

# **Open GIS Consortium Inc.**

Date: 2002-09-19

Reference number of this OpenGIS® Project Document: **OGC 02-070**

Version: 1.0.0

Category: Proposed OpenGIS® OGC Implementation Specification

Editor: William Lalonde

## **Styled Layer Descriptor Implementation Specification**

Document type:	OpenGIS® Implementation Specification
Document stage:	Adopted Specification
Document language:	English

Copyright 2000, 2001, 2002 Compusult Limited  
Copyright 2000, 2001, 2002 CubeWerx Inc.  
Copyright 2000, 2001, 2002 Environmental Systems Research Institute, Inc. (ESRI)  
Copyright 2000, 2001, 2002 Intergraph Corporation  
Copyright 2000, 2001, 2002 IONIC Software s.a.  
Copyright 2000, 2001, 2002 Laser-Scan Limited  
Copyright 2000, 2001, 2002 Syncline Inc.

The companies listed above have granted the Open GIS Consortium, Inc. (OGC) a nonexclusive, royalty-free, paid up, worldwide license to copy and distribute this document and to modify this document and distribute copies of the modified version.

This document does not represent a commitment to implement any portion of this specification in any company's products.

OGC's [Legal, IPR and Copyright Statements are found at http://www.opengis.org/legal/ipr.htm](http://www.opengis.org/legal/ipr.htm)

#### **NOTICE**

Permission to use, copy, and distribute this document in any medium for any purpose and without fee or royalty is hereby granted, provided that you include the above list of copyright holders and the entire text of this NOTICE.

We request that authorship attribution be provided in any software, documents, or other items or products that you create pursuant to the implementation of the contents of this document, or any portion thereof.

No right to create modifications or derivatives of OGC documents is granted pursuant to this license. However, if additional requirements (as documented in the Copyright FAQ at [http://www.opengis.org/legal/ipr\\_faq.htm](http://www.opengis.org/legal/ipr_faq.htm)) are satisfied, the right to create modifications or derivatives is sometimes granted by the OGC to individuals complying with those requirements.

THIS DOCUMENT IS PROVIDED "AS IS," AND COPYRIGHT HOLDERS MAKE NO REPRESENTATIONS OR WARRANTIES, EXPRESS OR IMPLIED, INCLUDING, BUT NOT LIMITED TO, WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE, NON-INFRINGEMENT, OR TITLE; THAT THE CONTENTS OF THE DOCUMENT ARE SUITABLE FOR ANY PURPOSE; NOR THAT THE IMPLEMENTATION OF SUCH CONTENTS WILL NOT INFRINGE ANY THIRD PARTY PATENTS, COPYRIGHTS, TRADEMARKS OR OTHER RIGHTS.

COPYRIGHT HOLDERS WILL NOT BE LIABLE FOR ANY DIRECT, INDIRECT, SPECIAL OR CONSEQUENTIAL DAMAGES ARISING OUT OF ANY USE OF THE DOCUMENT OR THE PERFORMANCE OR IMPLEMENTATION OF THE CONTENTS THEREOF.

The name and trademarks of copyright holders may NOT be used in advertising or publicity pertaining to this document or its contents without specific, written prior permission. Title to copyright in this document will at all times remain with copyright holders.

RESTRICTED RIGHTS LEGEND. Use, duplication, or disclosure by government is subject to restrictions as set forth in subdivision (c)(1)(ii) of the Right in Technical Data and Computer Software Clause at DFARS 252.227.7013

**OpenGIS® is a trademark or registered trademark of Open GIS Consortium, Inc. in the United States and in other countries.**

## Contents

i.	Preface .....	vi
ii.	Submitting organizations.....	vi
iii.	Document Contributor Contact Points .....	vi
iv.	Revision history .....	vii
v.	Changes to the OpenGIS® Abstract Specification .....	vii
	Foreword.....	viii
	Introduction .....	ix
1	Scope.....	1
2	Conformance.....	1
3	Normative references .....	1
4	Terms and definitions .....	2
5	Conventions.....	3
5.1.	Normative Verbs .....	3
5.2.	Symbols (and abbreviated terms) .....	3
6	Web-Map-Server Integration.....	4
6.1.	A Review of WMS 1.1.1 .....	4
6.2.	General HTTP Request Rules as used by WMS and SLD.....	6
6.3.	Styled-Layer Descriptor .....	7
6.4.	WMS Requests using an SLD .....	7
6.5.	Web Map Servers and Web Feature/Coverage Servers .....	11
6.6.	DescribeLayer Request .....	14
6.7.	Enhancements to WMS GetCapabilities.....	15
7	Layers .....	16
7.1.	SLD Root Element.....	16
7.2.	Named Layers.....	17
7.3.	User-Defined Layers .....	20
8	User-Defined Styles .....	23
9	FeatureTypeStyles .....	24
10	Rules .....	25
10.1.	Identification & Legends .....	26
10.2.	Scale Selection.....	26
10.3.	Feature Filtering.....	29

<b>11</b>	<b>Symbolizers.....</b>	<b>33</b>
<b>11.1.</b>	<b>Line Symbolizer.....</b>	<b>33</b>
<b>11.1.1</b>	<b>Format.....</b>	<b>33</b>
<b>11.1.2</b>	<b>Geometry.....</b>	<b>34</b>
<b>11.1.3</b>	<b>Stroke</b>	<b>34</b>
<b>11.1.4</b>	<b>Examples .....</b>	<b>37</b>
<b>11.2.</b>	<b>Polygon Symbolizer.....</b>	<b>38</b>
<b>11.2.1</b>	<b>Format.....</b>	<b>38</b>
<b>11.2.2</b>	<b>Fill</b>	<b>39</b>
<b>11.2.3</b>	<b>Example.....</b>	<b>39</b>
<b>11.3.</b>	<b>Point Symbolizer .....</b>	<b>40</b>
<b>11.3.1</b>	<b>Format.....</b>	<b>40</b>
<b>11.3.2</b>	<b>Graphic.....</b>	<b>41</b>
<b>11.3.3</b>	<b>Examples .....</b>	<b>43</b>
<b>11.4.</b>	<b>Text Symbolizer.....</b>	<b>45</b>
<b>11.4.1</b>	<b>Format.....</b>	<b>45</b>
<b>11.4.2</b>	<b>Label</b>	<b>45</b>
<b>11.4.3</b>	<b>Font</b>	<b>46</b>
<b>11.4.4</b>	<b>Label Placement .....</b>	<b>46</b>
<b>11.4.5</b>	<b>Halo</b>	<b>48</b>
<b>11.4.6</b>	<b>Example.....</b>	<b>49</b>
<b>11.5.</b>	<b>Raster Symbolizer .....</b>	<b>49</b>
<b>11.5.1</b>	<b>Format.....</b>	<b>49</b>
<b>11.5.2</b>	<b>Parameters .....</b>	<b>50</b>
<b>11.5.3</b>	<b>Examples .....</b>	<b>53</b>
<b>11.6.</b>	<b>Systems With Limited Capabilities .....</b>	<b>55</b>
<b>11.7.</b>	<b>Integrated SLD Examples .....</b>	<b>55</b>
<b>12</b>	<b>Map Legends.....</b>	<b>60</b>
<b>13</b>	<b>Symbology Management.....</b>	<b>64</b>
<b>13.1.</b>	<b>GetStyles.....</b>	<b>64</b>
<b>13.2.</b>	<b>PutStyles.....</b>	<b>64</b>
<b>14</b>	<b>Styling Standards .....</b>	<b>66</b>
<b>14.1.</b>	<b>GeoSym .....</b>	<b>66</b>
<b>14.2.</b>	<b>MIL2525B .....</b>	<b>70</b>
	<b>Annex A: Styled-Layer-Descriptor Schema .....</b>	<b>74</b>
	<b>Annex B: WMS_DescribeLayerResponse DTD .....</b>	<b>89</b>
	<b>Annex C: Conformance Tests .....</b>	<b>90</b>
	<b>Annex D: Future Work.....</b>	<b>91</b>
	<b>Annex E: RFC Changes.....</b>	<b>93</b>
<b>E.1.</b>	<b>Comments from commenter #1.....</b>	<b>93</b>
<b>E.1.1.</b>	<b>Comment #1 .....</b>	<b>93</b>
<b>E.1.2.</b>	<b>Comment #2 .....</b>	<b>93</b>

E.1.3. Comment #3 .....	94
E.1.4. Comment #4 .....	95
E.1.5. Comment #5 .....	95
E.1.6. Comment #6 .....	96
E.1.7. Comment #7 .....	97
E.1.8. Comment #8 .....	97
E.1.9. Comment #9 .....	98
E.1.10. Comment #10 .....	98
E.1.11. Comment #11 .....	99
E.2. Comments from commenter #2.....	99
E.2.1. Comment #1 .....	99
E.2.2. Comment #2 .....	100
E.2.3. Comment #3 .....	100
E.2.4. Comment #4 .....	101
E.2.5. Comment #5 .....	101
E.2.6. Comment #6 .....	101
E.2.7. Comment #7 .....	102
E.2.8. Comment #8 .....	102
E.2.9. Comment #9 .....	102
E.2.10. Comment #10 .....	102
E.2.11. Comment #11 .....	102
E.2.12. Comment #12 .....	103
E.2.13. Comment #13 .....	103
E.2.14. Comment #14 .....	103
Annex F: OGC SLD and ISO 19117.....	104
Bibliography .....	107

## i. Preface

This document explains how the Web Map Server (WMS 1.0 [1] & 1.1 [2]) specification can be extended to allow user-defined symbolization of feature data. It should be read in conjunction with the latest version WMS specification. At the time of writing the latest version WMS specification was defined by the WMS 1.1.1 Specification.

## ii. Submitting organizations

This Implementation Specification is being submitted to the OGC by the following organizations:

CubeWerx Inc (Editor).  
Syncline  
Ionic Software s.a.

## iii. Document Contributor Contact Points

All questions regarding this submission should be directed to the editor or the submitters listed in Section ii above. Following is a list of all contributors to the Implementation Specification.

COMPANY	CONTACT	ADDRESS	PHONE/FAX	EMAIL
Compusult Ltd.	Larry Bouzane			<a href="mailto:larry@compusult.nf.ca">larry@compusult.nf.ca</a>
CubeWerx Inc.	Peter Vretanos	200 rue Montcalm Suite R-13 Hull, Quebec Canada J8Y 3B5	Phone: 416-701-1985 Fax: 819-771-8388	<a href="mailto:pvretano@cubewerx.com">pvretano@cubewerx.com</a>
CubeWerx Inc.	Craig Bruce	200 rue Montcalm Suite R-13 Hull, Quebec	Phone: 819-771-8303 Ext 205	<a href="mailto:csbruce@cubewerx.com">csbruce@cubewerx.com</a>

		Canada J8Y 3B5	Fax: 819-771-8388	
ESRI	Marwa Mabrouk			mmabrouk@esri.com
ESRI	Ivan Cheung			icheung@esri.com
Galdos Systems Inc	Ron Lake			rlake@galdosinc.com
Intergraph Corp.	John Vincent			jtvincen@intergraph.com
Ionic Software s.a.	Dimitri Monie			dimitri.monie@ionicssoft.com
Laser-Scan Ltd.	Seb Lessware			sebl@lsl.co.uk
m-spatial	Adrian Cuthbert			adrian.cuthbert@m-spatial.com
Syncline	Raj Singh			rs@syncline.com
US Army ERDC	Dan Specht			specht@tec.army.mil

#### iv. Revision history

Date	Release	Author	Paragraph modified	Description
2001-02-07	01-028	Adrian Cuthbert	initial paper for SLD 0.7.0	WMT-2 Project-Discussion Paper
2001-08-31	01-028r2	Craig Bruce	re-write for SLD 0.7.1	MPP-1 Project-Discussion Paper
2001-11-30	01-028r3	Craig Bruce	update for SLD 0.7.2 and DIPR format	MPP-1.1 DIPR preview
2001-11-30	01-028r4	Craig Bruce	fixed up pre-pages, added GeoSym content	MPP-1.1 DIPR
2001-12-28	01-028r5	Craig Bruce	minor fixes, added 2525B content, example pictures	MPP-1.1 IPR
2002-03-12	02-013	Carl Reed Craig Bruce Bill Lalonde	Modified for submission and consideration as RFC Proposal Implementation Specification	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	RFC changes; see Annex E	Incorporated RFC comments

#### v. Changes to the OpenGIS® Abstract Specification

The OpenGIS® Abstract Specification does not require changes to accommodate the technical contents of this document.

## **Foreword**

Attention is drawn to the possibility that some of the elements of this part of OGC 02-013 may be the subject of patent rights. The Open GIS Consortium Inc. shall not be held responsible for identifying any or all such patent rights.

OGC 02-013r2 replaces OGC 01-028r5 and consists of the following part: Styled Layer Descriptor Implementation Specification



## 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.

This document addresses the need for geospatial consumers (either humans or machines) to control the visual portrayal of the data with which they work. The current OpenGIS 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 *StyledLayerDescriptor (SLD)*, is the main focus of this paper, and it can be used to portray the output of Web Map Servers, Web Feature Servers and Web Coverage Servers. In many cases, however, the client needs some information about the data residing on the remote server before he, she or it can make a sensible request. This led to the definition of new operations for the OGC services [see Section 6.6] in addition to the definition of the styling language.

There are two basic ways to style a data set. The simplest one is to color all features the same way. For example, one can imagine a layer advertised by a WMS as “hydrography” consisting of lines (rivers and streams) and polygons (lakes, ponds, oceans, etc.). A user might want to tell the server to color the insides of all polygons in a light blue, and color the boundaries of all polygons and all lines in a darker blue. This type of styling requires no knowledge of the attributes or “feature types” of the underlying data, only a language with which to describe these styles. This requirement is addressed by the **FeatureTypeStyle** element in the SLD document.

A more complicated requirement is to style features of the data differently depending on some attribute. For example, in a roads data set, style highways with a three-pixel red line; style four-lane roads in a two-pixel black line; and style two-lane roads in a one-pixel black line. Accomplishing this requires the user to be able to find out what attribute of the data set represents the road type. WMS already has an optional operation that fulfils this need, called **DescribeLayer**. This operation returns the feature types of the layer or layers specified in the request, and the attributes can be discovered with the **DescribeFeatureType** operation of a WFS interface.



# Styled Layer Descriptor Implementation Specification

## 1 Scope

This OpenGIS ® Interoperability-Program Report specifies the format of a map-styling language for producing georeferenced maps with user-defined styling. Different modes for utilizing this styling specification are discussed.

## 2 Conformance

Conformance and Interoperability Testing for this OGC Interoperability Program Report may be checked using all the relevant tests specified in Annex C (normative).

## 3 Normative references

The following normative documents contain provisions that, through reference in this text, constitute provisions of this specification. For dated references, subsequent amendments to, or revisions of, any of these publications do not apply. However, parties to agreements based on this specification are encouraged to investigate the possibility of applying the most recent editions of the normative documents indicated below. For undated references, the latest edition of the normative document referred to applies.

CGI, *The Common Gateway Interface*, National Center for Supercomputing Applications, <<http://hoohoo.ncsa.uiuc.edu/cgi/>>

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 1.1.1, February 2002, OGC document OGC 01-068r2.

OGC Adopted Implementation Specification: Web Map Server version 1.1.0, December 2001, OGC document OGC 01-047r2.

OGC Discussion Paper: Web Coverage Service version 0.7, April 2002, OGC document OGC 02-024.

XML 1.0 (October 2000), *Extensible Markup Language (XML) 1.0 (2nd edition)*, World Wide Web Consortium Recommendation, Bray, T., Paoli, J., Sperberg-McQueen, C.M., and Maler, E., eds., <<http://www.w3.org/TR/2000/REC-xml>>

## 4 Terms and definitions

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

### 4.1

#### **operation**

Specification of a transformation or query that an object may be called to execute [OGC AS 12]

### 4.2

#### **interface**

Named set of **operations** that characterize the behavior of an entity [OGC AS 12]

### 4.3

#### **service**

Distinct part of the functionality that is provided by an entity through **interfaces** [OGC AS 12]

### 4.4

#### **service instance**

#### **server**

Actual implementation of a **service**

### 4.5

#### **client**

Software component that can invoke an **operation** from a **server**

**4.6****request**

Invocation of an **operation** by a **client**

**4.7****response**

Result of an **operation** returned from a **server** to a **client**

**4.8****map**

Pictorial representation of geographic data

**4.9****capabilities XML**

Service-level metadata describing the **operations** and content available for a given **service instance**.

**5 Conventions****5.1. Normative Verbs**

In the sections labeled as normative, the key words "**required**", "**shall**", "**shall not**", "**should**", "**should not**", "**recommended**", "**may**", and "**optional**" in this document are to be interpreted as described in Internet RFC 2119 [15].

The verb "**deprecate**" provides notice that the referenced portion of the specification is being retained for backwards compatibility with earlier versions but may be removed from a future version of the specification without further notice.

**5.2. Symbols (and abbreviated terms)**

The following symbols and abbreviated terms are used in this document.

CGI	Common Gateway Interface
DCP	Distributed Computing Platform
DTD	Document Type Definition
EPSG	European Petroleum Survey Group
GIF	Graphics Interchange Format
GIS	Geographic Information System
GML	Geography Markup Language
HTTP	Hypertext Transfer Protocol
IETF	Internet Engineering Task Force
JPEG	Joint Photographic Experts Group
MIME	Multipurpose Internet Mail Extensions
OGC	Open GIS Consortium
OWS	OGC Web Service

PNG	Portable Network Graphics
RFC	Request for Comments
SLD	Styled Layer Descriptor
SVG	Scalable Vector Graphics
URL	Uniform Resource Locator
WebCGM	Web Computer Graphics Metafile
WCS	Web Coverage Service
WFS	Web Feature Service
WMS	Web Map Service
XML	Extensible Markup Language

## 6 Web-Map-Server Integration

### 6.1. A Review of WMS 1.1.1

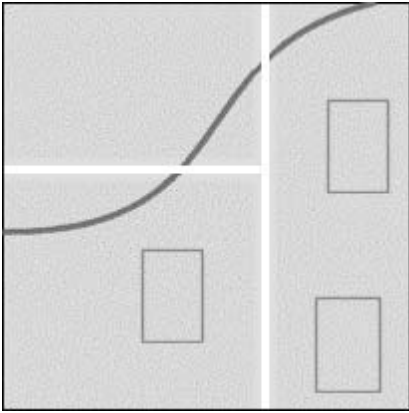
WMS 1.1.1 describes the appearance of a map in terms of ‘styled layers’. A styled layer can be considered as a transparent sheet with features symbolized upon it. A map is made up of a number of these styled layers put together in a specified order. The styled layers are said to be Z-ordered. Users can define more complex or simpler maps by adding or removing styled layers.

A styled layer itself represents a particular combination of ‘layer’ and a ‘style’ in which that layer can be symbolized. Conceptually, the layer defines a stream of features and the style defines how those features are symbolized. This concept is underlined by the fact that there may be multiple styles in which a layer can be symbolized.

