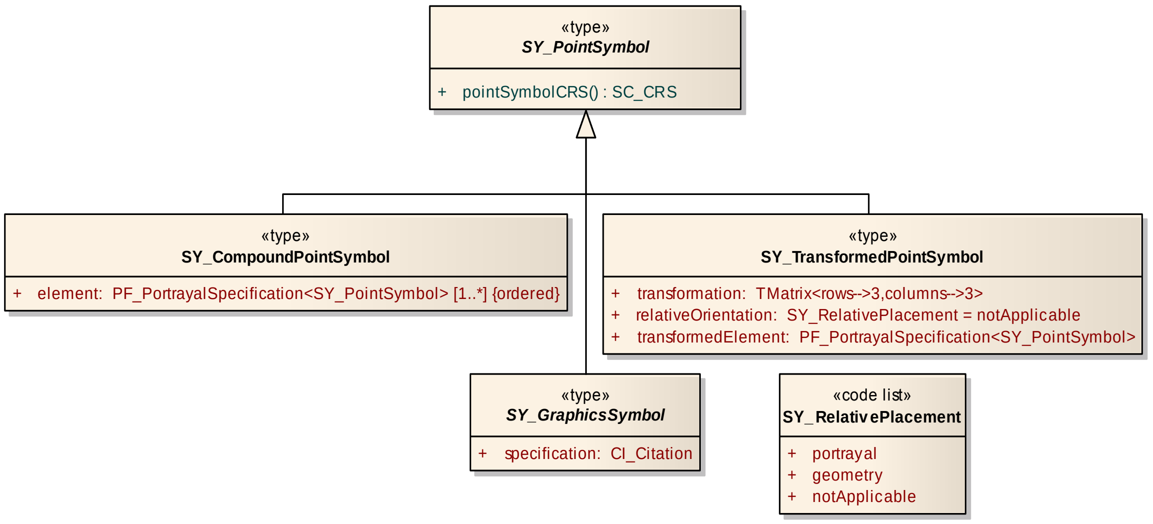

«type»
SY_PointSymbol
+ pointSymbolCRS() : SC_CRS
«type»
SY_CompoundPointSymbol
+ element: PF_PortrayalSpecification<SY_PointSymbol> [1..*] {ordered}
«type»
SY_TransformedPointSymbol
+ transformation: TMatrix<rows->3,columns->3>
+ relativeOrientation: SY_RelativePlacement = notApplicable
+ transformedElement: PF_PortrayalSpecification<SY_PointSymbol>
«type»
SY_GraphicsSymbol
+ specification: CI_Citation
«code list»
SY_RelativePlacement
+ portrayal
+ geometry
+ notApplicable

Figure 10: ISO Point Symbol

ISO **SY_CompoundPointSymbol** produces a composition of sub-symbols. The corresponding SE element is **CompositeGraphic**. The term "composite" is used instead of "compound" in SE since sub-graphics can overlap. If they do, they are rendered according to the painter's model.

ISO **SY_TransformedPointSymbol** produces an affine transformation of a point symbol using a 3×3 matrix of homogeneous coordinates. It includes a **relativeOrientation** property which can take the values **portrayal**, **geometry**, or **notApplicable**, which "specifies if the transformation is relative to the superordinate presentation, relative to the portrayal coordinate reference system, or not applicable" [ISO 19117:Revision]. It is unclear which is which between **portrayal** and **geometry**, and the exact semantics of each is not clear. Does this mechanism provide a means to show highway shields straight-up with respect to a map that are plotted along a line? If so, it seems a little odd that ISO includes the relative-orientation property within the icon-symbol definition itself, since this might limit the reuse of the icon. In SE, a relative orientation property is present only in a **GraphicStroke** to tell how to orient the contained **Graphic** with respect to the linear geometry.

In SE, transformations are available at two levels, in the **PointSymbolizer** for the feature geometry and in the **ExternalGraphic** and **MarkGraphic** elements for sub-graphic geometries, using the **Transform** sub-element which is discussed in Clause 6.7.4.

ISO **SY_GraphicSymbol** uses **GR_Graphic**, or perhaps an external graphic language, to construct graphic icons. **GR_Graphic** and related classes are defined in Figure 11 [ISO 19117:Revision].

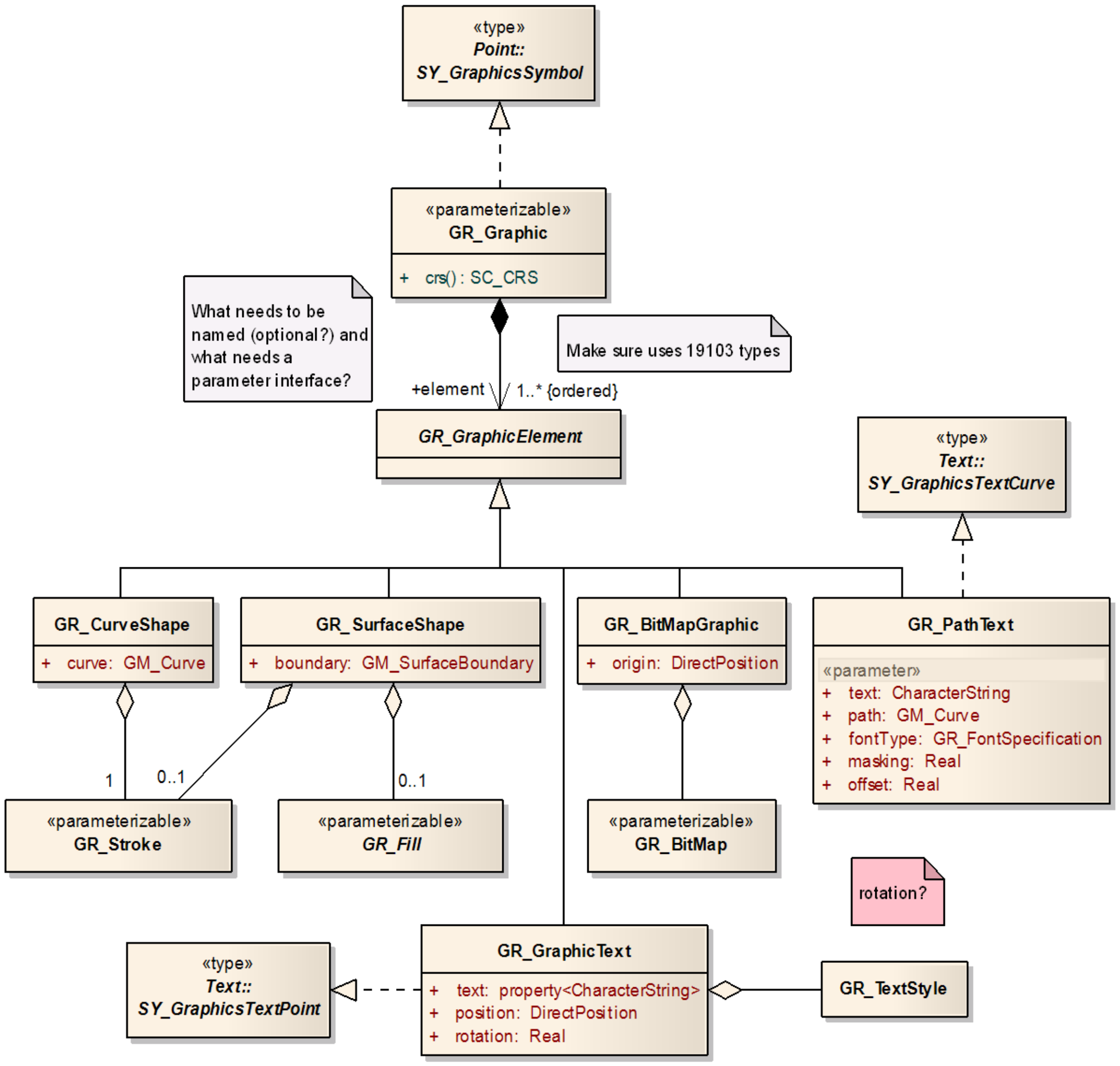


Figure 11: ISO Graphic

The **GR_Graphic** mechanism seems to be technically redundant in ISO. Since All of the symbols reference and transform sub-symbols and static features can be included in a symbology specification inline, this means that a graphic icon could be built up as being a mini-map of static features. However, passing source-data feature properties through to the mini-map would require features to be constructed from a static geometry plus the required properties of the source feature, and ISO provides this functionality. SE **Graphic** could be approached in a similar way. ISO **GR_Graphic** provides an easier-to-use mechanism than using symbols with static or constructed features, though it is less powerful since it is isolated from the transformation and complex-pattern mechanisms available in the symbols.

ISO **GR_Graphic** includes a list of **GR_GraphicElement**s, which derives various classes. **GR_CurveShape** and **GR_SurfaceShape** include a geometry of the appropriate type and reference stroke and/or fill styles as appropriate. **GR_GraphicText** and **GR_PathText** include a text label, a geometry, and font-styling information.

GR_BitMapGraphic gives an origin point and references a **GR_BitMap** image. Just giving an origin point is insufficient since the pixel size is unspecified. A bitmap cannot be rescaled to match the size of other graphic elements. A width and a height or equivalent in the graphic CRS are needed for rescaling. **GR_BitMapFill** includes scaling factors which can supply this information when used in that context.

GR_BitMap is defined in Figure 12 [ISO 19117:Revision].

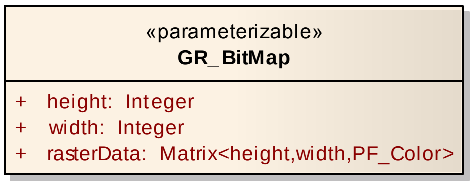


Figure 12: ISO Bit Map

The term “bitmap” is a somewhat of a misnomer since a literal bitmap stores only one bit of information for each pixel. Perhaps “pixel map” or “raster” would be a more appropriate term. Also, the matrix representation will be quite bulky since **GR_Colour** is a complex type, compression is not specified, and a structured realization in an encoding such as XML will add even more bulk. SE does not include an inherent raster concept but instead uses external formats such as PNG for this purpose. PNG is compact, compressed, and even when it is included inline in an XML graphic, it is base-64 encoded, which only adds 35% overhead (as opposed to a full XML structuring of every color component). Also, PNG is a standard format, meaning that there are many tools for manipulating and viewing such images.

GR_Colour is defined in Figure 13 [ISO 19117:Revision].

