Open Geospatial Consortium, Inc.

Date: 2009-09-11

Reference number of this document: OGC 09-016

Version: 0.3.0

Category: Public Engineering Report

Editor: Craig Bruce

OGC® OWS-6 Symbology Encoding (SE) Changes

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® Public 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

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

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

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 Chnage 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

OGC® OWS-6 Symbology Encoding (SE) Changes

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 extension of the OGC Symbology Encoding (SE) symbology format for improved capability and harmonization with ISO 19117 symbology, International Hydrographic Organization S-52 symbology, USGS Topomap symbology, and Homeland Security Emergency Management symbology.

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

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

1.2 Document contributor contact points

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

1.3 Revision history

Date	Release	Editor	Primary clauses modified	Description
2009-04-27	1.1.20	C. Bruce	Main body	OWS-6 project final release
2009-07-29	0.3.0	C. Reed	Various	Prepare for publication

1.4 Future work

Improvements in this document are desirable to further improve the capability of the SLD/SE format and improve its compatibility 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 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>

In addition to this document, this report includes several XML Schema Document files as specified in Annex A.

3 Terms and definitions

For the purposes of this report, the definitions specified in Clause 4 of the OWS Common Implementation Standard [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
GML	Geography Markup Language
HTTP	Hypertext Transfer Protocol
ISO	International Organization for Standardization
OGC	Open Geospatial Consortium
OWS	OGC Web Services

SE	Symbology Encoding
SLD	Styled Layer Descriptor
SQL	Structured Query Language
SVG	Scalable Vector Graphics
URI	Uniform Resource Identifier
USGS	United States Geological Survey
W3C	World Wide Web Consortium
WMS	Web Map Service
XML	Extensible Markup Language

4.2 Skeleton XML schemas

This document defines data types in XML Schema. A "skeleton" schema is one in which all comments and superfluous type definitions have been removed. The skeleton schemas included in the narrative are informative and the XML-Schema files distributed with this document are normative.

4.3 Narrative verb tenses

This document describes changes made for the proposed new design. Verb tenses are used saying that certain changes "have been" or that a change "is" made rather than saying that certain changes "should be" made to make this document easier to read. However, this document is only a proposal.

5 Symbology encoding 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 extension of the OGC Symbology Encoding (SE) symbology format for improved capability and harmonization with ISO 19117 symbology, International Hydrographic Organization S-52 symbology, USGS Topomap symbology, and Homeland Security Emergency Management symbology.

This report details changes made to the SE design at the common-elements, feature-type/coverage style, symbolizer, and function-extension levels.

6 Common elements

6.1 Version

A new element named **Version** is added and is used to replace every **version** attribute in the previous designs to allow the XML structures to be more portable to other structuring languages which do not have the unusual "attribute" concept of XML Schema.

The version number for the **VersionType** is changed to "**1.1.20**". This is to differentiate this proposed/development version of the SE specification from other versions and to facilitate testing implementation. In a previous OWS project to extend symbology, this same approach was taken (with version 1.0.20).

6.2 Description type and element

The **DescriptionType** definition is removed from SE and the OWS-Common 1.1.0 definition is used instead. **DescriptionType** was defined in SE before OWS-Common came into existence but there is now no point in keeping redundant definitions.

It would have been good to eliminate the SE **Description** element definition along with the **DescriptionType**, but unfortunately, OWS-Common does not define a **Description** element, only the data type, so this change cannot be made at this time.

6.3 Online resource

The **OnlineResource** element has the unfortunate properties of being overcomplicated and of using non-portable XML attributes. The only useful piece of information this element contains is the URI of the target resource. This could easily be replaced by an **HRef** element with **anyURI** content. On the other hand, the XLink mechanism is an official W3C Recommendation. No changes are recommended at this time, except to use the OWS-Common **OnlineResourceType** definition (which has the same syntax).

6.4 Feature-type name

FeatureTypeName is of type **xsd:QName**, whereas **Name** and **CoverageName** are of type **xsd:string**. This is because other specifications like GML use feature-type names of this sort. No change is recommended at this time.

6.5 Inline content

The **InlineContent** tag used an attribute to indicate the encoding type (XML or Base64), which is inconsistent with the goal of removing unnecessary attributes. It is updated to use **Encoding** and **Content** sub-elements. It now has the following skeleton XML schema:

```
<xsd:element name="InlineContent">
    <xsd:complexType>
        <xsd:sequence>
            <xsd:element ref="se:Encoding"/>
            <xsd:element ref="se:Content"/>
        </xsd:sequence>
    </xsd:complexType>
</xsd:element>

<xsd:element name="Encoding">
    <xsd:simpleType>
        <xsd:restriction base="xsd:string">
            <xsd:enumeration value="xml"/>
```

```
            <xsd:enumeration value="base64"/>
        </xsd:restriction>
    </xsd:simpleType>
</xsd:element>

<xsd:element name="Content">
    <xsd:complexType mixed="true">
        <xsd:sequence>
            <xsd:any minOccurs="0"/>
        </xsd:sequence>
    </xsd:complexType>
</xsd:element>
```

7 Feature type style

7.1 Filename

The XML-Schema file that holds the **FeatureTypeStyle** tag in SE 1.1.0 is named "**FeatureStyle.xsd**". This is a holdover from an intermediate change in version 1.0.20. The name is changed back to "**FeatureTypeStyle.xsd**" to reflect the name of the major root element in the schema file.

7.2 Version

The version attribute is replaced with an element as discussed in clause 6.1.

7.3 Online resource

The **OnlineResource** optional sub-element of **FeatureTypeStyle** for referencing remote rules is removed. The important grains for breaking SLD/SE into reusable components are: map, feature-type/coverage style, symbolizer, stroke, fill, and graphic. There is no worthwhile advantage referencing remote rules since rules function as a group and are critically tied together with their **FeatureTypeStyle**.

7.4 Feature-type-style reference

Naked **OnlineResource** elements should be avoided in SE and SLD in order to make it explicit what kind of resource is being referenced. To this end, a sibling element to **FeatureTypeStyle** called **FeatureTypeStyleReference** has been added which contains only an **OnlineResource** element that references a remote **FeatureTypeStyle**.

This element makes it clear what is being referenced — a remote SE **FeatureTypeStyle**. One could argue that the term "external" could be used instead in this name; however, this term has a slightly different meaning where it is used in **ExternalGraphic**, where it means that the graphic is in an externally-defined format. An **ExternalGraphic** may even have **InlineContent**.

8 Coverage style

8.1 Version

The version attribute is replaced with an element as discussed in clause 6.1.

8.2 Online resource

The **OnlineResource** optional sub-element of **CoverageStyle** is removed as it is for **FeatureTypeStyle** in clause 7.3.

8.3 Coverage-style reference

A **CoverageStyleReference** sibling element to **CoverageStyle** has been added for reasons discussed with **FeatureTypeStyleReference** in clause 7.4.

9 Rule

Rule now has the following skeleton XML schema:

```
<xsd:element name="Rule" type="se:RuleType">
    <xsd:complexType>
        <xsd:sequence>
            <xsd:element ref="se:Name" minOccurs="0"/>
            <xsd:element ref="se:Description" minOccurs="0"/>
            <xsd:choice minOccurs="0">
                <xsd:element ref="ogc:Filter"/>
                <xsd:element ref="se:ElseFilter"/>
            </xsd:choice>
            <xsd:element ref="se:DomainConstraints" minOccurs="0"/>
            <xsd:element ref="se:MinScaleDenominator" minOccurs="0"/>
            <xsd:element ref="se:MaxScaleDenominator" minOccurs="0"/>
            <xsd:element ref="se:Symbolizer"/>
        </xsd:sequence>
    </xsd:complexType>
</xsd:element>
```

9.1 Else filter

The definition of **ElseFilter** does not account for handling *Null* property values. Relational Database systems often use a ternary logic system that has truth values of **true**, **false**, and **null**. If expression *A* evaluates to **null**, so does **not** *A*, but only expressions that evaluate to **true** are selected for retrieval. Thus, the SE-1.1.0-suggested mechanism of using **not** *A* to implement the **ElseFilter** mechanism will not work properly with ternary-logic systems.

The verbiage in the specification should be updated to explain this situation and explicitly say that all features not selected by the other **Filter** conditions are selected by the **ElseFilter** condition. One could also consider eliminating the **ElseFilter** mechanism and requiring the user to incorporate such conditions manually.