In the WMS specification, the request for a map is encoded as an HTTP-GET request and the appearance for a map portrayal is specified by the **LAYERS** and **STYLES** parameters. Consider the following (incomplete) example map request (which is split over multiple lines for presentation purposes only):

```
http://yourfavoritesite.com/WMS?  
  VERSION=1.1.0&  
  REQUEST=GetMap&  
  BBOX=0.0,0.0,1.0,1.0&  
  LAYERS=Rivers,Roads,Houses&  
  STYLES=CenterLine,CenterLine,Outline
```

Results in the map portrayal shown below:



This is to be interpreted as three ‘styled layers’, namely:

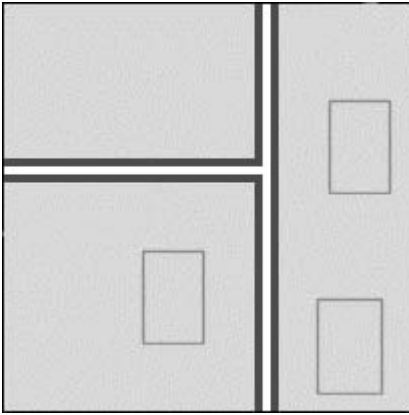
- **Houses:Outline**
- **Roads:CenterLine**
- **Rivers:CenterLine**

The colon notation is introduced only as a convenience to aid discussion. The **Rivers:CenterLine** styled layer is ‘below’ the **Roads:CenterLine** styled layer, as WMS uses the “painter’s model” and plots each successive layer in the **LAYER** list over top of the previously rendered layers. Consequently, the roads appear to ‘cross’ the river. It is possible for the same layer to appear more than once, although rarely with the same style.

A common ‘cartographic trick’ to generate what appears to be the boundaries of linear features is to draw them with a thick colored line and then draw them all again with a thinner, lighter line. This is done for the roads in the following (incomplete) map request:

```
http://yourfavoritesite.com/WMS?
  VERSION=1.1.0&
  REQUEST=GetMap&
  BBOX=0.0,0.0,1.0,1.0&
  LAYERS=Roads,Roads,Houses&
  STYLES=Casing,CenterLine,Outline
```

The resulting map portrayal based upon the above rule is:



This is to be interpreted as three styled layers, namely:

- **Houses:Outline**
- **Roads:CenterLine**
- **Roads:Casing**

It might be noted that the WMS cannot be interrogated for metadata to indicate which styled layers can be meaningfully combined and how. However, a flexible client would allow an end-user to explore the various possibilities.

The WMS 1.1.1 specification deals with styles and layers which are ‘known’ to the WMS and which are identified by name. For this reason, the rest of this document refers to the layers and styles that have been described above as “named layers” and “named styles”. The WMS specification provides only one way to define a styled layer, as a combination of a named layer and a named style.

## 6.2. General HTTP Request Rules as used by WMS and SLD

At present, the only distributed computing platform (DCP) explicitly supported by OGC Web Services is the World Wide Web itself, or more specifically Internet hosts implementing the Hypertext Transfer Protocol (HTTP) [IETF RFC 2616]. Thus the Online Resource of each operation supported by a service instance is an HTTP Uniform Resource Locator (URL). The URL may be different for each operation, or the same, at the discretion of the service provider. Each URL **shall** conform to the description in [IETF RFC 2616] (Section 3.2.2 "HTTP URL") but is otherwise implementation-dependent; only the parameters comprising the service request itself are mandated by the OGC Web Services specifications.

HTTP supports two request methods: GET and POST. One or both of these methods may be defined for a particular OGC Web Service type and offered by a service instance, and the use of the Online Resource URL differs in each case. The basic WMS



specification only defines HTTP GET for invoking operations. (A Styled Layer Descriptor WMS[10] defines HTTP POST for some operations.)

### 6.3. Styled-Layer Descriptor

Section 6.1 described how the appearance of a map in the WMS specification can be defined as a sequence of styled layers. Styling can also be described using a user-defined XML encoding of a map's appearance called a Styled-Layer Descriptor (SLD). The SLD format is discussed in detail in Sections 7 to 11. Briefly, an SLD includes a **StyledLayerDescriptor** XML element that contains a sequence of styled-layer definitions. These styled-layer definitions may use named or user-defined layers and named or user-defined styling. Here is an example simple SLD that corresponds to the first example from the previous section:

```
<StyledLayerDescriptor version="1.0.0">
  <NamedLayer>
    <Name>Rivers</Name>
    <NamedStyle>
      <Name>CenterLine</Name>
    </NamedStyle>
  </NamedLayer>
  <NamedLayer>
    <Name>Roads</Name>
    <NamedStyle>
      <Name>CenterLine</Name>
    </NamedStyle>
  </NamedLayer>
  <NamedLayer>
    <Name>Houses</Name>
    <NamedStyle>
      <Name>Outline</Name>
    </NamedStyle>
  </NamedLayer>
</StyledLayerDescriptor>
```

The **NamedLayer** and **NamedStyle** elements correspond to the **LAYERS** and **STYLES** of the CGI parameters and the “painter’s model” is also used for Z-ordering. An SLD XML document can become much more complex with user-defined styling. The WMS-1.2 styled-layer mechanism is compatible with the SLD-1.0.0 format.

### 6.4. WMS Requests using an SLD

Three approaches are defined to allow a client to take advantage of SLD symbology:

- The client interacts with the WMS using HTTP GET but the request can reference a remote SLD.
- The client uses the HTTP GET method but includes the SLD XML document in-line with the GET request in an **SLD\_BODY** CGI parameter (with appropriate character encoding).

- The client interacts with the WMS using HTTP POST with the **GetMap** request encoded in XML and including an embedded SLD.

The third method is technically superior but there has been a great lack of vendor support for the XML-POST **GetMap**-request method. Use of the second method, which is a compromise between the first and third methods, can encounter problems due to excessively long URLs.

It is important to note that in all cases the WMS has no prior knowledge of the SLD contents. There is a wide spectrum of possible clients. Some may allow a user to switch between a number of pre-defined maps, each specified by its own pre-defined SLD. Others may allow a user to interactively define how they wish a map to appear and construct the necessary SLD ‘on-the-fly’. All of the approaches described above allow a client application to do this but the first one requires that the client be able to place the SLD document in a Web location accessible to the WMS.

Consider the incomplete example GetMap request from the previous section:

```
http://yourfavoritesite.com/WMS?
  VERSION=1.1.0&
  REQUEST=GetMap&
  BBOX=0.0,0.0,1.0,1.0&
  LAYERS=Rivers,Roads,Houses&
  STYLES=CenterLine,CenterLine,Outline&
  WIDTH=400&
  HEIGHT=400&
  FORMAT=PNG
```

It has already been described, in Section 6.2, how the **LAYERS** and **STYLES** parameters could be encoded in an SLD. The request references the SLD using a **SLD** parameter, which replaces the **LAYERS** and **STYLES** parameters. The SLD itself must be accessible to the WMS and is identified using a URL. The URL must be encoded prior to inclusion as a parameter value, just as layer and style names are already encoded. Assuming the URL for the prepared SLD document is **http://myclientsite.com/mySLD.xml** then the above map request would be converted to look like:

```
http://yourfavoritesite.com/WMS?
  VERSION=1.0.5&
  REQUEST=GetMap&
  SRS=EPSG%3A4326&
  BBOX=0.0,0.0,1.0,1.0&
  SLD=http%3A%2F%2Fmyclientsite.com%2FmySLD.xml&
  WIDTH=400&
  HEIGHT=400&
  FORMAT=PNG
```

The prepared SLD document for this example would have the content of the **StyledLayerDescriptor** example from Section 6.2, with appropriate standard XML

header tags. The SLD document could also be included in-line with the GET request as in the following (long) example:

```
http://yourfavoritesite.com/WMS?
  VERSION=1.0.5&
  REQUEST=GetMap&
  SRS=EPSG%3A4326&
  BBOX=0.0,0.0,1.0,1.0&
  SLD_BODY=%3C%3Fxml+version%3D%221.0%22+encoding%3D%22UTF-8%22%
3F%3E%3C!DOCTYPE+StyledLayerDescriptor+SYSTEM+%22http%3A%2F%2Fsom
.site.com%2Fslid%2Fslid_072.xsd%22%3E%3CStyledLayerDescriptor+versi
on%3D%221.0.0%22%3E%3CNamedLayer%3E%3CName%3ERivers%3C%2FName%3E%
3CNamedStyle%3E%3CName%3ECenterLine%3C%2FName%3E%3C%2FNamedStyle%
3E%3C%2FNamedLayer%3E%3CNamedLayer%3E%3CName%3ERoads%3C%2FName%3E
%3CNamedStyle%3E%3CName%3ECenterLine%3C%2FName%3E%3C%2FNamedStyle
%3E%3C%2FNamedLayer%3E%3CNamedLayer%3E%3CName%3EHouses%3C%2FName%
3E%3CNamedStyle%3E%3CName%3EOutline%3C%2FName%3E%3C%2FNamedStyle%
3E%3C%2FNamedLayer%3E%3C%2FStyledLayerDescriptor%3E
  WIDTH=400&
  HEIGHT=400&
  FORMAT=PNG
```

There may be other complications in addition to the excessively long URLs with this approach if UTF-8 characters outside of the 7-bit ASCII range are used, as HTTP is defined to use the ISO Latin-1 character set. The advantages are that the client does not need to publish the SLD document on the Web and simple clients that are unable to use the POST method can use this method.

The alternative approach is to communicate with the WMS using HTTP POST with SLD as a component of WMS 1.2+. Using this method, the above example translates into the following XML encoding for POSTing:

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE GetMap
  SYSTEM "http://some.site.com/wms/GetMap.xsd">
<ogc:GetMap xmlns:ogc="http://www.opengis.net/ows"
  xmlns:gml="http://www.opengis.net/gml"
  env:encodingStyle="http://www.w3.org/2001/09/soap-
encoding"
  version="1.2.0" service="WMS">
  <StyledLayerDescriptor version="1.0.0">
    <NamedLayer>
      <Name>Rivers</Name>
      <NamedStyle>
        <Name>CenterLine</Name>
      </NamedStyle>
    </NamedLayer>
    <NamedLayer>
      <Name>Roads</Name>
      <NamedStyle>
        <Name>CenterLine</Name>
      </NamedStyle>
    </NamedLayer>
```

```

    <NamedLayer>
      <Name>Houses</Name>
      <NamedStyle>
        <Name>Outline</Name>
      </NamedStyle>
    </NamedLayer>
  </StyledLayerDescriptor>
  <BoundingBox
srsName="http://www.opengis.net/gml/srs/epsg.xml#4326">
    <gml:coord>
      <gml:X>-180.0</gml:X>
      <gml:Y>-90.0</gml:Y>
    </gml:coord>
    <gml:coord>
      <gml:X>180.0</gml:X>
      <gml:Y>90.0</gml:Y>
    </gml:coord>
  </BoundingBox>
  <Output>
    <Format>image/jpeg</Format>
    <Transparent>>false</Transparent>
    <Size>
      <Width>1024</Width>
      <Height>512</Height>
    </Size>
  </Output>
  <Exceptions>application/vnd.ogc.se+xml</Exceptions>
</ogc:GetMap>

```

Although this example describes how a WMS specification request can be converted to use an SLD, the mechanism can be used with any valid SLD. Specifically it can be used with an SLD that includes user-defined symbolization.

To make the HTTP-GET methods more practical for use, the SLD can also be used in one of two different modes depending on whether the **LAYERS** parameter is present in the request. If it is not present, then all layers identified in the SLD document are rendered with all defined styles, which is equivalent to the XML-POST method of usage. If the **LAYERS** parameter is present, then only the layers identified by that parameter are rendered and the SLD is used as a “style library”.

When an SLD is used as a style library, the **STYLES** CGI parameter is interpreted in the usual way in the **GetMap** request, except that the handling of the style names is organized so that the styles defined in the SLD take precedence over the named styles stored within the map server. The user-defined SLD styles can be given names and they can be marked as being the default style for a layer. To be more specific, if a style named “**CenterLine**” is referenced for a layer and a style with that name is defined for the corresponding layer in the SLD, then the SLD style definition is used. Otherwise, the standard named-style mechanism built into the map server is used. If the use of a default style is specified and a style is marked as being the default for the corresponding layer in the SLD, then the default style from the SLD is used; otherwise, the standard default style in the map server is used.

## 6.5. Web Map Servers and Web Feature/Coverage Servers

If a WMS is to symbolize features using a user-defined symbolization, it is necessary to identify the source of the feature data. This specification is designed to permit a wide variety of implementations of WMS that support user-defined symbolization. For example a WMS might symbolize feature or coverage data stored in a remote Web Feature Server (WFS) or Web Coverage Server (WCS), or it might only be able to symbolize data from a specific default feature/coverage store.

In support of this, optional parameters called **REMOTE\_OWS\_TYPE** and **REMOTE\_OWS\_URL** are introduced for HTTP-GET **GetMap** requests that can be used to direct the WMS to a remote WFS, WCS, or other OWS service as the ‘default’ source for feature/coverage data. The presently allowed values for the **REMOTE\_OWS\_TYPE** parameter are “**WFS**” and “**WCS**”, though more may be allowed in the future. The **REMOTE\_OWS\_URL** parameter gives the base URL of the service to use. (Previously, this mechanism was provided with a single **WFS** parameter, but that was too restrictive.) For example, if the URL for a WFS is **http://anothersite.com/WFS?** then the map request from the previous sub-section would be converted to look like:

```
http://yourfavoritesite.com/WMS?
  VERSION=1.0.5&
  REQUEST=GetMap&
  SRS=EPSG%3A4326&
  BBOX=0.0,0.0,1.0,1.0&
  SLD=http%3A%2F%2Fmyclientsite.com%2Fmysld.xml&
  WIDTH=400&
  HEIGHT=400&
  FORMAT=PNG&
  REMOTE_OWS_TYPE=WFS&
  REMOTE_OWS_URL=http%3A%2F%2Fanothersite.com%2FWFS%3F
```

This represents the simplest relationship between a WMS and a WFS/WCS. However there is a wide range of possible relationships. To clarify the discussion, this document introduces the concept of ‘component’ and ‘integrated’ servers:

- **Component servers:** these are servers designed to be loosely coupled and work in any combination. For example, a component WMS can symbolize feature/coverage data from any WFS/WCS to which it is directed.
- **Integrated servers:** these are servers that are closely coupled and can only work in particular configurations. For example, an integrated WMS might only be able to symbolize feature/coverage data from the WFS/WCS with which it is integrated.

Whether a particular server is a ‘component’ or ‘integrated’ server says something about how it is implemented. For example a WMS is a ‘valid’ OGC WMS provided it correctly supports the WMS interface. This makes no assumptions about how the WMS is implemented. However, a ‘component’ WMS that can symbolize feature/coverage data only interacts with the data through the WFS/WCS interface, this does say something

about the implementation. Of course, it is not important what type of WFS/WCS (component or integrated) that a component WMS is directed to. It is also worth noting that there will continue to be WMSes that can produce maps from sources other than feature data.

There will be a spectrum of WMS with the ability to support user-defined symbolization. This is best illustrated by describing in more detail what might be considered the ‘two ends’ of the spectrum, represented by a ‘component’ WMS at one end and ‘integrated’ WMS at the other.

- Component WMS

Essentially a portrayal engine that can symbolize feature data obtained from one or more remote WFS/WCSes. Typically it has these characteristics:

- A component WMS probably has no pre-defined ‘named’ layers or styles.
- A component WMS only supports the WMS interface.
- A component WMS can symbolize feature data from any compliant WFS/WCS.
- A component WMS supports both user-defined styles and user-defined layers.

It would be expected that WMS based upon XSLT technology would fit into this category.

- Integrated WMS

This is a server representing a closely coupled feature store and a portrayal engine. Typically it has these characteristics:

- An integrated WMS probably has pre-defined ‘named’ layers and styles.
- An integrated WMS supports the WMS interface and the **DescribeFeatureType** request of the WFS interface or the **GetCapabilities** or optional **DescribeCoverageLayer** requests of the WCS interface.
- An integrated WMS can only symbolize feature data from its own internal the feature store.
- An integrated WMS might only support user-defined styles being applied to pre-defined ‘named’ layers.

Whether one is using a component or integrated WMS, it must be possible to interrogate (albeit at a relatively superficial level) the underlying feature store. This is because user-defined symbolization makes use of concepts not previously required by WMS. For example, the WMS 1.1 specification makes it possible to interact with a WMS using concepts such as [Named]Layer and [Named]Style but without the need to use concepts such as feature type. By contrast user-defined symbolization needs to be able to define new layers and styles using feature types and feature-type properties. For example, a new layer might be defined as all the features of a particular feature type. This specification seeks to ensure that the bar to creating WMSes that support user-defined symbolization is as low as possible.

The underlying feature/coverage store is interrogated using the WFS/WCS interface. For a component WMS this is not a problem, since the feature/coverage store is indeed a remote WFS/WCS. For an integrated WMS, the server must support both the WMS interface and a minimal set of WFS/WCS operations. It must support the **DescribeFeatureType** request of a WFS or the **GetCapabilities** or optional **DescribeCoverageLayer** of a WCS. This describes the properties of a feature/coverage type specified by name in the request. And, if the WMS supports user-defined layers, then it must support the WFS **GetCapabilities** request. For a WFS this returns, among other things, the names of all the feature types supported by the WFS. Together the two WFS requests allow clients to retrieve all the information they require to construct user-defined symbolizations. The WCS **GetCapabilities** request alone is assumed to be sufficient for a WCS.

It is also necessary to indicate where a WMS should find the feature data that is to be symbolized. This is done using the following rules:

1. if the SLD specifies a WFS or WCS in the **RemoteOWS** element of the **UserLayer** element, then it should be used (see Section 7); otherwise
2. if the GetMap request included CGI **REMOTE\_OWS\_TYPE** and **REMOTE\_OWS\_URL** parameters then that remote service should be used; otherwise
3. the WMS should use a default WFS or WCS.

This approach does not permit features/coverages from different WFS/WCSes to be included in the same styled layer; however, it does allow different styled layers to be based on feature data from different WFS/WCSes. The first two options should only be used for a WMS that can be 'directed' to a remote WFS/WCS. The WMS will advertise this ability in response to a **GetCapabilities** request (see Section 6.7). For an integrated WMS, the default WFS/WCS is just the one with which it is integrated. However, there is no reason why a component WMS should not have a default WFS/WCS defined.

## 6.6. DescribeLayer Request

Defining a user-defined style requires information about the features being symbolized, or at least their feature type. Since user-defined styles can be applied to a named layer, there needs to be a mechanism by which a client can obtain feature/coverage-type information for a named layer. This is another example of bridging the gap between the WMS concepts of layers and styles and WFS/WCS concepts such as feature-type and coverage layer. To allow this, a WMS may optionally support the **DescribeLayer** request. This can be applied to multiple layers as shown in the example below:

```
http://yourfavoritesite.com/WMS?
  VERSION=1.1.0&
  REQUEST=DescribeLayer&
  LAYERS=Rivers,Roads,Houses
```

where **DescribeLayer** is a new option for the **REQUEST** parameter and **LAYERS** is the parameter that allows a number of named layers to be specified by name. This is thought to be a better approach than overloading the WMS Capabilities document even more.

The response should be an XML document describing the specified named layers. If any of the named layers are not present, the response is an XML document reporting an exception.

For each named layer, the description should indicate if it is indeed based on feature data and if so it should indicate the WFS/WCS (by a URL prefix) and the feature types. Note that it is perfectly valid for a named layer not to be describable in this way. It has been suggested that we reuse the WFS mechanism for indicating how one identifies feature types in a WFS, namely by using the **Query** element. Annex B gives the DTD for the response.

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE WMS_DescribeLayerResponse
  SYSTEM "http://some.site.com/sld/DSR.dtd" >
<WMS_DescribeLayerResponse version="1.1.0" >
  <!-- 'Layer_A' comes from the wfs specified by
  the prefix "http://www.mywfs.com/WFS?" and has features
  of types 'Road_FT' and 'Route_FT' -->
  <LayerDescription name="Layer_A"
    wfs="http://www.mywfs.com/WFS?">
    <Query typeName="Road_FT" />
    <Query typeName="Route_FT" />
  </LayerDescription>
  <!-- 'Layer_B' cannot be described in terms of
  a WFS and so has no wfs attribute and no contents -->
  <LayerDescription name="Layer_B">
  </LayerDescription>
</WMS_DescribeLayerResponse>
```

Note that even an integrated WMS should provide a WFS/WCS prefix, since this allows **DescribeFeatureType** requests to be made.