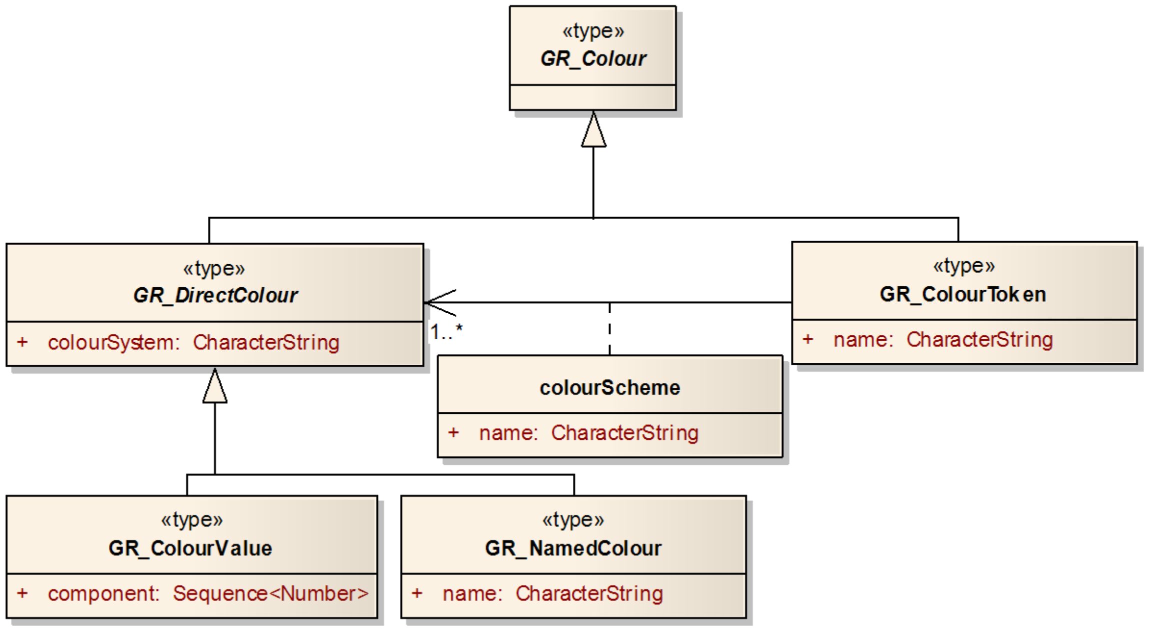


Figure 13: ISO Colour

This organization includes an enormous amount of structuring for such a basic component of styling. There will also be an issue with the British spelling of the term "color" instead of the American spelling, though some people will be irritated either way. SE uses the American spelling of the term and uses the simple and standard HTML/SVG "*#RRGGBB*" format. This format can also be computed at runtime as a string expression.

6.7.6.2 SE point symbolization

SE point symbolization is provided by **PointSymbolizer** which is derived from **Symbolizer**. It includes sub-elements **Geometry**, **UnitOfMeasure**, and **Transform** which are discussed in Clause 6.7.4 and **Graphic** which defines the graphic icon to plot.

Graphic is an abstract element with derived elements **ExternalGraphic**, **MarkGraphic**, **PointTextGraphic**, **AlternativeGraphics**, **CompositeGraphic**, and **GraphicReference**. SE could take a mini-map approach to defining graphic icons, but this would be semantically awkward since symbolizers deal with features rather than just geometries and SE does not include (and does not need) an internal feature-construction mechanism. On the other hand, the non-mini-map approach implies redundancy between the Graphic elements and the Symbolizer elements.

ExternalGraphic imports a graphic in an external encoding such as SVG for use with SE. This capability is crucial in practice, since most icons will already be available in some standard format, and converting them to an internal graphic language like with ISO would be awkward and time-consuming for the style designer any may result in an unprofessional appearance, since implementations may not support sophisticated rendering (e.g., antialiasing), whereas standard-format tools will (e.g., **librsvg** SVG rasterization library). ISO makes vague references to supporting external formats, but includes no specific mechanism for this purpose.

ExternalGraphic contains sub-elements **OnlineResource**/**InlineContent**, **Format**, **UnitOfMeasure**, **ViewBox**, **Transform**, **Opacity**, and **Halo**. **OnlineResource** references external-format content by hyperlink and **InlineContent** allows the external-format content to be included inline with the **ExternalGraphic** content encoded in XML or Base64. **Format** identifies the format of the external content using a MIME type (e.g., **image/svg+xml**). Implementations are expected to support common external formats.

ViewBox is optional and supplies a simple and convenient method to change the "view box" of the external graphic. The default view box will be based on the inherent coordinate values used in the external content. The view box is important in SE for two reasons: it determines the "anchor point" or "pivot point" that will be used to plot the graphic and it determines the physical size of the graphic when rendered, in the context of the **UnitOfMeasure**. The anchor point is the location within the graphic that will be aligned with the symbolizer control point when rendering and is the origin of the coordinate reference system of the view box (i.e., the (0,0) point). The coordinate space has the X axis pointing rightward and the Y axis pointing downward, as with the map viewport.

ViewBox contains an optional **Width** and an optional **Height**. The inherent view box of the external graphic and all of its internal coordinate values will be changed so that the view box will have the indicated width and/or height and will be centered around the origin of the CRS. If a width or height is negative, the coordinates will be flipped about the origin. This is useful with formats like TrueType fonts which are defined to have the Y axis pointing upward. If one of **Width** or **Height** is omitted, its value will be derived from the other based on the aspect ratio of the inherent view box. If both are omitted, the original view-box span will be retained, but it will be recentered around the origin.

ViewBox offers a very simple mechanism to recenter and resize external graphics, and **Transform** offers the full capacity to perform affine transformations on coordinate values. **Transform**, if present, is applied to the results of **ViewBox**, if present, or the inherent external-graphic coordinate values.

Opacity changes the opacity of the entire external graphic and **Halo** causes a halo to be rendered behind the graphic. Halos are discussed in Clause 0.

MarkGraphic is similar to **ExternalGraphic**, except that the format provides a geometry to be stroked and filled rather than a complete graphic. The geometry of a mark can be supplied as an external format using **WellKnownName**, using **OnlineResource/InlineContent** and **Format** and **MarkIndex** elements, or using a GML geometry. **WellKnownName** references a shape by a well-known name such as "**square**" or "**circle**". The external-format mechanism is similar to **ExternalGraphic** and includes an additional **MarkIndex** to dereference a specific mark within a collection of marks, such as a font file. Alternatively, the geometry can be given by an inline GML geometry object using any member of the **gml:_Geometry** abstract class, though points are not useful.

MarkGraphic includes sub-elements **ViewBox**, **UnitOfMeasure**, **Transform**, and **Halo** which were discussed with **ExternalGraphic**, plus **Fill** and **Stroke**, which fill and/or stroke the geometry according to user-supplied styles. **MarkGraphic** supplies the means for the use to build a graphic manually, using GML geometries, if the user should choose to do so.

PointTextGraphic provides a text label within a graphic icon at a point. It includes sub-elements **Position** giving the point, **UnitOfMeasure**, and **PointLabel** giving the label style. An example use case would be to paint the highway number on a highway shield. Label styles are discussed in Clause 0. These text labels should not be repositioned by the rendering system to deconflict them with other text labels, as this would damage the integral presentation of the graphic.

AlternativeGraphics provides a means for alternatives to be given for a graphic, in order of preference, in case a portrayal system does not support some external formats. SE portrayal is defined to be best-effort. A list of **Graphic**s to choose from is given. Each graphic should provide a semantically equivalent portrayal and the last one should be as simple as possible.

CompositeGraphic binds together a group of **Graphic**s to be treated as a single unit. Each member graphic needs to be compatible with the other members of the group. For example, they need to be laid out so their relative coordinate positions produce the desired appearance. The origin point of the CRS(es) is used for point positioning and the composite view box is the minimum bounding rectangle of the view boxes of the members.

GraphicReference refers to a **Graphic** through a hyperlink. This mechanism can be used to facilitate graphic-icon libraries.

6.7.7 Text symbolization

ISO **SY_TextSymbolizer** is defined in Figure 14.

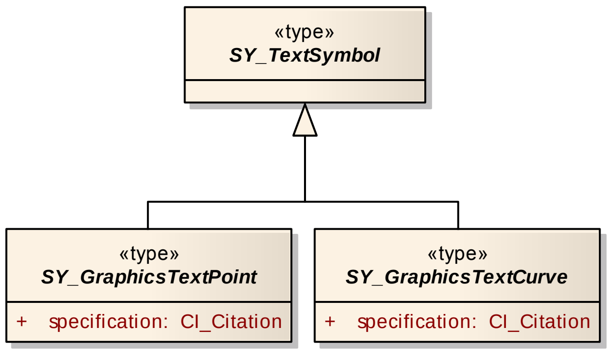


Figure 14: ISO Text Symbol

Text can be portrayed at a point or along a curve, and repetition of the text label may be useful if a very large area is given or a very long curve. The real work of text portrayal is performed using **GR_GraphicText** for points and **GR_PathText** for curves. These were both shown in Figure 11 in Clause 6.7.6.1.

SE **TextSymbolizer** provides text symbolization. It includes sub-elements **Geometry**, **UnitOfMeasure**, **PerpendicularOffset**, and **Transform**, which are discussed in Clause 6.7.4, plus **Label**. **Label** is an abstract element which has derived types **PointLabel** and **LineLabel**, which correspond to the expected ISO classes. It is odd that ISO uses the term “line” at the top symbolizer level, “curve” at the second symbolizer level, and “path” within the graphics library to refer to the same concept. SE uses “line” throughout.

ISO **GR_GraphicText** includes properties **text**, **position**, **rotation**, and a reference to a **GR_TextStyle** object. The **text** property gives the text label to be plotted and is parameterized, but the other properties are not parameterized. The **position** property gives direct coordinate values for the label location. This is suitable for plotting a label within a graphic icon, but it is unclear how to take the position from a feature geometry. The **rotation** property gives to rotation to apply to the label. **GR_TextStyle** is defined in Figure 15.

