Open Geospatial Consortium Inc.

Date: 2005-10-19

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

Version: 1.1.0 (draft)

Category: OpenGIS® Discussion Paper

Editor: Dr. Markus Müller

Symbology Encoding Implementation Specification

Copyright

Document type:	OpenGIS® Discussion Paper
Document stage:	Approved
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 document 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	Phone	Email
Larry Bouzane	Compusult Ltd.		larry@compusult.nf.ca
Dr. Craig Bruce	CubeWerx Inc.	+1 613-565-1344	csbruce@cubewerx.com
Ivan Cheung	ESRI		icheung@esri.com
Adrian Cuthbert	m-spatial		adrian.cuthbert@m-spatial.com
Reinhard Erstling	interactive instruments GmbH		erstling@interactive-instruments.de
Ron Lake	Galdos Systems Inc.		rlake@galdosinc.com
Seb Lessware	Laser-Scan Ltd.		sebl@lsl.co.uk
Marwa Mabrouk	ESRI		mmabrouk@esri.com
James Macgill	Penn State	+1 814-865-5642	jmacgill@psu.edu
Dimitri Monie	Ionic Software s.a.		dimitri.monie@ionicsoft.com
Dr. Markus Müller	lat/lon GmbH	+49 228 184960	mueller@lat-lon.de
Dr. Andreas Poth	lat/lon GmbH	+49 228 184960	poth@lat-lon.de
Raj Singh	Syncline		rs@syncline.com
Dan Specht	US Army ERDC		specht@tec.army.mil
John Vincent	Intergraph Corp.		jtvincen@intergraph.com
Peter Vretanos	CubeWerx Inc.	+1 416-701-1985	pvretano@cubewerx.com

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	Modified for submission and consideration as RFC Proposal for SLD	Implementation Specification

		Bruce Bill Lalonde	Implementation Specification	
2002-04-24	02-013r1	Bill Lalonde Greg Buehler	Minor formatting changes	Formating for Public Comment
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		Markus Müller	All	Split SLD specification in SLD profile for WMS and Symbology Encoding (this document)

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 OGC™ 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, October 2003, OGC document OGC 03-065r6, <https://portal.opengeospatial.org/files/?artifact_id=3837>.

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/TBD. 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 application 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 **FeatureStyle** and **CoverageStyle** root elements. These element include all information for styling the data such as Filter and different kinds of Symbols.

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 **FeatureStyle** or a **Coverage Style**. There are a number of elements used throughout Filter 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="OnlineResource">
        <xsd:complexType>
            <xsd:attributeGroup ref="xlink:simpleLink"/>
        </xsd:complexType>
</xsd:element>
```

This element is used with most “objects” defined by SE and SLD 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. A **wellKnownId** attribute is also available to allow a the fetching of an external web resource to be obviated if the content is already known. A community can define a set of well-known identifiers for common document instances.

8 Feature styles

The **FeatureStyle** defines the styling that is to be applied to a single feature type of a layer. This element may also be externally re-used outside of the scope of WMSes and layers. It is defined as follows:

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

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

A **FeatureStyle** 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 feature 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.

The **SemanticTypeIdentifier** is experimental and is intended to be used to identify what the feature style 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 **FeatureStyle** contains one or more **Rule** elements that allow conditional rendering. Rules are discussed in Clause 10.

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">
        <xsd:annotation>
            <xsd:documentation>
      A CoverageStyle contains styling information specific to one
      Coverage offering.  This is the SLD level that separates the
      'layer' handling from the 'coverage' handling.
    </xsd:documentation>
        </xsd:annotation>
        <xsd:complexType>
            <xsd:choice>
                <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:element ref="se:Rule" maxOccurs="unbounded"/>
                </xsd:sequence>
                <xsd:element ref="se:OnlineResource"/>
            </xsd:choice>
            <xsd:attribute name="version" type="se:VersionType"/>
        </xsd:complexType>
    </xsd:element>
```

Similar to **FeatureStyle**, 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 (in-layer) and that coverage must match the syntax and semantics of all coverage-property references inside of the **CoverageStyle**. Note that there is no restriction against a single UserStyle from including multiple **CoverageStyles** that reference the sa**me CoverageName.** However, this does not create an exception in the rendering semantics, since a map styler is expected to process all **CoverageStyles** in the order that they appear.