9.2 Domain constraints

The filtering mechanisms that were contained in the **LayerFeatureConstraints** and **LayerCoverageConstraints** elements that were removed from SLD 1.1.0 are now incorporated into the **Rule** element. This provides the extra filtering functionality for services other than just WMS and simplifies the SLD-level elements.

The SLD filtering elements to logically include here are **Extent**, **RangeAxis**, and **TimePeriod**. The SLD specification does not explicitly define the semantics of **Extent** for features and it actually seems to be equivalent to **RangeAxis**, so these are collapsed. Also, since the "domain" of a function is the set of "input" values and the "range" is the set of "output" values, it would seem that the correct term to use here is "Domain" instead of "Range". An input value would be a specific time or dimension interval and the output would be the relevant features or raster samples or coverage elements.

The new **DomainConstraints** sub-element of **Rule** has the following skeleton XML schema:

```
<xsd:element name="DomainConstraints">
    <xsd:complexType>
        <xsd:sequence>
            <xsd:element ref="se:DomainAxis" minOccurs="0" maxOccurs="unbounded"/>
            <xsd:element ref="se:TimePeriod" minOccurs="0"/>
        </xsd:sequence>
    </xsd:complexType>
</xsd:element>

<xsd:element name="DomainAxis">
    <xsd:complexType>
        <xsd:sequence>
            <xsd:element ref="se:Name"/>
            <xsd:element ref="se:DomainValue"/>
        </xsd:sequence>
    </xsd:complexType>
</xsd:element>

<xsd:element name="DomainValue" type="xsd:string"/>
<xsd:element name="TimePeriod" type="xsd:string"/>
```

DomainConstraints contains an optional list of **DomainAxis** elements with an optional **TimePeriod**. The **DomainAxis** element includes a **Name**/**DomainValue** pair.

9.3 Scale filtering

Scale filtering is presently incorporated into SE using the XML tags **MinScaleDenominator** and **MaxScaleDenominator**. An argument could be made to instead incorporate this into the Filter mechanism as an ordinary function which returns the current rendering-scale denominator. This would be more flexible, since conditions of any kind could be constructed, such as "**a=6 or scale() >= 10000**", or even "**a <= scale()**". However, this flexibility probably is not necessary and the explicit scale

intervals in SE are easy to optimize, execute, and transform into different representations. No change in representation is recommended at this time.

SE scale processing should also incorporate any pixel-size information supplied from the viewer client. For example, if the viewer client explicitly states that the rendering system that the real pixel size is 0.0847mm (printed paper), the scale constraints should be processed accordingly. A mechanism for specifying a real pixel size has been suggested for WMS 1.4.

9.4 Symbolizer

Rule has been modified to reference exactly one **Symbolizer** element for reasons discussed in clause 10.5.

10 Symbolizer

Symbolizer now has the following skeleton XML schema:

```
<xsd:element name="Symbolizer" type="se:SymbolizerType" abstract="true"/>
<xsd:complexType name="SymbolizerType" abstract="true">
    <xsd:sequence>
        <xsd:element ref="se:Version" minOccurs="0"/>
        <xsd:element ref="se:Name" minOccurs="0"/>
        <xsd:element ref="se:Description" minOccurs="0"/>
        <xsd:element ref="se:LegendGraphic" minOccurs="0"/>
        <xsd:element ref="se:ArgumentList" minOccurs="0"/>
        <xsd:element ref="se:FormalParameters" minOccurs="0"/>
    </xsd:sequence>
</xsd:complexType>
```

This is an abstract type from which all other symbolizers are derived, meaning they inherit all the above sub-elements.

10.1 Version

The version attribute is replaced by an element as discussed in clause 6.1.

10.2 Base symbolizer

The SE 1.1.0 mechanism for referencing external symbolizers is clunky and overcomplicated and needs to be replaced. This mechanism was introduced simply to reference remote symbols through a URI, but an unfortunate and unnecessary semantic was included that extends the remote symbolizer with additional inline parameters. This complicates implementations while providing little utility beyond referencing remote symbolizers.

The **BaseSymbolizer** element has been removed and replaced with **SymbolizerReference** for the above reasons plus reasons discussed in clause 7.4.

10.3 Parameterized symbolizers

An optional symbolizer-parameterization mechanism has been designed to offer complete symbolizer reusability between different, incompatible feature types. The mechanism consists of a list of formal-parameter names and an argument list. These elements appear in the schema in the order of processing flow.

10.3.1 Formal parameters

FormalParameters has the following skeleton XML schema:

```
<xsd:element name="FormalParameters">
    <xsd:complexType>
        <xsd:sequence>
            <xsd:element ref="se:Parameter" minOccurs="0" maxOccurs="unbounded"/>
        </xsd:sequence>
    </xsd:complexType>
</xsd:element>

<xsd:element name="Parameter">
    <xsd:complexType>
        <xsd:sequence>
            <xsd:element ref="se:Name"/>
            <xsd:element ref="se:Description" minOccurs="0"/>
        </xsd:sequence>
    </xsd:complexType>
</xsd:element>
```

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-library symbolizer.

Though many people may be tempted to extend the formal-parameter definition to include type declarations, this is not functionally necessary and should be avoided to ease implementation complexity. The provided **Description** element should be sufficient to tell the style designer using the symbolizer what the parameters mean.

The mechanism is relatively straightforward to implement, as parameter references inside of Filter expressions can be almost blindly substituted with the argument content. The substitution only needs to obey the Filter syntax (or the internal expression syntax of the runtime system).

10.3.2 Argument list

ArgumentList has the following skeleton XML schema:

```
<xsd:element name="ArgumentList">
    <xsd:complexType>
        <xsd:sequence>
            <xsd:element ref="se:Argument" minOccurs="0" maxOccurs="unbounded"/>
        </xsd:sequence>
    </xsd:complexType>
</xsd:element>

<xsd:element name="Argument">
    <xsd:complexType>
        <xsd:sequence>
            <xsd:element ref="se:Name"/>
            <xsd:element ref="se:Description" minOccurs="0"/>
            <xsd:element ref="se:Value"/>
        </xsd:sequence>
    </xsd:complexType>
</xsd:element>

<xsd:element name="Value" type="se:ParameterValueType"/>
```

10.4 Symbolizer reference

A **SymbolizerReference** element has been added as an extension of the abstract **Symbolizer** element to avoid using naked **OnlineResource** elements in SE. The **SymbolizerReference** element has as its only (extra) sub-element an **OnlineResource** element giving the URI of the online symbolizer. Argument lists will normally be supplied here for symbolizers with formal parameters.

10.5 Composite symbolizer

The new **CompositeSymbolizer** element derived from **Symbolizer** has been added to manage groups of descendant symbolizers as a single unit. This makes the logical grouping more explicit, but more importantly, it allows a group of symbolizers to be remotely referenced using a single URI. **CompositeSymbolizer** has the following skeleton XML schema:

```
<xsd:element name="CompositeSymbolizer" substitutionGroup="se:Symbolizer">
    <xsd:complexType>
        <xsd:complexContent>
            <xsd:extension base="se:SymbolizerType">
                <xsd:sequence>
                    <xsd:element ref="se:Symbolizer" maxOccurs="unbounded"/>
                </xsd:sequence>
            </xsd:extension>
        </xsd:complexContent>
    </xsd:complexType>
</xsd:element>
```

The **CompositeSymbolizer** element includes between one and an unbounded number of **Symbolizer** elements. This makes it fully recursive. For example, a **Rule** could include a **SymbolizerReference** that refers to a **CompositeSymbolizer** that contains a **LineSymbolizer**, another **CompositeSymbolizer**, a **PointSymbolizer**, a **SymbolizerReference**, etc.

Multiple symbolizers are used when a single feature is intended to produce multiple symbolizations that are to be plotted over top of one another. For example, a road with "casing" can be produced by plotting a thin line over a thick line, or a shadow effect can be produced by plotting with a geometry displacement. Multiple symbolizers were previously allowed to be included in a single **Rule** element.