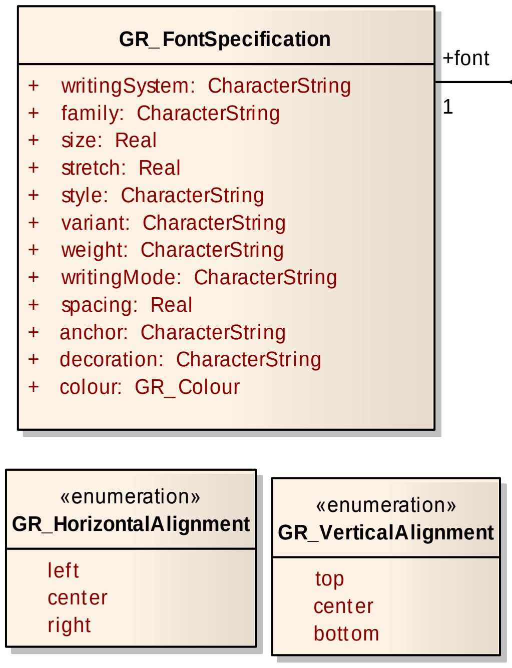

GR_FontSpecification
+ writingSystem: CharacterString
+ family: CharacterString
+ size: Real
+ stretch: Real
+ style: CharacterString
+ variant: CharacterString
+ weight: CharacterString
+ writingMode: CharacterString
+ spacing: Real
+ anchor: CharacterString
+ decoration: CharacterString
+ colour: GR_Colour
+font
1
GR_TextStyle
+ masking: property<Real> = 0
+ horizontalAlignment: property<PF_HorizontalAlignment>
+ verticalAlignment: property<PF_VerticalAlignment>
«enumeration»
GR_HorizontalAlignment
left
center
right
«enumeration»
GR_VerticalAlignment
top
center
bottom

Figure 15: ISO Text Style

ISO **GR_TextStyle** includes properties **masking**, **horizontalAlignment**, and **verticalAlignment**, and **font**. The masking property presumably supplies a radius around the text glyphs to blank out the super-pattern as it did with **SY_PointSymbolLinePattern** in Clause 6.7.4. The alignment properties indicate which point within the area occupied by the rendered point label is to be aligned with the plotting position, with three options for each alignment.

SE **PointLabel** includes sub-elements **UnitOfMeasure**, **LabelText**, **Font**, **HorizontalAlignment**, **VerticalAlignment**, **Rotation**, **ExclusionZone**, **Halo**, **Fill**, and **Stroke**. **LabelText** gives the text value to plot. The alignments give the anchor point within the label area and there is an additional vertical-alignment value over ISO of the **baseline** of the font. **Rotation** gives the label rotation in clockwise decimal degrees. **Fill** and **Stroke** have their usual meaning.

SE **ExclusionZone** provides either a circle or rectangle around the plotting point where the text label should not be plotted. This is to facilitate the combination of graphic icon with a text label in a composite symbolizer. A zone is given rather than a specific offset since it is recognized that a sophisticated portrayal system will include some kind of label-deconfliction mechanism. An explicit label alignment can be considered to be a hint of the most desired position. ISO does not include an exclusion-zone concept, which will make deconfliction more difficult to implement, as the portrayal system must automatically determine that it should be labeling a graphic icon rather than just a point in space in applicable cases.

SE **Halo** provides a fill underneath a text glyph (or graphic icon) to a specified radius around the exterior (and interior) of the shape. This can prevent text labels from becoming illegible when plotted over a cluttered background. ISO provides a masking, which erases the superpattern rather than providing a positive fill. Whether this will make text illegible sometimes depends on exactly what the term "superpattern" means. The most obvious meaning would be that it only applies to the styling of a single source feature type (or integrated group), meaning that other portrayed feature types may have content that crosses the label making it illegible. A positive-fill mechanism should be added to ISO.

ISO **GR_FontSpecification** includes many font properties, presumably derived from CSS/SVG. None of the properties is parameterizable, which is a significant limitation for **size**. SE **Font** includes only sub-elements **UnitOfMeasure**, **FontFamily**, **FontStyle**, **FontWeight**, and **FontSize**. The font-specific sub-elements are name after the CSS/SVG font parameters and have the same semantics, except that the unit of measure can only be supplied by the **UnitOfMeasure** element. CSS/SVG defines numerous font/text parameters and SE can be extended to include them using XML extensibility mechanisms. The previous **SvgParameter** mechanism of SE could supply any CSS/SVG parameters without needing XML schema changes.

ISO **GR_PathText** includes properties **text**, **path**, **fontType**, **masking**, and **offset**. These have familiar meanings, but it is not clear whether **offset** is along the line or perpendicular to the line. A perpendicular offset is needed to avoid the label over-plotting the line stroke if desired.

The organization of **GR_PathText** is inconsistent with **GR_GraphicText**, as **GR_PathText** includes its properties inline whereas **GR_GraphicText** splits the corresponding properties between itself and **GR_TextStyle** for no apparent reason. Nothing else refers to **GR_TextStyle**.

SE **LineLabel** has properties **UnitOfMeasure**, **LabelText**, **Font**, **HorizontalAlignment**, **VerticalAlignment**, **Halo**, **Fill**, and **Stroke**. These were all discussed with **PointLabel**. ISO does not include corresponding horizontal and vertical alignments. **HorizontalAlignment** indicates to which end of the line to anchor the label, which may loosely correspond to ISO **offset** if that is intended to be linear. However, a label-deconfliction mechanism may choose to plot the label at any position along the line, which would make an explicit value a hint. **VerticalAlignment** is an important parameter that ISO should include. It could be crudely simulated by ISO **offset** if that is intended to be perpendicular. SE also includes a **PerpendicularOffset** in the **TextSymbolizer** to move the label away from the line stroke in a composite symbolizer, when drawing labeled roads, for example. How would the example work in ISO? If two portrayal-feature types are needed, one for the road line and another for the road-label text, how would these show up in a map legend as a single integrated style?

6.8 Conclusions

ISO 19117:Revision and OGC SLD/SE are two different systems for encoding symbology. They are too conceptually different for a syntactic and/or semantic merge to be feasible at this time. SLD includes useful high-level Map and Layer concepts that ISO does not. ISO includes a complicated schema-remapping mechanism that appears to be experimental research and is therefore really not suitable for international standardization. SE achieves similar functionality using a simple condition-based mechanism that should be suitable in almost all practical use cases. The SE mechanism is easy to use, easy to implement, and easy to optimize. The ISO mechanism will be difficult to use, difficult to implement, and almost impossible to optimize.

The feature-symbolization mechanism in ISO is much more straightforward and sensible than the feature-remapping mechanism. SE symbolization has been extended in the OWS-6 project to provide all of the important functionality of the ISO design, while prioritizing implementation simplicity, concrete functionality in the Web environment, and compatibility with existing deployed versions of SE.

7 IHO S-52 symbology

7.1 Introduction

International Hydrography Organization (IHO) S-52 is both a symbology-encoding mechanism and a symbology library [IHO S-52]. The S-52 symbology-encoding mechanism uses terse commands encoded in character strings, for example, "**SPC;SW3;PU500,500,1000,1000;SCsample99,1;PD1000,500;**". An automatic translation between S-52 encoding and SE would be desirable, but is out of scope. The objective here is to make sure that SE has sufficient expressive power to encode the S-52 symbology library.

A report on S-52 harmonization with SE 1.1.0 was written by TENET Technology Ltd [TENET Report] and an additional change proposal was written by OSS Nokalva [OGC 09-043] relative to the TENET report and CubeWerx SE changes proposed for the OWS-6 project [OGC 09-016]. These reports discuss the issues of complex line styles, graphic-icon pivot points, and geometry-type delineations.

7.2 Complex line styles

SE 1.1.0 has a very simple mechanism for complex line patterns that is inadequate for many purposes. It includes only the capability to repeat a single graphic icon with gaps between. The intention was that the single graphic could be arbitrarily long and the rendering system could sensibly twist it around corners, but this is not realistic, considering that the graphic could be a raster and the rendering system would therefore have no means to analyze the content. The problem is probably also intractable even with a vector-graphic icon.

As described in Clause 6.7.4, SE has been extended for the OWS-6 project to include a much more capable complex-line-pattern facility. By using the **CompoundStroke** with **PenStroke** and **GraphicStroke** elements with alternatives, complex patterns can be realized.

The S-52 specification [ISO S-52 C&S] defines two different types of complex line styles, which can be summarized as follows [TENET]:

> *Single unit type*: this linestyle consists of a single repeating graphic symbol which is concatenated to form a string of symbols between two vertices of the line, using one orientation. Implemented verbatim, this type of complex linestyle can only symbolise a straight line. In order to change orientation at a vertex an additional simple linestyle is required to fill any gap between the last symbol and the vertex; a dashed style is typically used.
>
> *Composite type*: this linestyle uses a composite graphic symbol, constructed from a sequence of multiple sub-symbols and horizontal lines. This style is more suitable for rendering non-linear curves: the composite graphic symbol being repeated along the line but, being constructed from smaller symbols and horizontal lines, can change its orientation at the line's vertices. This complex linestyle can be thought of as a simple linestyle with additional symbols rendered at defined points along the line.

The S-52 Presentation Library Part I specification [ISO S-52 C&S], Clause 14.4.4 states:

> In order to fit all digitised lines (including curved lines), the complex linestyle is designed to bend around curves … If the curve is too sharp for the ECDIS to follow the digitised line exactly for part, or all, of the run-length of the line, the linestyle should default to a dashed line of the same color and lineweight as the original linestyle symbol (see 5.2.2).

Clause 5.2.2 states:

> ... If the run length of a linestyle symbol does not fit between two vertices of a line object, a simple linestyle should be used **to join the vertices**. A dashed line is preferred, but a solid line may be used.

The single-unit type line style does not seem to actually be used to define any symbols, just the composite type, so it is a guideline on how to handle awkward situations when rendering complex lines. Note in particular that the S-52 statements quoted above say "should" instead of "shall" or "must", meaning that the pen-line-style-substitution behavior described is not an absolute requirement. SE generally uses a best-effort approach to styling and leaves the fine details of symbology "finishing" to implementations, which should attempt to make their portrayals as aesthetically pleasing as feasible.

Some sample S-52 complex line styles and renderings are shown in Figure 16 [IHO S-52 C&S].