A WMS need not support the **DescribeLayer** request. However, it should be noted that it may be common for a WMS not to support **UserLayers** but to allow **UserStyles** to be applied to named layers. In such circumstances, supporting the **DescribeLayer** request is the only interoperable way in which a client can specify user-defined symbolization.

### 6.7. Enhancements to WMS GetCapabilities

The **GetCapabilities** request supported by a WMS must return enough information for a client to construct further, valid requests. For example, the capabilities of a WMS includes what named layers are known to a WMS. A **GetMap** request is not valid if it references a named layer not known to the WMS. To ensure valid interaction with a range of WMSes, including the integrated and component WMS described above, the capabilities response has been enhanced. Specifically it can describe the following information:

1. Does the WMS provide SLD support?
2. Can a client use HTTP POST (with an embedded SLD) and/or HTTP GET (with an **SLD** or **SLD\_BODY** parameter) to issue a request?
3. Does the WMS support user-defined layers?
4. Does the WMS support user-defined styles?
5. Can the WMS be ‘directed’ to remote OWS services (i.e., does the WMS support the **REMOTE\_OWS\_SERVICE** and **REMOTE\_OWS\_URL** parameters in HTTP-GET requests and the **RemoteOWS** attribute within an SLD)?
6. Does the WMS support the **DescribeLayer** request?

The fact that a WMS may not support named layers and named styles is indicated by not returning any **Layer** elements when describing its capabilities. This approach makes it possible to support user-defined styles without supporting user-defined layers. This situation is likely to be most prevalent with component WMSes that are used solely to render remote features. However, it should be noted that this description of the capabilities of a WMS makes no explicit mention of ‘integrated’ and ‘component’ types.

The first four items all represent various aspects of the ability to handle user-defined symbolization. The 1.1.0 WMS-Capabilities DTD introduces a **UserDefinedSymbolization** element contained by the **Capability** element. This new element contains information about the ability of the WMS to support user-defined symbolization:

```
<!-- Elements indicating the level of support the WMS provides
for user-defined symbolization. By default there is no support.
-->
```

```

<!--ELEMENT UserDefinedSymbolization (SupportedSLDVersion*)>
<!--ATTLIST UserDefinedSymbolization

<!-- If the WMS supports a StyledLayerDescriptor (SLD) then the
SLD parameter can be used in place of LAYERS and STYLES
parameters in a GetMap request. -->
    SupportSLD (0 | 1) "0"

<!-- If the WMS supports UserLayers then users can define their
own layers in the SLD in addition to NamedLayers already known to
the WMS. -->
    UserLayer (0 | 1) "0"

<!-- If the WMS supports UserStyles then users can define their
own styles in the SLD to be applied to previously specified
layers. -->
    UserStyle (0 | 1) "0"

<!-- If the WMS supports remote WFS or WCS then a remote service
can be specified as the source of features to be symbolized. -->
    RemoteWFS (0 | 1) "0"

    RemoteWCS (0 | 1) "0">

<!-- Indication of what SLD version(s) are supported by a WMS -->
<!--ELEMENT SupportedSLDVersion (#PCDATA)>

```

The last two items that need to be describable using capabilities are handled using the **Request** element of WMS capabilities, with the addition of a new request type represented by the **DescribeLayer** element. The **Request** element can be used to describe whether a request can be handled by HTTP (as a particular instance of a distributed computing platform) and, if so, whether the request can use HTTP GET and/or POST.

```

<!-- DescribeLayer interface: Presence of theDescribeLayer
element means this server can return a description of a specified
layer -->
<!--ELEMENT DescribeLayer (Format+, DCPTType+)>

```

Currently the only format supported by the **DescribeLayer** request is the XML encoding described in Annex B. For WMS 1.0.0 to 1.0.7, this format is called “**WMS\_XML**”. For WMS 1.0.8 and on, this is the MIME format “**application/vnd.ogc.wms\_xml**”.

## 7 Layers

### 7.1. SLD Root Element

An SLD document is defined as a sequence of styled layers. The root **StyledLayerDescriptor** is defined by the following XML-Schema fragment:

```

<xs:element name="StyledLayerDescriptor">
  <xs:complexType>
    <xs:choice minOccurs="0" maxOccurs="unbounded">
      <xs:element ref="sld:NamedLayer"/>
      <xs:element ref="sld:UserLayer"/>
    </xs:choice>
    <xs:attribute name="version" type="xs:string"
      use="required"/>
  </xs:complexType>
</xs:element>

```

The **version** attribute gives the SLD version of an SLD document, to facilitate backward compatibility with static documents stored in various different versions of the SLD spec. The string has the format “x.y.z”, the same as in other OpenGIS Web Server specs. For example, an SLD document stored according to this spec would have the version string “1.0.0”. The attribute is now required.

Note that version numbers are used differently with SLD from how they are with OGC Web services like WMS, for example. Version negotiation cannot be performed in the same way, since SLD is not a service. Instead, the SLD versions supported must either be implied by the web service they are associated with, or the web service must give an explicit list of supported SLD versions in its capabilities. This issue is unresolved.

The styled layers can correspond to either named layers (**NamedLayer**) or user-defined layers (**UserLayer**), which are described in subsequent sections. There may be any number of either type of styled layer, including zero, mixed in any order. The order that the layer references appear in the SLD document will be the order that the styled layers are rendered, with successive styled layers rendered over top of previous styled layers.

## 7.2. Named Layers

A “layer” is defined as a collection of features that can be potentially of various mixed feature types. A named layer is a layer that can be accessed from an OpenGIS Web Server using a well-known name. For example, the WMS interface uses the **LAYER** CGI parameter to reference named layers as in the example parameter from Section 6 of:

```
LAYERS=Rivers,Roads,Houses
```

The equivalent named-layer specification mechanism in SLD is defined by the following XML-Schema fragment:

```

<xs:element name="NamedLayer">
  <xs:complexType>
    <xs:sequence>
      <xs:element ref="sld:Name"/>
      <xs:element ref="sld:LayerFeatureConstraints"
        minOccurs="0"/>
      <xs:choice minOccurs="0" maxOccurs="unbounded">
        <xs:element ref="sld:NamedStyle"/>
        <xs:element ref="sld:UserStyle"/>
      </xs:choice>
    </xs:sequence>
  </xs:complexType>
</xs:element>

```

```

        </xs:choice>
      </xs:sequence>
    </xs:complexType>
  </xs:element>

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

```

The **Name** element identifies the well-known name of the layer being referenced, and is required. All possible well-known names are usually identified in the capabilities document for a server.

The **LayerFeatureConstraints** element is optional in a **NamedLayer** and allows the user to specify constraints on what features of what feature types are to be selected by the named-layer reference. It is essentially a filter that allows the selection of fewer features than are present in the named layer. This element is discussed in Section 7.3 where it is more apropos.

A named styled layer can include any number of named styles and user-defined styles, including zero, mixed in any order. If zero styles are specified, then the default styling for the specified named layer is to be used.

A named style, similar to a named layer, is referenced by a well-known name. A particular named style only has meaning when used in conjunction with a particular named layer. All available styles for each available layer are normally named in a capabilities document.

The WMS interface uses the **STYLES** CGI parameter to reference named styles relative to named **LAYERS**, as in the following example parameters:

```

LAYERS=Rivers,Roads,Houses&
STYLES=CenterLine,CenterLine,Outline

```

Parallel lists of corresponding layer names and style names are used in the CGI interface because the CGI interface is not powerful enough to properly nest the named-style references within the named-layer references. However, the SLD mechanism is. The equivalent named-style selection mechanism in SLD is defined by the following DTD fragment:

```

<xs:element name="NamedStyle">
  <xs:complexType>
    <xs:sequence>
      <xs:element ref="sld:Name"/>
    </xs:sequence>
  </xs:complexType>
</xs:element>

```

The **Name** element simply identifies the well-known style name.

The SLD example corresponding to the example WMS parameters above is:

```

<StyledLayerDescriptor version="1.0.0">
  <NamedLayer>
    <Name>Rivers</Name>
    <NamedStyle>
      <Name>CenterLine</Name>
    </NamedStyle>
  </NamedLayer>
  <NamedLayer>
    <Name>Roads</Name>
    <NamedStyle>
      <Name>CenterLine</Name>
    </NamedStyle>
  </NamedLayer>
  <NamedLayer>
    <Name>Houses</Name>
    <NamedStyle>
      <Name>Outline</Name>
    </NamedStyle>
  </NamedLayer>
</StyledLayerDescriptor>

```

Similarly a ‘cased’ roads example of:

```

LAYERS=Roads,Roads,Houses&
STYLES=Casing,CenterLine,Outline

```

becomes:

```

<StyledLayerDescriptor version="1.0.0">
  <NamedLayer>
    <Name>Roads</Name>
    <NamedStyle>
      <Name>Casing</Name>
    </NamedStyle>
  </NamedLayer>
  <NamedLayer>
    <Name>Roads</Name>
    <NamedStyle>
      <Name>CenterLine</Name>
    </NamedStyle>
  </NamedLayer>
  <NamedLayer>
    <Name>Houses</Name>
    <NamedStyle>
      <Name>Outline</Name>
    </NamedStyle>
  </NamedLayer>
</StyledLayerDescriptor>

```

However, this can be encoded in an alternative manner which does not require the repeated definition of the **NamedLayer** with name “Roads”, since each **NamedLayer** may include any number of style references:

```

<StyledLayerDescriptor version="1.0.0">
  <NamedLayer>
    <Name>Roads</Name>
    <NamedStyle>
      <Name>Casing</Name>
    </NamedStyle>
    <NamedStyle>
      <Name>CenterLine</Name>
    </NamedStyle>
  </NamedLayer>
  <NamedLayer>
    <Name>Houses</Name>
    <NamedStyle>
      <Name>Outline</Name>
    </NamedStyle>
  </NamedLayer>
</StyledLayerDescriptor>

```

Note that the above SLD defines three styled layers, even though there are only two **NamedLayer** elements.

User-defined styles (**UserStyle** in the Schema) are discussed in Section 8.

### 7.3. User-Defined Layers

In addition to using named layers, it is also useful to be able to define custom user-defined layers for rendering. The Schema fragment for user-defined layers is as follows:

```

<xs:element name="UserLayer">
  <xs:complexType>
    <xs:sequence>
      <xs:element ref="sld:Name" minOccurs="0"/>
      <xs:element ref="sld:RemoteOWS" minOccurs="0"/>
      <xs:element ref="sld:LayerFeatureConstraints"/>
      <xs:element ref="sld:UserStyle" maxOccurs="unbounded"/>
    </xs:sequence>
  </xs:complexType>
</xs:element>

```

Since a layer is defined as a collection of potentially mixed-type features, the **UserLayer** element must provide the means to identify the features to be used. All features to be rendered are assumed to be fetched from a Web Feature Server (WFS) or a Web Coverage Service (WCS, in which case the term “features” is used loosely).

The remote server to be used is identified by **RemoteOWS** (OGC Web Service) element which is defined as follows:

```

<xs:element name="RemoteOWS">
  <xs:complexType>
    <xs:sequence>
      <xs:element ref="sld:Service"/>
      <xs:element ref="sld:OnlineResource"/>
    </xs:sequence>
  </xs:complexType>
</xs:element>

```

```

    </xs:sequence>
  </xs:complexType>
</xs:element>

<xs:element name="Service">
  <xs:simpleType>
    <xs:restriction base="xs:string">
      <xs:enumeration value="WFS"/>
      <xs:enumeration value="WCS"/>
    </xs:restriction>
  </xs:simpleType>
</xs:element>

<xs:element name="OnlineResource">
  <xs:complexType>
    <xs:attributeGroup ref="xlink:simpleLink"/>
  </xs:complexType>
</xs:element>

```

It is hoped that this definition can be cleaned up to refer to schemas outside of the SLD definition. As indicated in Section 6.5, there are three ways to specify the WFS to be used: it may be identified by a **WFS** CGI parameter in a **GetMap** request; it may be given explicitly with the optional **RemoteOWS** element of the **UserLayer** element; or it may be the ‘default’ WFS for a WMS, which may be an implicit WFS built into the WMS. The **UserLayer** also has a **Name** element which can be used to give a name to a user-defined layer so that it could potentially be stored into a capable WMS, which is discussed in Section 13. A WCS is used similarly to a WFS in SLD, although WCS usage is considered “experimental”. The WFS and WCS mechanisms in SLD may change as service chaining in OGC Web Services becomes more formalized.

The **DescribeFeatureType** and WFS **GetCapabilities** mechanisms may be used to interrogate the WMS or WFS for what feature types are available to be referenced, also as discussed in Section 6.5.

The **LayerFeatureConstraints** element is used to specify what features of what feature types are to be included in a layer. It is defined as:

```

<xs:element name="LayerFeatureConstraints">
  <xs:complexType>
    <xs:sequence>
      <xs:element ref="sld:FeatureTypeConstraint"
        maxOccurs="unbounded"/>
    </xs:sequence>
  </xs:complexType>
</xs:element>

<xs:element name="FeatureTypeConstraint">
  <xs:complexType>
    <xs:sequence>
      <xs:element ref="sld:FeatureTypeName" minOccurs="0"/>
      <xs:element ref="ogc:Filter" minOccurs="0"/>
      <xs:element ref="sld:Extent" minOccurs="0"/>
    </xs:sequence>
  </xs:complexType>
</xs:element>

```

```

maxOccurs="unbounded"/>
</xs:sequence>
</xs:complexType>
</xs:element>
<xs:element name="FeatureTypeName" type="xs:string"/>

<xs:element name="Extent">
  <xs:complexType>
    <xs:sequence>
      <xs:element ref="sld:Name"/>
      <xs:element ref="sld:Value"/>
    </xs:sequence>
  </xs:complexType>
</xs:element>
<xs:element name="Value" type="xs:string"/>

```

The **Extent** reference should be refactored to refer to an externally defined WCS element. When used in a **UserLayer**, it defines what features are to be included in the layer and when used in a **NamedLayer**, it filters the features that are part of the named layer. The feature-filtering mechanism works analogously when applied to raster coverages, except that **Extent** constraints will normally be given instead of **Filter** constraints.

A **FeatureTypeConstraint** element is used to identify a feature type by a well-known name, using the **FeatureTypeName** element. Any positive number of **FeatureTypeConstraints** may be used to define the features of a layer, though all **FeatureTypeConstraints** in a **UserLayer** must come from the same WFS source.

Named styles cannot be used with user-defined layers, since there is no general way to know if a named style is suitable for use with an arbitrary user-defined layer. Only user-defined styles may be used with user-defined layers, and any positive number of them may be used. Using zero styles is not allowed since there will be no pre-defined default style for an arbitrarily constructed layer.

Here is a simple example of an SLD that uses a user-defined layer:

```

<StyledLayerDescriptor version="1.0.0">
  <UserLayer>
    <Name>MyLayer</Name>
    <RemoteOWS>
      <Service>WFS</Service>
      <OnlineResource
        xmlns:xlink="http://www.w3.org/1999/xlink"
        xlink:type="simple"
        xlink:href="http://some.site.com/WFS?" />
      </RemoteOWS>
    <LayerFeatureConstraints>
      <FeatureTypeConstraint>
        <FeatureTypeName>RoadFeatures</FeatureTypeName>
      </FeatureTypeConstraint>
    </LayerFeatureConstraints>
  </UserLayer>
</StyledLayerDescriptor>

```



```

    <UserStyle>
      [...]
    </UserStyle>
  </UserLayer>
</StyledLayerDescriptor>

```

The WFS is named explicitly in the **UserLayer**, only a single feature type is included in the layer, and the **UserStyle** element is incomplete.

## 8 User-Defined Styles

A user-defined style allows map styling to be defined externally from a system and to be passed around in an interoperable format. The XML-Schema fragment for the **UserStyle** SLD element is as follows:

```

<xs:element name="UserStyle">
  <xs:complexType>
    <xs:sequence>
      <xs:element ref="sld:Name" minOccurs="0"/>
      <xs:element ref="sld:Title" minOccurs="0"/>
      <xs:element ref="sld:Abstract" minOccurs="0"/>
      <xs:element ref="sld:IsDefault" minOccurs="0"/>
      <xs:element ref="sld:FeatureTypeStyle"
        maxOccurs="unbounded"/>
    </xs:sequence>
  </xs:complexType>
</xs:element>

<xs:element name="Title" type="xs:string"/>
<xs:element name="Abstract" type="xs:string"/>
<xs:element name="IsDefault" type="xs:string"/>

```

A **UserStyle** is at the same semantic level as a **NamedStyle** used in the context of a WMS. In a sense, a named style can be thought of as a reference to a hidden **UserStyle** that is stored inside of a map server.

The **Name**, **Title** and **Abstract** elements allow a user-defined style to be identified and are optional. The given **Name** is equivalent to the name of a WMS named style and is used to reference the style externally when an SLD is used in ‘library mode’ (Section 6.4) and identifies the named style to redefine when an SLD is inserted into a WMS (Section 13). The **Title** is a human-readable short description for the style that might be displayed in a GUI pick list, and the **Abstract** is a more exact description that may be a few paragraphs long.

The **IsDefault** element identifies whether a style is the default style of a layer, for use in SLD ‘library mode’ when rendering or for storing inside of a map server. **IsDefault** uses “1” for true and “0” for false. The default value is “0”.

A **UserStyle** can contain one or more **FeatureTypeStyles** which allow the rendering of features of specific types. Feature-type styles are described in Section 9.