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>
        <RasterSymbol>
            <ChannelSelection>
                <GrayChannel>
                    <SourceChannelName>Band.band1</SourceChannelName>
                </GrayChannel>
            </ChannelSelection>
            <ContrastEnhancement>
                <Normalize/>
            </ContrastEnhancement>
        </RasterSymbol>
    </Rule>
</CoverageStyle>
```

10 Rules

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

```
<xsd:element name="Rule">
    <xsd:complexType>
        <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:Symbol" maxOccurs="unbounded"/>
        </xsd:sequence>
    </xsd:complexType>
</xsd:element>
```

Only a single feature type can be in-context inside of a **Rule**, since the **FeatureStyle** element allows only a single feature type “through.” The elements of a **Rule** are described in the following clauses, except for the Symbol elements, which are described in Clause 11.

The ordering to use for the **Rule**s inside of a **FeatureStyle** is a bit of a tricky issue. On the one hand, the rules should most naturally be in the order of priority, with the most “important” rules coming first (e.g., with Highways coming before Minor Roads). On the other hand, the “painters model” is generally used for ordering in SE with the first item in a list being the first item plotted and hence being on the “bottom,” which is counter-intuitive for rule priorities, since Highways should generally be plotted over top of Minor Roads, for example. (It would also be awkward, though not invalid, for “**ElseFilter**” **Rule**s to come before regular-condition **Rule**s). Therefore, **Rule**s shall be placed in the order of “priority” in a **FeatureStyle**, and that stylers should attempt to render the higher-

priority rules over top of the lower-priority rules (exactly what this means is implementation-specific).

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** symbol (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">
    <xsd:complexType>
        <xsd:sequence>
            <xsd:element ref="se:Graphic"/>
        </xsd:sequence>
    </xsd:complexType>
</xsd:element>
```

10.2 Scale selection

T**he MinScaleDenominator and MaxScaleDenominator elements of a Rule de**fine 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 SLD), 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 SLD 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:

2º × (6378137m × 2 × π) ÷ 360º = 222638.9816m
1º × (6378137m × 2 × π) ÷ 360º = 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 ÷ 600 pixels ÷ 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 WFS 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 **FeatureStyle** 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.

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

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.

```
<FeatureStyle>
    <Rule>
        <Filter>...[A = 1]...</Filter>
        <PolygonSymbol> ...[red]... </PolygonSymbol>
```

```
    </Rule>
    <Rule>
        <ElseFilter/>
        <PolygonSymbol> ...[gray]... </PolygonSymbol>
    </Rule>
</FeatureStyle>
```

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.

```
<FeatureStyle>
    <Rule>
        <Filter>...[A = 1]...</Filter>
        <PolygonSymbol> ...[red]... </PolygonSymbol>
    </Rule>
</FeatureStyle>
```

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

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

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.

```
<FeatureStyle>
    <Rule>
        <Filter>...[A = 1]...</Filter>
        <MaxScaleDenominator>1e6</MaxScaleDenominator>
        <PolygonSymbol> ...[red]... </PolygonSymbol>
    </Rule>
    <Rule>
        <Filter>...[A = 2]...</Filter>
        <MaxScaleDenominator>1e6</MaxScaleDenominator>
        <PolygonSymbol> ...[yellow]... </PolygonSymbol>
    </Rule>
    <Rule>
        <ElseFilter/>
        <MaxScaleDenominator>1e6</MaxScaleDenominator>
        <PolygonSymbol> ...[blue]... </PolygonSymbol>
```

```
    </Rule>
    <Rule>
        <Filter>...[A = 1]...</Filter>
        <MinScaleDenominator>1e6</MinScaleDenominator>
        <MaxScaleDenominator>10e6</MinScaleDenominator>
        <PolygonSymbol> ...[purple]... </PolygonSymbol>
    </Rule>
    <Rule>
        <ElseFilter/>
        <MinScaleDenominator>1e6</MinScaleDenominator>
        <MaxScaleDenominator>10e6</MinScaleDenominator>
        <PolygonSymbol> ...[gray]... </PolygonSymbol>
    </Rule>
    <Rule>
        <Filter>...[A = 1]...</Filter>
        <MinScaleDenominator>10e6</MinScaleDenominator>
        <PolygonSymbol> ...[gray]... </PolygonSymbol>
    </Rule>
</FeatureStyle>
```