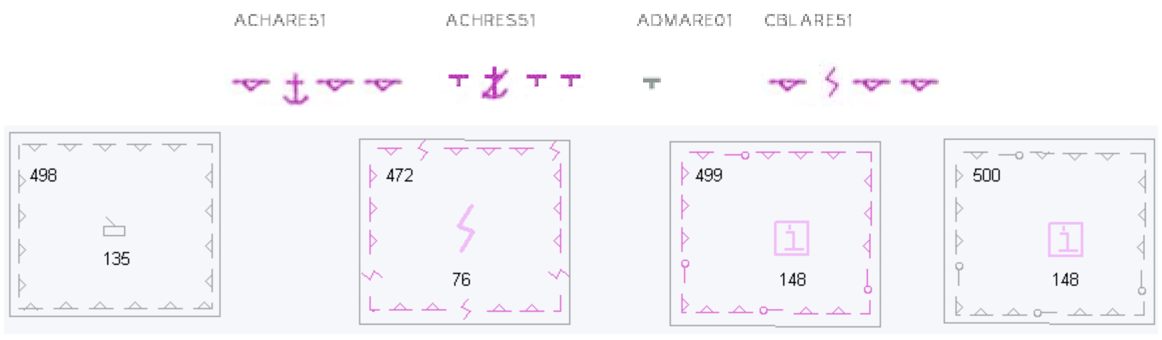


Figure 16: S-52 Sample Complex Line Styles And Portrayals

The “CBLARE51” at the top of the figure is the complex line style for the sample rendering labeled “472” at the bottom of the figure. The style is defined to have four consecutive graphic icons repeated in a row with gaps between them. The rendering process runs into a problem when plotting the bottom two corners of the sample rectangle in that the line segment makes a sharp 90° turn in the middle of the run length of the dash with the chevron. The sample rendering solves this problem by drawing a dash without a chevron, as the S-52 specification suggests. In the sample rendering labeled “498” at the lower right-hand corner, the renderer actually bends the dash component of the graphic icon around the corner. These approaches require either information in addition to the graphic icon or the ability to analyze the composition of the graphic icon, the latter of which is not feasible in the general SE environment. Other possible approaches would be to render the graphic icon beyond the corner of the line or to render it at some intermediate point and angle in the vicinity of the line span to reduce the discontinuity in the pattern. On the other hand, falling back to a pen stroke will portray the exact path of the line.

The OSS Nokalva proposal includes new elements within **StrokeElement** to address the line-bending problem called **DrawOnlyWhereLineIsStraight**, **DrawOnlyWhereLineIsNotStraight**, and **ReturnAfterDrawing**. They are defined as follows [OGC 09-043]:

> A **DrawOnlyWhereLineIsStraight** element with a value of true indicates that a copy of the stroke element is to be drawn only when it would represent a straight line segment. The default value is **false**.
>
> A **DrawOnlyWhereLineIsNotStraight** element with a value of true indicates that a copy of the stroke element is to be drawn only when it would represent either a line segment that is not straight or multiple consecutive line segments (e.g., at or near a vertex). The default value is **false**.
>
> A **ReturnAfterDrawing** element with a value of true indicates that the drawing position is to be reset to the current position once the stroke element (which may be any type of stroke) has been completely drawn. The default value is **false**.

The straight/not-straight elements are too specific. There will often be cases where there is only a small change in angle between two line segments and the graphic icons can be plotted more aesthetically than the (simple) alternative. There will also often be other cases where the the line segments of a geometry are very short relative to the graphic icons in a line style, which would cause (simple) not-straight stroke element to almost always be selected.

The CubeWerx SE proposal adds an more general element called **AlternativeStrokeElements** (based on the S-52 and OSS Nokalva designs) which includes a list of alternatives in order of preference for plotting the next stroke element. The renderer chooses among them using its own internal rules of aesthetics. For S-52 styles, the preferred stroke would be the **GraphicStroke** and the secondary stroke would be a simpler **PenStroke** that matches the color and dashing of the graphic icon. The fallback stroke may be ignored in simple implementations. This mechanism is sufficient to honor the S-52 semantics.

Over-plotting (compositing) stroke elements can also be utilized to a degree in the CubeWerx design. It can be an effective technique with the S-52 line styles "DWRTCL05" and "DWRTCL06" shown in Figure 17 [ISO S-52 C&S].

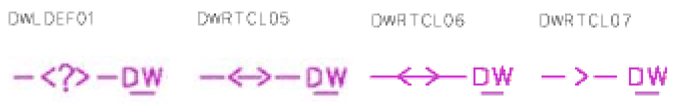


Figure 17: S-52 Line Styles With Line-Crossing Components

With these two line styles, the sideways-chevron graphic icon crosses the line only touching it at one point. A pen stroke can be used to draw the dashed or continuous lines in the patterns and the chevron can be over-plotted at a normal to the line at the intersection point using a **GraphicStroke** with a **Length** of **0**. This allows the otherwise complex pattern to bend around corners. The "DW" in the patterns would remain one awkwardly long graphic icon, however, since the underscore binds the two letters together.

Some S-52 line styles that are even more complex are shown in Figure 18 [ISO S-52 C&S].

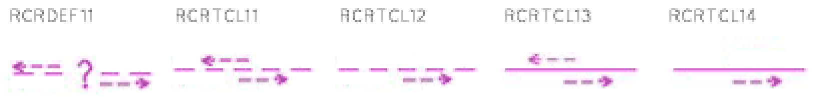


Figure 18: S-52 Line Styles With Perpendicularly Offset Components

These styles include components at a perpendicular offset to the main stroke pattern. The offset patterns can be realized with pen strokes plus a sideways chevron. OSS Nokalva argues for defining complex nested stroke patterns with internal perpendicular offsets and restarts to support these styles. Using internal perpendicular offsets will not work properly because the line lengths will be different on the insides and outsides of corners; the patterns will not stay aligned. A different approach could be used to realize all of the styles in Figure 18 except for the “RCRDEF11” style; the line could be styled as three line symbolizers inside of a composite symbolizer, one for the main line, one for the outer dashed-arrow line and one for the inner dashed-arrow line. The relative patterns would fall out of alignment, however, though this may not be a significant concern. With the “RCRDEF11” symbol, the question mark makes the pattern rigid and unsuitable for a compound symbolizer. The two sides would remain awkwardly long graphics that are substituted with a dashed pen stroke around sharp corners.

7.3 Pivot points

The TENET and OSS Nokalva reports both raise the issue of pivot points relative to SE 1.1.0 graphic icons. The SE 1.1.0 mechanism is ill-defined about allowing an anchor point to be outside of the bounding box of a graphic and lacks a mechanism to rotate a graphic about an explicit pivot point. The anchor point controls how the graphic icon is aligned with the plotting-destination point. The TENET report suggests clarifying the SE 1.1.0 design and the OSS Nokalva report suggests extending the poor SE 1.1.0 design.

The CubeWerx SE design changes SE pivot points completely by following the SVG graphic-transformation paradigm. Graphics exist in a coordinate space and the coordinate values of the graphic components can be manipulated using a 3×3 affine-transformation matrix (including simplified sub-operations like rotate and translate) using the new **Transform** element. The style designer can control the pivot point for rotation and the anchor/pivot point for aligning the graphic with the control point for plotting is defined to be the (0,0) point in the graphic coordinate space, which is allowed to be far outside the bounding box of the graphic components. ISO 19117:Revision does not have any explicit anchor/pivot points, so it probably works in the same way.

7.4 Geometry delineation

S-52 and other symbology standards include the concept of *delineation* and are used to style feature data that may have geometries of varying types (e.g., point, line, polygon) in different features within a single feature type. Delineation makes styles specific to features of a feature type with geometries of a specific dimensional classification.

The TENET report suggests five possible alternatives for handling delineations: add a specific delineation element to the **FeatureTypeStyle;** change the meaning of a feature type to be delineation specific, i.e., break a single source feature type into more than one logical feature type and process each independently; overload the **SemanticTypeIdentifier** element to select delineations; add a **Filter** function to select features of a specific type; or extend the **Filter PropertyName** mechanism to return a geometry of a specific type or a Null.

The OGC architecture does not treat delineation as an explicit concept and is in fact incompatible with the notion since an OGC feature can contain more than one geometry property, so a single feature may have multiple delineations, one for each geometry property. The only suggested approach that is compatible with the OGC architecture is to add a **Filter** function to check the delineation type of a feature.

The CubeWerx SE proposal includes a new **Filter** function called **Dimension** after the operations specified in ISO 19125-1 ([2] ISO 19125-1:2003 (E) (November 2003)). This function returns the inherent dimensionality of the geometry feature property gives as its argument. In other words, a point or multipoint geometry returns a 0, a curve returns a 1, a surface returns a 2, and a solid returns a 3. This function can be combined with a comparison to a numeric literal in the **Filter**s of the **Rule**s of a **FeatureTypeStyle** to achieve the style delineation. The **Dimension** function is generally useful and should be added directly to the **Filter** specification instead, like the string-manipulation functions defined in SE 1.1.0. The TENET reports suggests an **isKindOf** function, but **Dimension** is simpler in that it does not require enumerated geometry-type names to be available, only simple integers.

The OSS Nokalva goes off on a tangent about the need to handle features with a different geometry property for each different kind of geometry. This is unlikely to be an important consideration with any existing data; however, SE and Filter can handle this case by examining the different geometry properties and using the **PropertyIsNull** operation. OSS Nokalva also seems to indicate that different features may have different properties present, which breaks the concept of "feature type". But even so, a **Filter** function to detect a property being present or not (or overloading **PropertyIsNull**) still handles the situation.

7.5 ISO 19117 alignment

The TENET report discusses alignment issues between SE 1.1.0 and an intermediate revision of ISO 19117 between ISO 19117:2005 and ISO 19117:Revision that they discovered as part of their S-52 study.

TENET states "ISO 19117 models portrayal mapping conditions such as scale, lighting and display medium using the interface **PF_Context** attached to a higher level interface **PF_PortrayalMapping**; SE supports scale conditions at the level of the **FeatureTypeStyle** element."

The **PF_Context** class has been removed from ISO 19117:Revision. It included a static description of suitable viewing-environment conditions for using a **PF_PortrayalMapping** (which is roughly equivalent to an SE **Layer**). An important thing to keep in mind is that PF_Context gave passive information and was not executed like **Rule**s are. This means that **PF_Context** is roughly equivalent to the **Description** of an SLD **Style**. In principle, **Description** could include all kinds of metadata describing a style, including the information that was offered through **PF_Context**, though in practice this is mostly limited to providing a title.

TENET also states "A **PF_PortrayalRule** instance is associated with a single **SR_Symbol** — a composition of other subordinate **SR_Symbol**s and leaf **SR_SymbolElement**s; whereas an SE **Rule** element may consist of multiple **Symbolizer**s but these do not support the hierarchical structure of **SR_Symbol**."

The revised SE design has been changed to allow hierarchical **Symbolizer**s and have a **Rule** reference exactly one **Symbolizer**. This design is logically cleaner and offers easier reuse of styling.

7.6 S-52 symbology examples

Here are some additional S-52 symbology example portrayals to consider. Some buoy symbols are shown in Figure 19 [IHO S-52 C&S].