The following is an (incomplete) example of a **UserStyle** used with a **NamedLayer**:

```
<StyledLayerDescriptor version="1.0.0">
  <NamedLayer>
    <Name>Transportation</Name>
    <UserStyle>
      <Name>GS1</Name>
      <Title>GeoSym</Title>
      <Abstract>GeoSym style for transportation</Abstract>
      <FeatureTypeStyle>
        [...]
      </FeatureTypeStyle>
    </UserStyle>
  </NamedLayer>
</StyledLayerDescriptor>
```

## 9 FeatureTypeStyles

The **FeatureTypeStyle** 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:

```
<xs:element name="FeatureTypeStyle">
  <xs:complexType>
    <xs:sequence>
      <xs:element ref="sld:Name" minOccurs="0"/>
      <xs:element ref="sld:Title" minOccurs="0"/>
      <xs:element ref="sld:Abstract" minOccurs="0"/>
      <xs:element ref="sld:FeatureTypeName" minOccurs="0"/>
      <xs:element ref="sld:SemanticTypeIdentifier" minOccurs="0"
        maxOccurs="unbounded"/>
      <xs:element ref="sld:Rule" maxOccurs="unbounded"/>
    </xs:sequence>
  </xs:complexType>
</xs:element>

<xs:element name="SemanticTypeIdentifier" type="xs:string"/>
```

The **FeatureTypeStyle** element identifies that explicit separation in SLD between the handling of ‘layers’ and the handling of features of specific feature types. The ‘layer’ concept is unique to WMS and SLD, but features are used more generally, such as in WFS and GML, so this explicit separation is important.

Like a **UserStyle**, a **FeatureTypeStyle** can have a **Name**, **Title**, and **Abstract**. 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 **Title** and **Abstract** are for human-readable information.

The **FeatureTypeName** identifies the specific feature type that the feature-type style is for. It is allowed to be optional, but only if one feature type is in-context (in-layer) and that feature type must match the syntax and semantics of all feature-property references inside of the **FeatureTypeStyle**. Note that there is no restriction against a single **UserStyle** from including multiple **FeatureTypeStyles** that reference the same **FeatureTypeName**. This case does not create an exception in the rendering semantics, however, since a map styler is expected to process all **FeatureTypeStyles** in the order that they appear, regardless, plotting one instance over top of another.

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-type style).

The **FeatureTypeStyle** contains one or more **Rule** elements that allow conditional rendering. Rules are discussed in Section 10.

## 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:

```
<xs:element name="Rule">
  <xs:complexType>
    <xs:sequence>
      <xs:element ref="sld:Name" minOccurs="0"/>
      <xs:element ref="sld:Title" minOccurs="0"/>
      <xs:element ref="sld:Abstract" minOccurs="0"/>
      <xs:element ref="sld:LegendGraphic" minOccurs="0"/>
      <xs:choice minOccurs="0">
        <xs:element ref="ogc:Filter"/>
        <xs:element ref="sld:ElseFilter"/>
      </xs:choice>
      <xs:element ref="sld:MinScaleDenominator" minOccurs="0"/>
      <xs:element ref="sld:MaxScaleDenominator" minOccurs="0"/>
      <xs:choice minOccurs="0" maxOccurs="unbounded">
        <xs:element ref="sld:LineSymbolizer"/>
        <xs:element ref="sld:PolygonSymbolizer"/>
        <xs:element ref="sld:PointSymbolizer"/>
        <xs:element ref="sld:TextSymbolizer"/>
        <xs:element ref="sld:RasterSymbolizer"/>
      </xs:choice>
    </xs:sequence>
  </xs:complexType>
</xs:element>
```

```

    </xs:complexType>
  </xs:element>

```

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

The ordering to use for the **Rules** inside of a **FeatureTypeStyle** 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 SLD 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**” **Rules** to come before regular-condition **Rules**). Therefore, **Rules** should be placed in the order of “priority” in a **UserStyle**, 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 **Title** and **Abstract** 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** 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 SLD usage.

The **LegendGraphic** element gives an optional explicit **Graphic** symbol (described in Section 11.3) to be displayed in a legend for this rule. The use of this element and legends in general are discussed in Section 12. Its schema is:

```

<xs:element name="LegendGraphic">
  <xs:complexType>
    <xs:sequence>
      <xs:element ref="sld:Graphic"/>
    </xs:sequence>
  </xs:complexType>
</xs:element>

```

### 10.2. Scale Selection

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

```

<xs:element name="MinScaleDenominator" type="xs:double"/>
<xs:element name="MaxScaleDenominator" type="xs: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 or 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:

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

The actual scale denominator is translated into the “standard” scale denominator as:

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

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

$$\text{actual} \times \text{standard/actual} = \text{standard}$$

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:

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

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

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

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

$$2^\circ \times (6378137\text{m} \times 2 \times \pi) \div 360^\circ = 222638.9816\text{m}$$

$$1^\circ \times (6378137\text{m} \times 2 \times \pi) \div 360^\circ = 111319.4908\text{m}$$

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

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

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  $1\text{e-}6$ .

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

If a rule has no **Filter** element, the interpretation is that the rule condition is always true, i.e., all features are accepted and styled by the rule.

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:

```

<xs:element name="ElseFilter">
  <xs:complexType/>
</xs: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 section) and all rules for scale ranges that do not include the computed nominal scale are discarded from further processing. Then, the specific condition for the **ElseFilter** is computed by “or-ing” together all of the other filter conditions and take the global “not” of that condition. For example, if there are two rules in a style that have the (SQL) conditions “**a** = 6” and “**b** > 20”, then the condition for the **ElseFilter** would be:

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

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



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

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

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

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

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

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

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

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

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

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

```

        <MaxScale>250e3</MaxScale>
        <PolygonSymbol> ...[red]... </PolygonSymbol>
    </Rule>
    <Rule>
        <Filter>...[A = 1]...</Filter>
        <MinScale>250e3</MinScale>
        <MaxScale>5e6</MaxScale>
        <PolygonSymbol> ...[yellow]... </PolygonSymbol>
    </Rule>
    <Rule>
        <ElseFilter/>
        <PolygonSymbol> ...[gray]... </PolygonSymbol>
    </Rule>
</FeatureTypeStyle>

```

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

```

<FeatureTypeStyle>
    <Rule>
        <Filter>...[A = 1]...</Filter>
        <MaxScale>1e6</MaxScale>
        <PolygonSymbol> ...[red]... </PolygonSymbol>
    </Rule>
    <Rule>
        <Filter>...[A = 2]...</Filter>
        <MaxScale>1e6</MaxScale>
        <PolygonSymbol> ...[yellow]... </PolygonSymbol>
    </Rule>
    <Rule>
        <ElseFilter/>
        <MaxScale>1e6</MaxScale>
        <PolygonSymbol> ...[blue]... </PolygonSymbol>
    </Rule>
    <Rule>
        <Filter>...[A = 1]...</Filter>
        <MinScale>1e6</MinScale>
        <MaxScale>10e6</MinScale>
        <PolygonSymbol> ...[purple]... </PolygonSymbol>
    </Rule>
    <Rule>
        <ElseFilter/>
        <MinScale>1e6</MinScale>
        <MaxScale>10e6</MinScale>
        <PolygonSymbol> ...[gray]... </PolygonSymbol>
    </Rule>
    <Rule>
        <Filter>...[A = 1]...</Filter>
        <MinScale>10e6</MinScale>
        <PolygonSymbol> ...[gray]... </PolygonSymbol>
    </Rule>

```

```

    </Rule>
</FeatureTypeStyle>

```

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

## 11 Symbolizers

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

- Line
- Polygon
- Point
- Text
- Raster

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

### 11.1. Line Symbolizer

#### 11.1.1 Format

A **LineSymbolizer** is used to style a “stroke” along a linear geometry type, such as a string of line segments. It has the following simple definition:

```

<xs:element name="LineSymbolizer">
  <xs:complexType>
    <xs:sequence>
      <xs:element ref="sld:Geometry" minOccurs="0"/>
      <xs:element ref="sld:Stroke" minOccurs="0"/>
    </xs:sequence>
  </xs:complexType>
</xs:element>

```

The sub-elements are defined below.

### 11.1.2 Geometry

The **Geometry** element of a **LineSymbolizer** defines the linear geometry to be used for styling. The **Geometry** element is optional and if it is absent then the “default” geometry property of the feature type that is used in the containing **FeatureStyleType** is used. The precise meaning of “default” geometry property is system-dependent. Most frequently, feature types will have only a single geometry property. The format of the **Geometry** element is as follows:

```
<xs:element name="Geometry">
  <xs:complexType>
    <xs:sequence>
      <xs:element ref="ogc:PropertyName"/>
    </xs:sequence>
  </xs:complexType>
</xs:element>
```

The only method available for defining a geometry is to reference a geometry property using the **ogc:PropertyName** element (defined in the WFS 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 (Section 6.5). All symbolizer types can include a **Geometry** element also.

### 11.1.3 Stroke

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

```
<xs:element name="Stroke">
  <xs:complexType>
```

```

<xs:sequence>
  <xs:choice minOccurs="0">
    <xs:element ref="sld:GraphicFill"/>
    <xs:element ref="sld:GraphicStroke"/>
  </xs:choice>
  <xs:element ref="sld:CssParameter" minOccurs="0"
    maxOccurs="unbounded"/>
</xs:sequence>
</xs:complexType>
</xs: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 **LineSymbolizer** and other symbolizers), and its absence means that no stroke is to be rendered.

There are three basic types of strokes: solid-color, **GraphicFill** (stipple), and repeated linear **GraphicStroke**. A repeated linear graphic is plotted linearly and has its graphic 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 symbolizer will render a solid color.

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

```

<xs:element name="CssParameter" type="sld:ParameterValueType"/>

<xs:complexType name="ParameterValueType" mixed="true">
  <xs:choice minOccurs="0" maxOccurs="unbounded">
    <xs:element ref="ogc:expression"/>
  </xs:choice>
</xs: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:

```

<CssParameter name="stroke-width">3</CssParameter>

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

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

```

```
<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 **CssParameter** element.

The “**stroke**” **CssParameter** 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**” **CssParameter** element is absent, the default color is defined to be black (“**#000000**”) in the context of the **LineSymbolizer**.

The “**stroke-opacity**” **CssParameter** 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**” **CssParameter** element gives the absolute width (thickness) of a stroke in pixels encoded as a float. (Arguably, more units could be provided for encoding sizes, such as millimeters or typesetter's points.) 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**” **CssParameter** 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**” **CssParameter** element encodes a dash pattern as a series of space separated floats. The first number gives the length in pixels 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**” **CssParameter** 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:

```

<xs:element name="GraphicFill">
  <xs:complexType>
    <xs:sequence>
      <xs:element ref="sld:Graphic"/>
    </xs:sequence>
  </xs:complexType>
</xs: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 Section 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:

```

<xs:element name="GraphicStroke">
  <xs:complexType>
    <xs:sequence>
      <xs:element ref="sld:Graphic"/>
    </xs:sequence>
  </xs:complexType>
</xs: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.

#### 11.1.4 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:

```

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

```

The resulting map portrayal based upon the above rule is:



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

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

Some delinquent XML parses may require “<Stroke></Stroke>” instead of “<Stroke/>”.

## 11.2. Polygon Symbolizer

### 11.2.1 Format

A **PolygonSymbolizer** is used draw a polygon (or other area-type geometries), including filling its interior and stroking its border (outline). It has the following simple definition:

```
<xs:element name="PolygonSymbolizer">
  <xs:complexType>
    <xs:sequence>
      <xs:element ref="sld:Geometry" minOccurs="0"/>
      <xs:element ref="sld:Fill" minOccurs="0"/>
      <xs:element ref="sld:Stroke" minOccurs="0"/>
    </xs:sequence>
  </xs:complexType>
</xs:element>
```

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

### 11.2.2 Fill

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

```
<xs:element name="Fill">
  <xs:complexType>
    <xs:sequence>
      <xs:element ref="sld:GraphicFill" minOccurs="0"/>
      <xs:element ref="sld:CssParameter" minOccurs="0"
        maxOccurs="unbounded"/>
    </xs:sequence>
  </xs:complexType>
</xs: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 **CssParameter** elements are discussed in conjunction with the **Stroke** element in Section 11.1.3. Here, the **CssParameter** names are “**fill**” instead of “**stroke**” and “**fill-opacity**” instead of “**stroke-opacity**”. None of the other **CssParameters** 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 **PolygonSymbol** as follows:

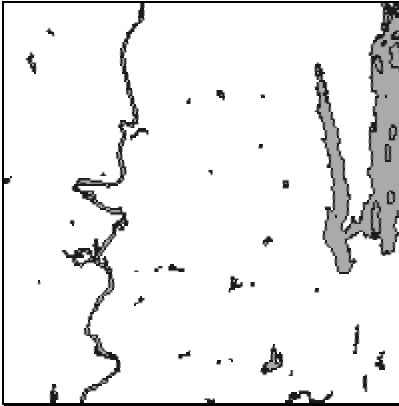
```
<PolygonSymbolizer>
  <Geometry>
    <ogc:PropertyName>the_area</ogc:PropertyName>
  </Geometry>
  <Fill>
    <CssParameter name="fill">#aaaaff</CssParameter>
  </Fill>
  <Stroke>
```

```

        <CssParameter name="stroke">#0000aa</CssParameter>
      </Stroke>
    </PolygonSymbolizer>

```

The resulting map portrayal based upon the above rule is:



A very simple styling with default parameters would be:

```

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

```

### 11.3. Point Symbolizer

#### 11.3.1 Format

A **PointSymbolizer** is used to draw “graphic” at a point. It has the following simple definition:

```

<xs:element name="PointSymbolizer">
  <xs:complexType>
    <xs:sequence>
      <xs:element ref="sld:Geometry" minOccurs="0"/>
      <xs:element ref="sld:Graphic" minOccurs="0"/>
    </xs:sequence>
  </xs:complexType>
</xs:element>

```

The **Geometry** element is discussed in Section 11.1.2. In this case, if a line, polygon, or raster geometry is used with this symbolizer, then the semantic is to use the centroid of the geometry, or any similar representative point. The **Graphic** element is described below.

### 11.3.2 Graphic

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

```
<xs:element name="Graphic">
  <xs:complexType>
    <xs:sequence>
      <xs:choice minOccurs="0" maxOccurs="unbounded">
        <xs:element ref="sld:ExternalGraphic"/>
        <xs:element ref="sld:Mark"/>
      </xs:choice>
      <xs:sequence>
        <xs:element ref="sld:Opacity" minOccurs="0"/>
        <xs:element ref="sld:Size" minOccurs="0"/>
        <xs:element ref="sld:Rotation" minOccurs="0"/>
      </xs:sequence>
    </xs:sequence>
  </xs:complexType>
</xs:element>

<xs:element name="ExternalGraphic">
  <xs:complexType>
    <xs:sequence>
      <xs:element ref="sld:OnlineResource"/>
      <xs:element ref="sld:Format"/>
    </xs:sequence>
  </xs:complexType>
</xs:element>
<xs:element name="Format" type="xs:string"/>

<xs:element name="Opacity" type="sld:ParameterValueType"/>
<xs:element name="Size" type="sld:ParameterValueType"/>
<xs:element name="Rotation" type="sld:ParameterValueType"/>
```

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. The **OnlineResource** sub-element (discussed in Section 7.3) 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 **Opacity** element gives the opacity of to use for rendering the graphic. It has the same semantics as the “**stroke-opacity**” and “**fill-opacity**” **CssParameter** 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) 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 **Mark** element of a **Graphic** defines a “shape” which has coloring applied to it. The element is defined as follows:

```
<xs:element name="Mark">
  <xs:complexType>
    <xs:sequence>
      <xs:element ref="sld:WellKnownName" minOccurs="0"/>
      <xs:element ref="sld:Fill" minOccurs="0"/>
      <xs:element ref="sld:Stroke" minOccurs="0"/>
    </xs:sequence>
  </xs:complexType>
</xs:element>
```

```

    </xs:sequence>
  </xs:complexType>

  <xs:element name="WellKnownName" type="xs:string"/>

```

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 Sections 11.2.2 and 11.1.3, respectively.

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 useable result in a best-effort rendering environment, provided that a simple **Mark** is included at the bottom of the list of sources for every **Graphic**.

### 11.3.3 Examples

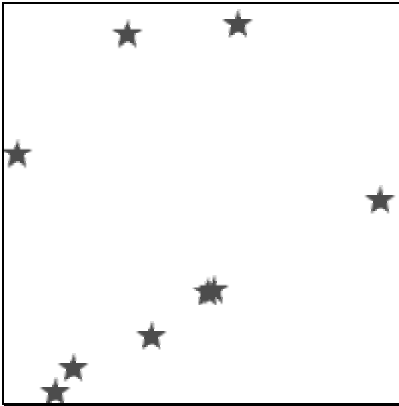
Consider the example of symbolizing ‘Hospital’ features that have a point geometry property called “**locatedAt**” as solid red stars centered on the hospital locations. The **PointSymbolizer** can be represented using a mark as follows:

```

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

```

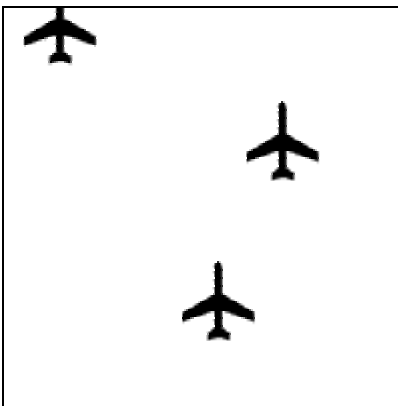
The resulting map portrayal based upon the preceding rule is:



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

```
<PointSymbolizer>
  <Graphic>
    <ExternalGraphic>
      <OnlineResource
        xmlns:xlink="http://www.w3.org/1999/xlink"
        xlink:type="simple"
        xlink:href="http://www.vendor.com/geosym/2267.svg"/>
      <Format>image/svg+xml</Format>
    </ExternalGraphic>
    <ExternalGraphic>
      <OnlineResource
        xmlns:xlink="http://www.w3.org/1999/xlink"
        xlink:type="simple"
        xlink:href="http://www.vendor.com/geosym/2267.png"/>
      <Format>image/png</Format>
    </ExternalGraphic>
    <Mark/>
    <Size>15.0</Size>
  </Graphic>
</PointSymbolizer>
```

The resulting map portrayal based upon the preceding rule is:



## 11.4. Text Symbolizer

### 11.4.1 Format

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

```
<xs:element name="TextSymbolizer">
  <xs:complexType>
    <xs:sequence>
      <xs:element ref="sld:Geometry" minOccurs="0"/>
      <xs:element ref="sld:Label" minOccurs="0"/>
      <xs:element ref="sld:Font" minOccurs="0"/>
      <xs:element ref="sld:LabelPlacement" minOccurs="0"/>
      <xs:element ref="sld:Halo" minOccurs="0"/>
      <xs:element ref="sld:Fill" minOccurs="0"/>
    </xs:sequence>
  </xs:complexType>
</xs:element>
```

These elements are discussed below, except for the **Geometry** and **Fill** elements, which were discussed in Sections 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 Section 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 Section 11.3.1 for point or Section 11.1.2 for line.