11 Line symbolizer

LineSymbolizer now has the following skeleton XML schema:

```
<xsd:element name="LineSymbolizer" substitutionGroup="se:Symbolizer">
    <xsd:complexType>
        <xsd:complexContent>
            <xsd:extension base="se:SymbolizerType">
                <xsd:sequence>
                    <xsd:element ref="se:Geometry" minOccurs="0"/>
                    <xsd:element ref="se:UnitOfMeasure" minOccurs="0"/>
                    <xsd:element ref="se:PerpendicularOffset" minOccurs="0"/>
                    <xsd:element ref="se:Transform" minOccurs="0"/>
                    <xsd:element ref="se:Stroke" minOccurs="0"/>
                </xsd:sequence>
            </xsd:extension>
        </xsd:complexContent>
    </xsd:complexType>
</xsd:element>
```

11.1 Geometry

11.1.1 Primary geometry

The SE 1.1.0 specification says, "The **Geometry** element is optional and if it is absent then the all geometry properties of the feature type that is used in the containing **FeatureType** are used." This is a bad idea, as a portrayal will be very confused if there is more than one geometry selected. The text should be changed to say, "The **Geometry** element is optional and if it is absent then the primary geometry property of the feature type is used. If no better information is available, then the first geometry property present in the feature schema shall be assumed to be the primary geometry."

11.1.2 Geometry functions

SE 1.1.0 defines the **Geometry** element to be allowed to refer only to a **PropertyName**. This is changed to allow it to include a full Filter expression, since some complex symbolizations may require computed geometries. Style designers should avoid using

this functionality unnecessarily, since it may be poorly supported by many implementations. **Geometry** has the following skeleton XML schema:

```
<xsd:element name="Geometry">
    <xsd:complexType>
        <xsd:sequence>
            <xsd:element ref="ogc:expression"/>
        </xsd:sequence>
    </xsd:complexType>
</xsd:element>
```

11.2 Unit of measure

UnitOfMeasure has the following skeleton XML schema:

```
<xsd:element name="UnitOfMeasure" type="xsd:anyURI"/>
```

11.2.1 Element name

The SE 1.1.0 **uom** attribute has been replaced by a **UnitOfMeasure** element. There is less incentive to keep element names short. **UnitOfMeasure** is the GML element name, though with different capitalization and GML has a very complex definition. The SE definition is very simple.

11.2.2 Additional units

SE 1.1.0 includes only three unit definitions which are insufficient. These include: portrayal pixels, ground meters, and ground feet. At least three additional units are needed to make measurements more portable between styling representations and rendering environments: portrayal millimeters, portrayal inches, and portrayal (printer's) points. The first two provide common metric and American printing measurements and font sizes are commonly specified in points. Percentages could be important, too. One could also argue for the inclusion of the SVG units of picas, font "ems", and font "exes".

The original SLD/SE specification used only portrayal pixels for units and this was sufficient for a time since most renderings were intended for video displays and most video displays have similar-size pixels. However, pixels are insufficient for rendering an image to be printed onto paper at high resolution, so absolute portrayal units are needed. Various official symbology standards also specify sizes in inches or millimeters.

Ground units are also important as they provide a convenient means to make portrayals scale directly with the zoom level. For example, one might define the line representing a road to always be the same width as the actual road. The rendering will always be a suitable size for the map, since the road is presumably a suitable size of the land.

Supporting different units in SE rendering implementations is quite straightforward, since converting absolute units to pixel sizes is simply a matter of multiplying by a constant. Even for ground units, the conversion factors are constant relative to the rendering scale. Supporting several units is not a burden on implementors.

11.2.3 Unit identifiers

GML includes a definition for units of measure, but unfortunately, GML does not appear to specify any particular identifiers to use, so people have been using their own ad-hoc values. The following and many others were found in GML documents using a simple web search on the term "gml uom":

- uom="urn:ogc:def:uom:EPSG::9102"
- uom="http://my.unidata.ucar.edu/content/software/udunits/udunits.txt#deg_C"
- uom="mm"
- uom="..Bbls"

This is what happens when one merely defines a conceptual framework rather than a concrete interoperable system. SE 1.1.0 defines the following additional identifiers:

- **http://www.opengeospatial.org/se/units/meter**
- **http://www.opengeospatial.org/se/units/foot**
- **http://www.opengeospatial.org/se/units/pixel**

The mixing of American and British spellings of names in SE is also unfortunate and inconsistent (e.g., **ColorMap** and **metre**). OGC needs to form a policy on this issue. Here it is recommended that American names be used consistently.

The following identifiers are defined for the new **UnitOfMeasure**:

- **urn:ogc:def:uom:se::px**
- **urn:ogc:def:uom:se::mm**
- **urn:ogc:def:uom:se::in**
- **urn:ogc:def:uom:se::pt**
- **urn:ogc:def:uom:se::percent**
- **urn:ogc:def:uom:se::gm**
- **urn:ogc:def:uom:se::gft**

The first four identifiers follow from the SVG portrayal units of pixels, millimeters, inches, and points. The next one refers to a percentage of the view box of an object. SVG uses a percentage sign, but this does not fit well with the URN syntax. The last two identifiers for meters and feet start with a "**g**" to specifically indicate that they refer to ground units rather than portrayal units.

11.2.4 Pixel units

In SVG, the pixel is considered to be a relative unit of measurement. The SVG authors recommend it to mean an angle of view of about 0.0227 degrees. For a normal display, it corresponds to about 0.28mm, which is the value that SE uses. For material printed on a laser printer, it corresponds to about 0.21mm, since one normally holds a sheet of paper closer when reading it.

In SE, there is confusion when a map is rendered into an image that is to be printed on a high-resolution printer. Lines measured in pixels will be very thin unless some kind of compensation is applied.

11.2.5 Granularity of usage

There are a few different possible granularities at which to allow units of measure to be selected. In SE 1.1.0, the **uom** attribute is only allowed on the Symbolizer elements. ISO 19117 includes coordinate-reference systems (including units of measure) in most classes but not scalar values. SVG allows a unit of measure on every scalar value.

SE now incorporates a unit of measure into selected elements. There needs to be one in every place where SE fragments could be independently developed and shared in symbology libraries, since the units selected likely will not be coordinated.

11.2.6 Unit source

The unit of measure to use for a parameter value is given by the innermost parent element that contains a unit of measure. If no parent includes an explict unit of measure, the default is pixels.

11.2.7 SVG syntax

SE 1.1.0 also supports the use of units given directly inside of graphic parameters using the same syntax that SVG uses, two-letter names appended to values, such as "**5px**". This is a bad idea, as it makes parsing values more complicated. This capability is removed.

11.3 Perpendicular offset

The **PerpendicularOffset** in SE 1.1.0 provides a transformation of a linear geometry that cannot be achieved with an affine transformation and therefore, it should be retained, even though it can be difficult to implement properly. Note that a perpendicular offset is different from a displacement in that the linear geometry in that a displaced line will have exactly the same shape as the original, whereas a perpendicular-offset line may have a different shape and overall length because of the need to keep a constant distance from the original line at the corners. One could imagine adding the parameters for specifying the method for generating the new corners and ends, but this is probably not necessary. Logically, the constant offset will produce rounded corners and butt ends (or one could even argue, disconnected corners).

An example application for the perpendicular offset is in producing double dashed lines for a road, similar to the pattern "====". This can be achieved using two symbolizers where one uses a positive perpendicular offset and the other, a negative. This effectively produces an outline of the original line, with open, butt ends.

The **PerpendicularOffset** element is moved to come directly after the **Geometry** and **UnitOfMeasure** elements in a **LineSymbolizer**, since this reflects the flow of the

processing. The linear geometry is fully determined before the **Stroke** is applied. **PerpendicularOffset** has the following skeleton XML schema:

```
<xsd:element name="PerpendicularOffset" type="se:ParameterValueType"/>
```

11.4Parameter value type

Most graphical parameters in SE are of type **ParameterValueType**, which has the following skeleton XML schema:

```
<xsd:complexType name="ParameterValueType" mixed="true">
    <xsd:sequence minOccurs="0">
        <xsd:element ref="ogc:expression"/>
    </xsd:sequence>
</xsd:complexType>
```

SE 1.1.0 included a definition for this which allowed the arbitrary mixing of literal characters and Filter expressions. This should be simplified to allow either a single literal character value or a single Filter expression. There is no need for mixing since the Filter-compatible **Concatenate** operation can accomplish the same task in a consistent way. One could argue that the literal-character option should be removed since it is equivalent to a single Filter **Literal** tag, however, this would likely result in SE documents that are significantly longer and more difficult to edit by hand. SVG units are also disallowed from literals in **ParameterValueType** since they are difficult to parse and units of measure are provided elsewhere.

11.5Transform

11.5.1 Element

A **Transform** element has been added to SE to perform general affine transformations using homogeneous coordinates on geometries and graphics. This idea is borrowed from SVG. This does mean that SE needs to deal with geometry coordinate spaces more explicitly.