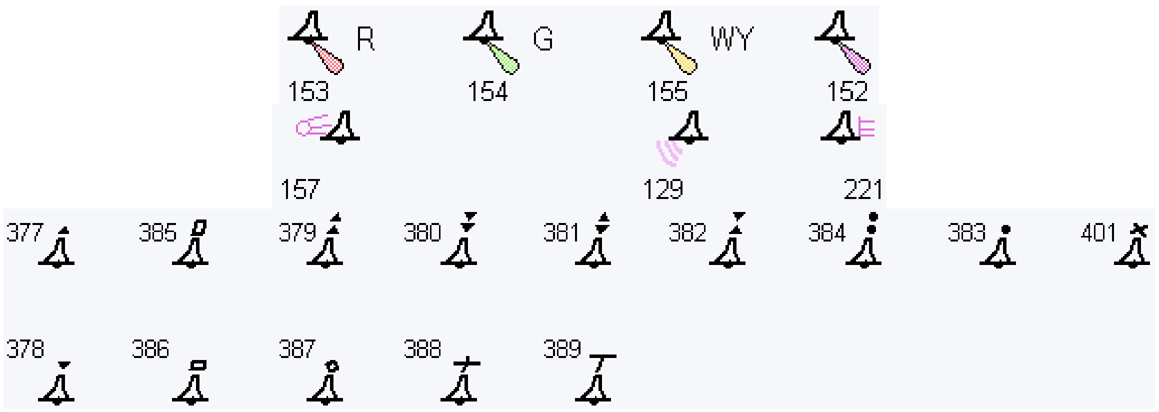


Figure 19: S-52 Buoy Examples

Buoys appear to have multiple definitions in S-52. The base bell-shaped symbol is defined as one icon, each light and other symbol are defined separately as one icon, and they are defined together as the icons on the top row of the figure. Other add-ons are shown in the second row. The third and fourth rows of buoys show different top marks, which are defined individually and in composite with the buoy. Perhaps there are more combinations of the elements than are shown above.

There are two ways that these symbols could be approached in SE, either as a single precomposed graphic icon or as different graphic icons that are plotted over top of each other. The former approach is easier and is presumably why the precomposed icons have specific symbol numbers. With the latter approach, the top marks and light graphics will need to be selected by different **Rule**s in the **FeatureTypeStyle**, since only **Rule**s can include the necessary conditions to select the different pieces. It is unclear whether a buoy is represented by a single point feature or by multiple point features, but a single feature with property values distinguishing the exact type of buoy makes the most sense.

Both composition approaches cause legend-generation problems. The most straightforward way to generate SE legends is to generate a different legend entry for each **Rule** that is present. In the precomposed case, this would equate to hundreds of legend entries just for buoys. In the post-composed case, there would be fewer legend entries, but each would be for only a component of a buoy portrayal. If the legend is generated in coordination with a map, only the **Rule**s that evaluated to **true** need to be included in the legend. Perhaps SE should provide an optional legend graphic for an entire **FeatureTypeStyle** so the style designer can supply a suitably abstracted legend graphic instead of allowing the rendering system to generate an overly bulky one.

Some composite line styles are shown in Figure 20 [IHO S-52 C&S].

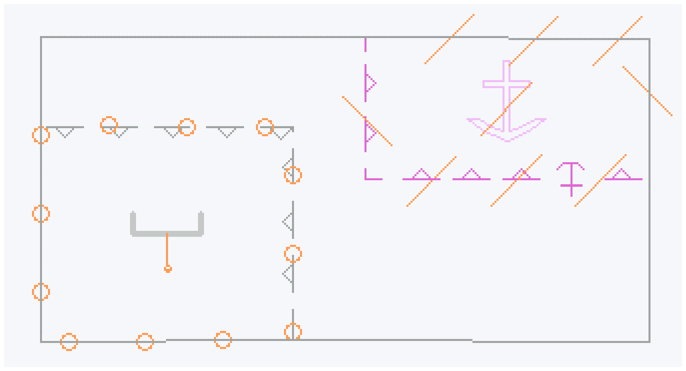


Figure 20: S-52 Composite-Line Examples

Composite line styles of this kind are easy to realize using a **CompoundSymbolizer**. Even the slash marks around the anchor area are just symbols. The central anchor symbol itself can be realized by using a **PointSymbol** with the anchor graphic icon within the **CompoundSymbolizer**, since a **PointSymbolizer** can be used with any feature-geometry type and a suitable point location(s) (perhaps the centroid if that is within the area) will be determined at runtime.

Some navigation symbols are show in Figure 21 [IHO S-52 C&S].

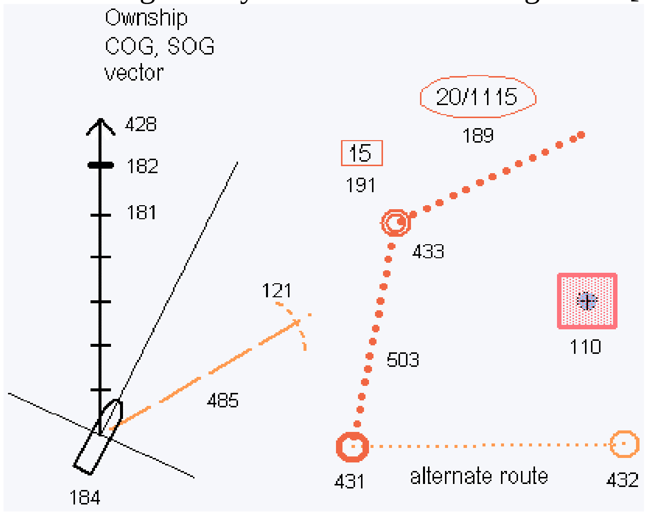


Figure 21: S-52 Navigation-Symbol Examples

Assuming that the boat information on the left of the figure is represented by a point feature, all of the information displayed can be built into a single **Graphic**, as they can be arbitrarily complex (though any conditional portrayal differences would need different **Rule**s in the **FeatureTypeStyle**). The routing information on the right of the figure can be represented by simple point and line symbolizers.

Some lines, areas, and icons are shown in Figure 22 [IHO S-52 C&S].

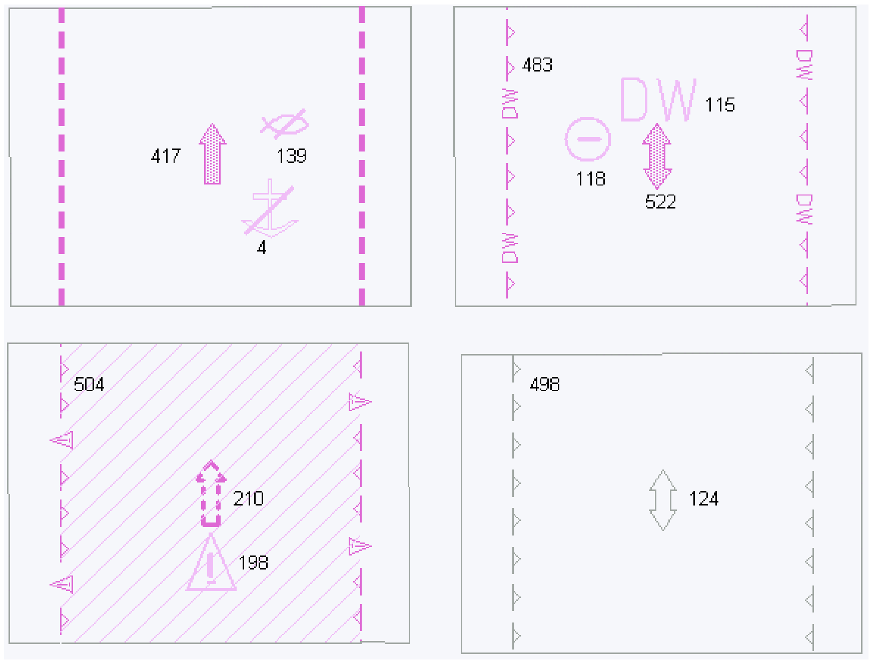


Figure 22: S-52 Traffic-Route Examples

The lines are realized using complex line styles discussed in Clause 7.2 and the central graphic icon can be realized using a **PointSymbolizer** within a **CompositeSymbolizer**. However, it is unclear what the source is of the second and/or third icons in three of the examples. If they are from attributes of the same route feature, they could be put into a **CompositeGraphic** (with appropriate offsets to avoid overlapping) and plotted with the **PointSymbolizer**. The "anchor point" of the composite graphic would be positioned in the center of the direction icon to maintain the appearance in the examples.

Oddly, in the hatch-filled example, the fill pattern is actually plotted over top of the strokes and the central graphic icons. Assuming that this is intentional and assuming that the route is represented by a polygon feature, the polygon will need to be stroked and filled in separate symbolizers within the composite symbolizer, since the fill of a single area symbolizer is plotted underneath the stroking. The fill-only area symbolizer will need to be the last one in the composite symbolizer.

An abstracted depth-contour example is shown in Figure 23 [IHO S-52 C&S].

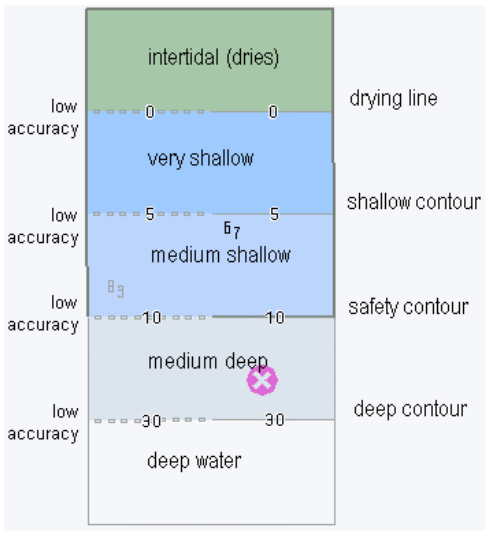


Figure 23: S-52 Depth-Contour Example

The solid contour lines can be drawn using a simple **PenStroke** and the boxes can be drawn using little rectangles in a **GraphicStroke** within a **CompoundStroke** to achieve the spacing. The fills can be drawn using a simple **SolidFill**, assuming that an area geometry is used, though sometimes the contour-line and fill-area data is available as two separate feature types, in which case two **FeatureTypeStyles** would be needed. Splitting this into two feature types allows different segments along the same fill area to have the different stroke styles (low-accuracy and normal accuracy). The depth values can be drawn using a **TextSymbolizer** with a white **Halo** within a **CompositeSymbolizer** for the contour-line feature type. The obstruction icon would presumably be in a separate feature type and would be drawn using a **PointSymbolizer** with the appropriate **Graphic**.

