Open Geospatial Consortium Inc.

Date: 2006-07-21

Reference number of this OGC® document: OGC 05-077r4

Version: 1.1.0 (revision 4)

Category: OpenGIS® Implementation Specification

Editor: Dr. Markus Müller

Symbology Encoding Implementation Specification

Document type: OpenGIS® Implementation Specification
Document stage: Final version
Document language: English

Contents

Page

i. Preface

This Specification defines Symbology Encoding, an XML language for styling information that can be applied to digital Feature and Coverage data.

This document is together with the Styled Layer Descriptor Profile for the Web Map Service Implementation Specification the direct follow-up of Styled Layer Descriptor Implementation Specification 1.0.0. The old specification document was split up into two documents to allow the parts that are not specific to WMS to be reused by other service specifications.

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.

ii. Document terms and definitions

This document uses the specification terms defined in Subclause 5.3 of [OGC 05-008], which is based on the ISO/IEC Directives, Part 2. Rules for the structure and drafting of International Standards. In particular, the word “shall” (not “must”) is the verb form used to indicate a requirement to be strictly followed to conform to this specification.

iii. Submitting organizations

The following organizations submitted this document to the Open Geospatial Consortium Inc.

CubeWerx Inc.
lat/lon GmbH (Editor)
Pennsylvania State University.
Syncline
Ionic Software s.a.

iv. Document contributor contact points

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

Name	Organization
Larry Bouzane	Compusult Ltd.
Dr. Craig Bruce	CubeWerx Inc.
Ivan Cheung	ESRI
Adrian Cuthbert	m-spatial
Reinhard Erstling	interactive instruments GmbH
Ron Lake	Galdos Systems Inc.
Seb Lessware	Laser-Scan Ltd.
Marwa Mabrouk	ESRI
James Macgill	Google Maps
Dimitri Monie	Ionic Software s.a.
Dr. Markus Müller	lat/lon GmbH
Dr. Andreas Poth	lat/lon GmbH
Raj Singh	Syncline
Dan Specht	US Army ERDC
John Vincent	Intergraph Corp.
Peter Vretanos	CubeWerx Inc.

v. Revision history

Date	Release	Editor	Primary clauses modified	Description
2001-02-07	01-028	Adrian Cuthbert	initial paper for SLD 0.7.0	WMT-2 Project-Discussion Paper
2001-08-31	01-028r2	Craig Bruce	re-write for SLD 0.7.1	MPP-1 Project-Discussion Paper
2001-11-30	01-028r3	Craig Bruce	update for SLD 0.7.2 and DIPR format	MPP-1.1 DIPR preview
2001-11-30	01-028r4	Craig Bruce	fixed up pre-pages, added GeoSym content	MPP-1.1 DIPR
2001-12-28	01-028r5	Craig Bruce	minor fixes, added 2525B content, example pictures	MPP-1.1 IPR
2002-03-12	02-013	Carl Reed Craig Bruce Bill Lalonde	Modified for submission and consideration as RFC Proposal for SLD Implementation Specification	Implementation Specification
2002-04-24	02-013r1	Bill Lalonde Greg	Minor formatting changes	Formating for Public Comment

		Buehler		
2002-08-15	02-013r2	Craig Bruce	Incorporated RFC changes	Incorporated RFC comments
2004-02-26	02-070r1	Craig Bruce	Incorporated SLD-1.0.20/ Style-Management-System changes	First draft for 1.1.0
2004-04-13	02-070r2	Donéa Luc	Incorporated 03-004 change proposal for coverage-data selection and styling	Second draft for 1.1.0
2004-05-01	02-070r3	Clemens Portele, Reinhard Erstling	Incorporated change request 03-095r1, general review for consistency	Third draft for 1.1.0
2004-12-17	02-070r4	Craig Bruce	Partial Incorporation of SLD-RWG & interactive instruments changes; see Annex E.	Fourth draft for 1.1.0
2005-4-11	02-070r5	James Macgill	Completed changes started in r4	Fith draft for 1.1.0
2005-04-29	02-070r6	Markus Müller, Andreas Poth	Incorporated change request 05-028	Sixth draft for 1.1.0
2005-08-22	02-070r7	Markus Müller, Andreas Poth	Finished changes regarding 05-028	Seventh draft for 1.1.0
2005-10-19	05-077	Markus Müller	All	Split SLD specification in SLD profile for WMS and Symbology Encoding (this document)
2006-04-21	05-077r1	Markus Müller, Reinhard Erstling	Included changes decided by SLD RWG for 02-070r4, Added SE functions.	First revision of draft SE 1.1.0 for review by SLD RWG.
2006-06-06	05-077r2	Markus Müller	Included changes induced by comments from Reinhard Erstling and Andreas Poth	Final 1.1.0 version
2006-07-20	05-077r3	Markus Müller, Reinhard Erstling	10.2: standardized rendering pixel size; 11.6: se:Function; editorial changes, All anonymous types of the schema changed to global type definitions.	Submitted version for 1.1.0
2006-08-21	05-077r4	Markus Müller	Editorial changes	Version for vote on publication

vi. Changes to the OGC Abstract Specification

The OGC™ Abstract Specification requires/does not require changes to accommodate the technical contents of this document.

Foreword

This document together with OGC 05-078 (Styled Layer Descriptor Profile of the Web Map Service Implementation Specification) replaces OGC 02-070 and consists of the following part: Symbology Encoding Implementation Specification

This document includes 3 annexes; Annexes A and B are normative, and Annex C is informative.

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

Introduction

The importance of the visual portrayal of geographic data cannot be overemphasized. The skill that goes into portraying data (whether it be geographic or tabular) is what transforms raw information into an explanatory or decision-support tool. From USGS' topographic map series to NOAA and NIMA's nautical charts to AAA's Triptik, fine-grained control of the graphical representation of data is a fundamental requirement for any professional mapping community.

The current OGC Web Map Service (WMS) specification supports the ability for an information provider to specify very basic styling options by advertising a preset collection of visual portrayals for each available data set. However, while a WMS currently can provide the user with a choice of style options, the WMS can only tell the user the name of each style. It cannot tell the user what portrayal will look like on the map. More importantly, the user has no way of defining their own styling rules. The ability for a human or machine client to define these rules requires a styling language that the client and server can both understand. Defining this language, called the ***Symbology Encoding (SE)*** is the focus of this specification. This language can be used to portray the output of Web Map Servers, Web Feature Servers and Web Coverage Servers.

Symbology Encoding Implementation Specification

1 Scope

This OpenGIS® Implementation Specification specifies the format of a map-styling language for producing georeferenced maps with user-defined styling.

2 Conformance

Conformance with this specification shall be checked using all the relevant tests specified in Annex A (normative).

3 Normative references

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

ISO 19105:2000, *Geographic information — Conformance and Testing*

IETF RFC 2045 (November 1996), *Multipurpose Internet Mail Extensions (MIME) Part One: Format of Internet Message Bodies*, Freed, N. and Borenstein N., eds., <http://www.ietf.org/rfc/rfc2045.txt>

IETF RFC 2616 (June 1999), *Hypertext Transfer Protocol – HTTP/1.1*, Gettys, J., Mogul, J., Frystyk, H., Masinter, L., Leach, P., and Berners-Lee, T., eds., <http://www.ietf.org/rfc/rfc2616.txt>

IETF RFC 2396 (August 1998), *Uniform Resource Identifiers (URI): Generic Syntax*, Berners-Lee, T., Fielding, N., and Masinter, L., eds., <http://www.ietf.org/rfc/rfc2396.txt>

OGC AS 12 (January 2002), *The OpenGIS Abstract Specification Topic 12: OpenGIS Service Architecture (Version 4.3)*, Percivall, G. (ed.), <http://www.opengis.org/techno/abstract/02-112.pdf>

OGC Adopted Implementation Specification: Web Map Server version 1.3, August 2004, OGC document OGC 04-024, <http://portal.opengis.org/files/?artifact_id=5316>.

OGC Adopted Implementation Specification: Web Feature Service version 1.1, May 2004, OGC document OGC 04-094, <https://portal.opengeospatial.org/files/?artifact_id=8339>.

OGC Adopted Implementation Specification: Filter Encoding version 1.1, May 2004, OGC document OGC 04-095 <https://portal.opengeospatial.org/files/?artifact_id=8340>.

OGC Adopted Implementation Specification: Geography Markup Language version 3.1.1, May 2004, OGC document OGC 04-095 < https://portal.opengeospatial.org/files/?artifact_id=4700>.

OGC Adopted Implementation Specification: Web Coverage Service version 1.0 (Corrigendum), October 2005, OGC document OGC 05-076, <https://portal.opengeospatial.org/files/?artifact_id=12582>.

In addition to this document, this specification includes several normative XML Schema files. Following approval of this document, these schemas will be posted online at the URL http://schemas.opengeospatial.net/SE/1.1.0. These XML Schema files are also bundled with the present document. In the event of a discrepancy between the bundled and online versions of the XML Schema files, the online files shall be considered authoritative.

4 Terms and definitions

For the purposes of this specification, the definitions specified in Clause 4 of the OWS Common Implementation Specification [OGC 05-008] shall apply. In addition, the following terms and definitions apply.

4.1
map
Pictorial representation of geographic data

5 Conventions

5.1 Abbreviated terms

Most of the abbreviated terms listed in Subclause 5.1 of the OWS Common Implementation Specification [OGC 05-008] apply to this document, plus the following abbreviated terms.

GIF	Graphics Interchange Format
JPEG	Joint Photographic Experts Group
PNG	Portable Network Graphics
SVG	Scalable Vector Graphic
WebCGM	Web Computer Graphics Metafile

WCS Web Coverage Service

WFS Web Feature Service

5.2 UML notation

Some diagrams that appear in this specification are presented using the Unified Modeling Language (UML) static structure diagram, as described in Subclause 5.2 of [OGC 05-008].

6 Symbology Encoding overview

This document defines an XML encoding that can be used for styling feature and coverage data. These styles apply either to specific feature types or coverage types, depending on the used data type.

Symbology Encoding includes the **FeatureTypeStyle** and **CoverageStyle** root elements. These elements include all information for styling the data such as Filter and different kinds of **Symbolizer**s.

As Symbology Encoding is a grammar for styling map data independent of any service interface specification it can be used flexibly by a number of services that style georeferenced information or store styling information that can be used by other services.

7 Symbology Encoding common elements

7.1 Introduction

Symbology Encoding defines elements used for rendering Features and Coverages. The root element of a Symbology Encoding is therefore a **FeatureTypeStyle** or a **Coverage Style**. There are a number of elements used throughout Symbology Encoding that will be described first.

7.2 Common elements

SE defines some basic elements used both by SE itself but also within SLD.

```
<xsd:simpleType name="VersionType">
        <xsd:restriction base="xsd:string">
            <xsd:enumeration value="1.1.0"/>
        </xsd:restriction>
</xsd:simpleType>

<xsd:element name="Name" type="xsd:string"/>

<xsd:element name="Description" type="se:DescriptionType">
</xsd:element>
<xsd:complexType name="DescriptionType">
    <xsd:sequence>
        <xsd:element name="Title" type="xsd:string" minOccurs="0"/>
        <xsd:element name="Abstract" type="xsd:string" minOccurs="0"/>
    </xsd:sequence>
</xsd:complexType>

<xsd:element name="OnlineResource" type="se:OnlineResourceType">
</xsd:element>
<xsd:complexType name="OnlineResourceType">
    <xsd:attributeGroup ref="xlink:simpleLink"/>
</xsd:complexType>
```

The **Name** element is used with most “objects” defined by SE to allow them to be referenced. Names must be unique in the context in which they are defined.

The **Description** element is also reused throughout SE and gives an informative description of the “object” being defined. This information can be extracted and used for such purposes as creating informal searchable metadata in catalogue systems. More metadata fields may be added to this element in the future. The **Name** is not considered to be part of a description since a name has a functional use that is more than just descriptive.

The **OnlineResource** element is used in various places in SE to refer to external documents. The most common usage pattern is to use the **xlink:type** and **xlink:href** fields to refer to a simple link, as in:

```
<OnlineResource xlink:type="simple" xlink:href="http://somesite.com/something.xml"/>
```

A frequent semantic for an **OnlineResource** that refers to an XML file is to process the content as if it appeared directly in-line.

8 Feature type styles

The **FeatureTypeStyle** defines the styling that is to be applied to a single feature type. It is defined as follows:

```
<xsd:element name="FeatureTypeStyle" type="se:FeatureTypeStyleType">
</xsd:element>
<xsd:complexType name="FeatureTypeStyleType">
    <xsd:sequence>
        <xsd:element ref="se:Name" minOccurs="0"/>
        <xsd:element ref="se:Description" minOccurs="0"/>
        <xsd:element ref="se:FeatureTypeName" minOccurs="0"/>
        <xsd:element ref="se:SemanticTypeIdentifier" minOccurs="0" maxOccurs="unbounded"/>
        <xsd:choice maxOccurs="unbounded">
            <xsd:element ref="se:Rule"/>
            <xsd:element ref="se:OnlineResource"/>
        </xsd:choice>
    </xsd:sequence>
    <xsd:attribute name="version" type="se:VersionType"/>
 </xsd:complexType>
```

The **version** is an optional attribute on the **FeatureTypeStyle** element that identifies the SE version number that the **FeatureTypeStyle** corresponds to.

A **FeatureTypeStyle** can have a **Name** and **Description**. The **Name** element does not have an explicit use at present, though it conceivably might be used to reference a featuretype style in some feature-style library. The **Description** is for human-readable information.

The **FeatureTypeName** identifies the specific feature type that the feature-type style is for. It is allowed to be optional but only if a feature type is in-context or if it is intended for usage for a number of feature types using **SemanticTypeIdentifier**.