SVG provides the following transformation operations:

- **translate**(tx), **translate**(tx, ty)
- **rotate**(angle), **rotate**(angle, x, y)
- **scale**(sxy), **scale**(sx, sy)
- **skewX**(skew_angle)
- **skewY**(skew_angle)
- **matrix**(a, b, c, d, e, f)

These all define affine transformations on a two-dimensional space. The theory and operation of these transformations is fairly straightforward, so SE can supply all of them without an undue burden on implementors. However, the **skewX** and **skewY** operations are uncommon and can be realized using the **matrix** operation, so they need not be

included in SE. Many general tutorials on affine transformations can be found by searching for the term on the Web **Error! Reference source not found.**.

Transform in SE has the following skeleton XML schema:

```
<xsd:element name="Transform">
    <xsd:complexType>
        <xsd:sequence>
            <xsd:choice minOccurs="1" maxOccurs="unbounded">
                <xsd:element ref="se:Translate" minOccurs="0"/>
                <xsd:element ref="se:Rotate" minOccurs="0"/>
                <xsd:element ref="se:Scale" minOccurs="0"/>
                <xsd:element ref="se:Matrix" minOccurs="0"/>
            </xsd:choice>
        </xsd:sequence>
    </xsd:complexType>
</xsd:element>
```

Transform includes children elements **Translate**, **Rotate**, **Scale**, and **Matrix** in a mixed list to be applied in turn.

Translate has child elements **X**, and **Y** which give the amount of translation (displacement) in the X and Y dimensions. The units of the translation are the units of the nearest parent. Default values for **X** and **Y** are zero units. All argument values used in **Transform** are of type **ParameterValueType**, meaning they can be computed expressions.

Rotate has child elements **Angle**, **X**, and **Y** which give the amount of rotation and the point about which the rotation should be performed. Performing a rotation about a point is equivalent to translating to the point to the origin, performing the rotation about the origin, and then translating back to the point. The **Angle** is always in clockwise degrees. The default value for all child elements is zero.

Scale has child element **XY** giving the scaling factor for both the X and Y dimensions, or individual **X** and **Y** elements giving the scaling factors in the individual dimensions. The scaling factors are always unit-less. The default values are 1.

Matrix has six children **A**, **B**, **C**, **D**, **E**, and **F** which form parameters of the 3×3 affine-transformation matrix with homogeneous coordinates:

$$\begin{bmatrix} x' \\ y' \\ 1 \end{bmatrix} = \begin{bmatrix} a & c & e \\ b & d & f \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ 1 \end{bmatrix}$$

or, expressed in the fashion of conventional programming languages:

```
x_new = a * x + c * y + e;
y_new = b * x + d * y + f;
```

This is the matrix-parameter order that SVG uses. The default values are **A**=1, **B**=0, **C**=0, **D**=1, **E**=0, **F**=0.

11.5.2 Coordinate reference systems

A coordinate transformation must be applied relative to a coordinate reference system. In SVG, coordinate values are explicit and fixed, but in SE, geometry-coordinate values are variable. This poses many problems. In SVG, the view box (rendering space) is defined by the user in the root element of the SVG document. SE is defined independently of its rendering environment, so it is difficult to integrate the two. Also in SE, the geometries to be styled are supplied at runtime, whereas in SVG, the geometries are fixed and have a predetermined relationship to the rendering view box. SVG also defines the Y axis of its view box to advance downward like the pixel coordinates inside of a raster. Unfortunately, this is a bit inconsistent with how Cartesian coordinates are usually defined. However, since SVG is a successful format and SE is already modeled after SVG, the new SE design adopts an approach similar to SVG for dealing with coordinate reference systems.

11.5.2.1 View-box coordinate space

The view box coordinate space in SE is defined as having its origin in the top-left corner of the rendering canvas with the X axis advancing to the right and the Y axis advancing downward. The units are defined to be the units of the unit of measure of the current symbolizer element. When transforming a feature geometry selected with the **Geometry** element of a symbolizer, the geometry is transformed from the map coordinate system (e.g., EPSG 4326 Lat/Long) into the view box coordinate space (often pixel coordinates within a raster) before any transformations from a **Transform** element included in a symbolizer element are applied. This matches SVG fairly closely, though the X and Y extents of the view box will be variable.

An example application for transforming a geometry would be to produce a shadow effect by defining a compound symbolizer in which the first sub-symbolizer translates the feature geometry 0.9 mm in the X direction and 0.9 mm in the Y direction and the second sub-symbolizer draws the feature geometry without translation. Another example would be to rotate every geometry by a specific angle about the center of the view box. This could make the map correspond to the direction in which a person is currently facing. The center of the view box could be located by using a unit of measure of **percent** and **X** and **Y** values of 50. Some future environmental control that allows a global transformation to be specified might be more suitable for this latter application.

11.5.2.2 Graphic coordinate space

The second major usage for transformations is to assemble composite graphics from multiple sub-graphics by moving standard components into position and to control the placement of graphics relative to the canvas view box or a control geometry. In this case, the coordinate spaces will be individual to the graphics for assembly and placement.

11.6 Stroke

In SE 1.1.0, the **Stroke** element has a clunky design that has been reformed to resemble the cleaner design of the ISO 19117 draft design of 2009-01-05 **Error! Reference source not found.**. ISO 19117 has a bit of redundancy in it that should be collapsed. Also, the **SvgParameter** element has been replaced with specific SE-specific elements for greater syntactic control. This could be an issue of controversy.

The stroke mechanism has been be redefined to have an abstract element called **Stroke** with one sub-element **UnitOfMeasure** and five derived elements: **PenStroke**, **GraphicStroke**, **TextStroke**, **CompoundStroke**, and **StrokeReference**. **Stroke** has the following skeleton XML schema:

```
<xsd:element name="Stroke" type="se:StrokeType" abstract="true"/>
<xsd:complexType name="StrokeType" abstract="true">
    <xsd:sequence>
        <xsd:element ref="se:UnitOfMeasure" minOccurs="0"/>
    </xsd:sequence>
</xsd:complexType>
```

11.6.1 Pen stroke

A **PenStroke** is a stroke that draws a line analogously to how a pen is used (though the ink could be a stippled pattern). **PenStroke** has the following skeleton XML schema:

```
<xsd:element name="PenStroke" substitutionGroup="se:Stroke"/>
    <xsd:complexType>
        <xsd:complexContent>
            <xsd:extension base="se:StrokeType">
                <xsd:sequence>
                    <xsd:choice minOccurs="0">
                        <xsd:element ref="se:Color"/>
                        <xsd:element ref="se:Stipple"/>
                    </xsd:choice>
                    <xsd:element ref="se:Opacity" minOccurs="0"/>
                    <xsd:element ref="se:Width" minOccurs="0"/>
                    <xsd:element ref="se:LineJoin" minOccurs="0"/>
                    <xsd:element ref="se:LineCap" minOccurs="0"/>
                    <xsd:element ref="se:DashArray" minOccurs="0"/>
                    <xsd:element ref="se:DashOffset" minOccurs="0"/>
                </xsd:sequence>
            </xsd:extension>
        </xsd:complexContent>
    </xsd:complexType>
</xsd:element>
```

PenStroke has a choice of either **Color** or **Stipple** to supply the color or color pattern to use to draw the stroke, plus **Opacity**, **Width**, **LineJoin**, **LineCap**, **DashArray**, and **DashOffset** as graphic-control parameters. **Color** is a direct replacement for **SvgParameter** with a name of "**stroke**" with the same semantics. A **Stipple** is equivalent to a **GraphicFill** (Clause 0). The name is changed to "stipple" since "fill" seems like an awkward term in this context. It would be conceptually simpler to always

use a **Fill** instead of allowing a **Color** or **Stipple**, but this would be inconsistent with SVG.

Opacity, **Width**, **LineJoin**, **LineCap**, **DashOffset**, and **DashArray** are direct replacements for **SvgParameter** with **name**s of "**stroke-opacity**", "**stroke-width**", "**stroke-linejoin**", "**stroke-linecap**", "**stroke-dashoffset**", and "**stroke-dasharray**", respectively. All parameters have values of type **ParameterValueType**. **Width**, **DashOffset**, and **DashArray** are in the **UnitOfMeasure** in scope.