A composite shore example is shown in Figure 24 [IHO S-52 C&S].

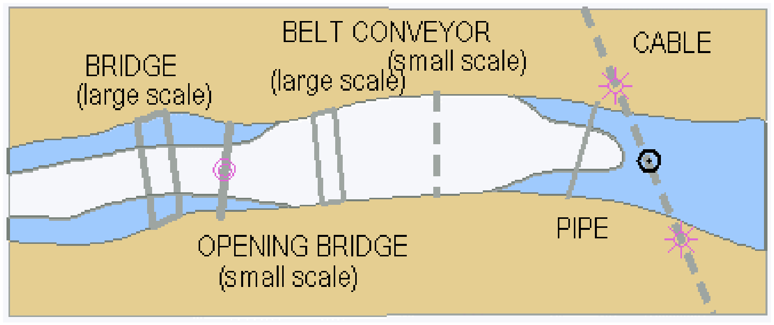


Figure 24: S-52 Composite Shore Example

The pen strokes and solid fills are easy to achieve. The cable towers are presumably supplied by an independent point-feature type, and are thus easy to draw. The purple icon on top of the bridge indicates that it is an opening bridge. A **StrokeAnnotationGraphic** with a relative position of **0.5** (the middle of the bridge) should be used to draw this. A **PointSymbolizer** could also be used with the line geometry, though the rendering system decides where to plot the icon. It is unclear what the target-shaped black icon in the middle of the cable represents. Perhaps it is a overhead obstruction from the lowest point of the cable. If it is supplied from a point feature that is independent from the cable, then it is easy to plot; otherwise, it will be very difficult to determine the proper plotting location.

8 USGS symbology

USGS provides Publication Symbols and Topographic Map Symbols [USGS].

8.1 Simple styles

The simpler USGS styles have been implemented using SLD 1.0.0 for National Hydrographic Data (NHD) ([4] USGS) available from:

http://frameworkwfs.usgs.gov/

A description of the development effort is available from:

http://frameworkwfs.usgs.gov/framework/nhd/nhd_styles.html

SLD encodings are available from:

http://frameworkwfs.usgs.gov/framework/sld/sld_content.html

A WMS serving this data as maps is available at:

http://frameworkwfs.usgs.gov/framework/wms/wms.cgi.

SLD 1.0.0 has the capacity to render pen strokes, arbitrarily complex (but unparameterized) fill patterns, and arbitrarily complex (but unparameterized) point graphic icons. Samples of these SLD styles are included in Figures 25, 26, and 27.

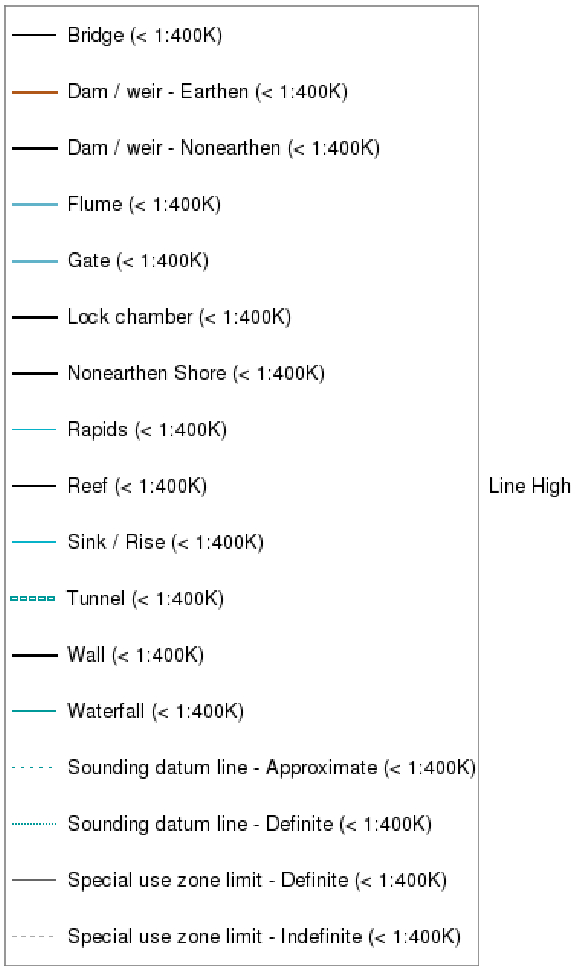

Bridge (< 1:400K)
Dam / weir - Earthen (< 1:400K)
Dam / weir - Nonearthen (< 1:400K)
Flume (< 1:400K)
Gate (< 1:400K)
Lock chamber (< 1:400K)
Nonearthen Shore (< 1:400K)
Rapids (< 1:400K)
Reef (< 1:400K)
Sink / Rise (< 1:400K)
Tunnel (< 1:400K)
Wall (< 1:400K)
Waterfall (< 1:400K)
Sounding datum line - Approximate (< 1:400K)
Sounding datum line - Definite (< 1:400K)
Special use zone limit - Definite (< 1:400K)
Special use zone limit - Indefinite (< 1:400K)
Line High

Figure 25: USGS Simple Line Styles

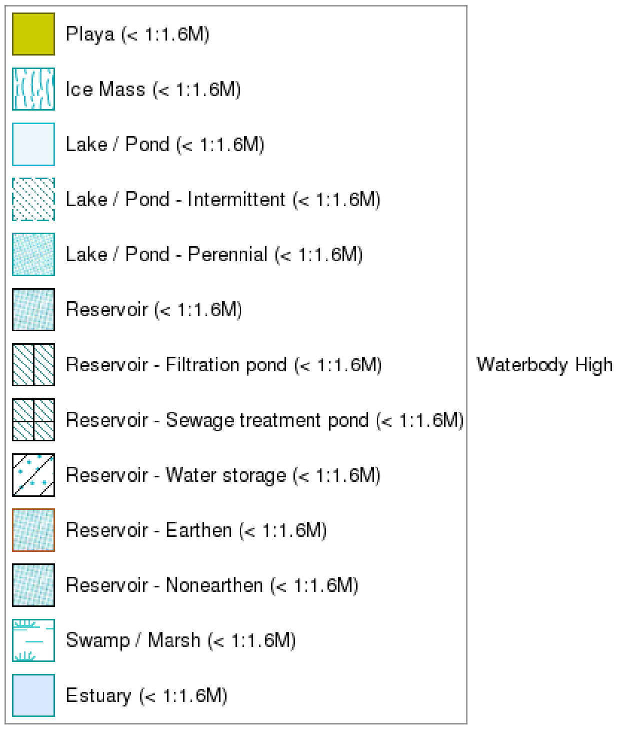

Playa (< 1:1.6M)
Ice Mass (< 1:1.6M)
Lake / Pond (< 1:1.6M)
Lake / Pond - Intermittent (< 1:1.6M)
Lake / Pond - Perennial (< 1:1.6M)
Reservoir (< 1:1.6M)
Reservoir - Filtration pond (< 1:1.6M)
Reservoir - Sewage treatment pond (< 1:1.6M)
Reservoir - Water storage (< 1:1.6M)
Reservoir - Earthen (< 1:1.6M)
Reservoir - Nonearthen (< 1:1.6M)
Swamp / Marsh (< 1:1.6M)
Estuary (< 1:1.6M)
Waterbody High

Figure 26: USGS Simple Area Styles

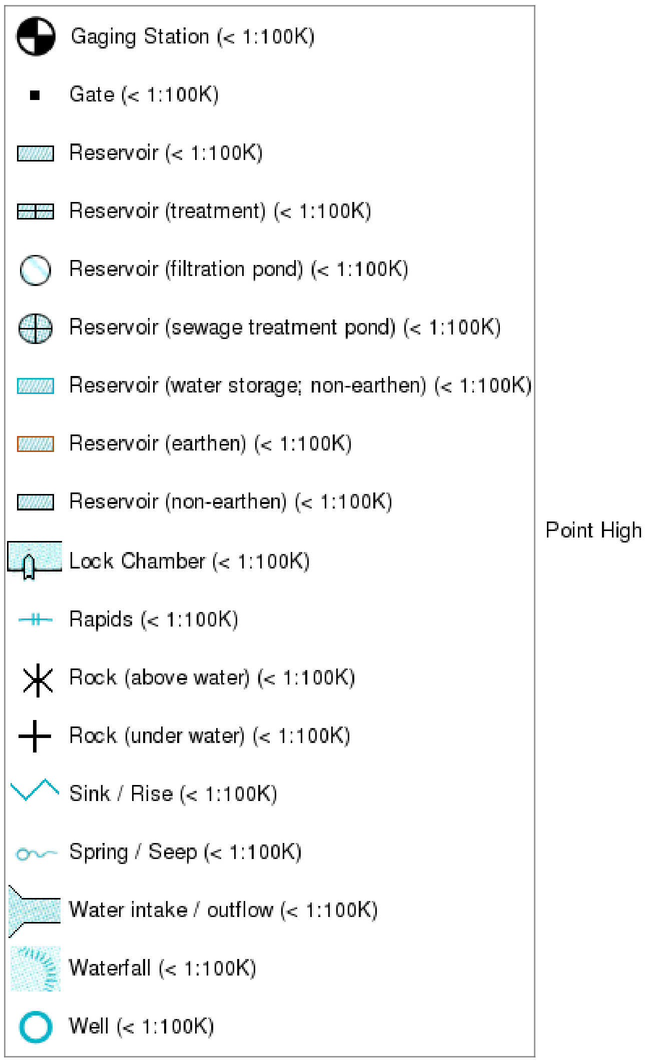

Gaging Station (< 1:100K)
Gate (< 1:100K)
Reservoir (< 1:100K)
Reservoir (treatment) (< 1:100K)
Reservoir (filtration pond) (< 1:100K)
Reservoir (sewage treatment pond) (< 1:100K)
Reservoir (water storage; non-earthen) (< 1:100K)
Reservoir (earthen) (< 1:100K)
Reservoir (non-earthen) (< 1:100K)
Lock Chamber (< 1:100K)
Rapids (< 1:100K)
Rock (above water) (< 1:100K)
Rock (under water) (< 1:100K)
Sink / Rise (< 1:100K)
Spring / Seep (< 1:100K)
Water intake / outflow (< 1:100K)
Waterfall (< 1:100K)
Well (< 1:100K)
Point High

Figure 27: USGS Simple Point Styles

8.2 Complex styles

USGS defines many complex styles and defines finishing rules for professional portrayals. Unfortunately, applying professional portrayal finishing is an artificial-intelligence problem. Only a certain degree of complexity is feasible to be expressed in SE and the rest will need to be programmed into portrayal systems.