### 11.4.2 Label

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

```
<xs:element name="Label" type="sld: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 **TextSymbol**, then no text will be rendered.

### 11.4.3 Font

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

```
<xs:element name="Font">
  <xs:complexType>
    <xs:sequence>
      <xs:element ref="sld:CssParameter" minOccurs="0"
        maxOccurs="unbounded"/>
    </xs:sequence>
  </xs:complexType>
</xs:element>
```

Four types of **CssParameter** are allowed, “**font-family**”, “**font-style**”, “**font-weight**”, and “**font-size**”.

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

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

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

The “**font-size**” **CssParameter** 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 or a line string and is defined as follows:

```
<xs:element name="LabelPlacement">
  <xs:complexType>
    <xs:choice>
      <xs:element ref="sld:PointPlacement"/>
      <xs:element ref="sld:LinePlacement"/>
    </xs:choice>
  </xs:complexType>
</xs:element>
```



```

    </xs:complexType>
  </xs:element>

  <xs:element name="PointPlacement">
    <xs:complexType>
      <xs:sequence>
        <xs:element ref="sld:AnchorPoint" minOccurs="0"/>
        <xs:element ref="sld:Displacement" minOccurs="0"/>
        <xs:element ref="sld:Rotation" minOccurs="0"/>
      </xs:sequence>
    </xs:complexType>
  </xs:element>

  <xs:element name="LinePlacement">
    <xs:complexType>
      <xs:sequence>
        <xs:element ref="sld:PerpendicularOffset" minOccurs="0"/>
      </xs:sequence>
    </xs:complexType>
  </xs:element>

```

This represents only a first attempt at addressing the problem in SLD and may be inadequate.

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 **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:

```

<xs:element name="AnchorPoint">
  <xs:complexType>
    <xs:sequence>
      <xs:element ref="sld:AnchorPointX"/>
      <xs:element ref="sld:AnchorPointY"/>
    </xs:sequence>
  </xs:complexType>
</xs:element>

<xs:element name="AnchorPointX" type="sld:ParameterValueType"/>
<xs:element name="AnchorPointY" type="sld: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, Y=0.5, which is at the middle height of the left-hand side of the label. 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 and is defined as:

```
<xs:element name="Displacement">
  <xs:complexType>
    <xs:sequence>
      <xs:element ref="sld:DisplacementX"/>
      <xs:element ref="sld:DisplacementY"/>
    </xs:sequence>
  </xs:complexType>
</xs:element>

<xs:element name="DisplacementX" type="sld:ParameterValueType"/>
<xs:element name="DisplacementY" type="sld:ParameterValueType"/>
```

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.

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 Section 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:

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

The distance is in pixels 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:

```
<xs:element name="Halo">
  <xs:complexType>
    <xs:sequence>
      <xs:element ref="sld:Radius" minOccurs="0"/>
      <xs:element ref="sld:Fill" minOccurs="0"/>
    </xs:sequence>
  </xs:complexType>
</xs:element>

<xs:element name="Radius" type="sld: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 **TextSymbolizer**, then no halo will be rendered.

#### 11.4.6 Example

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

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

### 11.5. Raster Symbolizer

#### 11.5.1 Format

The **RasterSymbolizer** describes how to render raster/matrix-coverage data (e.g., satellite photos, DEMs). It is defined as follows:

```
<xs:element name="RasterSymbolizer">
  <xs:complexType>
    <xs:sequence>
      <xs:element ref="sld:Geometry" minOccurs="0"/>
      <xs:element ref="sld:Opacity" minOccurs="0"/>
      <xs:element ref="sld:ChannelSelection" minOccurs="0"/>
      <xs:element ref="sld:OverlapBehavior" minOccurs="0"/>
      <xs:element ref="sld:ColorMap" minOccurs="0"/>
      <xs:element ref="sld:ContrastEnhancement" minOccurs="0"/>
      <xs:element ref="sld:ShadedRelief" minOccurs="0"/>
      <xs:element ref="sld:ImageOutline" minOccurs="0"/>
    </xs:sequence>
  </xs:complexType>
</xs:element>
```

```

    </xs:sequence>
  </xs:complexType>
</xs: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:

```

<xs:element name="ChannelSelection">
  <xs:complexType>
    <xs:choice>
      <xs:sequence>
        <xs:element ref="sld:RedChannel"/>
        <xs:element ref="sld:GreenChannel"/>
        <xs:element ref="sld:BlueChannel"/>
      </xs:sequence>
      <xs:element ref="sld:GrayChannel"/>
    </xs:choice>
  </xs:complexType>
</xs:element>

<xs:element name="RedChannel" type="sld:SelectedChannelType"/>
<xs:element name="GreenChannel" type="sld:SelectedChannelType"/>
<xs:element name="BlueChannel" type="sld:SelectedChannelType"/>
<xs:element name="GrayChannel" type="sld:SelectedChannelType"/>

<xs:complexType name="SelectedChannelType">
  <xs:sequence>
    <xs:element ref="sld:SourceChannelName"/>
    <xs:element ref="sld:ContrastEnhancement" minOccurs="0"/>
  </xs:sequence>
</xs:complexType>

<xs:element name="SourceChannelName" type="xs: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.

```
<xs:element name="OverlapBehavior">
  <xs:complexType>
    <xs:choice>
      <xs:element ref="sld:LATEST_ON_TOP"/>
      <xs:element ref="sld:EARLIEST_ON_TOP"/>
      <xs:element ref="sld:AVERAGE"/>
      <xs:element ref="sld:RANDOM"/>
    </xs:choice>
  </xs:complexType>
</xs:element>

<xs:element name="LATEST_ON_TOP">
  <xs:complexType/>
</xs:element>

<xs:element name="EARLIEST_ON_TOP">
  <xs:complexType/>
</xs:element>

<xs:element name="AVERAGE">
  <xs:complexType/>
</xs:element>

<xs:element name="RANDOM">
  <xs:complexType/>
</xs: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.

```
<xs:element name="ColorMap">
  <xs:complexType>
    <xs:choice minOccurs="0" maxOccurs="unbounded">
      <xs:element ref="sld:ColorMapEntry"/>
    </xs:choice>
  </xs:complexType>
</xs:element>

<xs:element name="ColorMapEntry">
  <xs:complexType>
    <xs:attribute name="color" type="xs:string" use="required"/>
    <xs:attribute name="opacity" type="xs:double"/>
```

```

    <xs:attribute name="quantity" type="xs:double"/>
    <xs:attribute name="label" type="xs:string"/>
  </xs:complexType>
</xs: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 matrixes 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:

```

<xs:element name="ContrastEnhancement">
  <xs:complexType>
    <xs:sequence>
      <xs:choice minOccurs="0">
        <xs:element ref="sld:Normalize"/>
        <xs:element ref="sld:Histogram"/>
      </xs:choice>
      <xs:element ref="sld:GammaValue" minOccurs="0"/>
    </xs:sequence>
  </xs:complexType>
</xs:element>

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

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

<xs:element name="GammaValue" type="xs: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 (value greater than **1.0**) or dim (value 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:

```
<xs:element name="ShadedRelief">
  <xs:complexType>
    <xs:sequence>
      <xs:element ref="sld:BrightnessOnly" minOccurs="0"/>
      <xs:element ref="sld:ReliefFactor" minOccurs="0"/>
    </xs:sequence>
  </xs:complexType>
</xs:element>

<xs:element name="BrightnessOnly" type="xs:boolean"/>

<xs:element name="ReliefFactor" type="xs:double"/>
```

Exact parameters of the shading are system-dependent (for now). If the **BrightnessOnly** flag is “0” (false, default), the shading is applied to the layer being rendered as the current **RasterSymbol**. If **BrightnessOnly** is “1” (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 **LineStringSymbol** or **PolygonSymbol**. It is defined as:

```
<xs:element name="ImageOutline">
  <xs:complexType>
    <xs:choice>
      <xs:element ref="sld:LineSymbolizer"/>
      <xs:element ref="sld:PolygonSymbolizer"/>
    </xs:choice>
  </xs:complexType>
</xs: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 **PolygonSymbolizer Fill** to allow the main-raster data be visible through the fill.

### 11.5.3 Examples

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

```
<RasterSymbolizer>
  <Opacity>1.0</Opacity>
  <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/>
</RasterSymbolizer>

```

Here is a rather artificial mutli-band raster symbol:

```

<RasterSymbolizer>
  <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>
</RasterSymbolizer>

```



```

    <OverlapBehavior>
      <LATEST_ON_TOP/>
    </OverlapBehavior>
    <ContrastEnhancement>
      <GammaValue>1.0</GammaValue>
    </ContrastEnhancement>
  </RasterSymbolizer>

```

## 11.6. Systems With Limited Capabilities

Systems with limited capabilities can use a “best-effort” approach to styling and rendering maps according to an SLD. For graphical capabilities, all systems are assumed to be able to render solid-color lines, solid-color fills, and simple internally-defined colored “marks”.

The best-effort approach is not well defined for activities such as processing **Filters**. If a system cannot process a **Filter** and an SLD includes one, then the wrong results will be obtained.

## 11.7. Integrated SLD Examples

Consider the simple case of supplying a user-defined symbolization to features of a single feature-type. Furthermore, let us assume that a NamedLayer already exists that represents those features. The features could be displayed using a NamedStyle by encoding a styled layer in the following manner:

```

<StyledLayerDescriptor version="1.0.0">
  <NamedLayer>
    <Name>Rivers</Name>
    <NamedStyle>
      <Name>CenterLine</Name>
    </NamedStyle>
  </NamedLayer>
</StyledLayerDescriptor>

```

The above SLD describes the styled layer **Rivers:CenterLine**. A client may wish to have the features in the “**Rivers**” named layer appear as a light-blue line, five pixels wide. Assuming that the features have a geometric property called “**center-line**” with line strings for values, then this can be done using a user-defined style, encoded as a **UserStyle** element.

```

<StyledLayerDescriptor version="1.0.0">
  <NamedLayer>
    <Name>Rivers</Name>
    <UserStyle>
      <Title>Blue River</Title>
      <FeatureTypeStyle>
        <Rule>
          <LineSymbolizer>
            <Geometry>
              <ogc:PropertyName>center-line</ogc:PropertyName>

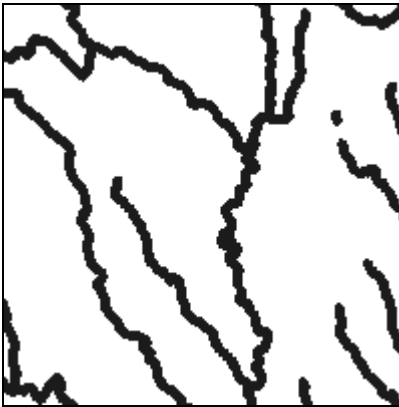
```

```

        </Geometry>
        <Stroke>
          <CssParameter name="stroke">#aaaaff</CssParameter>
          <CssParameter
            name="stroke-width">5.0</CssParameter>
        </Stroke>
      </LineSymbolizer>
    </Rule>
  </FeatureTypeStyle>
</UserStyle>
</NamedLayer>
</StyledLayerDescriptor>

```

The resulting map portrayal based upon the preceding rule is:



Note that the **UserStyle** element replaces the **NamedStyle** element, but that the overall layer/style structure remains. A symbol specifies how a feature is symbolized, a style determines which features are passed to which symbols. The **UserStyle** is a degenerate style that just passes features onto the contained symbols. When placing a **UserStyle** inside a **NamedLayer** there is no explicit mention of the types of feature being symbolized. Of course, the types are known to the WMS because a **NamedLayer** hides an internal definition.

The **Title** element in the **UserStyle** element means that it is possible to associate a user-defined title with any styled layer that is, even if only in part, user-defined. Since a WMS already has access to a title for named styles, this ensures that all styled layers can have a title associated with them. This can be used to help support legend functionality.

If we make the further assumption that the “**Rivers**” named layer just contains features of the feature-type “**main-river**”, then we can rewrite the above using a **UserLayer** element:

```

<StyledLayerDescriptor version="1.0.0">
  <UserLayer>
    <RemoteOWS>
      <Service>WFS</Service>
      <OnlineResource

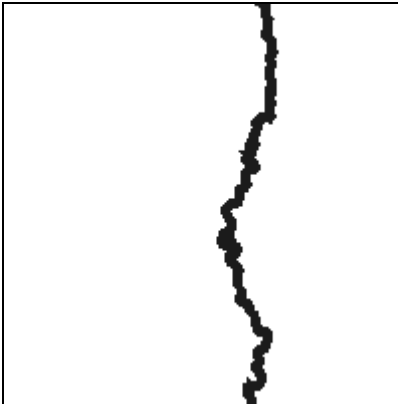
```

```

        xmlns:xlink="http://www.w3.org/1999/xlink"
        xlink:type="simple"
        xlink:href="http://some.site.com/WFS?"/>
    </RemoteOWS>
    <LayerFeatureConstraints>
        <FeatureTypeConstraint>
            <FeatureTypeName>main-river</FeatureTypeName>
        </FeatureTypeConstraint>
    </LayerFeatureConstraints>
    <UserStyle>
        <Title>Blue river</Title>
        <FeatureTypeStyle>
            <Rule>
                <LineSymbolizer>
                    <Geometry>
                        <ogc:PropertyName>center-line</ogc:PropertyName>
                    </Geometry>
                    <Stroke>
                        <CssParameter name="stroke">#aaaaff</CssParameter>
                        <CssParameter
                            name="stroke-width">5.0</CssParameter>
                    </Stroke>
                </LineSymbolizer>
            </Rule>
        </FeatureTypeStyle>
    </UserStyle>
</UserLayer>
</StyledLayerDescriptor>

```

The resulting map portrayal based upon the preceding rule is:



The **FeatureTypeConstraint** element is used to define the contents of the **UserLayer** in terms of a single feature-type. This somewhat restrictive definition of a **UserLayer** will be lifted in later versions of the specification. One might wish to define a **UserLayer** that only contains a subset of “Road” features, for example all those with a classification of “Interstate”. Alternatively one might wish to define a **UserLayer** containing features of various feature-types with a common thematic element, for example “Rivers” and “Streams” and “Lakes” pulled together into a “Hydrography” **UserLayer**.

Note that there is no need to name the **UserLayers** and **UserStyles** because the WMS retains no knowledge of them after the request. Currently there is no interoperable mechanism to define new named layers and named styles known to the WMS. The ability to do so would provide a interoperable mechanism for part of the initial configuration of a WMS, but it also introduces complex issues related to security.

Finally, let us go back to the ‘cased’ roads example. This could be encoded using **NamedLayers** and **NamedStyles** in an SLD as follows:

```
<StyledLayerDescriptor version="1.0.0">
  <NamedLayer>
    <Name>Roads</Name>
    <NamedStyle>
      <Name>Casing</Name>
    </NamedStyle>
    <NamedStyle>
      <Name>CenterLine</Name>
    </NamedStyle>
  </NamedLayer>
</StyledLayerDescriptor>
```

If the client wished to alter the color of the thick ‘casing’ then a **UserStyle** could be used to substitute for the “Casing” **NamedStyle**:

```
<StyledLayerDescriptor version="1.0.0">
  <NamedLayer>
    <Name>Roads</Name>
    <UserStyle>
      <FeatureStyleType>
        <Rule>
          <LineSymbolizer>
            <Geometry>
              <ogc:PropertyName>center-line</ogc:PropertyName>
            </Geometry>
            <Stroke>
              <CssParameter name="stroke">#aaaaff</CssParameter>
              <CssParameter
                name="stroke-width">5.0</CssParameter>
            </Stroke>
          </LineSymbolizer>
        </Rule>
      </FeatureTypeStyle>
    </UserStyle>
    <NamedStyle>
      <Name>CenterLine</Name>
    </NamedStyle>
  </NamedLayer>
</StyledLayerDescriptor>
```

The specification does not allow an existing named style to be ‘tweaked’ by just overriding individual parameters. Thus, the new ‘user-defined’ casing must supply all parameters that vary from their default value. So even if all we wished to do was alter

the color of the casing, we must now specify a stroke-width as well. In addition we must know something about the “Road” feature-type, specifically that it has a geometric property called “center-line” with a line string for a value. We can provide a **UserStyle** in place of the “CenterLine” **NamedStyle**:

```
<StyledLayerDescriptor version="1.0.0">
  <NamedLayer>
    <Name>Roads</Name>
    <UserStyle>
      <FeatureTypeStyle>
        <Rule>
          <LineSymbolizer>
            <Geometry>
              <ogc:PropertyName>center-line</ogc:PropertyName>
            </Geometry>
            <Stroke>
              <CssParameter name="stroke">#ff0000</CssParameter>
              <CssParameter
                name="stroke-width">5.0</CssParameter>
            </Stroke>
          </LineSymbolizer>
        </Rule>
      </FeatureTypeStyle>
    </UserStyle>
    <UserStyle>
      <FeatureTypeStyle>
        <Rule>
          <LineSymbolizer>
            <Geometry>
              <ogc:PropertyName>center-line</ogc:PropertyName>
            </Geometry>
            <Stroke>
              <CssParameter name="stroke">#ffffff</CssParameter>
              <CssParameter
                name="stroke-width">3.0</CssParameter>
            </Stroke>
          </LineSymbolizer>
        </Rule>
      </FeatureTypeStyle>
    </UserStyle>
  </NamedLayer>
</StyledLayerDescriptor>
```

This SLD defines two styled layers, which are drawn in Z-order. This ensures that the roads appear properly cased. However, the **UserStyle** element itself can contain multiple symbols. The interpretation is that the symbols are drawn in order for each feature in the layer. The following encoding defines a single styled layer in which each feature is displayed with two symbols:

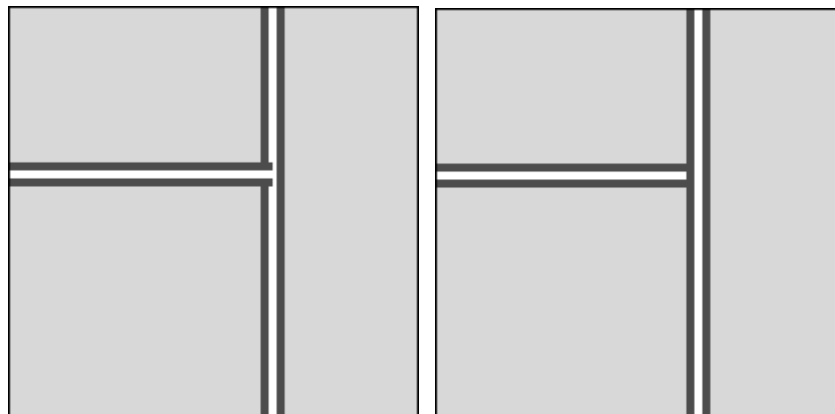
```
<StyledLayerDescriptor version="1.0.0">
  <NamedLayer>
    <Name>Roads</Name>
    <UserStyle>
```