11.6.2 Graphic stroke

In SE 1.1.0, the graphic-stroke concept is essentially that one graphic would be repeated over and over along a line and would be bent around corners in some aesthetic way. However, this does not really match how graphic strokes are used in practice. In practice, they are handled more like plotting text glyphs along a line; the graphics are relatively short and there may be more than one and they are rotated to follow the line with the individual graphics plotted as integral units, like letters.

GraphicStroke has the following skeleton XML schema:

```
<xsd:element name="GraphicStroke" substitutionGroup="se:Stroke">
    <xsd:complexType>
        <xsd:complexContent>
            <xsd:extension base="se:StrokeType">
                <xsd:sequence>
                    <xsd:element ref="se:Graphic"/>
                    <xsd:element ref="se:Length" minOccurs="0"/>
                    <xsd:element ref="se:RelativeOrientation" minOccurs="0"/>
                </xsd:sequence>
            </xsd:extension>
        </xsd:complexContent>
    </xsd:complexType>
</xsd:element>
```

The definition of a **GraphicStroke** is relatively simple as more complicated effects are supplied by **CompoundStroke**. **GraphicStroke** includes a single **Graphic** element which gives the graphic icon to plot, **Length**, and **RelativeOrientation**. Length gives the linear length of the geometry to reserve for the graphic icon and is allowed to be zero, which means not to advance the linear plotting position before or after drawing the graphic. The graphic is plotted at the midpoint of the linear length.

RelativeOrientation tells what angle to rotate the graphic to when it is plotted. Allowed values are "**normal**" (the default) meaning rotated to the mathematical normal (perpendicular) of the line at the plotting point, "**line**" which means to rotate to the direction of the line at the plotting point, "**portrayal**" which means always upright with respect to the parent portrayal environment (usually the map), and "**normalUp**" which means to rotate with respect to the normal of the line, but rotate an additional 180 degrees if the graphic would otherwise end up pointing downward with respect to the parent portrayal environment.

11.6.3 Text stroke

TextStroke allows compound strokes to include text labels as part of a complex pattern. This is needed to label the elevations of key contour lines, for example. **TextStroke** has the following skeleton XML schema:

```
<xsd:element name="TextStroke" substitutionGroup="se:Stroke">
    <xsd:complexType>
        <xsd:complexContent>
            <xsd:extension base="se:StrokeType">
                <xsd:sequence>
                    <xsd:element ref="se:LineLabel"/>
                </xsd:sequence>
            </xsd:extension>
        </xsd:complexContent>
    </xsd:complexType>
</xsd:element>
```

LineLabel is defined in the context of the text symbolizer in clause 0.

11.6.4 Compound stroke

A **CompoundStroke** allows multiple graphic and/or simpler strokes to be combined together. This functionality is needed to produce complex stroke styles such as rendering a sequence of graphic icons along a line or drawing simple dashed lines between boat-anchor icons. **CompoundStroke** has the following skeleton XML schema:

```
<xsd:element name="CompoundStroke" substitutionGroup="se:Stroke">
    <xsd:complexType>
        <xsd:complexContent>
            <xsd:extension base="se:StrokeType">
                <xsd:sequence>
                    <xsd:element ref="se:PreGap" minOccurs="0"/>
                    <xsd:choice maxOccurs="unbounded">
                        <xsd:element ref="se:StrokeElement"/>
                        <xsd:element ref="se:AlternativeStrokeElements"/>
                    </xsd:choice>
                    <xsd:element ref="se:PostGap" minOccurs="0"/>
                    <xsd:element ref="se:StrokeAnnotationGraphic" minOccurs="0"
                        maxOccurs="unbounded"/>
                </xsd:sequence>
            </xsd:extension>
        </xsd:complexContent>
    </xsd:complexType>
</xsd:element>
```

CompoundStroke contains a **PreGap**, a mix of one or more **StrokeElement**s and **AlternativeStrokeElements**, a **PostGap**, and an optional list of **StrokeAnnotationGraphic**s. The **PreGap** tells how far to advance along the line before starting to plot content and the **PostGap** tells how far from the end of the line to stop all plotting.

A **StrokeElement** renders a sub-stroke of the compound pattern and has the following skeleton XML schema:

```
<xsd:element name="StrokeElement"/>
    <xsd:complexType>
        <xsd:sequence>
            <xsd:element ref="se:PreGap" minOccurs="0"/>
            <xsd:choice>
                <xsd:element ref="se:PenStroke"/>
                <xsd:element ref="se:GraphicStroke"/>
                <xsd:element ref="se:TextStroke"/>
                <xsd:element ref=”se:StrokeReference”/>
            </xsd:choice>
            <xsd:element ref="se:Length" minOccurs="0"/>
            <xsd:element ref="se:PostGap" minOccurs="0"/>
        </xsd:sequence>
    </xsd:complexType>
</xsd:element>
```

StrokeElement contains a **PreGap**, a sub-stroke, a **Length**, and a **PostGap**. **PreGap** gives the linear distance to advance before drawing anything and **PostGap** gives the linear distance to advance after drawing the sub-stroke. The sub-stroke shall not contain any nested **CompoundStroke**, to limit implementation complexity. **Length** gives the length along the line to draw in the particular sub-stroke style. Lengths are all in the unit of measure in scope. If **Length** is omitted, the natural length of the sub-stroke is used. Beware that the natural length of a **PenStroke** is infinite. The natural length of a **GraphicStroke** is the value of its **Length** element or the width of the view box of the graphic.

AlternativeStrokeElements provides the option for the rendering system to choose from different **StrokeElements** and has the following skeleton XML schema:

```
<xsd:element name="AlternativeStrokeElements">
    <xsd:complexType>
        <xsd:sequence>
            <xsd:element ref="se:StrokeElement" maxOccurs="unbounded"/>
        </xsd:sequence>
    </xsd:complexType>
</xsd:element>
```

The included **StrokeElement**s are listed 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. This issue is discussed in the context of IHO S-52 symbology in the OWS-6 Symbology-Encoding Harmonization ER **Error! Reference source not found.**. This mechanism does not add fundamentally more implementation complexity to SE, and simple implementations can always choose the first alternative.

StrokeAnnotationGraphic allows graphic icons to be rendered at any position along a line and can be used to render arrowheads or street directions, for example. It has the following skeleton XML schema:

```
<xsd:element name="StrokeAnnotationGraphic">
    <xsd:complexType>
        <xsd:sequence>
            <xsd:element ref="se:Graphic" minOccurs="0"/>
            <xsd:element ref="se:RelativePosition" minOccurs="0"/>
            <xsd:element ref="se:RelativeOrientation" minOccurs="0"/>
        </xsd:sequence>
    </xsd:complexType>
</xsd:element>
```

StrokeAnnotationGraphic includes a **Graphic**, **RelativePosition**, and **RelativeOrientation**. **RelativePosition** is a decimal number between 0 and 1, where 0 means the start of the line geometry and 1 means the end.

11.6.5 Stroke reference

A **StrokeReference** allows a stroke to be accessed through a hyperlink. It has the following skeleton XML schema:

```
<xsd:element name="StrokeReference" substitutionGroup="se:Stroke">
    <xsd:complexType>
        <xsd:complexContent>
            <xsd:extension base="se:StrokeType">
                <xsd:sequence>
                    <xsd:element ref="se:OnlineResource"/>
                </xsd:sequence>
            </xsd:extension>
        </xsd:complexContent>
    </xsd:complexType>
</xsd:element>
```

12 Area symbolizer

The **PolygonSymbolizer** of SE 1.1.0 has been renamed to **AreaSymbolizer**. “Polygon” is too specific of a term, since the same mechanism can be used for rectangles, curved-path surfaces, etc. “Area” is the common name used in other symbology systems such as ISO 19117 and GeoSym.

AreaSymbolizer has the following skeleton XML schema:

```
<xsd:element name="AreaSymbolizer" substitutionGroup="se:Symbolizer">
    <xsd:complexType>
        <xsd:complexContent>
            <xsd:extension base="se:SymbolizerType">
                <xsd:sequence>
                    <xsd:element ref="se:Geometry" minOccurs="0"/>
                    <xsd:element ref="se:UnitOfMeasure" minOccurs="0"/>
                    <xsd:element ref="se:PerpendicularOffset" minOccurs="0"/>
```

```
                <xsd:element ref="se:Transform" minOccurs="0"/>
                <xsd:element ref="se:Fill" minOccurs="0"/>
                <xsd:element ref="se:Stroke" minOccurs="0"/>
            </xsd:sequence>
        </xsd:extension>
    </xsd:complexContent>
</xsd:complexType>
</xsd:element>
```

12.1 Perpendicular offset

The **PerpendicularOffset** element has been moved up to come immediately after the **Geometry** and **UnitOfMeasure** elements, as in the **LineSymbolizer**. This models the flow of the rendering processing. The perpendicular-offset functionality is only applied to the stroke of the area; the fill is unaffected by it.