Some USGS complex-symbol examples follow. Some coastal features are shown in Figure 28.

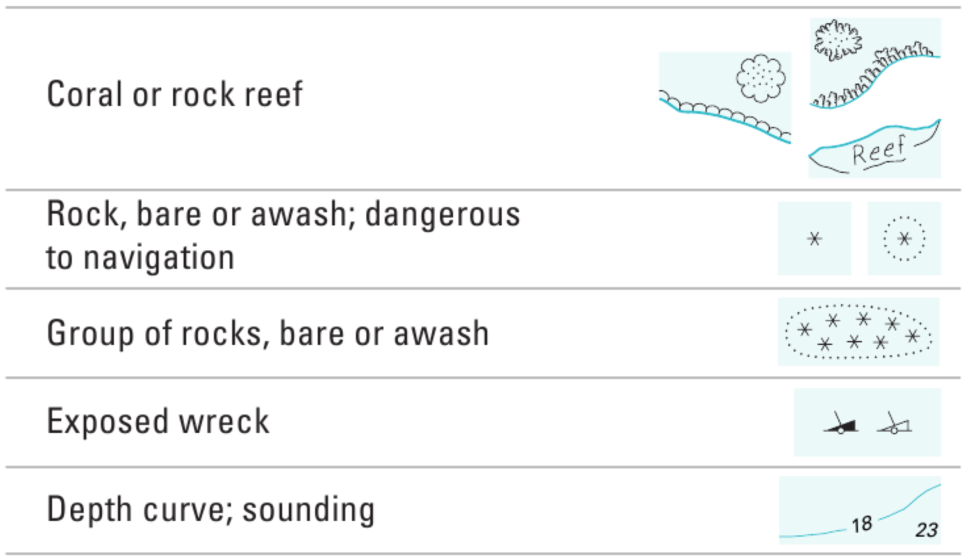

Coral or rock reef
Reef
Rock, bare or awash; dangerous to navigation
Group of rocks, bare or awash
Exposed wreck
Depth curve; sounding
18
23

Figure 28: USGS Coastal-Feature Examples

The scalloped edges in the corals and reefs can be supplied by **GraphicStroke**s within a **CompoundStroke** to allow the pattern to bend around curves smoothly. The coral-scallop pattern only needs a single **GraphicStroke** since only one shape is present. The scalloped-reef pattern appears to need 17 different scallops with explicitly given short **GraphicStroke Length**s that cause the taller graphics to overlap the shorter ones.

If the group of rocks is represented using an area geometry, it is easy to style using a dashed or graphic stroke and a fill pattern of asterisks. However, this would cause some of the asterisks to be cut off. If the rock locations are precise, then individual point features or a single multi-point feature would be needed for them. With individual point features, the outline would need to be supplied in the source data by a polygon or line feature type, since it would be very difficult to compute. With a multi-point geometry, functions could be supplied to compute a buffer zone around a convex hull of the geometry, though these functions are not defined in SE. The OGC **Filter** specification should define all of these common geometry and other functions.

The label within the depth curve can be rendered using a **TextStroke** within a **CompoundStroke** with gaps specified for both ends of the text stroke. This will cause the depth value to be printed repeatedly. A **RelativeOrientation** is not supplied inside of a **TextStroke**, but the only sensible choice is **normalUp**. If only one depth value is desired, the stroke could be defined to be infinitely long after the label is plotted.

Some contour-line examples are shown in Figure 29.

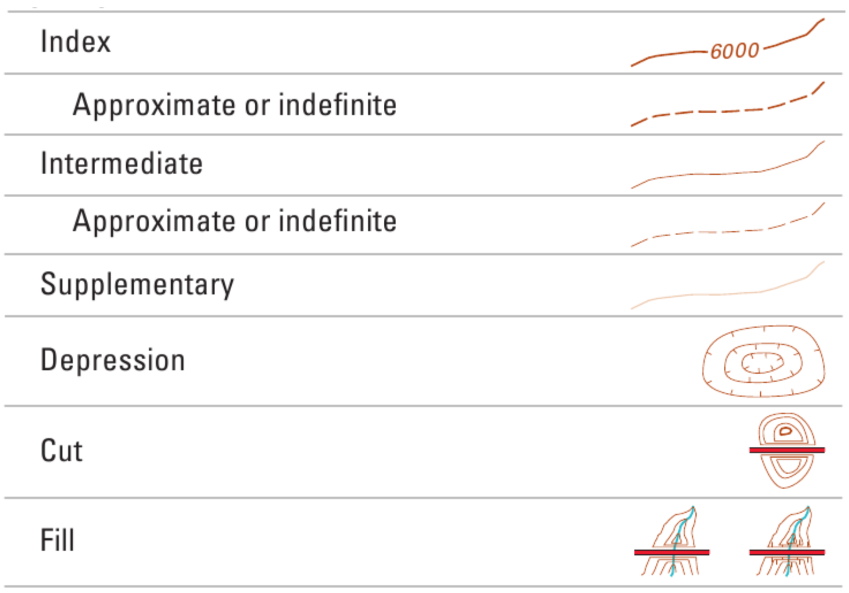


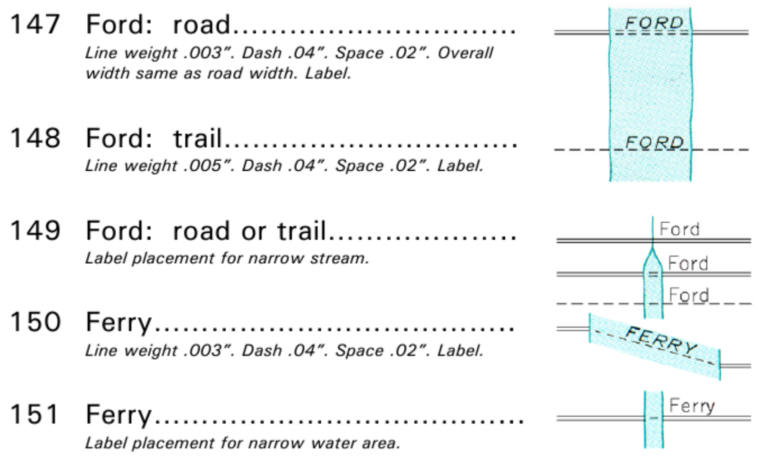
Figure 29: USGS Contour-Line Examples

The elevation value of an index contour line can be plotted in the same way as with depth curves above, or, since the gap is so narrow between the digits and the line, it could conceivably be plotted using a **CompositeSymbolizer** with a **PenStroke LineSymbolizer** and a **TextSymbolizer** with a white **Halo** of one or two pixels.

The barbs on the depression lines can be plotted using **CompositeSymbolizer** with a **LineSymbolizer** with a **PenStroke** and another **LineSymbolizer** with a **GraphicStroke** of a **Graphic** of a tick mark. Probably a better idea is to use a **CompoundStroke** with a **PenStroke** and a **GraphicStroke** with a **Length** of 0. This causes the rendering position not to advance after plotting the graphic, so the solid line continues. There is no plot-order issue since the line and the tick are the same color. In the example, the tick marks are more widely spaced apart in the longer rings. If desired, this could be achieved by using a function to compute the linear length of the contour line and fudging this value in some way and using it as the **Length** of the **GraphicStroke**. SE does not define such a function.

The bunching together of the contour lines with gaps between the roads in the Cut and Fill examples would need to be supplied by the source data, as this kind of inter-feature geometric manipulation would be very difficult or impossible to achieve. Fortunately, roads are relatively level, so the real world probably resembles this pattern.

Ford and ferry examples are shown in Figure 30. (A “ford” is a place where a body of water is shallow enough to be crossed by wading.)



147 Ford: road............................
Line weight .003”. Dash .04”. Space .02”. Overall width same as road width. Label.

148 Ford: trail............................
Line weight .005”. Dash .04”. Space .02”. Label.

149 Ford: road or trail...................
Label placement for narrow stream.

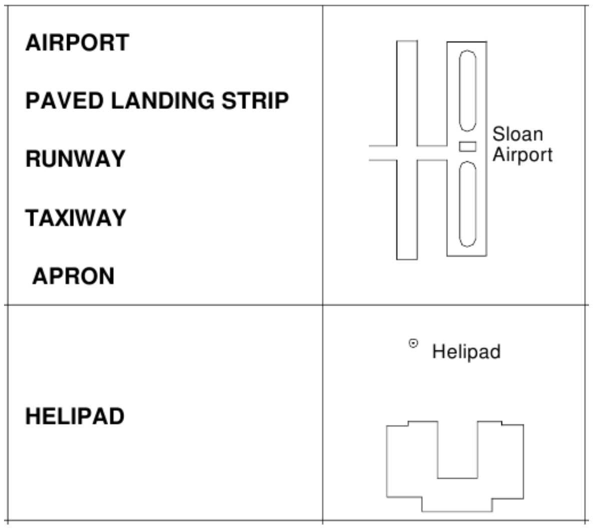
150 Ferry....................................
Line weight .003”. Dash .04”. Space .02”. Label.

151 Ferry....................................
Label placement for narrow water area.

Figure 30: USGS Ford & Ferry Examples

An issue with the ford and ferry crossings is the differing label placements for wide and narrow crossings. If the source data had a property that told which type was needed, this could be handled with two different **Rule**s. If not, then one could conceivably formulate an expression using geometry functions to decide whether a crossing is long or short. An issue with symbol 148 is that, assuming the ford is represented by a different feature from the road, the dash pattern will restart for the ford, unless the rendering system does a great deal of awkward work to try to match the patterns up.

Airport and helipad examples are shown in Figure 31.