The **SemanticTypeIdentifier** is experimental and is intended to be used to identify what the feature style (or coverages in case of usage inside a **CoverageStyle**) is suitable to be used for using community-controlled name(s). For example, a single style may be suitable to use with many different feature types. The syntax of the **SemanticTypeIdentifier** string is undefined, but the strings "**generic:line**", "**generic:polygon**", "**generic:point**", "**generic:text**", "**generic:raster**", and "**generic:any**" are reserved to indicate that a **FeatureTypeStyle** may be used with any feature type with the corresponding default geometry type (i.e., no feature properties are referenced in the feature style).

The **FeatureTypeStyle** contains one or more **Rule** elements that allow conditional rendering. Rules are discussed in Clause 10. The **Rule** can either be in-line or referenced by an **OnlineResource**.

9 Coverage styles

The **CoverageStyle** defines the styling that is to be applied to a subset of Coverage data. It is defined as follows:

```
<xsd:element name="CoverageStyle" type="se:CoverageStyleType">
</xsd:element>
<xsd:complexType name="CoverageStyleType">
    <xsd:sequence>
        <xsd:element ref="se:Name" minOccurs="0"/>
        <xsd:element ref="se:Description" minOccurs="0"/>
        <xsd:element ref="se:CoverageName" minOccurs="0"/>
        <xsd:element ref="se:SemanticTypeIdentifier" minOccurs="0" maxOccurs="unbounded"/>
        <xsd:choice maxOccurs="unbounded">
            <xsd:element ref="se:Rule"/>
            <xsd:element ref="se:OnlineResource"/>
        </xsd:choice>
    </xsd:sequence>
    <xsd:attribute name="version" type="se:VersionType"/>
</xsd:complexType>
```

Similar to **FeatureTypeStyle**, the **CoverageStyle** can have a **Name**, **Title**, and **Abstract**. The **Name** element does not have an explicit use at present, though it conceivably may be used to reference a coverage style in some coverage-style library. The **Title** and **Abstract** are for human-readable information.

The **CoverageName** identifies the specific coverage that the coverage style is for. It is allowed to be optional, but only if one coverage is in-context and that coverage must match the syntax and semantics of all coverage-property references inside of the **CoverageStyle**.

The following example displays the mapping of the coverage channel named 'band1' of a range axis named 'Band' on the Gray channel. A "Normalize" contrast enhancement is applied on the result of the Gray channel mapping.

```
<CoverageStyle>
    <Rule>
        <Name>ChannelSelection</Name>
        <Description>
            <Title>Gray channel mapping</Title>
        </Description>
        <RasterSymbolizer>
            <ChannelSelection>
                <GrayChannel>
                    <SourceChannelName>Band.band1</SourceChannelName>
                </GrayChannel>
            </ChannelSelection>
            <ContrastEnhancement>
                <Normalize/>
            </ContrastEnhancement>
        </RasterSymbolizer>
    </Rule>
</CoverageStyle>
```

Selected coverage data is indentified in the channel mapping using the following rule :

```
SourceChannelName = {RangeAxis name}.{RangeAxis value}
```

10 Rules

Rules are used to group rendering instructions by feature-property conditions and map scales. **Rule** definitions are placed immediately inside of featuretype- or coverage-style definitions. The format of a **Rule** is shown in the following XML-Schema fragment:

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

The elements of a **Rule** are described in the following clauses, except for the Symbolizer elements, which are described in Clause 11.

The ordering to use for the **Rule**s inside of a **FeatureTypeStyle** is following the “painters model” with the first item in a list being the first item plotted and hence being on the “bottom”.

10.1 Identification & legends

The **Description** elements give the familiar short title for display lists and longer description for the rule. Rules will typically be equated with different symbol appearances in a map legend, so it is useful to have at least the **Title** in the **Description** so it can be displayed in a legend. The **Name** element allows the rule to be referenced externally, which is needed in some methods of SE usage.

The **LegendGraphic** element allows an optional explicit **Graphic Symbolizer** (described in Subclause 11.3) to be displayed in a legend for this rule. The use of this element and legends in general are discussed in the SLD profile of WMS. Its schema is:

```
<xsd:element name="LegendGraphic" type="se:LegendGraphicType"/>
<xsd:complexType name="LegendGraphicType">
    <xsd:sequence>
        <xsd:element ref="se:Graphic"/>
    </xsd:sequence>
</xsd:complexType>
```

10.2 Scale selection

T**he MinScaleDenominator** and **MaxScaleDenominator** elements of a **Rule** define the range of map-rendering scales for which the rule should be applied. The schema is:

```
<xsd:element name="MinScaleDenominator" type="xsd:double"/>
<xsd:element name="MaxScaleDenominator" type="xsd:double"/>
```

The values used are actually the scale *denominators* relative to a "standardized rendering pixel size" (below). For example, an element-content value of "**`10000000`**" means a scale of **1:10-million**. Scientific notation is also allowed here (and for all non-integer numbers in SE), so a more convenient value of "**`10e6`**" could also be used for the element content for this example.

The **MinScaleDenominator** and **MaxScaleDenominator** elements, as their names suggest, are simply the minimum and maximum ranges of scale (denominators) of maps for which a rule should apply. The minimum scale is inclusive and the maximum scale is exclusive. So, for example, the following scale range:

```
<MinScaleDenominator>100e3</MinScaleDenominator>
<MaxScaleDenominator>1e6</MaxScaleDenominator>
```

corresponds to the logical condition (in C-like-language syntax):

```
scale_denom >= 100e3 && scale_denom < 1e6
```

Both of the elements are optional. The absence of a **MinScaleDenominator** element means there is no minimum-scale term to the condition or logically that the default value is **0**. The absence of a **MaxScaleDenominator** element means that there is no maximum-scale term to the condition of logically that the default value is infinity. So, the following scale constraint:

```
<MinScaleDenominator>100e3</MinScaleDenominator>
```

corresponds to the logical condition:

```
scale_denom >= 100e3
```

There is a similar correspondence for a lone **MaxScaleDenominator** element. The absence of both scale elements in a **Rule** mean that there is no scale constraint and that the rule is applicable to maps of all scales.

The "standardized rendering pixel size" is defined to be 0.28mm × 0.28mm (millimeters). Frequently, the true pixel size of the final rendering device is unknown in the web environment, and 0.28mm is a common actual size for contemporary video displays. If the map-rendering software has information available about the actual pixel size of the final display device, then an extra processing step will be needed (if the actual pixel size is different from the standard pixel size) to adjust the actual rendering scale to calculate the standard rendering scale, which will then be used to compare to the scale range of an SE rule. If the actual display device has non-square pixels, then a method of "linear equivalence" to square pixels should be used to calculate the standard rendering scale. For example:

```
actual_linear = sqrt(actual_x_size * actual_y_size)
```

As an example, suppose that a map is to be rendered on to a display with a known actual resolution of 100 dots per inch (square) and the linear distance of the coordinate system of the map is 200 meters per pixel. The actual scale (denominator) of the map to be rendered is computed as:

100dpi = 1/100 inches (actual pixel size in inches)
1/100 inches × 25.4mm/inch = 0.254mm (actual pixel size)
0.254mm × 1000mm/m = 0.000254m (actual pixel size in meters)
200m ÷ 0.000254m = 787401.5748 (actual scale denominator)

The actual scale denominator is translated into the "standard" scale denominator as:

0.28mm ÷ 0.254mm = 1.102362205 (multiplier for scale conversion)
787401.5748 × 1.102362205 = 868001.736 (standard scale denominator)

The standard scale denominator is approximately 868K. The "type" of the last calculation is correct since the types of its components have the form:

actual × standard/actual = standard

If the actual pixel size was instead 3cm × 2cm (e.g., from being projected onto a large screen) and the actual scale denominator was pre-computed to be 1M, the standard scale would be computed as:

0.28mm ÷ sqrt(30mm × 20mm) = 0.01143095213 (multiplier)
1000000 × 0.01143095213 = 11430.95213 (standard scale denominator)

The standard scale denominator is approximately 11.4K. This result makes sense because the standard pixels are much finer than the actual pixels, so they will have a “finer” scale.

Since it is common to integrate the output of multiple servers into a single displayed result in the web-mapping environment, it is important that different map servers have consistent behaviour with respect to processing scales, so that all of the independent servers will select or deselect rules at the same scales.

To insure consistent behaviour, scales relative to coordinate spaces must be handled consistently between map servers. For geographic coordinate systems, which use angular units, the angular coverage of a map should be converted to linear units for computation of scale by using the circumference of the Earth at the equator and by assuming perfectly square linear units. For linear coordinate systems, the size of the coordinate space should be used directly without compensating for distortions in it with respect to the shape of the real Earth. For example, if a map to be displayed covers a 2-degree by 1-degree area in the WGS-1984 geographic coordinate space, the linear size of this area for conversion to scales would be considered to be:

$2° \times (6378137m \times 2 \times \pi) \div 360° = 222638.9816m$
$1° \times (6378137m \times 2 \times \pi) \div 360° = 111319.4908m$

So, the map extent would be approximately 222639m × 111319m linear distance for the purpose of calculating the scale. If the image size for the map is 600×300 pixels, then the standard scale denominator for the map would be:

$222638.9816m \div 600\ pixels \div 0.00028m/pixel = 1325226.19$

or approximately 1.33M. (Only one dimension needs to be calculated since the coordinate-system space covered by each pixel is square.)

Floating-point roundoff-error control should also be applied to these calculations, to ensure consistency between systems. Since the scale denominators used in rules will often be “round” figures, such as 250000, if a calculation of the current scale results in a value of 249999.99999999, it should be considered to match 250000. A reasonable test for range matching of scale denominators would be defined as follows:

```
scale >= min_scale – epsilon && scale < max_scale + epsilon
```

where **`epsilon`** defined as **`1e-6`**.

An important issue relating to the use of standard scales is the question of what is truly intended by the use of a scale in the context of web mapping. Is the real intention that the translation between “actual” and “standard” scales always be completely accurate, or is the true intention about reducing the amount of “clutter” in the pixels of the image that is being rendered? The second case may be more important to some people. For example, projecting a cluttered image onto a wall will not make it any less cluttered, although it will change the “actual” scale of the map. This issue boils down to the idea that some may want to use the concept of a “standard” scale to control the amount of “clutter” in an image without ever connecting it to or calculating an “actual” scale for a map.

10.3 Feature filtering

The **Filter** and **ElseFilter** elements of a **Rule** allow the selection of features in rules to be controlled by attribute conditions. As discussed in the previous subclause, rule activation may also be controlled by the **MinScaleDenominator** and the **MaxScaleDenominator** elements as well as the map-rendering scale.

The **Filter** element has a relatively straightforward meaning. The syntax of the **Filter** element is defined in the Filter Encoding specification and allows both attribute (property) and spatial filtering. As a simple example, a feature type of "**Roads_FT**" might have a numerical attribute named "**num_lanes**". The following would be a valid **Filter** condition:

```
<ogc:Filter>
  <ogc:PropertyIsGreaterThanOrEqualTo>
    <ogc:PropertyName>num_lanes<ogc:PropertyName>
    <ogc:Literal>4</ogc:Literal>
  </ogc:PropertyIsGreaterThanOrEqualTo>
</ogc:Filter>
```

This would select only road features that have four or more lanes. For convenience, an SQL-like notation will be used for illustrative expressions in this section.

Filters used in different **Rule**s applicable to the same **FeatureTypeStyle** are allowed to overlap in terms of the features selected by each rule. The map styler must execute all rules that are applicable to a feature in the order that the rules appear. For example, if one rule for a user style has the (SQL) condition "**num_lanes >= 6**" and a subsequent rule has the condition "**num_lanes >= 4**", then all roads with four or more lanes would cause both rules to "fire". If the style of the first rule is to draw thick blue lines and the second it to draw thin black lines, then roads with six or more lanes would be drawn with thin black lines over top of thick blue ones. Whether all features are applied to each rule in sequence or whether all suitable rules are applied to each feature in sequence is implementation-specific, although there may be subtle differences in the appearance of maps resulting from each of the approaches.

If a rule has no **Filter** element, the interpretation is that the rule condition is always true, i.e., all features are accepted and styled by the rule. This behaviour is especially recommended for styles using a **SemanticTypeIdentifier**.

The **ElseFilter** allows rules to be specified that are activated for features that are not selected by any other rule in a feature-type style. The syntax is:

```
<xsd:element name="ElseFilter" type="se:ElseFilterType"/>
  <xsd:complexType name="ElseFilterType"/>
```

The **ElseFilter** element has a more complicated interpretation than the **Filter** element, and is interpreted as follows. The nominal scale of the map to be portrayed is computed (as described in the previous subclause) and all rules for scale ranges that do not include the computed nominal scale are discarded from further processing. Then, the specific condition for the **ElseFilter** is computed by "**or**-ing" together all of the other filter

conditions and take the global "**not**" of that condition. For example, if there are two rules in a style that have the (SQL) conditions "**a = 6**" and "**b > 20**", then the condition for the **ElseFilter** would be:

```
not( (a = 6) or (b > 20) )
```

This approach has a straightforward interpretation of building up the combination of the other rule conditions (**or**) and negating the meaning (**not**). However, it would also be logically equivalent, by De Morgan's Law of boolean algebra, to "**and**" together the negatives of the conditions of all other rules in the user style, as in:

```
not(a = 6) and not(b > 20)
```

This approach also has a straightforward interpretation of "features not matching any of the other rules". Note that both approaches handle overlapping **Filter** conditions, and that many execution systems, such as RDBMS query engines, will suitably optimize any awkwardly long conditions with overlapping propositions that might result.

A simple optimization for the above procedure is that if any rules of a user style have no **Filter** condition (i.e., are always "true"), then any **ElseFilter** rules can simply be discarded, since their selection condition will always be false. It is declared that there shall be no more than one active rule with an **ElseFilter** in any user style for any map scale.