12.2 Transform

The SE 1.1.0 **Displacement** element is replaced by the more general **Transform** element.

12.3 Fill

Fill has been made into an abstract element with three derived elements: **SolidFill**, **GraphicFill**, and **FillReference**. **Fill** has the following skeleton XML schema:

```
<xsd:element name="Fill" type="se:FillType" abstract="true"/>
<xsd:complexType name="FillType" abstract="true"/>
```

12.3.1 Solid fill

SolidFill defines a style for filling with a solid color and has the following skeleton XML schema:

```
<xsd:element name="SolidFill" substitutionGroup="se:Fill">
    <xsd:complexType>
        <xsd:complexContent>
            <xsd:extension base="se:FillType">
                <xsd:sequence>
                    <xsd:element ref="se:Color" minOccurs="0"/>
                    <xsd:element ref="se:Opacity" minOccurs="0"/>
                </xsd:sequence>
            </xsd:extension>
        </xsd:complexContent>
    </xsd:complexType>
</xsd:element>
```

It has two children **Color** and **Opacity** which replace the SE 1.1.0 **SvgParameter** names "**fill**" and "**fill-opacity**", respectively.

12.3.2 Graphic fill

GraphicFill repeats a rectangular tiling pattern over an area and has the following skeleton XML schema:

```
<xsd:element name="GraphicFill" substitutionGroup="se:Fill">
    <xsd:complexType name="GraphicFillType">
        <xsd:complexContent>
            <xsd:extension base="se:FillType">
                <xsd:sequence>
                    <xsd:element ref="se:UnitOfMeasure" minOccurs="0"/>
                    <xsd:element ref="se:Graphic"/>
                    <xsd:element ref="se:TileGap" minOccurs="0"/>
                </xsd:sequence>
            </xsd:extension>
        </xsd:complexContent>
    </xsd:complexType>
</xsd:element>
```

A **GraphicFill** has three children: **UnitOfMeasure**, **Graphic**, and **TileGap**. The graphic fill repeats the **Graphic** in a rectangular tiling pattern with a **TileGap** between tiles. The **TileGap** gives the **X** and **Y** displacements between successive tiles in the pattern, in the unit of measure. A gap is needed because the view box of the **Graphic** may be a tight bounding box that does not allow a gap to be specified internally. The default gap size is zero units in each direction.

Only a single **Graphic** is needed because complex repeating patterns can be precomposed within the **Graphic** itself. Cross hatching would be an example of a pattern that can be pre-composed within a **Graphic**, and hence, no specific hatching mechanism is needed in SE. There is no need for a compound fill for the same reason; a **Graphic** can contain a solid-fill region or any sub-graphics.

12.3.3 Fill reference

FillReference allows fill patterns to be reused and has the following skeleton XML schema:

```
<xsd:element name="FillReference" substitutionGroup="se:Fill">
    <xsd:complexType>
        <xsd:complexContent>
            <xsd:extension base="se:FillType">
                <xsd:sequence>
                    <xsd:element ref="se:OnlineResource"/>
                </xsd:sequence>
            </xsd:extension>
        </xsd:complexContent>
    </xsd:complexType>
</xsd:element>
```

13 Point symbolizer

The **PointSymbolizer** plots a **Graphic** icon at the feature-geometry point with the graphic coordinate-reference-system origin used as the anchor position and with the graphic scaled relative to the unit of measure of the graphic. It has the following skeleton schema:

```
<xsd:element name="PointSymbolizer" substitutionGroup="se:Symbolizer">
    <xsd:complexType>
        <xsd:complexContent>
            <xsd:extension base="se:SymbolizerType">
                <xsd:sequence>
                    <xsd:element ref="se:Geometry" minOccurs="0"/>
                    <xsd:element ref="se:UnitOfMeasure" minOccurs="0"/>
                    <xsd:element ref="se:Transform" minOccurs="0"/>
                    <xsd:element ref="se:Graphic"/>
                </xsd:sequence>
            </xsd:extension>
        </xsd:complexContent>
    </xsd:complexType>
</xsd:element>
```

13.1 Graphic

Graphic has been made into an abstract element with derived elements **ExternalGraphic**, **MarkGraphic**, **PointTextGraphic**, **AlternativeGraphics**, **CompositeGraphic**, and **GraphicReference**. The objective is have enough power to create arbitrarily complex composite graphic icons while using only the basic rendering capabilities that were already required by SE 1.1.0. The main extension is in the nesting of the previous mechanisms. **Graphic** has the following skeleton XML schema:

```
<xsd:element name="Graphic" type="se:GraphicType" abstract="true"/>
<xsd:complexType name="GraphicType" abstract="true"/>
```

13.2 External graphic

13.2.1 Base element

ExternalGraphic has the following skeleton schema:

```
<xsd:element name="ExternalGraphic" substitutionGroup="se:Graphic">
    <xsd:complexType>
        <xsd:complexContent>
            <xsd:extension base="se:GraphicType">
                <xsd:sequence>
                    <xsd:choice>
                        <xsd:element ref="se:OnlineResource"/>
                        <xsd:element ref="se:InlineContent"/>
                    </xsd:choice>
                    <xsd:element ref="se:Format"/>
                    <xsd:element ref="se:UnitOfMeasure" minOccurs="0"/>
                    <xsd:element ref="se:ViewBox" minOccurs="0"/>
                    <xsd:element ref="se:Transform" minOccurs="0"/>
```

```
                <xsd:element ref="se:Opacity" minOccurs="0"/>
                <xsd:element ref="se:Halo" minOccurs="0"/>
              </xsd:sequence>
            </xsd:extension>
          </xsd:complexContent>
        </xsd:complexType>
</xsd:element>
```

13.2.2 External graphic reference

The **OnlineResource**, **InlineContent**, **Format**, and **Opacity** elements have the same meaning as in SE 1.1.0. **ColorReplacement** has been removed because it is not a portable concept. It requires implementations to dig into the content of the external graphic format and substitute one color for another. Various implementations will support different formats to different degrees and expecting them to function consistently for such a critical parameter as color is not realistic. External images could also be antialiased which will change the effective colors used. One can use a **MarkGraphic** instead or refer to a family of pre-altered external graphics.

An **ExternalGraphic** refers to exactly one external-graphic resource. Compatible alternatives can be selected using the new **AlternativeGraphics** mechanism described in Clause 0. This has been changed because graphics will be too tightly tied to their internal coordinate spaces to allow a list of alternative graphics at a lower level.

13.2.3 Inherent graphic coordinate space

The coordinate space of a graphic is defined similarly to the coordinate space of the view box of the plotting environment. The X axis advances to the right of the origin and the Y axis advances downward, as in SVG.

When an external graphic is imported for rendering, its internal unit of measure and coordinate space are used by default, where applicable. Raster images are considered to have their origin at the top-left corner and have a unit of measure of pixel. The pixel registration shall use the cell-oriented approach described in WMS 1.3.0 with the origin on the outer edge of the top left pixel and the coordinates of the opposite corner being the width and height of raster.

SVG graphics have a coordinate space defined in their root elements. Unfortunately, SVG separates the concepts of view box and coordinate space. When imported, the coordinate space shall be considered to be canonical but the unit of measure will be a custom size computed from the coordinate-space and view-box sizes.

For other external-graphic formats, the coordinate values shall be used literally and the unit of measure in the parent scope shall be assumed if none is specified or implied by the format. In some formats like font files, the Y axis advances upward, which means that graphics will be drawn upside-down if they are not transformed (by using **ViewBox** or by scaling by −1 on the Y axis).

13.2.4 Coordinate-space manipulation

The **UnitOfMeasure**, **ViewBox**, **PerpendicularOffset**, and **Transform** elements manipulate the coordinate space of a graphic icon. The manipulations are processed in sequence. These elements replace the **Size**, **Rotation**, **AnchorPoint**, and **Displacement** elements of SE 1.1.0.

The **UnitOfMeasure** element overrides the inherent unit of measure of the graphic coordinate space. The unit of measure determines how large the graphic will be when it is plotted.