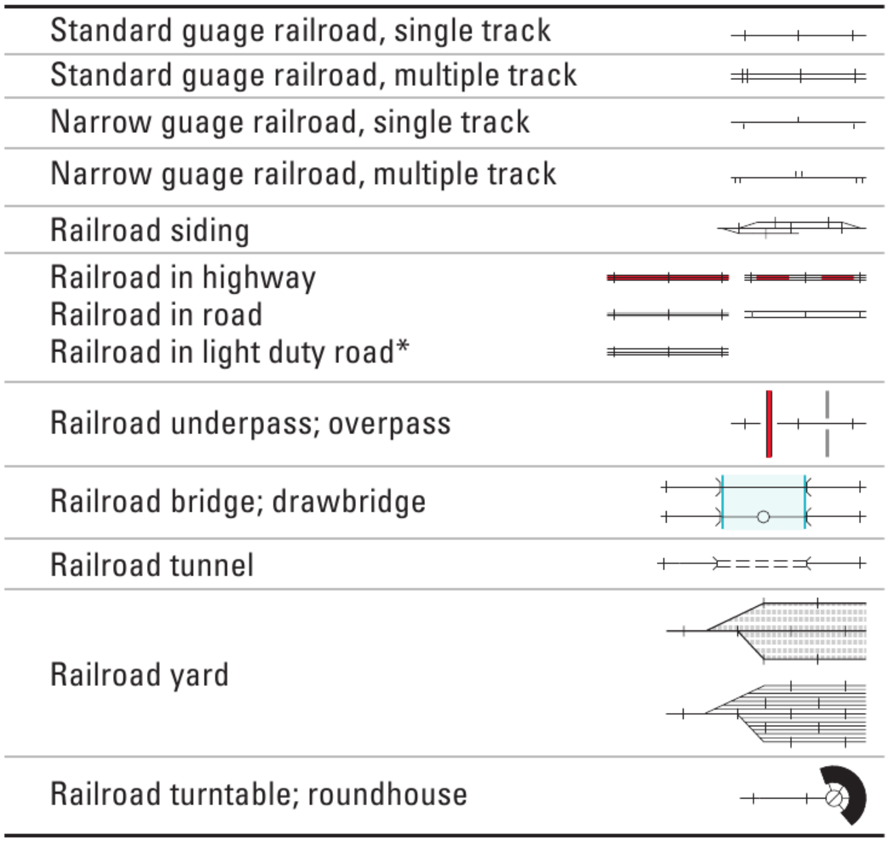


AIRPORT PAVED LANDING STRIP RUNWAY TAXIWAY APRON	Sloan Airport
HELIPAD	Helipad

Figure 31: USGS Airport & Helipad Examples

Both styles have different rules depending on how much space the features occupy on the map. Special geometry and rendering-environment-access functions would be needed to compute this.

Some railroad examples are shown in Figure 32.



Standard guage railroad, single track
Standard guage railroad, multiple track
Narrow guage railroad, single track
Narrow guage railroad, multiple track
Railroad siding
Railroad in highway Railroad in road Railroad in light duty road*
Railroad underpass; overpass
Railroad bridge; drawbridge
Railroad tunnel
Railroad yard
Railroad turntable; roundhouse

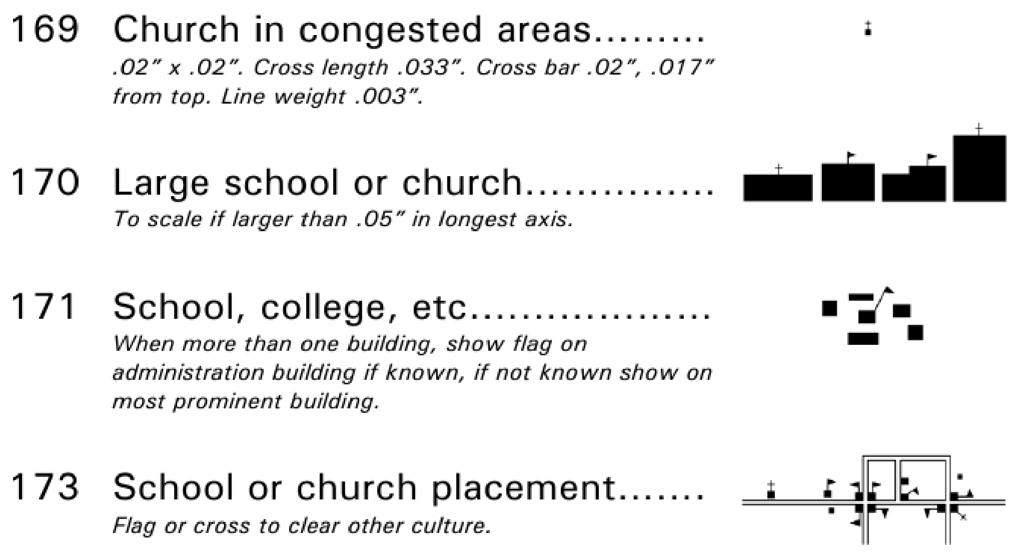
Figure 32: USGS Railroad Examples

The tick marks can be supplied using a **CompoundStroke** with a **PenStroke** and a **GraphicStroke** with **Length** 0. The multiple parallel lines used in the railroad in highway and railroad tunnel can be plotted using a **CompositeSymbolizer** with multiple different **LineSymbolizers** with different **PerpendicularOffsets**, plus the symbolizer for the highway. The ticks can be attached to a specific rail line and made long enough to cover all of them. The tunnel and bridge edges can be plotted using **StrokeAnnotationGraphic**s. The ring in the middle of the drawbridge poses a difficult problem, since the bridge line needs to be not drawn beneath it. This could be crudely simulated using a solid fill of the water color in the middle of the ring graphic.

The overpass and underpass cases pose a more general problem of the same kind as the drawbridge: the road or railway beneath the overpass needs to be not drawn within 0.02" of the overpass (called a "break space"). USGS highways have the same rule in the more difficult circumstance that features of the same type may pass over each other, so the distinction would be an elevation coordinate or property value. SE presently provides no means to address break spacing. The most plausible general way to approach this would be to provide an erasing mechanism with an eraser shape, some means of specifying when content should be erased, and a means of identifying what content should be erased. The application of erasing would be easy to implement in a raster portrayal environment, as it would mean simply changing the pixel opacity values to 0, but in a vector portrayal environment, it would mean clipping graphic elements. Break spacing is an issue to consider in the future.

Railroad yards are also difficult to deal with. The instruction for the first railroad yard style is "Show primary line(s) through RAILROAD YARD and the outermost tracks. Show fill on black separate (plate 515B)." This presents the problem of knowing which tracks are outermost and/or primary and computing the area of the railroad yard for the fill. In principle, the source features could have a property indicating the class of the rail line and railroad-yard areas could be supplied by a separate feature type. Computing this information on the fly would be next to impossible. Also note that the tick marks on the various rail lines in the railroad yard have a high degree of alignment. This would also be difficult to achieve. It would be best left to the portrayal system to try to align similar things as part of its portrayal "finishing" procedures.

Some church and school examples are shown in Figure 33.



169 Church in congested areas.........
.02" x .02". Cross length .033". Cross bar .02", .017" from top. Line weight .003".

170 Large school or church..............
To scale if larger than .05" in longest axis.

171 School, college, etc...................
When more than one building, show flag on administration building if known, if not known show on most prominent building.

173 School or church placement.......
Flag or cross to clear other culture.

Figure 33: USGS Church & School Examples

The style 169 requires that the portrayal system know when an area is "congested" and that some **Filter** function be available to SE to convey this information. Style 171 indicates that a suitable building be chosen and that the flag be drawn in such a way as to avoid other features, which is not something that can be reasonably expressed in SE. Style 173 has a similar issue in that the flags and crosses are drawn in such a way as to avoid other features.

In general, USGS and probably all professional symbology definitions include numerous "finishing rules" that would be difficult to express adequately in an interoperable language and be very difficult to implement. ISO 19117 also defers "finishing rules" to be considered later. Professional mapping is an artificial-intelligence problem that is normally performed in practice with iterations of manual and automatic adjustments to a specific concrete map. SE can address many of the USGS styling requirements, but is unable to address them all.

9 Emergency management symbology

Homeland Security Point Symbology for Emergency Management [ANSI INCITS 415-2006] defines only point symbols which are very easy to encode in SE. In fact, these very symbols were encoded in SLD 1.0.0 for a previous OGC project in 2005, "Emergency Mapping Symbology, Phase 1" (EMS-1) ([5] OGC).

The symbols were supplied to the project in SVG format and the SLD/SE encoding used very simple **PointSymbolizer**s to refer to the SVG files. Some of these SLD files are still available online, for example:

http://demo.cubewerx.com/sld/libraries/ers/EMS_SVG.xml

The SVG files are also still available online, for example:

http://demo.cubewerx.com/sld/libraries/ers/graphics/SVG_ERS_Symbols/CGM_Operations_S1/Fire_Station_S1.svg

The emergency-management symbols are officially distributed in TrueType font files:

http://www.mirrorservice.org/sites/www.ibiblio.org/gentoo/distfiles/ers_v220.zip

One issue with the font files is that all of the the supplied symbols include an outline frame with each font glyph, which means that extra steps would be needed to render the symbols with frames colored according to Annex A of the EMS specification. In other words, font glyphs are meant to be rendered in a uniform color, and there is no way to distinguish the symbol from frame within a glyph. However, glyphs including only frames are also supplied, so they can be plotted over the symbol glyphs, though there may still be issues with the underlying black frames bleeding through around the translucent edges of the frames. If the frame coloring is required, the symbols should be distributed without included frames but should be aligned to be composited with the frame-only glyphs.

Figure 34 shows an ambulance symbol with a frame and a frame symbol alone (the frame style here means "destroyed or totally incapacitated"). The ambulance should be rendered as black, but the frame should be rendered as red. There is no glyph supplied with only the ambulance and not the frame.

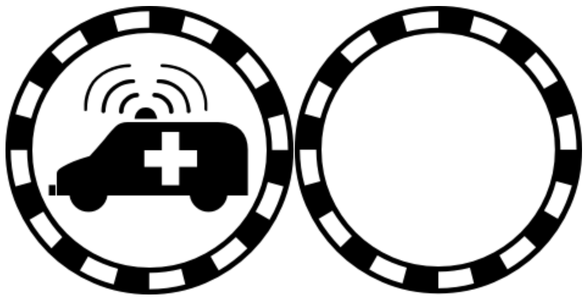


Figure 34: EMS Ambulance Symbol & Frame

Bibliography

[1] OGC 06-042 (2006), *OpenGIS Web Map Service (WMS) Implementation Specification*, <http://portal.opengeospatial.org/files/?artifact_id=14416>

[2] ISO 19125-1:2003 (E) (November 2003), *Geographic information — Simple feature access — Part 1: Common architecture*

[3] Wikipedia, *Relational algebra*, <http://en.wikipedia.org/wiki/Relational_algebra>

[4] USGS, *Framework Web Feature Services*, <http://frameworkwfs.usgs.gov/>

[5] OGC, *Emergency Mapping Symbology, Phase 1* (project), <http://www.opengeospatial.org/projects/initiatives/ems1>

[6] OGC RFQ (July 2008), *Request for Quotation (RFQ) And Call for Participation — OGC Web Services Initiative - Phase 6 (OWS-6)*, pp. 106–107, 117–121.