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 Symbols

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

a) Line
b) Polygon
c) Point
d) Text
e) Raster
f) MappedColor

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

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

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

```
<PolygonSymbol uom="http://www.opengeospatial.org/sld/units/metre"/>
<PolygonSymbol uom="http://www.opengeospatial.org/sld/units/feet"/>
<PolygonSymbol uom="http://www.opengeospatial.org/sld/units/pixel"/>
```

All values set inside this **Symbol** shall use this unit for drawing the corresponding elements. It is also possible to use pixel values inside a **Symbol** 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 symbol

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

11.1.1 Format

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

```
<xsd:element name="LineSymbol" substitutionGroup="se:Symbol">
    <xsd:complexType>
        <xsd:complexContent>
            <xsd:extension base="se:SymbolType">
                <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>
</xsd:element>
```

The sub-elements are defined below.

11.1.2 Geometry

The **Geometry** element of a **LineSymbol** defines the linear geometry to be used for styling. The **Geometry** element is optional and if it is absent then the "default" geometry property of the feature type that is used in the containing **FeatureType** is used. The precise meaning of "default" geometry property is system-dependent. 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">
    <xsd:complexType>
        <xsd:sequence>
            <xsd:element ref="ogc:PropertyName"/>
        </xsd:sequence>
    </xsd:complexType>
</xsd:element>
```

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>
```

The properties that are present in a geometry can be interrogated using the **`DescribeFeatureType`** call of the WFS interface (Subclause **Error! Reference source not found.**). All symbol types can include a **Geometry** element also.

11.1.3 Stroke

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

```
<xsd:element name="Stroke">
    <xsd:complexType>
        <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>
</xsd:element>
```

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 **LineSym**`bol` and other symbols), 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 symbol 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 symbol 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">
    <xsd:complexType mixed="true">
        <xsd:complexContent>
            <xsd:extension base="se:ParameterValueType">
                <xsd:attribute name="name" type="xsd:string" use="required"/>
            </xsd:extension>
        </xsd:complexContent>
    </xsd:complexType>
</xsd:element>


<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>
```

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 **LineSymbol**.

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 **LineSymbol** 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">
    <xsd:complexType>
        <xsd:sequence>
            <xsd:element ref="se:Graphic"/>
        </xsd:sequence>
    </xsd:complexType>
</xsd:element>
```

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">
    <xsd:annotation>
        <xsd:documentation>
 A "GraphicStroke" defines a repated-linear graphic pattern to be used
 for stroking a line.
```

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

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:annotation>
        <xsd:documentation>
 Initialgap defines the initial empty space, before the first Graphic or Label should be rendered.
</xsd:documentation>
    </xsd:annotation>
</xsd:element>
<xsd:element name="Gap" type="se:ParameterValueType">
    <xsd:annotation>
        <xsd:documentation>
 Gap defines the empty space between two Graphics or Labels.
</xsd:documentation>
    </xsd:annotation>
</xsd:element>
```

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 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 symbol:

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

```
  </Geometry>
  <Stroke>
    <SvgParameter name="stroke">#0000ff</SvgParameter>
    <SvgParameter name="stroke-width">2</SvgParameter>
  </Stroke>
</LineSymbol>
```

The resulting map portrayal based upon the above rule is:

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

```
<LineSymbol>
  <Stroke/>
</LineSymbol>
```

11.2 Polygon symbol

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

11.2.1 Format

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

```
<xsd:element name="PolygonSymbol" substitutionGroup="se:Symbol">
    <xsd:complexType>
        <xsd:complexContent>
            <xsd:extension base="se:SymbolType">
                <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:sequence>
            </xsd:extension>
        </xsd:complexContent>
    </xsd:complexType>
</xsd:element>
```

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 **Polygon**Symbol 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 graphic symbols used as part of the same point symbol 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**.