If a user style contains no active **ElseFilter** and there are features (of a layer) that do not match the condition of any active style, then those features are simply not styled (i.e., are discarded). The order of rules with **Filters** and **ElseFilters** in a user style is unimportant for the determination of the **ElseFilter** condition.

Note that the above is a description of the semantics of the **ElseFilter** and not a requirement that systems implement exactly the procedural method described; any semantically equivalent method will suffice. The semantics described above allow for scale-dependent and scale-independent (global) "else" conditions for user styles. Some (incomplete) examples follow.

```
<FeatureTypeStyle>
    <Rule>
        <Filter>...[A = 1]...</Filter>
        <PolygonSymbolizer> ...[red]... </PolygonSymbolizer>
    </Rule>
    <Rule>
        <ElseFilter/>
        <PolygonSymbolizer> ...[gray]... </PolygonSymbolizer>
    </Rule>
</FeatureTypeStyle>
```

Above, all features in the layer will be rendered. Features with attribute '**A**' equal to **1** will be rendered in red and all other features will be rendered in gray.

```
<FeatureTypeStyle>
    <Rule>
        <Filter>...[A = 1]...</Filter>
        <PolygonSymbolizer> ...[red]... </PolygonSymbolizer>
    </Rule>
</FeatureTypeStyle>
```

Above, only features with **A=1** will be rendered. All other features will not be rendered.

```
<FeatureTypeStyle>
    <Rule>
        <Filter>...[A = 1]...</Filter>
        <MaxScaleDenominator>250e3</MaxScaleDenominator>
        <PolygonSymbolizer> ...[red]... </PolygonSymbolizer>
    </Rule>
    <Rule>
        <Filter>...[A = 1]...</Filter>
        <MinScaleDenominator>250e3</MinScaleDenominator>
        <MaxScaleDenominator>5e6</MaxScaleDenominator>
        <PolygonSymbolizer> ...[yellow]... </PolygonSymbolizer>
    </Rule>
    <Rule>
        <ElseFilter/>
        <PolygonSymbolizer> ...[gray]... </PolygonSymbolizer>
    </Rule>
</FeatureTypeStyle>
```

Above, all features in the layer will be rendered. For map-scale denominators up to 250K, all features with **A=1** will be rendered in red. For map scale denominators between 250K and 5M, all features with **A=1** will be rendered in yellow. All features with **A != 1** at all scales and all features with **A=1** at scale denominators above or equal to 5M will be rendered in gray.

```
<FeatureTypeStyle>
    <Rule>
        <Filter>...[A = 1]...</Filter>
        <MaxScaleDenominator>1e6</MaxScaleDenominator>
        <PolygonSymbolizer> ...[red]... </PolygonSymbolizer>
    </Rule>
    <Rule>
        <Filter>...[A = 2]...</Filter>
        <MaxScaleDenominator>1e6</MaxScaleDenominator>
        <PolygonSymbolizer> ...[yellow]... </PolygonSymbolizer>
    </Rule>
    <Rule>
        <ElseFilter/>
        <MaxScaleDenominator>1e6</MaxScaleDenominator>
        <PolygonSymbolizer> ...[blue]... </PolygonSymbolizer>
    </Rule>
    <Rule>
        <Filter>...[A = 1]...</Filter>
        <MinScaleDenominator>1e6</MinScaleDenominator>
        <MaxScaleDenominator>10e6</MinScaleDenominator>
        <PolygonSymbolizer> ...[purple]... </PolygonSymbolizer>
    </Rule>
    <Rule>
        <ElseFilter/>
        <MinScaleDenominator>1e6</MinScaleDenominator>
        <MaxScaleDenominator>10e6</MinScaleDenominator>
```

```
        <PolygonSymbolizer> ...[gray]... </PolygonSymbolizer>
    </Rule>
    <Rule>
        <Filter>...[A = 1]...</Filter>
        <MinScaleDenominator>10e6</MinScaleDenominator>
        <PolygonSymbolizer> ...[gray]... </PolygonSymbolizer>
    </Rule>
</FeatureTypeStyle>
```

Above, for a scale denominators less than 1M, all features with **A=1** will be rendered as red; all features with **A=2** will be rendered as yellow, and all other features will be rendered as blue. For scale denominators between 1M and 10M, all features with **A=1** will be rendered as purple and all other features will be rendered as gray. For scale denominators at or above 10M, features with **A=1** will be rendered as gray and all other features will not be rendered.

11 Symbolizers

Embedded inside of **Rule**s, which group conditions for styling features, are Symbolizers. A Symbolizer describes how a feature is to appear on a map. The Symbolizer describes not just the shape that should appear but also such graphical properties as color and opacity. A Symbolizer is obtained by specifying one of a small number of different types of Symbolizers and then supplying parameters to override its default behaviour. Five types of Symbolizers are defined:

- Line
- Polygon
- Point
- Text
- Raster

These Symbolizer types are described in turn below. SVG/CSS2 terminology and syntax are used as appropriate.

All **Symbolizer**s include an optional **gml:uom**-attribute as used by GML (this is set inside the abstract **SymbolizerType** and therefore inherited by all **Symbolizer**s). This applies to all elements included inside a Symbolizer such as **stroke-width**, **Size**, **font-size**, **Gap**, **InitialGap**, **Displacement** and **PerpendicularOffset**. If no **uom** is set inside of **Symbolizer**, all units are measured in pixel, the behaviour used by SLD 1.0.0.

The following **uom** definitions are recommended to be used:

```
<PolygonSymbolizer uom="http://www.opengeospatial.org/se/units/metre"/>
<PolygonSymbolizer uom="http://www.opengeospatial.org/se/units/foot"/>
<PolygonSymbolizer uom="http://www.opengeospatial.org/se/units/pixel"/>
```

Pixel is interpreted as a paper unit, referring to the size of the map, while metre, foot and other similar units are “ground” units referring to the actual size of real-world objects. All values set inside this **Symbolizer** shall use this unit for drawing the corresponding elements. It is also possible to use pixel values inside a **Symbolizer** that uses a **uom**: px has to be appended to the corresponding values in this case (e.g. 5px stands for 5 pixel).

11.1 Line Symbolizer

A **LineSymbolizer** is used to style a “stroke” along a linear geometry type, such as a string of line segments.

11.1.1 Format

The **LineSymbolizer** has the following simple definition:

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

The sub-elements are defined below.

11.1.2 Geometry

The **Geometry** element of a **LineSymbolizer** defines the linear geometry to be used for styling. 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. Most frequently, though, feature types will have only a single geometry property. The format of the **Geometry** element is as follows:

```
<xsd:element name="Geometry" type="se:GeometryType"/>
<xsd:complexType name="GeometryType">
    <xsd:sequence>
        <xsd:element ref="ogc:PropertyName"/>
    </xsd:sequence>
</xsd:complexType>
```

The only method available for defining a geometry is to reference a geometry property using the **ogc:PropertyName** element (defined in the Filter Specification). The content of the element gives the property name in XPath syntax. In principle, a fixed geometry could be defined using GML or operators could be defined for computing the geometry from references or literals. However, using a feature property directly is by far the most commonly useful method.

Geometry types other than inherently linear types can also be used. If a point geometry is used, it should be interpreted as a line of "epsilon" (arbitrarily small) length with a horizontal orientation centered on the point, and should be rendered with two end caps. If a polygon is used (or other "area" type), then its closed outline is used as the line string (with no end caps). If a raster geometry is used, its coverage-area outline is used for the line, rendered with no end caps.

Here is an example usage of this element, referencing a property of a feature called "**centerline**":

```
<Geometry>
  <ogc:PropertyName>centerline</ogc:PropertyName>
</Geometry>
```

In case the Symbolizer is applied to WFS data , the properties that are present in a geometry can be interrogated using the **DescribeFeatureType** call of the WFS interface. All **Symbolizer** types can include a **Geometry** element also.

11.1.3 Stroke

The **Stroke** element of the **LineSymbolizer** encapsulates the graphical-Symbolization parameters for linear geometries. The definition of the **Stroke** element is:

```
<xsd:element name="Stroke" type="se:StrokeType"/>
<xsd:complexType name="StrokeType">
    <xsd:sequence>
        <xsd:choice minOccurs="0">
            <xsd:element ref="se:GraphicFill"/>
            <xsd:element ref="se:GraphicStroke"/>
        </xsd:choice>
        <xsd:element ref="se:SvgParameter" minOccurs="0" maxOccurs="unbounded"/>
    </xsd:sequence>
 </xsd:complexType>
```

The graphical parameters and their values are derived from SVG/CSS2 standards with identical names and semantics. The values for the parameters are given as the contents of the elements. The **Stroke** element is optional inside of **LineSymbolizer** (and other Symbolizers), and its absence means that no stroke is to be rendered.

There are three basic types of strokes: solid-color, **GraphicFill** (stipple), and repeated linear **GraphicStroke**. A repeated linear graphic is plotted linearly and has its graphic Symbolizer bent around the curves of the line string, and a graphic fill has the pixels of the line rendered with a repeating area-fill pattern. If neither a **GraphicFill** nor **GraphicStroke** element is given, then the line Symbolizer will render a solid color.

The simple SVG/CSS2 styling parameters are given with the **SvgParameter** element, which is defined as follows:

```
<xsd:element name="SvgParameter" type="se:SvgParameterType"/>
<xsd:complexType name="SvgParameterType" mixed="true">
    <xsd:complexContent mixed="true">
        <xsd:extension base="se:ParameterValueType">
            <xsd:attribute name="name" type="xsd:string" use="required"/>
        </xsd:extension>
    </xsd:complexContent>
 </xsd:complexType>

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

The parameter values are allowed to be complex expressions for maximum flexibility. The '**mixed="true"**' definition means that regular text may be mixed in with various sub-expressions, implying a text-substitution model for parameter values. Numeric and character-string data types are not distinguished, which may cause some complications. Here are some usage examples:

```
<SvgParameter name="stroke-width">3</SvgParameter>

<SvgParameter name="stroke-width">
  <ogc:Literal>3</ogc:Literal>
</SvgParameter>

<SvgParameter name="stroke-width">
  <ogc:Add>
    <ogc:PropertyName>A</ogc:PropertyName>
    <ogc:Literal>2</ogc:Literal>
  </ogc:Add>
</SvgParameter>

<Label>"This is city "<ogc:PropertyName>NAME</ogc:PropertyName>" of state
"<ogc:PropertyName>STATE</ogc:PropertyName></Label>
```

For usage within Symbology Encoding, SE-specific functions evaluating to a string value are defined, too. Functions can be used inside SvgParameter-elements. The structure and usage of functions is defined in subclause 11.6.

The allowed SVG/CSS styling parameters for a stroke are: "**stroke**" (color), "**stroke-opacity**", "**stroke-width**", "**stroke-linejoin**", "**stroke-linecap**", "**stroke-dasharray**", and "**stroke-dashoffset**". The chosen parameter is given by the **name** attribute of the **SvgParameter** element.

The "**stroke**" **SvgParameter** element gives the solid color that will be used for a stroke. The color value is RGB-encoded using two hexadecimal digits per primary-color component, in the order Red, Green, Blue, prefixed with a hash (#) sign. The hexadecimal digits between **A** and **F** may be in either uppercase or lowercase. For example, full red is encoded as "**#ff0000**" (with no quotation marks). If the "**stroke**" **SvgParameter** element is absent, the default color is defined to be black ("**#000000**") in the context of the **LineSymbolizer**.

The "**stroke-opacity**" **SvgParameter** element specifies the level of translucency to use when rendering the stroke. The value is encoded as a floating-point value ("float") between 0.0 and 1.0 with 0.0 representing completely transparent and 1.0 representing completely opaque, with a linear scale of translucency for intermediate values. For example**, "0.65**" would represent 65% opacity. The default value is 1.0 (opaque).

The **"stroke-width" SvgParameter** element gives the absolute width (thickness) of a stroke in units of measure as defined in the **LineSymbolizer** encoded as a float. The default is 1.0. Fractional numbers are allowed (with a system-dependent interpretation) but negative numbers are not.

The "**stroke-linejoin**" and "**stroke-linecap**" **SvgParameter** elements encode enumerated values telling how line strings should be joined (between line segments) and capped (at the two ends of the line string). The values are represented as content strings. The allowed values for line join are "**mitre**", "**round**", and "**bevel**", and the allowed values for line cap are "**butt**", "**round**", and "**square**". The default values are system-dependent.

The "**stroke-dasharray**" **SvgParameter** element encodes a dash pattern as a series of space separated floats. The first number gives the length in uoms of dash to draw, the second gives the amount of space to leave, and this pattern repeats. If an odd number of values is given, then the pattern is expanded by repeating it twice to give an even number of values. Decimal values have a system-dependent interpretation (usually depending on whether antialiasing is being used). The default is to draw an unbroken line.

The "**stroke-dashoffset**" **SvgParameter** element specifies the distance as a float into the "**stroke-dasharray**" pattern at which to start drawing.

The **GraphicFill** element both indicates that a stipple-fill repeated graphic will be used and specifies the fill graphic. Its syntax is:

```
<xsd:element name="GraphicFill" type="se:GraphicFillType"/>
<xsd:complexType name="GraphicFillType">
    <xsd:sequence>
        <xsd:element ref="se:Graphic"/>
    </xsd:sequence>
</xsd:complexType>
```

A "graphic" can be defined very informally as "a little picture". The appearance of the graphic is defined with the embedded **Graphic** element, which is discussed in Subclause 11.3.2. Additional parameters for the **GraphicFill** may be provided in the future to provide more control the exact style of filling.

The **GraphicStroke** element both indicates that a repeated-linear-graphic stroke type will be used. Its syntax is:

```
<xsd:element name="GraphicStroke" type="se:GraphicStrokeType"/>
<xsd:complexType name="GraphicStrokeType">
    <xsd:sequence>
        <xsd:element ref="se:Graphic"/>
        <xsd:element ref="se:InitialGap" minOccurs="0"/>
        <xsd:element ref="se:Gap" minOccurs="0"/>
    </xsd:sequence>
</xsd:complexType>
```

The **Graphic** sub-element specifies the linear graphic. Proper stroking with a linear graphic requires two "hot-spot" points within the space of the graphic to indicate where the rendering line starts and stops. In the case of raster images with no special mark-up, this line will be assumed to be middle pixel row of the image, starting from the first pixel column and ending at the last pixel column.

InitialGap specifies how far away the first graphic will be drawn relative to the start of the rendering line, while **Gap** gives the distance between two graphics. The XML schema definition is as follows.

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

11.1.4 PerpendicularOffset

PerpendicularOffset allows to draw lines in parallel to the original geometry. For complex line strings these parallel lines have to be constructed so that the distance between original geometry and drawn line stays equal. This construction can result in drawn lines that are actually smaller or longer than the original geometry. It is defined as:

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

The distance is in **uom**s and is positive to the left-hand side of the line string. Negative numbers mean right. The default offset is **`0`**.

11.1.5 Examples

Consider that there is a layer defined with all the features of the type '**River**' that is to be displayed as a blue line two pixels wide. Here is the example **Symbolizer**:

```
<LineSymbolizer>
  <Geometry>
    <ogc:PropertyName>centerline</ogc:PropertyName>
  </Geometry>
  <Stroke>
    <SvgParameter name="stroke">#0000ff</SvgParameter>
    <SvgParameter name="stroke-width">2</SvgParameter>
  </Stroke>
</LineSymbolizer>
```

The resulting map portrayal based upon the above rule is:

Here is a simple example using default stroking of the default geometry property:

```
<LineSymbolizer>
  <Stroke/>
</LineSymbolizer>
```

11.2 Polygon Symbolizer

A **PolygonSymbolizer** is used to draw a polygon (or other area-type geometries), including filling its interior and stroking its border (outline).

11.2.1 Format

The **PolygonSymbolizer** has the following simple definition:

```
<xsd:element name="PolygonSymbolizer" type="se:PolygonSymbolizerType" substitutionGroup="se:Symbolizer"/>
<xsd:complexType name="PolygonSymbolizerType">
    <xsd:complexContent>
        <xsd:extension base="se:SymbolizerType">
            <xsd:sequence>
                <xsd:element ref="se:Geometry" minOccurs="0"/>
                <xsd:element ref="se:Fill" minOccurs="0"/>
                <xsd:element ref="se:Stroke" minOccurs="0"/>
                <xsd:element ref="se:Displacement" minOccurs="0"/>
                <xsd:element ref="se:PerpendicularOffset" minOccurs="0"/>
            </xsd:sequence>
        </xsd:extension>
    </xsd:complexContent>
</xsd:complexType>
```

The **Geometry** element is primarily discussed in Subclause 11.1.2. If a polygon has "holes," then they are not filled, but the borders around the holes are stroked in the usual way. "Islands" within holes **are** filled and stroked, and so on. If a point geometry is referenced instead of a polygon, then a small, square, ortho-normal polygon should be constructed for rendering. If a line is referenced, then the line (string) is closed for filling (only) by connecting its end point to its start point, any line crossings are corrected in

some way, and only the original line is stroked. If a raster geometry is used, then the raster-coverage area is used as the polygon. A missing Geometry element selects the "default" geometry for a feature type.

The **Fill** and **Stroke** elements are contained in the **PolygonSymbolizer** in the conceptual order that they are used and plotted using the "painters model", where the **Fill** will be rendered first, and then the **Stroke** will be rendered on top of the fill.

The **Stroke** element is discussed in Subclause 11.1.3, and a missing **Stroke** element means that the geometry will not be stroked. The **Fill** element is discussed below.

The **Displacement** gives the X and Y displacements from the original geometry. This element may be used to avoid over-plotting of multiple **PolygonSymbolizer**s for one geometry or supplying "shadows" of polygon gemeotries. The displacements are in units of pixels above and to the right of the point. The default displacement i**s X=0, Y=0**.

PerpendicularOffset works as defined for **LineSymbolizer**, allowing to draw polygons smaller or larger than their actual geometry. The distance is in **uom**s and is positive to the outside of the polygon. Negative numbers mean drawing the polygon smaller. The default offset is **0**.

11.2.2 Fill

The **Fill** element specifies how the area of the geometry will be filled. Here is the XML-Schema definition:

```
<xsd:element name="Fill" type="se:FillType"/>
<xsd:complexType name="FillType">
    <xsd:sequence>
        <xsd:element ref="se:GraphicFill" minOccurs="0"/>
        <xsd:element ref="se:SvgParameter" minOccurs="0" maxOccurs="unbounded"/>
    </xsd:sequence>
 </xsd:complexType>
```

There are two types of fills, solid-color and repeated **GraphicFill**. The repeated-graphic fill is selected only if the **GraphicFill** element is present. If the **Fill** element is omitted from its parent element, then no fill will be rendered. The **GraphicFill** and **SvgParameter** elements are discussed in conjunction with the **Stroke** element in Subclause 11.1.3. Here, the **SvgParameter** names are "**fill**" instead of "**stroke**" and "**fill-opacity**" instead of "**stroke-opacity**". None of the other **SvgParameters** in **Stroke** are available for filling and the default value for the fill color in this context is 50% gray (value "**#808080**").

11.2.3 Example

Consider the example of a 'Lake' feature type with a Polygon property called '**geometry**' that we wish to symbolize as a 'light-blue' filled polygon with its boundary drawn as a 'dark blue' line. The lake can be both filled and its boundary drawn using the **PolygonSymbolizer** as follows:

```
<PolygonSymbolizer>
  <Geometry>
    <ogc:PropertyName>the_area</ogc:PropertyName>
  </Geometry>
  <Fill>
    <SvgParameter name="fill">#aaaaff</SvgParameter>
  </Fill>
  <Stroke>
    <SvgParameter name="stroke">#0000aa</SvgParameter>
  </Stroke>
</PolygonSymbolizer>
```

The resulting map portrayal based upon the above rule is:

A very simple styling with default parameters would be:

```
<PolygonSymbolizer>
  <Fill/>
  <Stroke/>
</PolygonSymbolizer>
```

11.3 Point Symbolizer

A **PointSymbolizer** is used to draw a “graphic” at a point.

11.3.1 Format

The **PointSymbolizer** has the following simple definition:

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

The **Geometry** element is discussed in Subclause 11.1.2. In this case, if a line, polygon, or raster geometry is used with this Symbolizer, then the semantic is to use the centroid of the geometry, or any similar representative point. The **Graphic** element is described below.
It may occur multiple times so that a point Symbolizer may be composed of multiple graphics.

11.3.2 Graphic

A **Graphic** is a “graphic symbol” with an inherent shape, color(s), and possibly size. A “graphic” can be very informally defined as “a little picture” and can be of either a raster or vector-graphic source type. The term “graphic” is used since the term “symbol” is similar to “Symbolizer” which is used in a different context in SE. The high-level definition of a **Graphic** element is:

```
<xsd:element name="Graphic" type="se:GraphicType"/>
<xsd:complexType name="GraphicType">
    <xsd:sequence>
        <xsd:choice minOccurs="0" maxOccurs="unbounded">
            <xsd:element ref="se:ExternalGraphic"/>
            <xsd:element ref="se:Mark"/>
        </xsd:choice>
        <xsd:element ref="se:Opacity" minOccurs="0"/>
        <xsd:element ref="se:Size" minOccurs="0"/>
        <xsd:element ref="se:Rotation" minOccurs="0"/>
        <xsd:element ref="se:AnchorPoint" minOccurs="0"/>
        <xsd:element ref="se:Displacement" minOccurs="0"/>
    </xsd:sequence>
 </xsd:complexType>

<xsd:element name="Opacity" type="se:ParameterValueType"/>
<xsd:element name="Size" type="se:ParameterValueType"/>
<xsd:element name="Rotation" type="se:ParameterValueType"/>


<xsd:element name="AnchorPoint" type="se:AnchorPointType"/>
<xsd:complexType name="AnchorPointType">
    <xsd:sequence>
        <xsd:element ref="se:AnchorPointX"/>
        <xsd:element ref="se:AnchorPointY"/>
    </xsd:sequence>
</xsd:complexType>

<xsd:element name="AnchorPointX" type="se:ParameterValueType"/>
<xsd:element name="AnchorPointY" type="se:ParameterValueType"/>


<xsd:element name="Displacement" type="se:DisplacementType"/>
<xsd:complexType name="DisplacementType">
    <xsd:sequence>
        <xsd:element ref="se:DisplacementX"/>
        <xsd:element ref="se:DisplacementY"/>
    </xsd:sequence>
</xsd:complexType>

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

```
<xsd:element name="ExternalGraphic" type="se:ExternalGraphicType"/>
<xsd:complexType name="ExternalGraphicType">
    <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:ColorReplacement" minOccurs="0" maxOccurs="unbounded"/>
    </xsd:sequence>
</xsd:complexType>


<xsd:element name="ColorReplacement" type="se:ColorReplacementType"/>
<xsd:complexType name="ColorReplacementType">
    <xsd:sequence>
        <xsd:element ref="se:Recode"/>
    </xsd:sequence>
</xsd:complexType>


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

If the **Graphic** element is omitted from the parent element, then nothing will be plotted. The **Mark** element is defined and discussed below.

Graphics can either be referenced from an external URL in a common format (such as GIF or SVG) or may be derived from a **Mark**. Multiple external URLs and marks may be referenced with the semantic that they all provide the equivalent graphic in different formats. The “hot spot” to use for positioning the rendering at a point must either be inherent in the external format or is defined to be the “central point” of the graphic, where the exact definition “central point” is system-dependent.

The default if neither an **ExternalGraphic** nor a **Mark** is specified is to use the default mark of a “**square**” with a 50%-gray fill and a black outline, with a size of 6 pixels, unless an explicit **Size** is specified. This definition allows a reasonable display to be selected very simply.

The **ExternalGraphic** element allows a reference to be made to an external graphic file with a Web URL or to in-line content. The **OnlineResource** sub-element (discussed in Subclause 7.2) gives the URL and the **Format** sub-element identifies the expected document MIME type of a successful fetch. Knowing the MIME type in advance allows the styler to select the best-supported format from the list of URLs with equivalent content. Users should avoid referencing external graphics that may change at arbitrary times, since many systems may cache or permanently store graphic content for improved efficiency and reliability. Graphic content should be static when at all possible.

The **InlineContent** sub-element allows the content of an external graphic object to be included in-line. The two choices for encoding are XML and Base-64-encoded binary, as indicated by the **encoding** attribute. An issue with the XML encoding is that the **<?xml** ...**?>** tag of the object cannot be present inside of the **InlineContent** tag. The

external graphic object will be extracted and used like the content fetched from an **ExternalContent** tag.

The **ColorReplacement** element, which may occur multiple times, allows to replace a color in the **ExternalGraphic**, the color specified in the OriginalColor sub-element, by another color as a result of a recode function as defined in 11.6. **LookUpValue** is in this case set to **ExternalGraphic**, both **Data** and **Value** elements are set to color values.

The **Opacity** element gives the opacity to use for rendering the graphic. It has the same semantics as the "**stroke-opacity**" and "**fill-opacity**" **SvgParameter** elements. The default value is "**1.0**".

The **Size** element gives the absolute size of the graphic in **uom**s encoded as a floating-point number. The default size for an object is context-dependent. Negative values are not allowed.

The default size of an image format (such as GIF) is the inherent size of the image. The default size of a format without an inherent size (such as SVG which are not specially marked) is defined to be 16 pixels in height and the corresponding aspect in width. If a size is specified, the height of the graphic will be scaled to that size and the corresponding aspect will be used for the width. An expected common use case will be for image graphics to be on the order of 200 pixels in linear size and to be scaled to lower sizes. On systems that can resample these graphic images "smoothly," the results will be visually pleasing.

The **Rotation** element gives the rotation of a graphic in the clockwise direction about its center point in decimal degrees, encoded as a floating-point number. Negative values mean counter-clockwise rotation. The default value is 0.0 (no rotation). Note that there is no connection between source geometry types and rotations; the point used for plotting has no inherent direction. Also, the point within the graphic about which it is rotated is format dependent. If a format does not include an inherent rotation point, then the point of rotation should be the centroid.

The **AnchorPoint** element of a **PointSymbolizer** gives the location inside of a **Graphic** (or label - see 11.4.4) to use for anchoring the graphic to the main-geometry point. The coordinates are given as two floating-point numbers in the **AnchorPointX** and **AnchorPointY** elements each with values between 0.0 and 1.0 inclusive. The bounding box of the graphic/label to be rendered is considered to be in a coordinate space from 0.0 (lower-left corner) to 1.0 (upper-right corner), and the anchor position is specified as a point in this space. The default point is X=**0.5**, Y=**0.5**, which is at the middle height and middle length of the graphic/label text. A system may choose different anchor points to de-conflict graphics/labels.

The **Displacement** gives the X and Y displacements from the "hot-spot" point. This element may be used to avoid over-plotting of multiple graphic symbols used as part of the same point symbol. The displacements are in units of measure above and to the right of the point. The default displacement is X=0, Y=0.

If **Displacement** is used in conjunction with **Size** and/or **Rotation** then the graphic symbol shall be scaled and/or rotated before it is displaced.

The **Mark** element of a **Graphic** defines a "shape" which has coloring applied to it. The element is defined as follows:

```
<xsd:element name="Mark" type="se:MarkType"/>
<xsd:complexType name="MarkType">
    <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:choice>
        <xsd:element ref="se:Fill" minOccurs="0"/>
        <xsd:element ref="se:Stroke" minOccurs="0"/>
    </xsd:sequence>
</xsd:complexType>