```

<FeatureTypeStyle>
  <Rule>
    <LineSymbolizer>
      <Geometry>
        <ogc:PropertyName>center-line</ogc:PropertyName>
      </Geometry>
      <Stroke>
        <CssParameter name="stroke">#ff0000</CssParameter>
        <CssParameter
          name="stroke-width">5.0</CssParameter>
      </Stroke>
    </LineSymbolizer>
    <LineSymbolizer>
      <Geometry>
        <ogc:PropertyName>center-line</ogc:PropertyName>
      </Geometry>
      <Stroke>
        <CssParameter name="stroke">#ffffff</CssParameter>
        <CssParameter
          name="stroke-width">3.0</CssParameter>
      </Stroke>
    </LineSymbolizer>
  </Rule>
</FeatureTypeStyle>
</UserStyle>
</NamedLayer>
</StyledLayerDescriptor>

```




This would process each road feature one at a time, with the road drawn first with a thick red line and then a thin white line. This would give rise to one of the following, depending on which order the features are processed in a layer:



## 12 Map Legends

Legends are normally included with maps to indicate to the user how various features are represented in the map. It is therefore important to be able to produce a legend on a map-display client for styles that are represented in SLD format.

The structuring of SLD **UserStyles** into **FeatureTypeStyles** and **Rules** provides convenient packaging for this purpose, since rules identify each different kind of graphic symbolization that may be present in a map. Given the information in an SLD **UserStyle**, a map-viewer client could generate a legend entry for a layer in the following format:

<b>Roads Layer</b>	
	<b>Highways</b>
	<b>Collector Roads</b>
	<b>Minor Roads</b>

The icon symbols are graphics (images) showing how the rule is rendered. Abstracts or conditions could be displayed by clicking on the titles, etc. The exact presentation of this information is at the discretion of the viewer client.

Generating this kind of display may involve a significant amount of processing on the client. The client will need to examine the selected SLD style and determine which rules apply at the currently used map scale. Then, it will generate something analogous to the above ‘form’ with the Layer and Rule titles in HTML or Java Swing or whatever the environment, using references for the images. Alternatively, the job of producing a meaningful legend entry for a style could be passed on to the server side in a similar way to how it is done now with WMS **LegendURLs**.

The image references make use of the **GetLegendGraphic** operation of the SLD-WMS interface, with a separate reference for each image. The parameterization of the operation needs to be “overloaded” in the same way that parameters for the **GetMap** with an SLD are overloaded in order to handle the different kinds of clients. I.e., there is an XML-based HTTP-POST method for executing **GetMap** (which is the way that SLD is really intended to be used but nobody implements it), and there are HTTP-GET methods using **SLD=** and **SLD\_BODY=** parameters to reference/transport the SLD for use in either “literal” or “library” mode (depending on whether the **LAYERS=** parameter is present).

The GET parameters of the **GetLegendGraphic** operation are defined as follows:

<b>Parameter</b>	<b>Required</b>	<b>Description</b>
<b>VERSION</b>	Required	Version as required by OGC interfaces.

<b>REQUEST</b>	Required	Value must be “ <b>GetLegendRequest</b> ”.
<b>LAYER</b>	Required	Layer for which to produce legend graphic.
<b>STYLE</b>	Optional	Style of layer for which to produce legend graphic. If not present, the default style is selected. The style may be any valid style available for a layer, including non-SLD internally-defined styles.
<b>FEATURETYPE</b>	Optional	Feature type for which to produce the legend graphic. This is not needed if the layer has only a single feature type.
<b>RULE</b>	Optional	Rule of style to produce legend graphic for, if applicable. In the case that a style has multiple rules but no specific rule is selected, then the map server is obligated to produce a graphic that is representative of all of the rules of the style.
<b>SCALE</b>	Optional	In the case that a <b>RULE</b> is not specified for a style, this parameter may assist the server in selecting a more appropriate representative graphic by eliminating internal rules that are out-of-scope. This value is a standardized scale denominator, defined in Section 10.2
<b>SLD</b>	Optional	This parameter specifies a reference to an external SLD document. It works in the same way as the <b>SLD=</b> parameter of the WMS <b>GetMap</b> operation.
<b>SLD_BODY</b>	Optional	This parameter allows an SLD document to be included directly in an HTTP-GET request. It works in the same way as the <b>SLD_BODY=</b> parameter of the WMS <b>GetMap</b> operation.
<b>FORMAT</b>	Required	This gives the MIME type of the file format in which to return the legend graphic. Allowed values are the same as for the <b>FORMAT=</b> parameter of the WMS <b>GetMap</b> request.
<b>WIDTH</b>	Optional	This gives a hint for the width of the returned graphic in pixels. Vector-graphics can use this value as a hint for the level of detail to include.
<b>HEIGHT</b>	Optional	This gives a hint for the height of the returned graphic in pixels.
<b>EXCEPTIONS</b>	Optional	This gives the MIME type of the format in which to return exceptions. Allowed values are the same as for the <b>EXCEPTIONS=</b> parameter of the WMS <b>GetMap</b> request.



The **GetLegendGraphic** operation itself is optional for an SLD-enabled WMS. It provides a general mechanism for acquiring legend symbols, beyond the **LegendURL** reference of WMS Capabilities. Servers supporting the **GetLegendGraphic** call might code **LegendURL** references as **GetLegendGraphic** for interface consistency. Vendor-specific parameters may be added to **GetLegendGraphic** requests and all of the usual OGC-interface options and rules apply. No XML-POST method for **GetLegendGraphic** is presently defined.

Here is an example invocation:

```
http://www.vendor.com/wms.cgi?
  VERSION=1.1.0&
  REQUEST=GetLegendGraphic&
  LAYER=ROADL_1M%3Alocal_data&
  STYLE=my_style&
  RULE=highways
  SLD=http%3A%2F%2Fwww.sld.com%2Fstyles%2Fkpp01.xml
  WIDTH=16&
  HEIGHT=16&
  FORMAT=image%2Fgif&
```

which would produce a 16x16 icon for the **Rule** named “**highways**” defined within layer “**ROADL\_1M:local\_data**” in the SLD. The list of available formats for legend graphics and exceptions can be assumed to be the same as are available for a map in the WMS **GetMap** request.

An alternative approach to using a **GetLegendGraphic** operation would be for the viewer client to render a style sample directly itself using the style description. This would save some interactions between the client and server and would allow the viewer client to present consistent sample shapes (across remote map servers from different vendors), although the legend graphics might look different from the graphics actually rendered in the map since the viewer and server may have different rendering engines and different graphical capabilities.

The **LegendGraphic** element of an SLD **Rule** (defined in Section 10.1) actually only has a limited role in building legends. For vector types, a map server would normally render a standard vector geometry (such as a box) with the given symbolization for a rule. But for some layers, such as for Digital Elevation Model (DEM) data, there is not really a “standard” geometry that can be rendered in order to get a good representative image. So, this is what the **LegendGraphic** SLD element is intended for, to provide a substitute representative image for a **Rule**. For example, it might reference a remote URL for a DEM layer called “**GTOPO30**”:

```
http://www.vendor.com/sld/icons/COLORMAP_GTOPO30.png
```

## 13 Symbology Management

This section describes methods to store and retrieve user-defined styles to and from a map server using SLD format.

### 13.1. GetStyles

The **GetStyles** operation is used to retrieve user-defined styles from a WMS. The parameters of the HTTP-GET method are defined as follows:

Parameter	Required	Description
<b>VERSION</b>	Required	Version as required by OGC interfaces.
<b>REQUEST</b>	Required	Value must be “ <b>GetStyles</b> ”.
<b>LAYERS</b>	Required	Comma-separated list of named layers for which to retrieve style descriptions.
<b>SLDVER</b>	Optional	The SLD version requested for the SLD document. The default is to return the highest version supported by the server. If a version is requested that the server does not support, then the next lower supported version is supported by the server is returned, or the server’s lowest supported version. This is similar to WMS version negotiation.

All of the styles of multiple layers may be retrieved at a time for efficiency. On successful extraction, an SLD document is returned of MIME type “**application/vnd.ogc.sld+xml**”. All requested styles that can in fact be described by SLD will be returned as **UserStyle** elements, and styles that cannot be will returned as **NamedStyle** elements. (It is understood that map servers may have internally-encoded styling that cannot be properly described by SLD.) The map server should normally return the layer information as **NamedLayers** for better integration with the **PutStyles** operation. A **UserLayer** will provide more information about the internal organization of a layer for greater WFS/WCS integration.

On error, a standard “**application/vnd.ogc.se\_xml**” service-exception document will be returned.

### 13.2. PutStyles

The **PutStyles** operation is used to store user-defined styles and user-defined layers into a WMS. Many styles for many different layers may be stored. On successful insertion, these styles will subsequently be available from the WMS for use as named styles. The parameters of the HTTP-GET method are defined as follows:

Parameter	Required	Description
<b>VERSION</b>	Required	Version as required by OGC interfaces.
<b>REQUEST</b>	Required	Value must be “ <b>PutStyles</b> ”.
<b>MODE</b>	Required	This gives the mode of the ‘put’: either “ <b>InsertAndReplace</b> ” or “ <b>ReplaceAll</b> ”. In <b>InsertAndReplace</b> mode, all new styles for a layer are inserted and all existing styles which are defined in the SLD are replaced. In <b>ReplaceAll</b> mode, all existing styles for a layer are logically deleted, and then the SLD-defined styles are inserted. This is similar to <b>InsertAndReplace</b> mode, except that all styles not in the SLD are deleted.
<b>SLD</b>	Optional	This parameter specifies a reference to an external SLD document. It works in the same way as the <b>SLD=</b> parameter of the WMS <b>GetMap</b> operation.
<b>SLD_BODY</b>	Optional	This parameter allows an SLD document to be included directly in an HTTP-GET request. It works in the same way as the <b>SLD_BODY=</b> parameter of the WMS <b>GetMap</b> operation.

The format is implied to be SLD. Both the **SLD=** and **SLD\_BODY=** parameters are individually marked as “Optional”, but one of them must be used to pass the SLD. This operation could also hypothetically be used to define new user-defined layers, but that is not the immediate intention in this version of the SLD Specification. No HTTP-POST method for the **PutStyles** operation is defined at this time.

The **InsertAndReplace** and **ReplaceAll** modes are provided to handle the common usage cases of either tweaking an individual style or of performing a bulk update of all styles for a layer or layers. A “lost-update” problem may be encountered when performing a bulk update on styles that were retrieved with a previous **GetStyles** operation. Any updates to the styles that were made by some other user in the meantime may be lost. Lost updates are a general problem in distributed systems and the general solution is to use some kind of locking mechanism, but no solution is provided here.

All SLD **NamedStyles** that are present in the SLD in a **PutStyles** operation in **InsertAndReplace** mode are ignored. (These may be present as a result of a previous **GetStyles** query for which there was no suitable SLD description of some styles.) In the **ReplaceAll** mode of **PutStyles**, the current definition of every **NamedStyle** is retained. (If no reference at all is provided, then styles are deleted in this mode.)

The semantics for updating the default style are as follows. If a style for a layer is marked as being default in the SLD, then that style will become the new default, superseding the existing default style in the map server. If **ReplaceAll** mode is used and the existing default style for a layer is implicitly deleted but no new style is inserted

from the SLD that is marked as being the new default, then the action to be taken is system-specific. A WMS must always have a default style for every layer.

Any user-defined layers that are present in the given SLD will cause a layer to be created inside of the map server of the given name and definition. If the layer already exists, it will be replaced with the new definition. A map server shall be assumed to support user-defined layer updates if and only if it indicates that it supports the **PutStyles** operation and user-defined layers in its capabilities document.

On successful insertion, a simple return of MIME type **"application/vnd.ogc.success+xml"** will be returned. It is defined according to the following DTD:

```
<!ELEMENT OGC_OWS_Success EMPTY >
```

On error, a standard **"application/vnd.ogc.se\_xml"** service-exception document will be returned. This document is described in the WMS 1.1.0 Specification. The insertion of new styles is formally defined to be "atomic", meaning that either all style updates to all layers are made or none are. In practice, however, some systems may not live up to this formal definition.

## 14 Styling Standards

### 14.1. GeoSym

The functionality of SLD is sufficient to render GeoSym styles. GeoSym makes use of solid, dash-patterned, and repeated-linear graphic strokes; solid and repeated-graphic fills; the rendering of graphic symbols at a point; and the rendering of text.

Some examples follow of GeoSym-compatible styles for VMAP0 feature types. Here is the style for a simple polygon-fill type for Built-Up Areas. The areas are filled with a yellow color.

```
<UserStyle>
  <Name>GEOSYM</Name>
  <FeatureTypeStyle>
    <Rule>
      <PolygonSymbolizer>
        <Fill>
          <CssParameter name="fill">#FFF053</CssParameter>
        </Fill>
      </PolygonSymbolizer>
    </Rule>
  </FeatureTypeStyle>
</UserStyle>
```

Here is a more complex example for Roads. There are four different rules for portraying different kinds of roads. The WFS-Filter conditions are somewhat verbose.

```

<UserStyle>
  <Name>GEOSYM</Name>
  <FeatureTypeStyle>
    <Rule>
      <Name>Operational-Median</Name>
      <Title>Operational, Median</Title>
      <ogc:Filter> <!-- EXS = 28 AND MED = 1 -->
        <ogc:And>
          <ogc:PropertyIsEqualTo>
            <ogc:PropertyName>EXS</ogc:PropertyName>
            <ogc:Literal>28</ogc:Literal>
          </ogc:PropertyIsEqualTo>
          <ogc:PropertyIsEqualTo>
            <ogc:PropertyName>MED</ogc:PropertyName>
            <ogc:Literal>1</ogc:Literal>
          </ogc:PropertyIsEqualTo>
        </ogc:And>
      </ogc:Filter>
      <LineSymbolizer>
        <Stroke>
          <CssParameter name="stroke">#940100</CssParameter>
          <CssParameter name="stroke-width">2.0</CssParameter>
        </Stroke>
      </LineSymbolizer>
    </Rule>

    <Rule>
      <Name>NonOperational-Median</Name>
      <Title>Non-Operational, Median</Title>
      <ogc:Filter> <!-- EXS != 28 AND MED = 1 -->
        <ogc:And>
          <ogc:Not>
            <ogc:PropertyIsEqualTo>
              <ogc:PropertyName>EXS</ogc:PropertyName>
              <ogc:Literal>28</ogc:Literal>
            </ogc:PropertyIsEqualTo>
          </ogc:Not>
          <ogc:PropertyIsEqualTo>
            <ogc:PropertyName>MED</ogc:PropertyName>
            <ogc:Literal>1</ogc:Literal>
          </ogc:PropertyIsEqualTo>
        </ogc:And>
      </ogc:Filter>
      <LineSymbolizer>
        <Stroke>
          <CssParameter name="stroke">#940100</CssParameter>
          <CssParameter name="stroke-width">2.0</CssParameter>
          <CssParameter
            name="stroke-dasharray">5 3</CssParameter>
        </Stroke>
      </LineSymbolizer>
    </Rule>

    <Rule>
      <Name>Operational-NoMedian</Name>
      <Title>Operational, No Median</Title>

```

```

<ogc:Filter> <!-- EXS = 28 AND MED != 1 -->
  <ogc:And>
    <ogc:PropertyIsEqualTo>
      <ogc:PropertyName>EXS</ogc:PropertyName>
      <ogc:Literal>28</ogc:Literal>
    </ogc:PropertyIsEqualTo>
    <ogc:Not>
      <ogc:PropertyIsEqualTo>
        <ogc:PropertyName>MED</ogc:PropertyName>
        <ogc:Literal>1</ogc:Literal>
      </ogc:PropertyIsEqualTo>
    </ogc:Not>
  </ogc:And>
</ogc:Filter>
<LineSymbolizer>
  <Stroke>
    <CssParameter name="stroke">#940100</CssParameter>
    <CssParameter name="stroke-width">1.0</CssParameter>
  </Stroke>
</LineSymbolizer>
</Rule>

<Rule>
  <Name>NonOperational-NoMedian</Name>
  <Title>Non-Operational, No Median</Title>
  <ogc:Filter> <!-- EXS != 28 AND MED != 1 -->
    <ogc:And>
      <ogc:Not>
        <ogc:PropertyIsEqualTo>
          <ogc:PropertyName>EXS</ogc:PropertyName>
          <ogc:Literal>28</ogc:Literal>
        </ogc:PropertyIsEqualTo>
      </ogc:Not>
      <ogc:Not>
        <ogc:PropertyIsEqualTo>
          <ogc:PropertyName>MED</ogc:PropertyName>
          <ogc:Literal>1</ogc:Literal>
        </ogc:PropertyIsEqualTo>
      </ogc:Not>
    </ogc:And>
  </ogc:Filter>
  <LineSymbolizer>
    <Stroke>
      <CssParameter name="stroke">#940100</CssParameter>
      <CssParameter name="stroke-width">1.0</CssParameter>
      <CssParameter
        name="stroke-dasharray">6 2</CssParameter>
    </Stroke>
  </LineSymbolizer>
</Rule>
</FeatureTypeStyle>
</UserStyle>

```

Finally, here is the style for Airport points. Two types of airports are symbolized by different external graphical images in PNG format.