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

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 **Polygon**Symbol as follows:

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

The resulting map portrayal based upon the above rule is:

A very simple styling with default parameters would be:

```
<PolygonSymbol>
  <Fill/>
  <Stroke/>
</PolygonSymbol>
```

11.3 Point symbol

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

11.3.1 Format

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

```
    <xsd:element name="PointSymbol" substitutionGroup="se:Symbol">
        <xsd:annotation>
            <xsd:documentation>
      A "PointSymbol" specifies the rendering of a "graphic symbol"
      at a point.
    </xsd:documentation>
        </xsd:annotation>
        <xsd:complexType>
            <xsd:complexContent>
                <xsd:extension base="se:SymbolType">
                    <xsd:sequence>
                        <xsd:element ref="se:Geometry" minOccurs="0"/>
                        <xsd:element ref="se:Graphic" minOccurs="0"/>
                    </xsd:sequence>
                </xsd:extension>
```

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

The **Geometry** element is discussed in Subclause 11.1.2. In this case, if a line, polygon, or raster geometry is used with this symbol, 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 time so that a point symbol 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 "symbol" which is used in a different context in SLD. The high-level definition of a **Graphic** element is:

```
<xsd:element name="Graphic">
    <xsd:complexType>
        <xsd:sequence>
            <xsd:choice minOccurs="0" maxOccurs="unbounded">
                <xsd:element ref="se:ExternalGraphic"/>
                <xsd:element ref="se:Mark"/>
            </xsd:choice>
            <xsd:sequence>
                <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:sequence>
    </xsd:complexType>
</xsd:element>



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


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

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

<xsd:element name="ExternalGraphic">
    <xsd:complexType>
        <xsd:sequence>
            <xsd:choice>
```

```
                <xsd:element ref="se:OnlineResource"/>
                <xsd:element ref="se:InlineContent"/>
            </xsd:choice>
            <xsd:element ref="se:Format"/>
        </xsd:sequence>
    </xsd:complexType>
</xsd:element>


<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 **Opacity** element gives the opacity of 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 pixels encoded as a floating-point number. This element is also used in other contexts than graphic size and pixel units are still used even for font size. 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 **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">
    <xsd:complexType>
        <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>

<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 **Point**Symbol can be represented using a mark as follows:

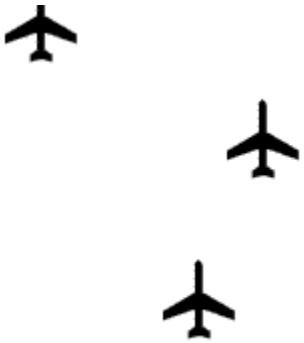
```
<PointSymbol>
  <Geometry>
    <ogc:PropertyName>locatedAt</ogc:PropertyName>
  </Geometry>
  <Graphic>
    <Mark>
      <WellKnownName>star</WellKnownName>
      <Fill>
        <SvgParameter name="fill">#ff0000</SvgParameter>
      </Fill>
    </Mark>
    <Size>8.0</Size>
  </Graphic>
</PointSymbol>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

The resulting map portrayal based upon the preceding rule is:

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

```
<PointSymbol>
  <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>
</PointSymbol>
```

The resulting map portrayal based upon the preceding rule is:



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

```
<PointSymbol>
  <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>
    </PointSymbol>
```

The resulting map portrayal based upon the preceding rule is:

11.4Text symbol

11.4.1 Format

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

```
<xsd:element name="TextSymbol" substitutionGroup="se:Symbol">
    <xsd:complexType>
        <xsd:complexContent>
            <xsd:extension base="se:SymbolType">
                <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>
</xsd:element>
```