<xsd:element name="WellKnownName" type="xsd:string"/>
<xsd:element name="MarkIndex" type="xsd:integer"/>
```

The **WellKnownName** element gives the well-known name of the shape of the mark. Allowed values include at least "**square**", "**circle**", "**triangle**", "**star**", "**cross**", and "**x**", though map servers may draw a different symbol instead if they don't have a shape for all of these. The default **WellKnownName** is "**square**". Renderings of these marks may be made solid or hollow depending on **Fill** and **Stroke** elements. These elements are discussed in Subclauses 11.2.2 and 11.1.3, respectively.

The alternative to a **WellKnownName** is an external mark format. The **MarkIndex** allows an individual mark in a mark archive to be selected. An example format for an external mark achive would be a TrueType font file, with **MarkIndex** being used to select an individual glyph from that file.

The **Mark** element serves two purposes. It allows the selection of simple shapes, and, in combination with the capability to select and mix multiple external-URL graphics and marks, it allows a style to be specified that can produce a usable result in a best-effort rendering environment, provided that a simple **Mark** is included at the bottom of the list of sources for every **Graphic**.

11.3.3 Examples

Consider the example of symbolizing 'Hospital' features that have a point geometry property called "**locatedAt**" as solid red stars centered on the hospital locations. The **PointSymbolizer** can be represented using a mark as follows:

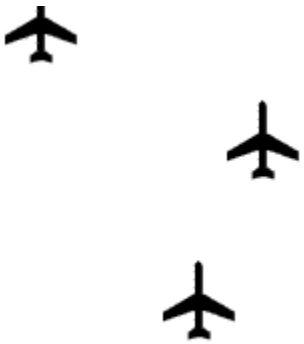
```
<PointSymbolizer>
  <Geometry>
    <ogc:PropertyName>locatedAt</ogc:PropertyName>
  </Geometry>
  <Graphic>
    <Mark>
      <WellKnownName>star</WellKnownName>
      <Fill>
        <SvgParameter name="fill">#ff0000</SvgParameter>
      </Fill>
    </Mark>
    <Size>8.0</Size>
  </Graphic>
</PointSymbolizer>
```

The resulting map portrayal based upon the preceding rule is:

Airports could be symbolized by using the following external-URL example:

```
<PointSymbolizer>
  <Graphic>
    <ExternalGraphic>
      <OnlineResource xlink:type="simple"
        xlink:href="http://www.vendor.com/geosym/2267.svg"/>
      <Format>image/svg+xml</Format>
    </ExternalGraphic>
    <ExternalGraphic>
      <OnlineResource xlink:type="simple"
        xlink:href="http://www.vendor.com/geosym/2267.png"/>
      <Format>image/png</Format>
    </ExternalGraphic>
    <Mark/>
    <Size>15.0</Size>
  </Graphic>
</PointSymbolizer>
```

The resulting map portrayal based upon the preceding rule is:



A point Symbolizer composed of two graphics of which one is displaced may be defined as:

```
<PointSymbolizer>
 <Graphic>
  <ExternalGraphic>
   <OnlineResource xlink:type="simple"
    xlink:href="http://www.vendor.com/geosym/0512.svg"/>
   <Format>image/svg+xml</Format>
  </ExternalGraphic>
 </Graphic>
 <Graphic>
  <ExternalGraphic>
   <OnlineResource xlink:type="simple"
    xlink:href="http://www.vendor.com/geosym/2011.svg"/>
   <Format>image/svg+xml</Format>
  </ExternalGraphic>
  <Size>6.0</Size>
  <Displacement>
   <DisplacementX>11.0</DisplacementX>
   <DisplacementY>8.0</DisplacementY>
  </Displacement>
 </Graphic>
</PointSymbolizer>
```

The resulting map portrayal based upon the preceding rule is:

11.4Text Symbolizer

11.4.1 Format

The **TextSymbolizer** is used for styling text labels and its format is defined as follows:

```
<xsd:element name="TextSymbolizer" type="se:TextSymbolizerType" substitutionGroup="se:Symbolizer"/>
<xsd:complexType name="TextSymbolizerType">
    <xsd:complexContent>
        <xsd:extension base="se:SymbolizerType">
            <xsd:sequence>
                <xsd:element ref="se:Geometry" minOccurs="0"/>
                <xsd:element ref="se:Label" minOccurs="0"/>
                <xsd:element ref="se:Font" minOccurs="0"/>
                <xsd:element ref="se:LabelPlacement" minOccurs="0"/>
                <xsd:element ref="se:Halo" minOccurs="0"/>
                <xsd:element ref="se:Fill" minOccurs="0"/>
            </xsd:sequence>
        </xsd:extension>
    </xsd:complexContent>
</xsd:complexType>
```

These elements are discussed below, except for the **Geometry** and **Fill** elements, which were discussed in Subclauses 11.1.2 and 11.2.2, respectively. The geometry type is interpreted as being either a point or a line as needed by the **LabelPlacement** discussed in Subclause 11.4.4. If the given geometry is not of point or line type as appropriate, it shall be transformed into the appropriate type as discussed in Subclause 11.3.1 for point or Subclause 12.1.2 for line.

11.4.2 Label

The **Label** element is used to provide text-label content. It is defined as follows:

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

The **ParameterValueType** may refer to a complex value and the type of the property/expression is unimportant as the system is expected to provide a text-string version of the property/expression for rendering whatever its type. If a **Label** element is not provided in a **TextSymbolizer**, then no text shall be rendered.

11.4.3 Font

The **Font** element identifies a font of a certain family, style, and size. Its format is defined as:

```
<xsd:element name="Font" type="se:FontType"/>
<xsd:complexType name="FontType">
    <xsd:sequence>
        <xsd:element ref="se:SvgParameter" minOccurs="0" maxOccurs="unbounded"/>
    </xsd:sequence>
</xsd:complexType>
```

Four types of **SvgParameter** are allowed, "**font-family**", "**font-style**", "**font-weight**", and "**font-size**".

The "**font-family**" **SvgParameter** element gives the family name of a font to use. Allowed values are system-dependent. Any number of font-family **SvgParameter** elements may be given and they are assumed to be in preferred order.

The "**font-style**" **SvgParameter** element gives the style to use for a font. The allowed values are "**normal**", "**italic**", and "**oblique**".

The "**font-weight**" **SvgParameter** element gives the amount of weight or boldness to use for a font. Allowed values are "**normal**" and "**bold**".

The "**font-size**" **SvgParameter** element gives the size to use for the font in pixels. The default is defined to be **10** pixels, though various systems may have restrictions on what sizes are available.

When handling vendor-specific fonts, some reasonable interpretation of the CSS font parameters should be used. For example, with a vendor-specific vector-based font, the font family could be interpreted as the basename of the filename including the font; the font style of "**italic**" could be interpreted as an oblique slant; and the weight of "**bold**" could be interpreted as using thicker lines (such as two or three pixels thick).

11.4.4 Label placement

The **LabelPlacement** element is used to position a label relative to a point, line string or polygon and is defined as follows:

```
<xsd:element name="LabelPlacement" type="se:LabelPlacementType"/>
<xsd:complexType name="LabelPlacementType">
    <xsd:choice>
        <xsd:element ref="se:PointPlacement"/>
        <xsd:element ref="se:LinePlacement"/>
    </xsd:choice>
</xsd:complexType>

<xsd:element name="PointPlacement" type="se:PointPlacementType"/>
<xsd:complexType name="PointPlacementType">
    <xsd:sequence>
        <xsd:element ref="se:AnchorPoint" minOccurs="0"/>
        <xsd:element ref="se:Displacement" minOccurs="0"/>
        <xsd:element ref="se:Rotation" minOccurs="0"/>
    </xsd:sequence>
</xsd:complexType>

<xsd:element name="LinePlacement" type="se:LinePlacementType"/>
<xsd:complexType name="LinePlacementType">
    <xsd:sequence>
        <xsd:element ref="se:PerpendicularOffset" minOccurs="0"/>
        <xsd:element ref="se:IsRepeated" minOccurs="0"/>
        <xsd:element ref="se:InitialGap" minOccurs="0"/>
        <xsd:element ref="se:Gap" minOccurs="0"/>
        <xsd:element ref="se:IsAligned" minOccurs="0"/>
        <xsd:element ref="se:GeneralizeLine" minOccurs="0"/>
    </xsd:sequence>
</xsd:complexType>
<xsd:element name="IsRepeated" type="xsd:boolean"/>
<xsd:element name="IsAligned" type="xsd:boolean"/>
<xsd:element name="GeneralizeLine" type="xsd:boolean"/>
```

For a **PointPlacement**, the anchor point of the label and a linear displacement from the point can be specified, to allow a graphic Symbolizer to be plotted directly at the point. This might be useful to label a city, for example. For a **LinePlacement**, a perpendicular offset can be specified, to allow the line itself to be plotted also. This might be useful for labelling a road or a river, for example. The default behaviour of LinePlacement is to draw of the label along the line. If IsRepeated is "true", the label will be repeatedly drawn along the line with **InitialGap** and **Gap** (see clause 11.1.3) defining the spaces at the beginning and between labels. **GeneralizeLine** allows the actual geometry, be it a linestring or polygon to be generalized for label placement. This is e.g. useful for labelling polygons inside their interior when there is need for the label to resemble the shape of the polygon.

Labels can either be aligned to the line geometry if IsAligned is "true" (the default) or are drawn horizontally.

The **AnchorPoint** element of a **PointPlacement** gives the location inside of a label to use for anchoring the label to the main-geometry point. It is formally defined in subclause 11.3.2.The **Displacement** element of a **PointPlacement** gives the X and Y displacements from the main-geometry point to render a text label (see 11.3.2).

This will often be used to avoid over-plotting a graphic symbol marking a city or some such feature. The displacements are in units of pixels above and to the right of the point. A system may reflect this displacement about the X and/or Y axes to de-conflict labels. The default displacement is X=**0**, Y=**0**. **Displacement** is formally defined in Section 11.3.2.

The **Rotation** of a **PointPlacement** gives the clockwise rotation of the label in degrees from the normal direction for a font (left-to-right for Latin-derived human languages at least). **Rotation** is formally defined in Subclause 11.3.2.

The **PerpendicularOffset** element of a **LinePlacement** gives the perpendicular distance away from a line to draw a label. It is defined simply as:

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

The distance is in **uom**s and is positive to the left-hand side of the line string. Negative numbers mean right. The default offset is **0**.

11.4.5 Halo

A **Halo** is a type of **Fill** that is applied to the backgrounds of font glyphs. The use of halos greatly improves the readability of text labels. **Halo** is defined as:

```
<xsd:element name="Halo" type="se:HaloType"/>
<xsd:complexType name="HaloType">
    <xsd:sequence>
        <xsd:element ref="se:Radius" minOccurs="0"/>
        <xsd:element ref="se:Fill" minOccurs="0"/>
    </xsd:sequence>
 </xsd:complexType>

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

The **Radius** element gives the absolute size of a halo radius in pixels encoded as a floating-point number. The radius is taken from the outside edge of a font glyph to extend the area of coverage of the glyph (and the inside edge of "holes" in the glyphs). The halo of a text label is considered to be a single shape. The default radius is one pixel. Negative values are not allowed. The default halo fill is solid white (**Color** "**#FFFFFF**"). The glyph's fill is plotted on top of the halo. The default font fill is solid black (**Color** "**#000000**"). If no **Halo** is selected in the containing **Text**Symbolizer, then no halo will be rendered.

11.4.6 Example

Consider displaying the value of a "**hospitalName**" property of hospital features as a label. Here is an example **TextSymbolizer**:

```
<TextSymbolizer>
  <Geometry>
    <ogc:PropertyName>locatedAt</ogc:PropertyName>
  </Geometry>
  <Label>
    <ogc:PropertyName>hospitalName</ogc:PropertyName>
  </Label>
  <Font>
    <SvgParameter name="font-family">Arial</SvgParameter>
    <SvgParameter name="font-family">Sans-Serif</SvgParameter>
    <SvgParameter name="font-style">italic</SvgParameter>
    <SvgParameter name="font-size">10</SvgParameter>
  </Font>
  <Halo/>
  <Fill>
    <SvgParameter name="fill">#000000</SvgParameter>
  </Fill>
</TextSymbolizer>
```

11.5 Raster Symbolizer

The **RasterSymbolizer** describes how to render raster/matrix-coverage data (e.g., satellite photos, DEMs).

11.5.1 Format

The **RasterSymbolizer** format is defined as follows:

```
    <xsd:element name="RasterSymbolizer" type="se:RasterSymbolizerType"
substitutionGroup="se:Symbolizer"/>
    <xsd:complexType name="RasterSymbolizerType">
        <xsd:complexContent>
            <xsd:extension base="se:SymbolizerType">
                <xsd:sequence>
                    <xsd:element ref="se:Geometry" minOccurs="0"/>
                    <xsd:element ref="se:Opacity" minOccurs="0"/>
                    <xsd:element ref="se:ChannelSelection" minOccurs="0"/>
                    <xsd:element ref="se:OverlapBehavior" minOccurs="0"/>
                    <xsd:element ref="se:ColorMap" minOccurs="0"/>
                    <xsd:element ref="se:ContrastEnhancement" minOccurs="0"/>
                    <xsd:element ref="se:ShadedRelief" minOccurs="0"/>
                    <xsd:element ref="se:ImageOutline" minOccurs="0"/>
```

```
            </xsd:sequence>
        </xsd:extension>
    </xsd:complexContent>
</xsd:complexType>
```

The interpretation of **Geometry** is system-dependent, as raster data may be organized differently from feature data, though omitting this element selects the default raster-data source. Geometry-type transformations are also system-dependent and it is assumed that this capability will be little used. **Opacity** has the usual meaning. The meanings of the other parameters are described with their element definitions. Default values are system or data dependent.