ViewBox has the following skeleton XML schema:

```
<xsd:element name="ViewBox">
    <xsd:complexType>
        <xsd:sequence>
            <xsd:element ref="se:Width" minOccurs="0"/>
            <xsd:element ref="se:Height" minOccurs="0"/>
        </xsd:sequence>
    </xsd:complexType>
</xsd:element>
```

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).

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 given 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.

The **PerpendicularOffset** alters the outline of a **MarkGraphic** geometry only for stroking.

The **Transform** element applies various affine transformations to the graphic, such as rotation, scaling, and translation. The unit of measure is not altered. Note that additional transformations may be applied to the graphic in the context in which it is used. For example, to make a **StrokeAnnotationGraphic** draw an arrowhead at the start of a line that points in the opposite direction of the line, one could transform the graphic

arrowhead so that it points downward in the graphic space and then use a **RelativeOrientation** of **line** in the **StrokeAnnotationGraphic**.

13.2.5 Halo

Halo is added to **ExternalGraphic** to allow graphics to have halos like text labels can in SE 1.1.0.

13.3Mark graphic

A **MarkGraphic** allows a graphic to be created by stroking and filling a geometric line or shape. It has the following skeleton XML schema:

```
<xsd:element name="ExternalGraphic" substitutionGroup="se:Graphic">
    <xsd:complexType>
        <xsd:complexContent>
            <xsd:extension base="se:GraphicType">
                <xsd:sequence>
                    <xsd:choice minOccurs="0">
                        <xsd:element ref="se:WellKnownName"/>
                        <xsd:sequence>
                            <xsd:choice>
                                <xsd:element ref="se:OnlineResource"/>
                                <xsd:element ref="se:InlineContent"/>
                            </xsd:choice>
                            <xsd:element ref="se:Format"/>
                            <xsd:element ref="se:MarkIndex" minOccurs="0"/>
                        </xsd:sequence>
                        <xsd:element ref="gml:_Geometry"/>
                    </xsd:choice>
                    <xsd:element ref="se:UnitOfMeasure" minOccurs="0"/>
                    <xsd:element ref="se:ViewBox" minOccurs="0"/>
                    <xsd:element ref="se:PerpendicularOffset" minOccurs="0"/>
                    <xsd:element ref="se:Transform" minOccurs="0"/>
                    <xsd:element ref="se:Halo" minOccurs="0"/>
                    <xsd:element ref="se:Fill" minOccurs="0"/>
                    <xsd:element ref="se:Stroke" minOccurs="0"/>
                </xsd:sequence>
            </xsd:extension>
        </xsd:complexContent>
    </xsd:complexType>
</xsd:element>
```

This is similar to **ExternalGraphic** except that a **WellKnownName** or GML geometry can be used instead of an external format and a **MarkIndex** can be given for an external format (such as a font file). Also, a fill and a stroke may be applied to the geometry. Since SLD 1.1.0 already includes GML features, the inclusion of a GML geometry does not increase the implementation complexity. The GML-geometry coordinate values shall be interpreted literally in the unit of measure in the parent scope (if not overridden).

13.4 Point-text graphic

A **PointTextGraphic** allows text to be plotted at a given point within a composite graphic. It has the following skeleton XML schema:

```
<xsd:element name="PointTextGraphic" substitutionGroup="se:Graphic">
    <xsd:complexType>
        <xsd:complexContent>
            <xsd:extension base="se:GraphicType">
                <xsd:sequence>
                    <xsd:element ref="se:Position"/>
                    <xsd:element ref="se:UnitOfMeasure" minOccurs="0"/>
                    <xsd:element ref="se:PointLabel" minOccurs="0"/>
                </xsd:sequence>
            </xsd:extension>
        </xsd:complexContent>
    </xsd:complexType>
</xsd:element>
```

Position is given directly in **X** and **Y** sub-elements. The default unit of measure is taken from the parent scope. The **PointLabel** element is described in clause 14.2.

PointTextGraphic should only be used to paint text labels within composite graphic elements, such as in highway shields. 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 icon. **TextSymbolizer** should be used for labeling features.

13.5 Alternative graphics

An **AlternativeGraphics** element allows one of multiple alternative graphic elements to be selected in case an implementation does not support some formats. It has the following skeleton XML schema:

```
<xsd:element name="AlternativeGraphics" substitutionGroup="se:Graphic">
    <xsd:complexType>
        <xsd:complexContent>
            <xsd:extension base="se:GraphicType">
                <xsd:sequence>
                    <xsd:element ref="se:GraphicElement" minOccurs="0"
                        maxOccurs="unbounded"/>
                </xsd:sequence>
            </xsd:extension>
        </xsd:complexContent>
    </xsd:complexType>
</xsd:element>

<xsd:element name="GraphicElement">
    <xsd:complexType>
        <xsd:sequence>
            <xsd:element ref="se:Graphic"/>
        </xsd:sequence>
```

```
    </xsd:complexType>
</xsd:element>
```

The list of **GraphicElements** is in order of most-preferfed alternative to least-preferred. All of the alternatives must have their coordinate spaces and units of measure coordinated to make them function sensibly as semantic equivalents. The last item in a list should be a well-known mark graphic to be sure one will be selected.

It is unclear if this mechanism which is present in all previous versions of SE has actually seen much use in practice.

13.6 Composite graphic

A **CompositeGraphic** binds a group of individual graphics into a single composite graphic. It has the following skeleton schema:

```
<xsd:element name="CompoundGraphic" substitutionGroup="se:Graphic">
    <xsd:complexType>
        <xsd:sequence>
            <xsd:element ref="se:GraphicElement" minOccurs="0" maxOccurs="unbounded"/>
        </xsd:sequence>
    </xsd:complexType>
</xsd:element>
```

The list of **GraphicElements** is the painter's order with the first one being plotted on the bottom. All of the elements must have their coordinate spaces and units of measure coordinated to produce a suitable composite 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.

13.7 Graphic reference

A **GraphicReference** allows a remote SE graphic to be fetched and logically included inline. It has the following skeleton XML schema:

```
<xsd:element name="GraphicReference" substitutionGroup="se:Graphic">
    <xsd:complexType>
        <xsd:complexContent>
            <xsd:extension base="se:GraphicType">
                <xsd:sequence>
                    <xsd:element ref="se:OnlineResource"/>
                </xsd:sequence>
            </xsd:extension>
        </xsd:complexContent>
    </xsd:complexType>
</xsd:element>
```

This mechanism can be used to facilitate online graphic libraries.

14 Text symbolizer

The **TextSymbolizer** has been updated to be parallel to the other symbolizers and abstract out point and line labels in a similar way to how strokes and fills are abstracted and reused outside of line and area symbolizers. The updated skeleton XML schema is:

```
<xsd:element name="TextSymbolizer" substitutionGroup="se:Symbolizer">
    <xsd:complexType>
        <xsd:complexContent>
            <xsd:extension base="se:SymbolizerType">
                <xsd:sequence>
                    <xsd:element ref="se:Geometry" minOccurs="0"/>
                    <xsd:element ref="se:UnitOfMeasure" minOccurs="0"/>
                    <xsd:element ref="se:PerpendicularOffset" minOccurs="0"/>
                    <xsd:element ref="se:Transform" minOccurs="0"/>
                    <xsd:element ref="se:Label" minOccurs="0"/>
                </xsd:sequence>
            </xsd:extension>
        </xsd:complexContent>
    </xsd:complexType>
</xsd:element>
```

14.1 Label

The abstract **Label** element groups **PointLabel** and **LineLabel**. It has the following skeleton XML schema:

```
<xsd:element name="Label" type="se:LabelType" abstract="true"/>
<xsd:complexType name="LabelType" abstract="true"/>
```

14.2 Point label

A **PointLabel** is used to draw a text label relative to a point. It has the following skeleton XML schema:

```
<xsd:element name="PointLabel" substitutionGroup="se:Label">
    <xsd:complexType>
        <xsd:complexContent>
            <xsd:extension base="se:LabelType">
                <xsd:sequence>
                    <xsd:element ref="se:UnitOfMeasure" minOccurs="0"/>
                    <xsd:element ref="se:LabelText" minOccurs="0"/>
                    <xsd:element ref="se:Font" minOccurs="0"/>
                    <xsd:element ref="se:HorizontalAlignment" minOccurs="0"/>
                    <xsd:element ref="se:VerticalAlignment" minOccurs="0"/>
                    <xsd:element ref="se:Rotation" minOccurs="0"/>
                    <xsd:element ref="se:ExclusionZone" minOccurs="0"/>
                    <xsd:element ref="se:Halo" minOccurs="0"/>
                    <xsd:element ref="se:Fill" minOccurs="0"/>
                    <xsd:element ref="se:Stroke" minOccurs="0"/>
                </xsd:sequence>
            </xsd:extension>
        </xsd:complexContent>
```

```
    </xsd:complexType>
</xsd:element>
```

The plotting point is supplied from the context of the parent element and may be in the form of any geometry type. The centroid of a non-point geometry may be used or the label may be repeatedly plotted throughout the area of a large polygon, for example. The exact behavior is implementation dependent. Sophisticated systems will also perform label deconfliction to avoid labels being plotted over top of each other.