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 **Text**Symbol, 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">
    <xsd:complexType>
        <xsd:sequence>
            <xsd:element ref="se:SvgParameter" minOccurs="0" maxOccurs="unbounded"/>
        </xsd:sequence>
    </xsd:complexType>
</xsd:element>
```

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">
    <xsd:complexType>
        <xsd:choice>
            <xsd:element ref="se:PointPlacement"/>
            <xsd:element ref="se:LinePlacement"/>
            <xsd:element ref="se:PolygonPlacement"/>
        </xsd:choice>
    </xsd:complexType>
</xsd:element>

<xsd:element name="PointPlacement">
    <xsd:complexType>
        <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>


<xsd:element name="LinePlacement">
    <xsd:annotation>
        <xsd:documentation>
 A "LinePlacement" specifies how a text label should be rendered
 relative to a linear geometry.
</xsd:documentation>
    </xsd:annotation>
    <xsd:complexType>
        <xsd:sequence>
            <xsd:element ref="se:PerpendicularOffset" minOccurs="0"/>
        </xsd:sequence>
    </xsd:complexType>
 </xsd:element>



<xsd:element name="LinePlacement">
    <xsd:complexType>
        <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:sequence>
    </xsd:complexType>
</xsd:element>
<xsd:element name="IsRepeated" type="xsd:boolean"/>
<xsd:element name="IsAligned" type="xsd:boolean"/>


<xsd:element name="PolygonPlacement">
    <xsd:complexType>
        <xsd:sequence>
            <xsd:element ref="se:UseRepresentativePolygonLine"/>
        </xsd:sequence>
    </xsd:complexType>
</xsd:element>

<xsd:element name="UseRepresentativePolygonLine" 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 symbol 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 one label at a central point of the line. If IsRepeated ist "true", the label will be repeatedly drawn along the line with InitialGap and Gap defining the spaces at the beginning and between labels.

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

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 defined as:

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

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

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 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 label text. A system may choose different anchor points to de-conflict labels.

The **Displacement** element of a **PointPlacement** gives the X and Y displacements from the main-geometry point to render a text label.

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 uoms 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">
    <xsd:complexType>
```

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


    <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**Symbol, 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 **TextSymbol**:

```
<TextSymbol>
 <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>
 <Fill>
  <SvgParameter name="fill">#000000</SvgParameter>
 </Fill>
 <Halo/>
  </TextSymbol>
```

11.5 Raster symbol

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

11.5.1 Format

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

```
    <xsd:element name="RasterSymbol" substitutionGroup="se:Symbol">
        <xsd:complexType>
            <xsd:complexContent>
```

```
            <xsd:extension base="se:SymbolType">
                <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>
</xsd:element>
```

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">
    <xsd:complexType>
        <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>

<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.

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 either the colors of a palette-type raster source or the mapping of fixed-numeric pixel values to colors.

```
<xsd:element name="ColorMap">
    <xsd:complexType>
        <xsd:choice minOccurs="0" maxOccurs="unbounded">
            <xsd:element ref="se:ColorMapEntry"/>
        </xsd:choice>
    </xsd:complexType>
</xsd:element>
<xsd:element name="ColorMapEntry">
    <xsd:complexType>
        <xsd:attribute name="color" type="xsd:string" use="required"/>
        <xsd:attribute name="opacity" type="xsd:double"/>
        <xsd:attribute name="quantity" type="xsd:double"/>
        <xsd:attribute name="label" type="xsd:string"/>
    </xsd:complexType>
</xsd:element>
```

For example, a DEM raster giving elevations in meters above sea level can be translated to a colored image with a **ColorMap**. The **quantity** attributes of a color-map are used for translating between numeric matrices and color rasters and the **ColorMap** entries should be in order of increasing numeric **quantity** so that intermediate numeric values can be matched to a color (or be interpolated between two colors). Labels may be used for legends or may be used in the future to match character values. Not all systems can

support **opacity** in colormaps. The default opacity is **1.0** (fully opaque). Defaults for **quantity** and **label** are system-dependent.