11.5.2 Parameters

The **ChannelSelection** element specifies the false-color channel selection for a multi-spectral raster source (such as a multi-band satellite-imagery source). It is defined as:

```
<xsd:element name="ChannelSelection" type="se:ChannelSelectionType"/>
<xsd:complexType name="ChannelSelectionType">
    <xsd:choice>
        <xsd:sequence>
            <xsd:element ref="se:RedChannel"/>
            <xsd:element ref="se:GreenChannel"/>
            <xsd:element ref="se:BlueChannel"/>
        </xsd:sequence>
        <xsd:element ref="se:GrayChannel"/>
    </xsd:choice>
</xsd:complexType>

<xsd:element name="RedChannel" type="se:SelectedChannelType"/>
<xsd:element name="GreenChannel" type="se:SelectedChannelType"/>
<xsd:element name="BlueChannel" type="se:SelectedChannelType"/>
<xsd:element name="GrayChannel" type="se:SelectedChannelType"/>


<xsd:complexType name="SelectedChannelType">
    <xsd:sequence>
        <xsd:element ref="se:SourceChannelName"/>
        <xsd:element ref="se:ContrastEnhancement" minOccurs="0"/>
    </xsd:sequence>
</xsd:complexType>
<xsd:element name="SourceChannelName" type="xsd:string"/>
```

Either a channel may be selected to display in each of red, green, and blue, or a single channel may be selected to display in grayscale. (The spelling “gray” is used since it seems to be more common on the Web than “grey” by a ratio of about 3:1.) Contrast enhancement may be applied to each channel in isolation. Channels are identified by a system and data-dependent character identifier. Commonly, channels will be labelled as “**1**”, “**2**”, etc. or as defined in clause 9.

The **OverlapBehavior** element tells a system how to behave when multiple raster images in a layer overlap each other, for example with satellite-image scenes.

```
<xsd:element name="OverlapBehavior">
    <xsd:simpleType>
        <xsd:restriction base="xsd:string">
            <xsd:enumeration value="LATEST_ON_TOP"/>
            <xsd:enumeration value="EARLIEST_ON_TOP"/>
            <xsd:enumeration value="AVERAGE"/>
            <xsd:enumeration value="RANDOM"/>
        </xsd:restriction>
    </xsd:simpleType>
</xsd:element>
```

LATEST_ON_TOP and **EARLIEST_ON_TOP** refer to the time the scene was captured. **AVERAGE** means to average multiple scenes together. This can produce blurry results if the source images are not perfectly aligned in their geo-referencing. **RANDOM** means to select an image (or piece thereof) randomly and place it on top. This can produce crisper results than **AVERAGE** potentially more efficiently than **LATEST_ON_TOP** or **EARLIEST_ON_TOP**. The default behaviour is system-dependent.

The **ColorMap** element defines the mapping of palette-type raster colors or fixed-numeric pixel values to colors using an Interpolate or Categorize SE function as defined in subsection 11.6. The **LookUpValue** is in this case set to **Rasterdata**.

```
<xsd:element name="ColorMap" type="se:ColorMapType"/>
<xsd:complexType name="ColorMapType">
    <xsd:choice>
        <xsd:element ref="se:Categorize"/>
        <xsd:element ref="se:Interpolate"/>
    </xsd:choice>
</xsd:complexType>
```

For example, a DEM raster giving elevations in meters above sea level can be translated to a colored image with a **ColorMap** using a **Categorize** function.

The **ContrastEnhancement** element defines contrast enhancement for a channel of a false-color image or for a color image. Its format is:

```
<xsd:element name="ContrastEnhancement" type="se:ContrastEnhancementType"/>
<xsd:complexType name="ContrastEnhancementType">
    <xsd:sequence>
        <xsd:choice minOccurs="0">
            <xsd:element ref="se:Normalize"/>
            <xsd:element ref="se:Histogram"/>
        </xsd:choice>
        <xsd:element ref="se:GammaValue" minOccurs="0"/>
    </xsd:sequence>
</xsd:complexType>

<xsd:element name="Normalize" type="se:NormalizeType"/>
<xsd:complexType name="NormalizeType"/>
<xsd:element name="Histogram" type="se:HistogramType"/>
<xsd:complexType name="HistogramType"/>
<xsd:element name="GammaValue" type="xsd:double"/>
```

In the case of a color image, the relative grayscale brightness of a pixel color is used. "**Normalize**" means to stretch the contrast so that the dimmest color is stretched to black and the brightest color is stretched to white, with all colors in between stretched out linearly. "**Histogram**" means to stretch the contrast based on a histogram of how many colors are at each brightness level on input, with the goal of producing equal number of pixels in the image at each brightness level on output. This has the effect of revealing many subtle ground features. A "**GammaValue**" tells how much to brighten (values greater than **1.0**) or dim (values less than **1.0**) an image. The default **GammaValue** is **1.0** (no change). If none of **Normalize**, **Histogram**, or **GammaValue** are selected in a **ContrastEnhancement**, then no enhancement is performed.

The **ShadedRelief** element selects the application of relief shading (or "hill shading") to an image for a three-dimensional visual effect. It is defined as:

```
<xsd:element name="ShadedRelief" type="se:ShadedReliefType"/>
<xsd:complexType name="ShadedReliefType">
    <xsd:sequence>
        <xsd:element ref="se:BrightnessOnly" minOccurs="0"/>
        <xsd:element ref="se:ReliefFactor" minOccurs="0"/>
    </xsd:sequence>
</xsd:complexType>

<xsd:element name="BrightnessOnly" type="xsd:boolean"/>
<xsd:element name="ReliefFactor" type="xsd:double"/>
```

Exact parameters of the shading are system-dependent (for now). If the **BrightnessOnly** flag is "**0**" or "**false**" (false, default), the shading is applied to the layer being rendered as the current **RasterSymbolizer**. If **BrightnessOnly** is "**1**" or "**true**" (true), the shading is applied to the brightness of the colors in the rendering canvas generated so far by other layers, with the effect of relief-shading these other layers. The default for **BrightnessOnly** is "**0**" (false). The **ReliefFactor** gives the amount of exaggeration to use for the height of the "hills." A value of around **55** (times) gives reasonable results for Earth-based DEMs. The default value is system-dependent.

The **ImageOutline** element specifies that individual source rasters in a multi-raster set (such as a set of satellite-image scenes) should be outlined with either a **LineSymbolizer** or **PolygonSymbolizer**. It is defined as:

```
<xsd:element name="ImageOutline" type="se:ImageOutlineType"/>
<xsd:complexType name="ImageOutlineType">
    <xsd:choice>
        <xsd:element ref="se:LineSymbolizer"/>
        <xsd:element ref="se:PolygonSymbolizer"/>
    </xsd:choice>
 </xsd:complexType>
```

An **Opacity** of **0.0** can be selected for the main raster to avoid rendering the main-raster pixels, or an opacity can be used for a **PolygonSymbolizer Fill** to allow the main-raster data be visible through the fill.

11.5.3 Examples

The following example applies a coloring to elevation (DEM) data (quantities are in meters):

```
<RasterSymbolizer>
    <Opacity>1.0</Opacity>
    <OverlapBehavior>AVERAGE</OverlapBehavior>
    <ColorMap>
        <Categorize>
            <LookupValue>Rasterdata</LookupValue>
            <Value>#00ff00</Value>
            <Threshold>-417</Threshold>
            <Value>#00fa00</Value>
            <Threshold>-333</Threshold>
            <Value>#14f500</Value>
            <Threshold>-250</Threshold>
            <Value>#28f502</Value>
            <Threshold>-167</Threshold>
            <Value>#3cf505</Value>
            <Threshold>-83</Threshold>
            <Value>#50f50a</Value>
            <Threshold>-1</Threshold>
            <Value>#64f014</Value>
            <Threshold>0</Threshold>
            <Value>#7deb32</Value>
            <Threshold>30</Threshold>
            <Value>#78c818</Value>
            <Threshold>105</Threshold>
            <Value>#38840c</Value>
            <Threshold>300</Threshold>
            <Value>#2c4b04</Value>
            <Threshold>400</Threshold>
            <Value>#ffff00</Value>
            <Threshold>700</Threshold>
            <Value>#dcdc00</Value>
            <Threshold>1200</Threshold>
            <Value>#b47800</Value>
            <Threshold>1400</Threshold>
            <Value>#c85000</Value>
            <Threshold>1600</Threshold>
            <Value>#be4100</Value>
            <Threshold>2000</Threshold>
            <Value>#963000</Value>
            <Threshold>3000</Threshold>
            <Value>#3c0200</Value>
            <Threshold>5000</Threshold>
            <Value>#ffffff</Value>
            <Threshold>13000</Threshold>
            <Value>#ffffff</Value>
        </Categorize>
    </ColorMap>
    <ShadedRelief/>
</RasterSymbolizer>
```

Here is a rather artificial multi-band raster **Symbolizer**:

```
<RasterSymbolizer>
    <Opacity>1.0</Opacity>
    <ChannelSelection>
        <RedChannel>
            <SourceChannelName>1</SourceChannelName>
            <ContrastEnhancement>
                <Histogram/>
            </ContrastEnhancement>
        </RedChannel>
        <GreenChannel>
            <SourceChannelName>2</SourceChannelName>
            <ContrastEnhancement>
                <GammaValue>2.5</GammaValue>
            </ContrastEnhancement>
        </GreenChannel>
        <BlueChannel>
            <SourceChannelName>3</SourceChannelName>
            <ContrastEnhancement>
                <Normalize/>
            </ContrastEnhancement>
        </BlueChannel>
    </ChannelSelection>
    <OverlapBehavior>LATEST_ON_TOP</OverlapBehavior>
    <ColorMap>
        <Interpolate>
            <LookupValue>Rasterdata</LookupValue>
            <InterpolationPoint>
                <Data>0</Data>
                <Value>#000000</Value>
            </InterpolationPoint>
            <InterpolationPoint>
                <Data>255</Data>
                <Value>#ffffff</Value>
            </InterpolationPoint>
        </Interpolate>
    </ColorMap>
    <ContrastEnhancement>
        <GammaValue>1.0</GammaValue>
    </ContrastEnhancement>
</RasterSymbolizer>
```

11.6 Symbology Encoding Functions

SE extends the concept of **ogc:expression**s inherited from Filter Encoding to adequately support the needs of symbolization in transforming and editing data. It does so by introducing a couple of functions (**se:Function**s) which are substitutable for **ogc:expression**s.

There are two general groups of functions for usage in SE. The first group is used to transform raw values into “symbolizable” quantities. This especially comprises the processes of categorization, recoding, and interpolation. This group of functions is especially useful in all places using **SvgParameters**, making them dynamically related to data values. The second group defines means for formatting data items like numbers, strings, and dates. These functions are especially helpful to set up the **Label** element of **TextSymbolizers.**

Function is an abstract expression and is defined as follows:

```
<xsd:element name="Function" type="se:FunctionType" abstract="true"
         substitutionGroup="ogc:expression"/>
<xsd:complexType name="FunctionType" abstract="true">
    <xsd:complexContent>
        <xsd:extension base="ogc:FunctionType">
            <xsd:attribute name="fallbackValue" type="xsd:string" use="required"/>
        </xsd:extension>
    </xsd:complexContent>
</xsd:complexType>
```

The value of the **fallbackValue** attribute is used as a default value, if the SE implementation does not support the function. If the implementation supports the function, then the result value is determined by executing the function.

A conformant SE implementation need not implement any of the described functions, however, if it *does* implement any of the functions specified, their working **shall** be as described in this specification. The functions have to exhibit the described behavior only if used in the context of **Symbolizer**s. It is not necessary (though allowed) for an implementation to make the functions also available in the context of filter expressions elsewhere.

11.6.1 Numeric formatting function

One of the most needed is a function for formatting numbers to make them human readable. You need such a function whenever a **TextSymbolizer** is employed to output a numeric property value.

It is defined as follows:

```
<xsd:element name="FormatNumber" type="se:FormatNumberType" substitutionGroup="se:Function"/>
<xsd:complexType name="FormatNumberType">
    <xsd:complexContent>
        <xsd:extension base="se:FunctionType">
            <xsd:sequence>
                <xsd:element ref="se:NumericValue"/>
                <xsd:element ref="se:Pattern"/>
                <xsd:element ref="se:NegativePattern" minOccurs="0"/>
            </xsd:sequence>
            <xsd:attribute name="decimalPoint" type="xsd:string" use="optional" default="."/>
            <xsd:attribute name="groupingSeparator" type="xsd:string" use="optional" default=","/>
        </xsd:extension>
    </xsd:complexContent>
</xsd:complexType>


<xsd:element name="NumericValue" type="se:ParameterValueType"/>
<xsd:element name="Pattern" type="xsd:string"/>
<xsd:element name="NegativePattern" type="xsd:string"/>
```

The **Pattern** is build-up of characters as follows

#	digit, which is omitted if irrelevant (leading or trailing zero)
0	digit
.	decimal point
,	grouping separator
E	exponent indicator
-	minus sign
'	apostrophe (for quoting)

Due to required distinctions in the notation of numbers, two of the pattern characters can be defined as attributes of the **FormatNumber** element:

- decimalPoint="."
- groupingSeperator=","

If there is no **NegativePattern**, "-" is prefixed to the **Pattern**. The first argument is converted to a numeric value if necessary. The semantics shall be as described as for XSLT and the Java DecimalFormat class.