```

<UserStyle>
  <Name>GEOSYM</Name>
  <FeatureTypeStyle>
    <Rule>
      <Name>NonMilitary</Name>
      <Title>Non Military</Title>
      <ogc:Filter> <!-- USE != 8 AND USE != 22 -->
        <ogc:And>
          <ogc:Not>
            <ogc:PropertyIsEqualTo>
              <ogc:PropertyName>USE</ogc:PropertyName>
              <ogc:Literal>8</ogc:Literal>
            </ogc:PropertyIsEqualTo>
          </ogc:Not>
          <ogc:Not>
            <ogc:PropertyIsEqualTo>
              <ogc:PropertyName>USE</ogc:PropertyName>
              <ogc:Literal>22</ogc:Literal>
            </ogc:PropertyIsEqualTo>
          </ogc:Not>
        </ogc:And>
      </ogc:Filter>
      <PointSymbolizer>
        <Graphic>
          <ExternalGraphic>
            <OnlineResource xlink:type="simple"
              xlink:href="http://www.vendor.com/geosym/2267.png"/>
            <Format>image/png</Format>
          </ExternalGraphic>
          <Size>16.0</Size>
        </Graphic>
      </PointSymbolizer>
    </Rule>

    <Rule>
      <Name>Military</Name>
      <Title>Military</Title>
      <ogc:Filter> <!-- USE = 8 OR USE = 22 -->
        <ogc:Or>
          <ogc:PropertyIsEqualTo>
            <ogc:PropertyName>USE</ogc:PropertyName>
            <ogc:Literal>8</ogc:Literal>
          </ogc:PropertyIsEqualTo>
          <ogc:PropertyIsEqualTo>
            <ogc:PropertyName>USE</ogc:PropertyName>
            <ogc:Literal>22</ogc:Literal>
          </ogc:PropertyIsEqualTo>
        </ogc:Or>
      </ogc:Filter>
      <PointSymbolizer>
        <Graphic>
          <ExternalGraphic>
            <OnlineResource xlink:type="simple"
              xlink:href="http://www.vendor.com/geosym/2269.png"/>
            <Format>image/png</Format>
          </ExternalGraphic>
        </Graphic>
      </PointSymbolizer>
    </Rule>
  </FeatureTypeStyle>
</UserStyle>

```

```

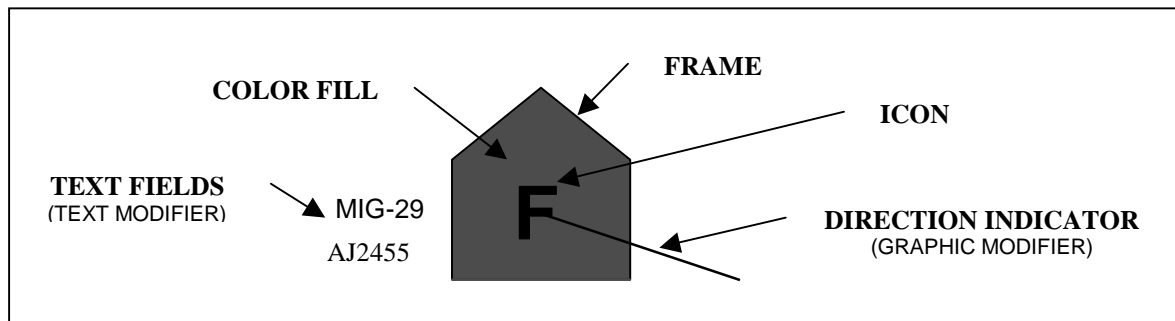
        <Size>16.0</Size>
      </Graphic>
    </PointSymbolizer>
  </Rule>
</FeatureTypeStyle>
</UserStyle>

```

## 14.2. MIL2525B

MIL-STD-2525B is a Department of Defense Interface Standard that defines Common Warfighting Symbolology. SLD contains many of the constructs needed to support MIL-STD-2525B; however, due to the scope and complexity of MIL-STD-2525B, it is likely that additional work will need to be done to provide a fully compliant SLD solution. MIL-STD-2525B symbolology is divided into the two broad categories: Tactical Symbols and Tactical Graphics. Tactical Symbols are used to describe point locations while Tactical Graphics may extend to a line or area.

Tactical Symbols are composed of frame and fill (which under certain conditions are omitted), an icon and zero or more Graphics and Text Modifiers. These components provide information about the symbol's affiliation, battle dimension, status, and mission. Tactical Symbol size may be adjusted by the user, but should be independent of the scale of the map display. An example of the Tactical Symbols with labels showing the names of the components is shown below.



The fill color, frame shape and, in some cases, text modifiers depict the affiliation (Friend, Neutral, Hostile, etc.) and the battle dimension (Land, Air, Sea, etc.) of the symbol. There are nine possible values for affiliation and nine for battle dimension (although some combinations are not used). Some affiliation indicators use a Text Modifier to reflect a pending or presumed value. The frame outline may also be drawn with a dotted line to indicate a planned or anticipated future status. While not trivial, these frame shapes, line styles, and fill colors could be selected based on a rule set codified using SLD.

Icons represent a much larger set of possibilities with many lending themselves to being broken down into more primitive components for potential processing by a ruleset. Although the icon illustrated above is a simple text character, many of the icons have



much more intricate graphical contents. There are also graphical indicators for mobility, auxiliary equipment, headquarters staff, and other attributes. Some of these are independent of the frame shape and are likely to be drawn using “one-size-fits-all” primitives. Others, such as a diagonal line across the icon, are dependent on the frame shape and a developer may choose to incorporate them into the set of frame primitives or to have different primitives for each of the frame shapes.

Text Fields are mapped to Text Modifier locations with some locations used for multiple purposes. For the most part, Text Modifiers are used independently from the other symbol attributes. The SLD Label Placement elements (see Section 11.4.4) should adequately support MIL-STD-2525B Text Modifiers.

The frame, fill and icon are communicated using a 15-character code. The symbol shown above has the code “SHAPMFF---\*\*\*\*\*”, two text modifiers, and one graphics modifier. Dissecting this code character by character gives us “S” for the code scheme, “H” for Affiliation - Hostile, “A” for Battle Dimension - Air, “P” for Status - Present, “MFF---” for Function Id - Military, Fixed Wing, Fighter, and “\*\*\*” for size/mobility, “\*\*\*” for Country Code, and “\*” for the Order of Battle. The dashes indicate unused positions and asterisks indicate user-defined positions based on specific symbol circumstances. The Type Text Modifier is “MIG-29” and the Unique Designation Text Modifier is “AJ2455”. The Direction of Movement Graphic Modifier indicates an ESE direction of travel.

Tactical Graphics may include icon-based symbols but cannot be represented as Tactical Symbols alone. Therefore, in addition to the SLD constructs necessary for Tactical Symbols, Tactical Graphics will use additional constructs to support line and area styling. The Tactical Graphics symbolization may be an extension of existing SLD Line Styling to include support for line and area styles such as tick marks, jagged lines, and rounded corners, some of which may not be able to be represented using the current stroke and fill SLD Style elements.

Two conceivable approaches for support of the basic MIL-STD-2525B symbol set would be to map each anticipated 15-character code to a complete, pre-defined symbol, or by building up MIL-STD-2525B styles using more primitive graphic elements. The first approach would require a pre-defined symbol set with hundreds or perhaps thousands of symbol files. The second approach would be more desirable, if possible, since it is more general and can be applied to other external styling standards. Below is an example of an SLD ruleset to display a symbol built up from primitives to form the Tactical Symbol shown above. Note that the functionality to support the Direction Indicator Graphic Modifier is not presently available.

```
<StyledLayerDescriptor version="1.0.0">
  <NamedLayer>
    <Name>MIL2525B-Data</Name>
    <UserStyle>
      <Name>MIL2525B</Name>
      <FeatureTypeStyle>
```

```

<Rule>
  <Name>HostileAircraft</Name>
  <Title>Hostile Aircraft</Title>
  <ogc:Filter> <!-- Affiliation = Hostile -->
    <ogc:And>
      <ogc:PropertyIsEqualTo>
        <ogc:PropertyName>Affiliation</ogc:PropertyName>
        <ogc:Literal>H</ogc:Literal>
      </ogc:PropertyIsEqualTo>
      <ogc:PropertyIsEqualTo>
        <ogc:PropertyName>Dimension</ogc:PropertyName>
        <ogc:Literal>A</ogc:Literal>
      </ogc:PropertyIsEqualTo>
    </ogc:And>
  </ogc:Filter>
  <PointSymbolizer>
    <Graphic>
      <ExternalGraphic>
        <OnlineResource xlink:type="simple"
xlink:href="http://www.vendor.com/m2525b/AirHostile.png"/>
        <Format>image/png</Format>
      </ExternalGraphic>
      <Size>16.0</Size>
    </Graphic>
  </PointSymbolizer>
</Rule>

<Rule>
  <Name>MilitaryFixedWingFighter</Name>
  <Title>Military Fixed-Wing Fighter</Title>
  <ogc:Filter> <!-- Function ID = Mil. Fixed Wing Fighter
-->
    <ogc:PropertyIsEqualTo>
      <ogc:PropertyName>FUNCTIONID</ogc:PropertyName>
      <ogc:Literal>MFF</ogc:Literal>
    </ogc:PropertyIsEqualTo>
  </ogc:Filter>
  <PointSymbolizer>
    <Graphic>
      <ExternalGraphic>
        <OnlineResource xlink:type="simple"
xlink:href="http://www.vendor.com/m2525b/MilFixedFighter.png"/>
        <Format>image/png</Format>
      </ExternalGraphic>
      <Size>16.0</Size>
    </Graphic>
  </PointSymbolizer>
  <TextSymbolizer>
    <Label>
      <ogc:PropertyName>TypeTextModifier</ogc:PropertyName>
    </Label>
    <Font>
      <CssParameter name="font-

```

```

family">Arial</CssParameter>
    <CssParameter name="font-family">Sans-
Serif</CssParameter>
    <CssParameter name="font-
style">italic</CssParameter>
    <CssParameter name="font-size">10</CssParameter>
  </Font>
  <Fill>
    <CssParameter name="fill">#000000</CssParameter>
  </Fill>
  <Displacement>
    <DisplacementX>-50</DisplacementX>
    <DisplacementY>0</DisplacementY>
  </Displacement>
</TextSymbolizer>
<TextSymbolizer>
  <Label>

<ogc:PropertyName>UniqueIDTextModifier</ogc:PropertyName>
  </Label>
  <Font>
    <CssParameter name="font-
family">Arial</CssParameter>
    <CssParameter name="font-family">Sans-
Serif</CssParameter>
    <CssParameter name="font-
style">italic</CssParameter>
    <CssParameter name="font-size">10</CssParameter>
  </Font>
  <Fill>
    <CssParameter name="fill">#000000</CssParameter>
  </Fill>
  <Displacement>
    <DisplacementX>-50</DisplacementX>
    <DisplacementY>20</DisplacementY>
  </Displacement>
</TextSymbolizer>
</Rule>
</FeatureTypeStyle>
</UserStyle>
</NamedLayer>
</StyledLayerDescriptor>

```

## Annex A: Styled-Layer-Descriptor Schema

(Normative)

This annex contains the Styled-Layer-Descriptor XML Schema corresponding to this version of the specification. Comments in the schema are informative; in case of conflict with the main body of this specification, the main body takes precedence. The schema definition here (without comments) is normative; in case of conflict with schema definitions in the main body of this specification, this schema takes precedence.

```
<?xml version="1.0" encoding="UTF-8"?>
<xsd:schema targetNamespace="http://www.opengis.net/sld"
  xmlns:sld="http://www.opengis.net/sld"
  xmlns:ogc="http://www.opengis.net/ogc"
  xmlns:xlink="http://www.w3.org/1999/xlink"
  xmlns:xsd="http://www.w3.org/2001/XMLSchema"
  elementFormDefault="qualified">
  <xsd:import namespace="http://www.w3.org/1999/xlink"
    schemaLocation="../../../gml/2.1/xlinks.xsd"/>
  <xsd:import namespace="http://www.opengis.net/ogc"
    schemaLocation="../../../filter/1.0.0/Filter.xsd"/>

  <!-- ***** -->
  <xsd:annotation>
    <xsd:documentation>
      STYLED LAYER DESCRIPTOR version 1.0.0 (2002-08-16)
    </xsd:documentation>
  </xsd:annotation>

  <xsd:element name="StyledLayerDescriptor">
    <xsd:annotation>
      <xsd:documentation>
        A StyledLayerDescriptor is a sequence of styled layers, represented
        at the first level by NamedLayer and UserLayer elements.
      </xsd:documentation>
    </xsd:annotation>
    <xsd:complexType>
      <xsd:sequence>
        <xsd:element ref="sld:Name" minOccurs="0"/>
        <xsd:element ref="sld:Title" minOccurs="0"/>
        <xsd:element ref="sld:Abstract" minOccurs="0"/>
        <xsd:choice minOccurs="0" maxOccurs="unbounded">
          <xsd:element ref="sld:NamedLayer"/>
          <xsd:element ref="sld:UserLayer"/>
        </xsd:choice>
      </xsd:sequence>
      <xsd:attribute name="version" type="xsd:string" use="required"/>
    </xsd:complexType>
  </xsd:element>
  <xsd:element name="Name" type="xsd:string"/>
  <xsd:element name="Title" type="xsd:string"/>
  <xsd:element name="Abstract" type="xsd:string"/>
```

```

<!-- ***** -->
<xsd:annotation>
  <xsd:documentation>
    LAYERS AND STYLES
  </xsd:documentation>
</xsd:annotation>

<xsd:element name="NamedLayer">
  <xsd:annotation>
    <xsd:documentation>
      A NamedLayer is a layer of data that has a name advertised by a WMS.
    </xsd:documentation>
  </xsd:annotation>
  <xsd:complexType>
    <xsd:sequence>
      <xsd:element ref="sld:Name"/>
      <xsd:element ref="sld:LayerFeatureConstraints" minOccurs="0"/>
      <xsd:choice minOccurs="0" maxOccurs="unbounded">
        <xsd:element ref="sld:NamedStyle"/>
        <xsd:element ref="sld:UserStyle"/>
      </xsd:choice>
    </xsd:sequence>
  </xsd:complexType>
</xsd:element>

<xsd:element name="NamedStyle">
  <xsd:annotation>
    <xsd:documentation>
      A NamedStyle is used to refer to a style that has a name in a WMS.
    </xsd:documentation>
  </xsd:annotation>
  <xsd:complexType>
    <xsd:sequence>
      <xsd:element ref="sld:Name"/>
    </xsd:sequence>
  </xsd:complexType>
</xsd:element>

<xsd:element name="UserLayer">
  <xsd:annotation>
    <xsd:documentation>
      A UserLayer allows a user-defined layer to be built from WFS and
      WCS data.
    </xsd:documentation>
  </xsd:annotation>
  <xsd:complexType>
    <xsd:sequence>
      <xsd:element ref="sld:Name" minOccurs="0"/>
      <xsd:element ref="sld:RemoteOWS" minOccurs="0"/>
      <xsd:element ref="sld:LayerFeatureConstraints"/>
      <xsd:element ref="sld:UserStyle" maxOccurs="unbounded"/>
    </xsd:sequence>
  </xsd:complexType>
</xsd:element>

<xsd:element name="RemoteOWS">
  <xsd:annotation>
    <xsd:documentation>
      A RemoteOWS gives a reference to a remote WFS/WCS/other-OWS server.
    </xsd:documentation>
  </xsd:annotation>
  <xsd:complexType>

```

```

    <xsd:sequence>
      <xsd:element ref="sld:Service"/>
      <xsd:element ref="sld:OnlineResource"/>
    </xsd:sequence>
  </xsd:complexType>
</xsd:element>

<xsd:element name="Service">
  <xsd:annotation>
    <xsd:documentation>
      A Service refers to the type of a remote OWS server.
    </xsd:documentation>
  </xsd:annotation>
  <xsd:simpleType>
    <xsd:restriction base="xsd:string">
      <xsd:enumeration value="WFS"/>
      <xsd:enumeration value="WCS"/>
    </xsd:restriction>
  </xsd:simpleType>
</xsd:element>

<xsd:element name="OnlineResource">
  <xsd:annotation>
    <xsd:documentation>
      An OnlineResource is typically used to refer to an HTTP URL.
    </xsd:documentation>
  </xsd:annotation>
  <xsd:complexType>
    <xsd:attributeGroup ref="xlink:simpleLink"/>
  </xsd:complexType>
</xsd:element>

<xsd:element name="LayerFeatureConstraints">
  <xsd:annotation>
    <xsd:documentation>
      LayerFeatureConstraints define what features & feature types are
      referenced in a layer.
    </xsd:documentation>
  </xsd:annotation>
  <xsd:complexType>
    <xsd:sequence>
      <xsd:element ref="sld:FeatureTypeConstraint" maxOccurs="unbounded"/>
    </xsd:sequence>
  </xsd:complexType>
</xsd:element>

<xsd:element name="FeatureTypeConstraint">
  <xsd:annotation>
    <xsd:documentation>
      A FeatureTypeConstraint identifies a specific feature type and
      supplies filtering.
    </xsd:documentation>
  </xsd:annotation>
  <xsd:complexType>
    <xsd:sequence>
      <xsd:element ref="sld:FeatureTypeName" minOccurs="0"/>
      <xsd:element ref="ogc:Filter" minOccurs="0"/>
      <xsd:element ref="sld:Extent" minOccurs="0" maxOccurs="unbounded"/>
    </xsd:sequence>
  </xsd:complexType>
</xsd:element>
<xsd:element name="FeatureTypeName" type="xsd:string"/>

```

```

<xsd:element name="Extent">
  <xsd:annotation>
    <xsd:documentation>
      An Extent gives feature/coverage/raster/matrix dimension extent.
    </xsd:documentation>
  </xsd:annotation>
  <xsd:complexType>
    <xsd:sequence>
      <xsd:element ref="sld:Name"/>
      <xsd:element ref="sld:Value"/>
    </xsd:sequence>
  </xsd:complexType>
</xsd:element>
<xsd:element name="Value" type="xsd:string"/>

<xsd:element name="UserStyle">
  <xsd:annotation>
    <xsd:documentation>
      A UserStyle allows user-defined styling and is semantically
      equivalent to a WMS named style.
    </xsd:documentation>
  </xsd:annotation>
  <xsd:complexType>
    <xsd:sequence>
      <xsd:element ref="sld:Name" minOccurs="0"/>
      <xsd:element ref="sld:Title" minOccurs="0"/>
      <xsd:element ref="sld:Abstract" minOccurs="0"/>
      <xsd:element ref="sld:IsDefault" minOccurs="0"/>
      <xsd:element ref="sld:FeatureTypeStyle" maxOccurs="unbounded"/>
    </xsd:sequence>
  </xsd:complexType>
</xsd:element>
<xsd:element name="IsDefault" type="xsd:string"/>

<!-- ***** -->
<xsd:annotation>
  <xsd:documentation>
    FEATURE-TYPE STYLING
  </xsd:documentation>
</xsd:annotation>

<xsd:element name="FeatureTypeStyle">
  <xsd:annotation>
    <xsd:documentation>
      A FeatureTypeStyle contains styling information specific to one
      feature type. This is the SLD level that separates the 'layer'
      handling from the 'feature' handling.
    </xsd:documentation>
  </xsd:annotation>
  <xsd:complexType>
    <xsd:sequence>
      <xsd:element ref="sld:Name" minOccurs="0"/>
      <xsd:element ref="sld:Title" minOccurs="0"/>
      <xsd:element ref="sld:Abstract" minOccurs="0"/>
      <xsd:element ref="sld:FeatureTypeName" minOccurs="0"/>
      <xsd:element ref="sld:SemanticTypeIdentifier" minOccurs="0"
        maxOccurs="unbounded"/>
      <xsd:element ref="sld:Rule" maxOccurs="unbounded"/>
    </xsd:sequence>
  </xsd:complexType>
</xsd:element>

```

```

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

<xsd:element name="Rule">
  <xsd:annotation>
    <xsd:documentation>
      A Rule is used to attach property/scale conditions to and group
      the individual symbolizers used for rendering.
    </xsd:documentation>
  </xsd:annotation>
  <xsd:complexType>
    <xsd:sequence>
      <xsd:element ref="sld:Name" minOccurs="0"/>
      <xsd:element ref="sld:Title" minOccurs="0"/>
      <xsd:element ref="sld:Abstract" minOccurs="0"/>
      <xsd:element ref="sld:LegendGraphic" minOccurs="0"/>
      <xsd:choice minOccurs="0">
        <xsd:element ref="ogc:Filter"/>
        <xsd:element ref="sld:ElseFilter"/>
      </xsd:choice>
      <xsd:element ref="sld:MinScaleDenominator" minOccurs="0"/>
      <xsd:element ref="sld:MaxScaleDenominator" minOccurs="0"/>
      <xsd:choice minOccurs="0" maxOccurs="unbounded">
        <xsd:element ref="sld:LineSymbolizer"/>
        <xsd:element ref="sld:PolygonSymbolizer"/>
        <xsd:element ref="sld:PointSymbolizer"/>
        <xsd:element ref="sld:TextSymbolizer"/>
        <xsd:element ref="sld:RasterSymbolizer"/>
      </xsd:choice>
    </xsd:sequence>
  </xsd:complexType>
</xsd:element>
<xsd:element name="LegendGraphic">
  <xsd:complexType>
    <xsd:sequence>
      <xsd:element ref="sld:Graphic"/>
    </xsd:sequence>
  </xsd:complexType>
</xsd:element>
<xsd:element name="ElseFilter">
  <xsd:complexType/>
</xsd:element>
<xsd:element name="MinScaleDenominator" type="xsd:double"/>
<xsd:element name="MaxScaleDenominator" type="xsd:double"/>