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">
    <xsd:complexType>
        <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>

<xsd:element name="Normalize">
    <xsd:complexType/>
</xsd:element>

<xsd:element name="Histogram">
    <xsd:complexType/>
</xsd:element>

<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">
    <xsd:complexType>
        <xsd:sequence>
            <xsd:element ref="se:BrightnessOnly" minOccurs="0"/>
            <xsd:element ref="se:ReliefFactor" minOccurs="0"/>
        </xsd:sequence>
    </xsd:complexType>
</xsd:element>

<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 **Raster**Symbol. 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 **LineSymbol** or **PolygonSymbol**. It is defined as:

```
<xsd:element name="ImageOutline">
    <xsd:complexType>
        <xsd:choice>
            <xsd:element ref="se:LineSymbol"/>
            <xsd:element ref="se:PolygonSymbol"/>
        </xsd:choice>
    </xsd:complexType>
</xsd:element>
```

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 **Polygon**Symbol **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):

```
<RasterSymbol>
 <Opacity>1.0</Opacity>
 <ColorMap>
  <ColorMapEntry color="#00ff00" quantity="-500"/>
  <ColorMapEntry color="#00fa00" quantity="-417"/>
  <ColorMapEntry color="#14f500" quantity="-333"/>
  <ColorMapEntry color="#28f502" quantity="-250"/>
  <ColorMapEntry color="#3cf505" quantity="-167"/>
  <ColorMapEntry color="#50f50a" quantity="-83"/>
  <ColorMapEntry color="#64f014" quantity="-1"/>
  <ColorMapEntry color="#7deb32" quantity="0"/>
  <ColorMapEntry color="#78c818" quantity="30"/>
  <ColorMapEntry color="#38840c" quantity="105"/>
  <ColorMapEntry color="#2c4b04" quantity="300"/>
  <ColorMapEntry color="#ffff00" quantity="400"/>
  <ColorMapEntry color="#dcdc00" quantity="700"/>
  <ColorMapEntry color="#b47800" quantity="1200"/>
  <ColorMapEntry color="#c85000" quantity="1400"/>
  <ColorMapEntry color="#be4100" quantity="1600"/>
  <ColorMapEntry color="#963000" quantity="2000"/>
  <ColorMapEntry color="#3c0200" quantity="3000"/>
  <ColorMapEntry color="#ffffff" quantity="5000"/>
  <ColorMapEntry color="#ffffff" quantity="13000"/>
 </ColorMap>
```

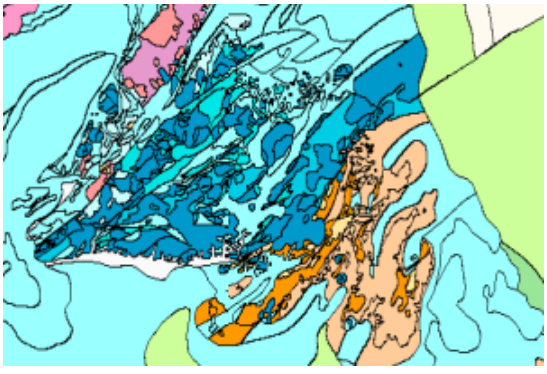
```
  <OverlapBehavior>AVERAGE</OverlapBehavior>
  <ShadedRelief/>
</RasterSymbol>
```

Here is a rather artificial mutli-band raster symbol:

```
<RasterSymbol>
  <Opacity>1.0</Opacity>
  <ColorMap>
    <ColorMapEntry color="#000000" quantity="0"/>
    <ColorMapEntry color="#ffffff" quantity="255"/>
  </ColorMap>
  <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>
   <ContrastEnhancement>
      <GammaValue>1.0</GammaValue>
   </ContrastEnhancement>
    </RasterSymbol>
```