The **LabelText** supplies the string of text characters to be plotted. If the source data type is not a text string, then the content is converted to a text string.

The **Font** element has the following skeleton XML schema and follows the SE 1.1.0 and SVG definitions:

```
<xsd:element name="Font">
    <xsd:complexType>
        <xsd:sequence>
            <xsd:element ref="se:UnitOfMeasure" minOccurs="0"/>
            <xsd:element ref="se:FontFamily" minOccurs="0"/>
            <xsd:element ref="se:FontStyle" minOccurs="0"/>
            <xsd:element ref="se:FontWeight" minOccurs="0"/>
            <xsd:element ref="se:FontSize" minOccurs="0"/>
        </xsd:sequence>
    </xsd:complexType>
</xsd:element>
```

The semantics of the **FontFamily** content is changed to match the CSS/SVG semantics: it includes a comma separated list of font families in order of preference. SE 1.1.0 allowed multiple font-family elements. Also, CSS/SVG provides many more definitions and the removal of the **SvgParameter** element relative to SE 1.1.0 disallows these to be included in a consistent way. Additional font parameters may need to be canonized in the future. The normal procedure will be to add an element with the SVG name converted to UpperCamelCase.

HorizontalAlignment and **VerticalAlignment** give enumerated values to control the alignment of the text label relative to the plotting point. The allowed values for **HorizontalAlignment** are "**left**", "**center**", and "**right**". The allowed values for **VerticalAlignment** are "**top**", "**middle**", "**baseline**", and "**bottom**". SE 1.1.0 used numeric values for this purpose, but their meaning was rather obscure and there was no provision for aligning with respect to the font baseline of a label. If the rendering system incorporates label deconfliction, these values shall be regarded as preference hints.

The **Rotation** is unfortunately similar to the **Rotate** of **Translate**, but **Rotation** only includes a direct parameter value. **Rotation** gives the number of clockwise degrees to rotate the label.

An **ExclusionZone** is a new concept for SE. It defines either a circular or rectangular region around the control point which the label should not intersect. This is useful when a graphic is drawn at the control point with a different symbolizer that should not be

over-plotted with the label. Previously, this could be crudely approximated with a fixed displacement, but the new control gives label-deconfliction mechanisms explicit information.

ExclusionZone is an abstract type that has derived types **ExclusionRadius** and **ExlcusionRectangle**. **ExclusionRadius** includes **UnitOfMeasure** and **Radius** elements. **ExclusionRectangle** includes **UnitOfMeasure**, **X**, and **Y** elements. The **X** and **Y** elements give the symmetric distance in the indicated dimension about the control point of the exclusion rectangle. For instance, X=10 and Y=5 indicates a rectangle of 20 by 10 units with the control point at the center.

Halo and **Fill** retain their definitions from SE 1.1.0, though **Halo** now has a **UnitOfMeasure** and **Fill** has been reorganized. **Stroke** has been added to allow the stroking of the font glyphs, though many implementations may not support this.

14.3Line label

A **LineLabel** is used to draw a label relative to a linear geometry. It has the following skeleton XML schema:

```
<xsd:element name="LineLabel" substitutionGroup="se:Label">
    <xsd:complexType>
        <xsd:complexContent>
            <xsd:extension base="se:LabelType">
                <xsd:sequence>
                    <xsd:element ref="se:UnitOfMeasure" minOccurs="0"/>
                    <xsd:element ref="se:LabelText" minOccurs="0"/>
                    <xsd:element ref="se:Font" minOccurs="0"/>
                    <xsd:element ref="se:HorizontalAlignment" minOccurs="0"/>
                    <xsd:element ref="se:VerticalAlignment" minOccurs="0"/>
                    <xsd:element ref="se:Halo" minOccurs="0"/>
                    <xsd:element ref="se:Fill" minOccurs="0"/>
                    <xsd:element ref="se:Stroke" minOccurs="0"/>
                </xsd:sequence>
            </xsd:extension>
        </xsd:complexContent>
    </xsd:complexType>
</xsd:element>
```

This is very similar to **PointLabel** except that **Rotation** and **ExclusionZone** are removed. **Rotation** is made irrelevant by the purpose of a line label, as every glyph will be individually rotated according to the line. One could argue that a **RelativeOrientation** used with **GraphicStroke** could be used here, though it is difficult to imagine a case where anyone would want to use any value other than "**normalUp**" for drawing labels on a map. **ExclusionZone** is also irrelevant.

HorizontalAlignment may not be useful here, but it is included anyway. It can be considered as a preference hint of where to position the label on a line. A label-deconfliction mechanism will normally choose a suitable placement according to its internal algorithms.

SE 1.1.0 had several dubious additional parameters for line-label placement, including **IsRepeated**, **InitialGap**, **Gap**, **IsAligned**, and **GeneralizeLine**. A **TextSymbolizer** really should not be used in this way; it should be used to give an ordinary label to a map feature. The SE 1.1.0 extensions really are specifying a stroke style. In the revised design, a **CompoundStroke** with a **TextStroke** component (and perhaps other components) is a more capable replacement. Also, it is difficult to imagine a map renderer that does not generalize its line geometries in some way which makes the utility of **GeneralizeLine** questionable.

15 Raster symbolizer

The **RasterSymbolizer** definition has been sufficient up to now and the OWS-6 project imposes no need to change its functionality. It remains as-is.

16 Symbology-related Filter functions

16.1 Organization

The symbology-encoding functions are split off from the **Symbolizer.xsd** schema file into a separate file called **Function.xsd**. This is a logical splitting point because functions are not symbolizers and are used elsewhere in the SE hierarchy. The function schema content is also somewhat bulky.

These general function design can mostly stay as-is, though it would be better if Filter came with a set of standard functions so that SE does not need to re-invent string concatenate, etc. Or perhaps these should use the **<Function name="blah">** Filter syntax rather than being direct operations.

The **fallbackValue** attribute on **se:Function** has been made **optional**, since requiring it is awkward and problematic. Suppose that an SE expression is being generated from SQL the string-concatenation expression "**a || b**". What **fallbackValue** would the SE generator generate if it is required to? It would need to make something up out of thin air. This is a general problem with making inherently optional items be required — at the end of the process, one cannot tell the real items from the phony made-up items.

16.2 Dimension

A new function called **Dimension** has been added to help with geometry-class delineation in SE rules. It returns the dimensionality of the given geometry argument. I.e., it returns 0 for points, 1 for curves, 2 for areas, and 3 for solids. Its definition is taken from ISO 19125-1 **Error! Reference source not found.**. It has the following skeleton XML schema:

```
<xsd:element name="Dimension" substitutionGroup="se:Function">
    <xsd:complexType>
        <xsd:sequence>
            <xsd:element ref="se:Geometry"/>
```

```
        </xsd:sequence>
    </xsd:complexType>
</xsd:element>
```

Annex A

XML Schema Documents

In addition to this document, this report includes several XML Schema Documents. These XML Schema Documents are bundled in a zip file with the present document.

The symbology abilities now specified in this document use symbology specified XML Schema Documents included in the zip file with this document. These XML Schema Documents combine the XML schema fragments listed in various subclauses of this document, eliminating duplications. These XML Schema Documents are named:

FeatureTypeStyle.xsd

Symbolizer.xsd

Function.xsd

common.xsd

These XML Schema Documents use and build on the other OGC XML Schema Documents.

Bibliography

[1] OGC 09-012 (April 2009), OWS-6 Symbology-Encoding Harmonization (Engineering Report), Craig Bruce (ed.)

[2] *SVG Tutorial*, <http://www.xml.com/pub/a/2001/03/21/svg.html>

[3] *SVG Coordinate Systems, Transformations, and Units*, <http://www.w3.org/TR/SVG/coords.html>

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

[5] ISO 19117 Revision Draft (January 2009), *Geographic information — Portrayal*

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