11.6.2 Date formatting function

This function is used for several date types. The argument of the function can consist of one of the following ISO 8601 XML schema types:

```
dateTime
time
date
gYearMonth
gMonthDay
gDay
gMonth
gYear
```

or

```
gml:TimeInstant
```

```
<xsd:element name="FormatDate" type="se:FormatDateType" substitutionGroup="se:Function"/>
<xsd:complexType name="FormatDateType">
    <xsd:complexContent>
        <xsd:extension base="se:FunctionType">
            <xsd:sequence>
                <xsd:element ref="se:DateValue"/>
                <xsd:element ref="se:Pattern"/>
            </xsd:sequence>
        </xsd:extension>
    </xsd:complexContent>
</xsd:complexType>
```

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

Standard output is as defined in ISO 8601: **YYYY-MM-DDThh:mm:ss**

The **Pattern** is build-up of characters as follows

YYYY	Four digit year
YY	Two digit year (without century and millennium)
MM	Two digit month
M	Month, leading zero omitted
MMM	Month displayed by three letter acronym (“FEB”), ISO 639 two-letter language codes as defined by ISO 639 can be appended to create language-dependent variants (**MMMde** would yield “DEZ” instead of “DEC”)
MMMMM	for display of full month (“February”). The two-letter language code can be appended (**MMMMMde** would result in ‘Februar’).
DD	Two digit day
D	Day, leading zero omitted
hh	hour, **h** is used to omit a leading zero
mm	minute, **m** is used to omit a leading zero
ss	second, **s** is used to omit a leading zero
.	point, will appear literally in the result
/	slash, literally
:	colon. literally
-	minus, literally
****	backslash is employed to quote any character, which is to appear literally in the result.
a	am/pm marker
z	**z**: time zone (if present e.g. Pacific Standard Time; PST; GMT-08:00)

The following example:

```
<FormatDate>
  <DateValue><ogc:PropertyName>aDate</ogc:PropertyName></DateValue>
  <Pattern>DD.MM.YYYY</Pattern>
</FormatDate>
```

would lead to a typical German representation of date (e.g. 20.02.2006)

11.6.3 String formatting functions

SE supports a repertoire of string manipulation functions. The following is a collection of the most important functions for the purpose of symbolization of text.

```
<xsd:element name="Substring" type="se:SubstringType" substitutionGroup="se:Function"/>
<xsd:complexType name="SubstringType">
    <xsd:complexContent>
        <xsd:extension base="se:FunctionType">
            <xsd:sequence>
                <xsd:element ref="se:StringValue"/>
                <xsd:element ref="se:Position" minOccurs="0"/>
                <xsd:element ref="se:Length" minOccurs="0"/>
            </xsd:sequence>
        </xsd:extension>
    </xsd:complexContent>
</xsd:complexType>

<xsd:element name="StringValue" type="se:ParameterValueType"/>
<xsd:element name="Position" type="se:ParameterValueType"/>
<xsd:element name="Length" type="se:ParameterValueType"/>
```

Returns the substring at position **Position** (counting from 1) with length **Length**.

The first argument **StringValue** is converted to a string value before applying the substring operation. If **Position** is not specified it is assumed as 1. The default value for **Length** is the remaining length starting at **Position**.

The function shall react friendly on invalid **Position** and **Length** contents. **Position**s and **Length**s less or equal 0 shall yield the empty string. If the actual string length is less the defined substring the existing part of the substring shall be returned.

```
<xsd:element name="Concatenate" type="se:ConcatenateType" substitutionGroup="se:Function"/>
<xsd:complexType name="ConcatenateType">
    <xsd:complexContent>
        <xsd:extension base="se:FunctionType">
            <xsd:sequence>
                <xsd:element ref="se:StringValue" maxOccurs="unbounded"/>
            </xsd:sequence>
        </xsd:extension>
    </xsd:complexContent>
</xsd:complexType>
```

The function concatenates strings. It is used to create concatenated strings as arguments of functions.

```
<xsd:element name="ChangeCase" type="se:ChangeCaseType" substitutionGroup="se:Function"/>
<xsd:complexType name="ChangeCaseType">
    <xsd:complexContent>
        <xsd:extension base="se:FunctionType">
            <xsd:sequence>
                <xsd:element ref="se:StringValue"/>
            </xsd:sequence>
            <xsd:attribute name="direction" type="se:directionType"/>
        </xsd:extension>
    </xsd:complexContent>
</xsd:complexType>
<xsd:simpleType name="directionType">
    <xsd:restriction base="xsd:token">
        <xsd:enumeration value="toUpper"/>
        <xsd:enumeration value="toLower"/>
    </xsd:restriction>
</xsd:simpleType>
```

The function changes the case of the **StringValue** as indicated by the attribute **direction**. Possible values of the latter are "toUpper" and "toLower", where the former is the default value.

```
<xsd:element name="Trim" type="se:TrimType" substitutionGroup="se:Function"/>
<xsd:complexType name="TrimType">
    <xsd:complexContent>
        <xsd:extension base="se:FunctionType">
            <xsd:sequence>
                <xsd:element ref="se:StringValue"/>
            </xsd:sequence>
            <xsd:attribute name="stripOffPosition" type="se:stripOffPositionType"/>
            <xsd:attribute name="stripOffChar" type="xsd:string"/>
        </xsd:extension>
    </xsd:complexContent>
</xsd:complexType>
<xsd:simpleType name="stripOffPositionType">
    <xsd:restriction base="xsd:token">
        <xsd:enumeration value="leading"/>
        <xsd:enumeration value="trailing"/>
        <xsd:enumeration value="both"/>
    </xsd:restriction>
</xsd:simpleType>
```

The function strips off "leading", "trailing", or "both" chars from a string value. Attributes control the mode of stripping and the character stripped. Defaults are "leading" and blank.

```
<xsd:element name="StringPosition" type="se:StringPositionType" substitutionGroup="se:Function"/>
<xsd:complexType name="StringPositionType">
    <xsd:complexContent>
        <xsd:extension base="se:FunctionType">
            <xsd:sequence>
                <xsd:element ref="se:LookupString"/>
                <xsd:element ref="se:StringValue"/>
            </xsd:sequence>
            <xsd:attribute name="searchDirection" type="se:searchDirectionType"/>
        </xsd:extension>
    </xsd:complexContent>
</xsd:complexType>
<xsd:element name="LookupString" type="se:ParameterValueType"/>
<xsd:simpleType name="searchDirectionType">
    <xsd:restriction base="xsd:token">
        <xsd:enumeration value="frontToBack"/>
        <xsd:enumeration value="backToFront"/>
    </xsd:restriction>
</xsd:simpleType>
```

This function returns the position of the first occurrence (counting from 1) of the **LookupString** in **StringValue**. Zero is returned in case of search failure. The direction of search is determined by the attribute **searchdirection**, which can take values "frontToBack" and "backToFront", where the former is the default.

```
<xsd:element name="StringLength" type="se:StringLengthType" substitutionGroup="se:Function"/>
<xsd:complexType name="StringLengthType">
    <xsd:complexContent>
        <xsd:extension base="se:FunctionType">
            <xsd:sequence>
                <xsd:element ref="se:StringValue"/>
            </xsd:sequence>
        </xsd:extension>
    </xsd:complexContent>
 </xsd:complexType>
```

The function returns the number of characters in a string. The argument is converted to a string before computing its length.

11.6.4 Transformation functions

This specification defines three pre-defined functions for transforming raw data:

1. Categorization: The transformation of continuous values to distinct values. This is for example needed to generate choropleth maps from continuous attributes. Another example would be the stepwise selection of different text heights or line widths in dependence from such an attribute.

2. Interpolation: Transformation of continuous values by a function defined on a number of nodes. This is used to adjust the value distribution of an attribute to the desired distribution of a continuous symbolization control variable (like size, width, color, etc).

3. Recoding: Transformation of discrete values to any other values. This is needed when integers have to be translated into text or, reversely, text contents into other texts or numeric values or colors.

Categorization is defined using the **Categorize** element:

```
<xsd:element name="Categorize" type="se:CategorizeType" substitutionGroup="se:Function"/>
<xsd:complexType name="CategorizeType">
    <xsd:complexContent>
        <xsd:extension base="se:FunctionType">
            <xsd:sequence>
                <xsd:element ref="se:LookupValue"/>
                <xsd:element ref="se:Value"/>
                <xsd:sequence minOccurs="0" maxOccurs="unbounded">
                    <xsd:element ref="se:Threshold"/>
                    <xsd:element ref="se:Value"/>
                </xsd:sequence>
            </xsd:sequence>
            <xsd:attribute name="threshholdsBelongTo" type="se:ThreshholdsBelongToType" use="optional"/>
        </xsd:extension>
    </xsd:complexContent>
</xsd:complexType>

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

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

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

<xsd:simpleType name="ThreshholdsBelongToType">
    <xsd:restriction base="xsd:token">
        <xsd:enumeration value="succeeding"/>
        <xsd:enumeration value="preceding"/>
    </xsd:restriction>
</xsd:simpleType>
```

The **Thresholds** have to be specified in ascending order and (like the **LookupValue**) have to be of a uniform and orderable type. The value of the function is determined by looking up into which interval between two thresholds the **LookupValue** falls. The first interval ranges from *-Infinity* to the first given threshold and the last one accordingly from the last threshold to *+Infinity*.

In case the **Categorize (**or **Interpolate**) function is used inside a **RasterSymbolizer as a** ColorMap, the **LookupValue** is set to the fixed value “Rasterdata”.

The **Values** can be of any type, dependent on which symbolization context the function is employed. Color values (like #00ffff) or numeric values are typical.

Whether the **Threshold** values themselves belong to the preceding or the succeeding interval can be controlled by the attribute thresholdsBelongTo= with the possible values "preceding" and "succeeding" the latter being the default.

In the following example using the **Categorize** function the stroke width of a line symbol representing a road segment depends on the number of average vehiles passing per hour (4999 or less: 1 pixel; 5000..14999: 2 pixel; 15000..39999: 3 pixel; 40000..74999: 4 pixel; 75000+: 5 pixel).

```
<SvgParameter name="stroke-width">
    <Categorize fallbackValue="1">
        <LookupValue>
            <ogc:PropertyName>vehiclesPerHour</ogc:PropertyName>
        </LookupValue>
        <Value>1</Value>
        <Threshold>5000</Threshold>
        <Value>2</Value>
        <Threshold>15000</Threshold>
        <Value>3</Value>
        <Threshold>40000</Threshold>
        <Value>4</Value>
        <Threshold>75000</Threshold>
        <Value>5</Value>
    </Categorize>
</SvgParameter>
```

The Interpolate function is defined in the following:

```
    <xsd:element name="Interpolate" type="se:InterpolateType" substitutionGroup="se:Function"/>
    <xsd:complexType name="InterpolateType">
        <xsd:complexContent>
            <xsd:extension base="se:FunctionType">
                <xsd:sequence>
                    <xsd:element ref="se:LookupValue"/>
                    <xsd:element ref="se:InterpolationPoint" maxOccurs="unbounded"/>
                </xsd:sequence>
                <xsd:attribute name="mode" type="se:ModeType"/>
                <xsd:attribute name="method" type="se:MethodType"/>
            </xsd:extension>
        </xsd:complexContent>
    </xsd:complexType>

    <xsd:element name="InterpolationPoint" type="se:InterpolationPointType" substitutionGroup="ogc:expression"/>
    <xsd:complexType name="InterpolationPointType">
        <xsd:complexContent>
            <xsd:extension base="ogc:ExpressionType">
                <xsd:sequence>
                    <xsd:element ref="se:Data"/>
                    <xsd:element ref="se:Value"/>
                </xsd:sequence>
            </xsd:extension>
        </xsd:complexContent>
    </xsd:complexType>

    <xsd:element name="Data" type="xsd:double"/>

    <xsd:simpleType name="ModeType">
        <xsd:restriction base="xsd:token">
            <xsd:enumeration value="linear"/>
            <xsd:enumeration value="cosine"/>
            <xsd:enumeration value="cubic"/>
        </xsd:restriction>
    </xsd:simpleType>
    <xsd:simpleType name="MethodType">
        <xsd:restriction base="xsd:token">
            <xsd:enumeration value="numeric"/>
            <xsd:enumeration value="color"/>
        </xsd:restriction>
```

```
</xsd:simpleType>
```

The **InterpolationPoint**s have to be specified in ascending order of **Data**. They define a graph of points. **LookupValues** less than the **Data** value of the first **InterpolationPoint** are mapped to its corresponding **Value**. Accordingly, **LookupValues** greater than the **Data** value of the last **InterpolationPoint** are mapped to the **Value** of this one. **LookupValue**s between two **InterpolationPoint**s are interpolated between the corresponding **Value**s.

Only numeric quantities are allowed for **LookupValue** and **Data**. **Value**s are usually numeric as well. The interpolation of color-values requires the attribute mode="color" at the **Interpolate** element.

Recoding is defined by the **Recode** element:

```
<xsd:element name="Recode" type="se:RecodeType" substitutionGroup="se:Function"/>
<xsd:complexType name="RecodeType">
    <xsd:complexContent>
        <xsd:extension base="se:FunctionType">
            <xsd:sequence>
                <xsd:element ref="se:LookupValue"/>
                <xsd:element ref="se:MapItem" maxOccurs="unbounded"/>
            </xsd:sequence>
        </xsd:extension>
    </xsd:complexContent>
</xsd:complexType>

<xsd:element name="MapItem" type="se:MapItemType" substitutionGroup="ogc:expression"/>
<xsd:complexType name="MapItemType">
    <xsd:complexContent>
        <xsd:extension base="ogc:ExpressionType">
            <xsd:sequence>
                <xsd:element ref="se:Data"/>
                <xsd:element ref="se:Value"/>
            </xsd:sequence>
        </xsd:extension>
    </xsd:complexContent>
</xsd:complexType>
```

This function recodes values from a property or expression into corresponding values of arbitrary type. The comparisons are performed checking for identical values.