11.6 Mapped-color symbol

A **MappedColorSymbol** is used to provide simple variable-color rendering for vector symbol types. This is also called “choropleth” or “thematic” symbolization. An example of a choropleth follows for geologic rock types and ages:



11.6.1 Format

The **MappedColorSymbol** is defined as:

```
<xsd:element name="MappedColorSymbol" substitutionGroup="se:Symbol">
    <xsd:complexType>
        <xsd:complexContent>
            <xsd:extension base="se:SymbolType">
                <xsd:sequence>
                    <xsd:element ref="se:LookupValue"/>
                    <xsd:element ref="se:ColorMap"/>
                    <xsd:element ref="se:MappedColorSubSymbol"/>
                </xsd:sequence>
            </xsd:extension>
        </xsd:complexContent>
    </xsd:complexType>
</xsd:element>


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

<xsd:element name="MappedColorSubSymbol">
    <xsd:complexType>
        <xsd:choice>
            <xsd:element ref="se:LineSymbol"/>
            <xsd:element ref="se:PolygonSymbol"/>
            <xsd:element ref="se:PointSymbol"/>
            <xsd:element ref="se:TextSymbol"/>
            <xsd:element ref="se:RasterSymbol"/>
        </xsd:choice>
        <xsd:attribute name="matchTo" use="required">
            <xsd:simpleType>
                <xsd:restriction base="xsd:string">
                    <xsd:enumeration value="quantity"/>
                    <xsd:enumeration value="label"/>
                </xsd:restriction>
            </xsd:simpleType>
        </xsd:attribute>
    </xsd:complexType>
</xsd:element>
```

11.6.2 Parameters

The **LookupValue** element determines what value will be used to select the coloring of the choropleth features. It may be any expression.

The **ColorMap** specifies what the selectable colors are. The **ColorMap** element format is defined with the **RasterSymbol**. Only a simple, manually-defined colormap is available.

The **MappedColorSubSymbol** specifies how the matching is made between lookup values and colormap entries and includes the symbols to be colored. The **matchTo** attribute can be given the value "**quantity**" or "**label**" to indicate that the lookup will be based either the **quantity** or **label** attributes of the colormap entires, respectively. These lookup types are used for numeric and string expressions, respectively.

The way that the coloring is applied is that the selected color is used as the default color for the included sub-symbol, overriding the regular meaning of the default color for strokes, fills, and halos. Within a sub-**RasterSymbol**, the default color only applies to embedded **ImageOutline** symbols.

11.6.3 Example

The following example renders polygons with black strokes and variable-colored fills based on the **type_code** numeric feature property.

```
<MultiColorSymbol>
  <LookupValue>
    <ogc:PropertyName>type_code</ogc:PropertyName>
  </LookupValue>
  <ColorMap>
    <ColorMapEntry color="#ffcc99" quantity="1"/>
    <ColorMapEntry color="#0099cc" quantity="2"/>
    <ColorMapEntry color="#ccff99" quantity="3"/>
    <ColorMapEntry color="#99ffff" quantity="4"/>
  </ColorMap>
  <MappedColorSubSymbol matchTo="quantity">
    <PolygonSymbol>
      <Stroke>
        <SvgParameter name="stroke">#000000</SvgParameter>
      </Stroke>
      <Fill/>
    </PolygonSymbol>
  </MappedColorSubSymbol>
</MultiColorSymbol>
```

Annex A
(normative)

Abstract test suite

A.1 General

A paragraph.

In each Implementation Specification document, Annex A shall specify the Abstract Test Suite, as specified in Clause 9 and Annex A of ISO 19105. That Clause and Annex specify the ISO/TC 211 requirements for Abstract Test Suites. Examples of Abstract Test Suites are available in an annex of most ISO 191XX documents, one of the more useful is in ISO 191TBD. Note that this guidance may be more abstract than needed in an OGC Implementation Specification.

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/(TBD) 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

Symbol.xsd

FeautureStyle.xsd

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

This annex can be included if useful to provide more XML document examples.

D.1 Introduction

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

D.2 TBD