Annex A
(normative)

Abstract test suite

A.1 General

Symbology Encoding is independent of a specific service specification and can be implemented in a number of ways. Compliance to the specification can therefore be defined on a minimal syntactic level regarding XML validity and on functionality level regarding the range of SE elements an implementation supports. Conformance verification of the output of a system implementing SE requires in most cases visual interpretation. Therefore the possibilities for automatic testing of SE are very limited.

Symbology Encoding encompasses Symbolizers pertaining to features and coverages. A compliant implementation of SE has to support at least basic Feature Style Functionality (Class A and B) or or Basic Coverage Style functionality (Class A and C).

A full SE implementation would have to support all elements of either FeatureStyle or CoverageStyle with the exception of SE functions. Implementations of this category would be typed “Full Symbology Encoding (Feature Type style) implementation (Class A and D) or “Full Symbology Encoding (Coverage style) implementation” (Class A and E). An implementation fulfilling classes A, D and E would be would be typed a “Full Symbology Encoding Implementation”.

A.2 Conformance Class A. Basic Schema conformance

a) Test purpose: To verify that an SE document is valid Symbology Encoding

b) Test method: Validate XML instance document against SE schemas

c) Reference: Annex B

d) Test type: Basic Test

A 3 Conformance Class B. Basic Feature Styling Functionality

a) Test purpose: To verify that an SE implementation supports basic functionality for feature styling

b) Test method: Generate graphic output using FeatureTypeStyle elements including mandatory sub-elements

c) Reference: Clause 8

d) Test type: Capability Test

A 3 Conformance Class C. Basic Coverage Styling Functionality

a) Test purpose: To verify that an SE implementation supports basic functionality for coverage styling

b) Test method: Generate graphic output using CoverageStyle elements including mandatory sub-elements

c) Reference: Clause 9

d) Test type: Capability Test

A4 Conformance Class D. Full Feature Styling Functionality

a) Test purpose: To verify that an SE implementation supports full functionality for feature styling

b) Test method: Generate graphic output using FeatureTypeStyle elements including all sub-elements

c) Reference: Clause 8

d) Test type: Capability Test

A 5 Conformance Class E. Full Coverage Styling Functionality

a) Test purpose: To verify that an SE implementation supports full functionality for coverage styling

b) Test method: Generate graphic output using CoverageStyle elements including all sub-elements

c) Reference: Clause 9

d) Test type: Capability Test

Annex B
(normative)

XML schemas

In addition to this document, this specification includes several normative XML Schema files. These are posted online at the URL http://schemas.opengeospatial.net/se where a lower level directory is used for this Version 1.1.0. These XML Schema files are also bundled in a zip file with the present document. In the event of a discrepancy between the bundled and online versions of the XML Schema files, the online files shall be considered authoritative.

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

common.xsd

Symbolizer.xsd

FeatureStyle.xsd

Although FeatureStyle.xsd contains elements pertaining to FeatureTypes and Coverages, as Features are a superclass of both these elements, this file name was chosen.All these XML Schemas contain documentation of the meaning of each element and attribute, and this documentation shall be considered normative as specified in Subclause 11.6.3 of [OGC 05-008].

Annex C
(informative)

Example XML documents

C.1 Introduction

This annex provides more example XML documents than given in the body of this document.

C.2 FeatureTypeStyle

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<FeatureTypeStyle version="1.1.0" xsi:schemaLocation="http://www.opengis.net/se FeatureStyle.xsd"
xmlns="http://www.opengis.net/se" xmlns:ogc="http://www.opengis.net/ogc"
xmlns:xlink="http://www.w3.org/1999/xlink" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns:oceansea="http://www.myurl.net/oceansea">
    <FeatureTypeName>oceansea:Foundation</FeatureTypeName>
    <Rule>
        <Name>main</Name>
        <PolygonSymbolizer uom="http://www.opengeospatial.org/sld/units/pixel">
            <Fill>
                <SvgParameter name="fill">#96C3F5</SvgParameter>
            </Fill>
        </PolygonSymbolizer>
    </Rule>
</FeatureTypeStyle>
```

C.3 CoverageStyle

```
<?xml version="1.0" encoding="UTF-8"?>
<CoverageStyle version="1.1.0" xsi:schemaLocation="http://www.opengis.net/se FeatureStyle.xsd"
xmlns="http://www.opengis.net/se" xmlns:ogc="http://www.opengis.net/ogc"
xmlns:xlink="http://www.w3.org/1999/xlink" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
    <Rule>
        <Name>ChannelSelection</Name>
        <Description>
            <Title>Gray channel mapping</Title>
        </Description>
        <RasterSymbolizer>
            <ChannelSelection>
                <GrayChannel>
                    <SourceChannelName>Band.band1</SourceChannelName>
                </GrayChannel>
            </ChannelSelection>
            <ContrastEnhancement>
                <Normalize/>
            </ContrastEnhancement>
        </RasterSymbolizer>
    </Rule>
</CoverageStyle>
```

C.4 LineSymbolizer

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<LineSymbolizer version="1.1.0" xsi:schemaLocation="http://www.opengis.net/se Symbolizer.xsd"
xmlns="http://www.opengis.net/se" xmlns:ogc="http://www.opengis.net/ogc"
xmlns:xlink="http://www.w3.org/1999/xlink" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
uom="http://www.opengeospatial.org/se/units/metre">
    <Name>MyLineSymbolizer</Name>
    <Description>
        <Title>Example Symbol</Title>
        <Abstract>This is just a simple example of a line symbolizer.</Abstract>
    </Description>
    <Stroke>
        <SvgParameter name="stroke">#0000ff</SvgParameter>
        <SvgParameter name="stroke-width">2</SvgParameter>
    </Stroke>
</LineSymbolizer>
```

C.5 PolygonSymbolizer

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<PolygonSymbolizer version="1.1.0" xsi:schemaLocation="http://www.opengis.net/se Symbolizer.xsd"
xmlns="http://www.opengis.net/se" xmlns:ogc="http://www.opengis.net/ogc"
xmlns:xlink="http://www.w3.org/1999/xlink" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
uom="http://www.opengeospatial.org/se/units/pixel">
    <Name>MyPolygonSymbolizer</Name>
    <Description>
        <Title>Example PolygonSymbolizer</Title>
        <Abstract>This is just a simple example of a polygon symbolizer.</Abstract>
    </Description>
    <Fill>
        <SvgParameter name="fill">#aaaaff</SvgParameter>
    </Fill>
    <Stroke>
        <SvgParameter name="stroke">#0000aa</SvgParameter>
    </Stroke>
</PolygonSymbolizer>
```

C.6 PointSymbolizer 1

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<PointSymbolizer version="1.1.0" xsi:schemaLocation="http://www.opengis.net/se Symbolizer.xsd"
xmlns="http://www.opengis.net/se" xmlns:ogc="http://www.opengis.net/ogc"
xmlns:xlink="http://www.w3.org/1999/xlink" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
uom="http://www.opengeospatial.org/se/units/metre">
    <Name>MyPointSymbolizer</Name>
    <Description>
        <Title>Example Pointsymbolizer</Title>
        <Abstract>This is just a simple example of a point symbolizer.</Abstract>
    </Description>
    <Graphic>
        <Mark>
            <WellKnownName>star</WellKnownName>
            <Fill>
                <SvgParameter name="fill">#ff0000</SvgParameter>
            </Fill>
        </Mark>
        <Size>8.0</Size>
    </Graphic>
</PointSymbolizer>
```

C.7 PointSymbolizer 2

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<PointSymbolizer version="1.1.0" xsi:schemaLocation="http://www.opengis.net/se Symbolizer.xsd"
xmlns="http://www.opengis.net/se" xmlns:ogc="http://www.opengis.net/ogc"
xmlns:xlink="http://www.w3.org/1999/xlink" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
uom="http://www.opengeospatial.org/se/units/pixel">
    <Name>MyPointSymbolizer</Name>
    <Description>
        <Title>Example Pointsymbolizer</Title>
        <Abstract>This is just a simple example of a point symbolizer.</Abstract>
    </Description>
    <Graphic>
        <ExternalGraphic>
            <OnlineResource xlink:type="simple" xlink:href="http://www.vendor.com/geosym/2267.svg"/>
            <Format>image/svg+xml</Format>
        </ExternalGraphic>
        <ExternalGraphic>
            <OnlineResource xlink:type="simple" xlink:href="http://www.vendor.com/geosym/2267.png"/>
            <Format>image/png</Format>
        </ExternalGraphic>
        <Mark/>
        <Size>15.0</Size>
    </Graphic>
</PointSymbolizer>
```

C.8 TextSymbolizer

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<TextSymbolizer version="1.1.0" xsi:schemaLocation="http://www.opengis.net/se Symbolizer.xsd"
xmlns="http://www.opengis.net/se" xmlns:ogc="http://www.opengis.net/ogc"
xmlns:xlink="http://www.w3.org/1999/xlink" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
uom="http://www.opengeospatial.org/se/units/pixel">
    <Name>MyTextSymbolizer</Name>
    <Description>
        <Title>Example TextSymbolizer</Title>
        <Abstract>This is just an example of a text symbolizer using the FormatNumber function.</Abstract>
    </Description>
    <Geometry>
        <ogc:PropertyName>locatedAt</ogc:PropertyName>
    </Geometry>
    <Label>
        <ogc:PropertyName>hospitalName</ogc:PropertyName> (
            <FormatNumber fallbackValue="">
            <NumericValue>
                <ogc:PropertyName>numberOfBeds</ogc:PropertyName>
            </NumericValue>
            <Pattern>#####</Pattern>
        </FormatNumber>)
    </Label>
    <Font>
        <SvgParameter name="font-family">Arial</SvgParameter>
        <SvgParameter name="font-family">Sans-Serif</SvgParameter>
        <SvgParameter name="font-style">italic</SvgParameter>
        <SvgParameter name="font-size">10</SvgParameter>
    </Font>
    <Halo/>
    <Fill>
        <SvgParameter name="fill">#000000</SvgParameter>
    </Fill>
</TextSymbolizer>
```

C.9 RasterSymbolizer 1

```
<?xml version="1.0" encoding="UTF-8"?>
<RasterSymbolizer version="1.1.0" xsi:schemaLocation="http://www.opengis.net/se Symbolizer.xsd"
xmlns="http://www.opengis.net/se" xmlns:ogc="http://www.opengis.net/ogc"
xmlns:xlink="http://www.w3.org/1999/xlink" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
    <Opacity>1.0</Opacity>
    <OverlapBehavior>AVERAGE</OverlapBehavior>
    <ColorMap>
        <Categorize fallbackValue="#78c818">
            <LookupValue>Rasterdata</LookupValue>
            <Value>#00ff00</Value>
            <Threshold>-417</Threshold>
            <Value>#00fa00</Value>
            <Threshold>-333</Threshold>
            <Value>#14f500</Value>
            <Threshold>-250</Threshold>
            <Value>#28f502</Value>
            <Threshold>-167</Threshold>
            <Value>#3cf505</Value>
            <Threshold>-83</Threshold>
            <Value>#50f50a</Value>
            <Threshold>-1</Threshold>
            <Value>#64f014</Value>
            <Threshold>0</Threshold>
            <Value>#7deb32</Value>
            <Threshold>30</Threshold>
            <Value>#78c818</Value>
            <Threshold>105</Threshold>
            <Value>#38840c</Value>
            <Threshold>300</Threshold>
            <Value>#2c4b04</Value>
            <Threshold>400</Threshold>
            <Value>#ffff00</Value>
            <Threshold>700</Threshold>
            <Value>#dcdc00</Value>
            <Threshold>1200</Threshold>
            <Value>#b47800</Value>
            <Threshold>1400</Threshold>
            <Value>#c85000</Value>
            <Threshold>1600</Threshold>
            <Value>#be4100</Value>
            <Threshold>2000</Threshold>
            <Value>#963000</Value>
            <Threshold>3000</Threshold>
            <Value>#3c0200</Value>
            <Threshold>5000</Threshold>
            <Value>#ffffff</Value>
            <Threshold>13000</Threshold>
            <Value>#ffffff</Value>
        </Categorize>
    </ColorMap>
    <ShadedRelief/>
</RasterSymbolizer>
```

C.10 RasterSymbolizer 2

```
<?xml version="1.0" encoding="UTF-8"?>
<RasterSymbolizer version="1.1.0" xsi:schemaLocation="http://www.opengis.net/se Symbolizer.xsd"
xmlns="http://www.opengis.net/se" xmlns:ogc="http://www.opengis.net/ogc"
xmlns:xlink="http://www.w3.org/1999/xlink" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
    <Opacity>1.0</Opacity>
    <ChannelSelection>
        <RedChannel>
            <SourceChannelName>1</SourceChannelName>
            <ContrastEnhancement>
                <Histogram/>
            </ContrastEnhancement>
        </RedChannel>
        <GreenChannel>
            <SourceChannelName>2</SourceChannelName>
            <ContrastEnhancement>
                <GammaValue>2.5</GammaValue>
            </ContrastEnhancement>
        </GreenChannel>
        <BlueChannel>
            <SourceChannelName>3</SourceChannelName>
            <ContrastEnhancement>
                <Normalize/>
            </ContrastEnhancement>
        </BlueChannel>
    </ChannelSelection>
    <OverlapBehavior>LATEST_ON_TOP</OverlapBehavior>
    <ColorMap>
        <Interpolate fallbackValue="#dddddd">
            <LookupValue>Rasterdata</LookupValue>
            <InterpolationPoint>
                <Data>0</Data>
                <Value>#000000</Value>
            </InterpolationPoint>
            <InterpolationPoint>
                <Data>255</Data>
                <Value>#ffffff</Value>
            </InterpolationPoint>
        </Interpolate>
    </ColorMap>
    <ContrastEnhancement>
        <GammaValue>1.0</GammaValue>
    </ContrastEnhancement>
</RasterSymbolizer>
```