<!-- ***** -->
<xsd:annotation>
  <xsd:documentation>
    SYMBOLIZERS
  </xsd:documentation>
</xsd:annotation>

<xsd:element name="Symbolizer" type="sld:SymbolizerType" abstract="true"/>

<xsd:complexType name="SymbolizerType" abstract="true">
  <xsd:annotation>
    <xsd:documentation>
      A "SymbolizerType" is an abstract type for encoding the graphical
      properties used to portray geographic information. Concrete symbol
      types are derived from this base type.
    </xsd:documentation>
  </xsd:annotation>
</xsd:complexType>

```



```

<!-- ***** -->
<xsd:annotation>
  <xsd:documentation>
    LINE SYMBOLIZER
  </xsd:documentation>
</xsd:annotation>

<xsd:element name="LineSymbolizer" substitutionGroup="sld:Symbolizer">
  <xsd:annotation>
    <xsd:documentation>
      A LineSymbolizer is used to render a "stroke" along a linear geometry.
    </xsd:documentation>
  </xsd:annotation>
  <xsd:complexType>
    <xsd:complexContent>
      <xsd:extension base="sld:SymbolizerType">
        <xsd:sequence>
          <xsd:element ref="sld:Geometry" minOccurs="0"/>
          <xsd:element ref="sld:Stroke" minOccurs="0"/>
        </xsd:sequence>
      </xsd:extension>
    </xsd:complexContent>
  </xsd:complexType>
</xsd:element>

<xsd:element name="Geometry">
  <xsd:annotation>
    <xsd:documentation>
      A Geometry gives reference to a (the) geometry property of a
      feature to be used for rendering.
    </xsd:documentation>
  </xsd:annotation>
  <xsd:complexType>
    <xsd:sequence>
      <xsd:element ref="ogc:PropertyName"/>
    </xsd:sequence>
  </xsd:complexType>
</xsd:element>

<xsd:element name="Stroke">
  <xsd:annotation>
    <xsd:documentation>
      A "Stroke" specifies the appearance of a linear geometry. It is
      defined in parallel with SVG strokes. The following CssParameters
      may be used: "stroke" (color), "stroke-opacity", "stroke-width",
      "stroke-linejoin", "stroke-linecap", "stroke-dasharray", and
      "stroke-dashoffset".
    </xsd:documentation>
  </xsd:annotation>
  <xsd:complexType>
    <xsd:sequence>
      <xsd:choice minOccurs="0">
        <xsd:element ref="sld:GraphicFill"/>
        <xsd:element ref="sld:GraphicStroke"/>
      </xsd:choice>
      <xsd:element ref="sld:CssParameter" minOccurs="0"
        maxOccurs="unbounded"/>
    </xsd:sequence>
  </xsd:complexType>
</xsd:element>

```

```

<xsd:element name="CssParameter">
  <xsd:annotation>
    <xsd:documentation>
      A "CssParameter" refers to an SVG/CSS graphical-formatting
      parameter. The parameter is identified using the "name" attribute
      and the content of the element gives the SVG/CSS-coded value.
    </xsd:documentation>
  </xsd:annotation>
  <xsd:complexType mixed="true">
    <xsd:complexContent>
      <xsd:extension base="sld: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:annotation>
    <xsd:documentation>
      The "ParameterValueType" uses WFS-Filter expressions to give
      values for SLD graphic parameters. A "mixed" element-content
      model is used with textual substitution for values.
    </xsd:documentation>
  </xsd:annotation>
  <xsd:choice minOccurs="0" maxOccurs="unbounded">
    <xsd:element ref="ogc:expression"/>
  </xsd:choice>
</xsd:complexType>

<xsd:element name="GraphicFill">
  <xsd:annotation>
    <xsd:documentation>
      A "GraphicFill" defines repeated-graphic filling (stippling)
      pattern for an area geometry.
    </xsd:documentation>
  </xsd:annotation>
  <xsd:complexType>
    <xsd:sequence>
      <xsd:element ref="sld:Graphic"/>
    </xsd:sequence>
  </xsd:complexType>
</xsd:element>

<xsd:element name="GraphicStroke">
  <xsd:annotation>
    <xsd:documentation>
      A "GraphicStroke" defines a repeated-linear graphic pattern to be used
      for stroking a line.
    </xsd:documentation>
  </xsd:annotation>
  <xsd:complexType>
    <xsd:sequence>
      <xsd:element ref="sld:Graphic"/>
    </xsd:sequence>
  </xsd:complexType>
</xsd:element>

<!-- ***** -->
  <xsd:annotation>
    <xsd:documentation>
      POLYGON SYMBOLIZER
    </xsd:documentation>
  </xsd:annotation>

```

```

    </xsd:documentation>
  </xsd:annotation>

  <xsd:element name="PolygonSymbolizer" substitutionGroup="sld:Symbolizer">
    <xsd:annotation>
      <xsd:documentation>
        A "PolygonSymbolizer" specifies the rendering of a polygon or
        area geometry, including its interior fill and border stroke.
      </xsd:documentation>
    </xsd:annotation>
    <xsd:complexType>
      <xsd:complexContent>
        <xsd:extension base="sld:SymbolizerType">
          <xsd:sequence>
            <xsd:element ref="sld:Geometry" minOccurs="0"/>
            <xsd:element ref="sld:Fill" minOccurs="0"/>
            <xsd:element ref="sld:Stroke" minOccurs="0"/>
          </xsd:sequence>
        </xsd:extension>
      </xsd:complexContent>
    </xsd:complexType>
  </xsd:element>

  <xsd:element name="Fill">
    <xsd:annotation>
      <xsd:documentation>
        A "Fill" specifies the pattern for filling an area geometry.
        The allowed CssParameters are: "fill" (color) and "fill-opacity".
      </xsd:documentation>
    </xsd:annotation>
    <xsd:complexType>
      <xsd:sequence>
        <xsd:element ref="sld:GraphicFill" minOccurs="0"/>
        <xsd:element ref="sld:CssParameter" minOccurs="0" maxOccurs="unbounded"/>
      </xsd:sequence>
    </xsd:complexType>
  </xsd:element>

  <!-- ***** -->
  <xsd:annotation>
    <xsd:documentation>
      POINT SYMBOLIZER
    </xsd:documentation>
  </xsd:annotation>

  <xsd:element name="PointSymbolizer">
    <xsd:annotation>
      <xsd:documentation>
        A "PointSymbolizer" specifies the rendering of a "graphic symbol"
        at a point.
      </xsd:documentation>
    </xsd:annotation>
    <xsd:complexType>
      <xsd:complexContent>
        <xsd:extension base="sld:SymbolizerType">
          <xsd:sequence>
            <xsd:element ref="sld:Geometry" minOccurs="0"/>
            <xsd:element ref="sld:Graphic" minOccurs="0"/>
          </xsd:sequence>
        </xsd:extension>
      </xsd:complexContent>
    </xsd:complexType>
  </xsd:element>

```

```

</xsd:element>

<xsd:element name="Graphic">
  <xsd:annotation>
    <xsd:documentation>
      A "Graphic" specifies or refers to a "graphic symbol" with inherent
      shape, size, and coloring.
    </xsd:documentation>
  </xsd:annotation>
  <xsd:complexType>
    <xsd:sequence>
      <xsd:choice minOccurs="0" maxOccurs="unbounded">
        <xsd:element ref="sld:ExternalGraphic"/>
        <xsd:element ref="sld:Mark"/>
      </xsd:choice>
      <xsd:sequence>
        <xsd:element ref="sld:Opacity" minOccurs="0"/>
        <xsd:element ref="sld:Size" minOccurs="0"/>
        <xsd:element ref="sld:Rotation" minOccurs="0"/>
      </xsd:sequence>
    </xsd:sequence>
  </xsd:complexType>
</xsd:element>
<xsd:element name="Opacity" type="sld:ParameterValueType"/>
<xsd:element name="Size" type="sld:ParameterValueType"/>
<xsd:element name="Rotation" type="sld:ParameterValueType"/>

<xsd:element name="ExternalGraphic">
  <xsd:annotation>
    <xsd:documentation>
      An "ExternalGraphic" gives a reference to an external raster or
      vector graphical object.
    </xsd:documentation>
  </xsd:annotation>
  <xsd:complexType>
    <xsd:sequence>
      <xsd:element ref="sld:OnlineResource"/>
      <xsd:element ref="sld:Format"/>
    </xsd:sequence>
  </xsd:complexType>
</xsd:element>
<xsd:element name="Format" type="xsd:string"/>

<xsd:element name="Mark">
  <xsd:annotation>
    <xsd:documentation>
      A "Mark" specifies a geometric shape and applies coloring to it.
    </xsd:documentation>
  </xsd:annotation>
  <xsd:complexType>
    <xsd:sequence>
      <xsd:element ref="sld:WellKnownName" minOccurs="0"/>
      <xsd:element ref="sld:Fill" minOccurs="0"/>
      <xsd:element ref="sld:Stroke" minOccurs="0"/>
    </xsd:sequence>
  </xsd:complexType>
</xsd:element>
<xsd:element name="WellKnownName" type="xsd:string"/>

<!-- ***** -->
<xsd:annotation>
  <xsd:documentation>

```

```

    TEXT SYMBOLIZER
  </xsd:documentation>
</xsd:annotation>

<xsd:element name="TextSymbolizer" substitutionGroup="sld:Symbolizer">
  <xsd:annotation>
    <xsd:documentation>
      A "TextSymbolizer" is used to render text labels according to
      various graphical parameters.
    </xsd:documentation>
  </xsd:annotation>
  <xsd:complexType>
    <xsd:complexContent>
      <xsd:extension base="sld:SymbolizerType">
        <xsd:sequence>
          <xsd:element ref="sld:Geometry" minOccurs="0"/>
          <xsd:element ref="sld:Label" minOccurs="0"/>
          <xsd:element ref="sld:Font" minOccurs="0"/>
          <xsd:element ref="sld:LabelPlacement" minOccurs="0"/>
          <xsd:element ref="sld:Halo" minOccurs="0"/>
          <xsd:element ref="sld:Fill" minOccurs="0"/>
        </xsd:sequence>
      </xsd:extension>
    </xsd:complexContent>
  </xsd:complexType>
</xsd:element>

<xsd:element name="Label" type="sld:ParameterValueType">
  <xsd:annotation>
    <xsd:documentation>
      A "Label" specifies the textual content to be rendered.
    </xsd:documentation>
  </xsd:annotation>
</xsd:element>

<xsd:element name="Font">
  <xsd:annotation>
    <xsd:documentation>
      A "Font" element specifies the text font to use. The allowed
      CssParameters are: "font-family", "font-style", "font-weight",
      and "font-size".
    </xsd:documentation>
  </xsd:annotation>
  <xsd:complexType>
    <xsd:sequence>
      <xsd:element ref="sld:CssParameter" minOccurs="0" maxOccurs="unbounded"/>
    </xsd:sequence>
  </xsd:complexType>
</xsd:element>

<xsd:element name="LabelPlacement">
  <xsd:annotation>
    <xsd:documentation>
      The "LabelPlacement" specifies where and how a text label should
      be rendered relative to a geometry. The present mechanism is
      poorly aligned with CSS/SVG.
    </xsd:documentation>
  </xsd:annotation>
  <xsd:complexType>
    <xsd:choice>
      <xsd:element ref="sld:PointPlacement"/>
      <xsd:element ref="sld:LinePlacement"/>
    </xsd:choice>
  </xsd:complexType>
</xsd:element>

```

```

    </xsd:choice>
  </xsd:complexType>
</xsd:element>

<xsd:element name="PointPlacement">
  <xsd:annotation>
    <xsd:documentation>
      A "PointPlacement" specifies how a text label should be rendered
      relative to a geometric point.
    </xsd:documentation>
  </xsd:annotation>
  <xsd:complexType>
    <xsd:sequence>
      <xsd:element ref="sld:AnchorPoint" minOccurs="0"/>
      <xsd:element ref="sld:Displacement" minOccurs="0"/>
      <xsd:element ref="sld:Rotation" minOccurs="0"/>
    </xsd:sequence>
  </xsd:complexType>
</xsd:element>

<xsd:element name="AnchorPoint">
  <xsd:annotation>
    <xsd:documentation>
      An "AnchorPoint" identifies the location inside of a text label to
      use an an 'anchor' for positioning it relative to a point geometry.
    </xsd:documentation>
  </xsd:annotation>
  <xsd:complexType>
    <xsd:sequence>
      <xsd:element ref="sld:AnchorPointX"/>
      <xsd:element ref="sld:AnchorPointY"/>
    </xsd:sequence>
  </xsd:complexType>
</xsd:element>
<xsd:element name="AnchorPointX" type="sld:ParameterValueType"/>
<xsd:element name="AnchorPointY" type="sld:ParameterValueType"/>

<xsd:element name="Displacement">
  <xsd:annotation>
    <xsd:documentation>
      A "Displacement" gives X and Y offset displacements to use for
      rendering a text label near a point.
    </xsd:documentation>
  </xsd:annotation>
  <xsd:complexType>
    <xsd:sequence>
      <xsd:element ref="sld:DisplacementX"/>
      <xsd:element ref="sld:DisplacementY"/>
    </xsd:sequence>
  </xsd:complexType>
</xsd:element>
<xsd:element name="DisplacementX" type="sld:ParameterValueType"/>
<xsd:element name="DisplacementY" type="sld:ParameterValueType"/>

<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="sld:PerpendicularOffset" minOccurs="0"/>
        </xsd:sequence>
      </xsd:complexType>
    </xsd:element>

    <xsd:element name="PerpendicularOffset" type="sld:ParameterValueType">
      <xsd:annotation>
        <xsd:documentation>
          A "PerpendicularOffset" gives the perpendicular distance away
          from a line to draw a label.
        </xsd:documentation>
      </xsd:annotation>
    </xsd:element>

    <xsd:element name="Halo">
      <xsd:annotation>
        <xsd:documentation>
          A "Halo" fills an extended area outside the glyphs of a rendered
          text label to make the label easier to read over a background.
        </xsd:documentation>
      </xsd:annotation>
      <xsd:complexType>
        <xsd:sequence>
          <xsd:element ref="sld:Radius" minOccurs="0"/>
          <xsd:element ref="sld:Fill" minOccurs="0"/>
        </xsd:sequence>
      </xsd:complexType>
    </xsd:element>
    <xsd:element name="Radius" type="sld:ParameterValueType"/>

<!-- ***** -->
    <xsd:annotation>
      <xsd:documentation>
        RASTER SYMBOLIZER
      </xsd:documentation>
    </xsd:annotation>

    <xsd:element name="RasterSymbolizer" substitutionGroup="sld:Symbolizer">
      <xsd:annotation>
        <xsd:documentation>
          A "RasterSymbolizer" is used to specify the rendering of raster/
          matrix-coverage data (e.g., satellite images, DEMs).
        </xsd:documentation>
      </xsd:annotation>
      <xsd:complexType>
        <xsd:complexContent>
          <xsd:extension base="sld:SymbolizerType">
            <xsd:sequence>
              <xsd:element ref="sld:Geometry" minOccurs="0"/>
              <xsd:element ref="sld:Opacity" minOccurs="0"/>
              <xsd:element ref="sld:ChannelSelection" minOccurs="0"/>
              <xsd:element ref="sld:OverlapBehavior" minOccurs="0"/>
              <xsd:element ref="sld:ColorMap" minOccurs="0"/>
              <xsd:element ref="sld:ContrastEnhancement" minOccurs="0"/>
              <xsd:element ref="sld:ShadedRelief" minOccurs="0"/>
              <xsd:element ref="sld:ImageOutline" minOccurs="0"/>
            </xsd:sequence>
          </xsd:extension>
        </xsd:complexContent>
      </xsd:complexType>
    </xsd:element>

```

```

<xsd:element name="ChannelSelection">
  <xsd:annotation>
    <xsd:documentation>
      "ChannelSelection" specifies the false-color channel selection
      for a multi-spectral raster source.
    </xsd:documentation>
  </xsd:annotation>
  <xsd:complexType>
    <xsd:choice>
      <xsd:sequence>
        <xsd:element ref="sld:RedChannel"/>
        <xsd:element ref="sld:GreenChannel"/>
        <xsd:element ref="sld:BlueChannel"/>
      </xsd:sequence>
      <xsd:element ref="sld:GrayChannel"/>
    </xsd:choice>
  </xsd:complexType>
</xsd:element>
<xsd:element name="RedChannel" type="sld:SelectedChannelType"/>
<xsd:element name="GreenChannel" type="sld:SelectedChannelType"/>
<xsd:element name="BlueChannel" type="sld:SelectedChannelType"/>
<xsd:element name="GrayChannel" type="sld:SelectedChannelType"/>
<xsd:complexType name="SelectedChannelType">
  <xsd:sequence>
    <xsd:element ref="sld:SourceChannelName"/>
    <xsd:element ref="sld:ContrastEnhancement" minOccurs="0"/>
  </xsd:sequence>
</xsd:complexType>
<xsd:element name="SourceChannelName" type="xsd:string"/>

<xsd:element name="OverlapBehavior">
  <xsd:annotation>
    <xsd:documentation>
      "OverlapBehavior" tells a system how to behave when multiple
      raster images in a layer overlap each other, for example with
      satellite-image scenes.
    </xsd:documentation>
  </xsd:annotation>
  <xsd:complexType>
    <xsd:choice>
      <xsd:element ref="sld:LATEST_ON_TOP"/>
      <xsd:element ref="sld:EARLIEST_ON_TOP"/>
      <xsd:element ref="sld:AVERAGE"/>
      <xsd:element ref="sld:RANDOM"/>
    </xsd:choice>
  </xsd:complexType>
</xsd:element>
<xsd:element name="LATEST_ON_TOP">
  <xsd:complexType/>
</xsd:element>
<xsd:element name="EARLIEST_ON_TOP">
  <xsd:complexType/>
</xsd:element>
<xsd:element name="AVERAGE">
  <xsd:complexType/>
</xsd:element>
<xsd:element name="RANDOM">
  <xsd:complexType/>
</xsd:element>

<xsd:element name="ColorMap">

```



```

<xsd:annotation>
  <xsd:documentation>
    A "ColorMap" defines either the colors of a pallet-type raster
    source or the mapping of numeric pixel values to colors.
  </xsd:documentation>
</xsd:annotation>
<xsd:complexType>
  <xsd:choice minOccurs="0" maxOccurs="unbounded">
    <xsd:element ref="sld: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>

<xsd:element name="ContrastEnhancement">
  <xsd:annotation>
    <xsd:documentation>
      "ContrastEnhancement" defines the 'stretching' of contrast for a
      channel of a false-color image or for a whole grey/color image.
      Contrast enhancement is used to make ground features in images
      more visible.
    </xsd:documentation>
  </xsd:annotation>
  <xsd:complexType>
    <xsd:sequence>
      <xsd:choice minOccurs="0">
        <xsd:element ref="sld:Normalize"/>
        <xsd:element ref="sld:Histogram"/>
      </xsd:choice>
      <xsd:element ref="sld: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"/>

<xsd:element name="ShadedRelief">
  <xsd:annotation>
    <xsd:documentation>
      "ShadedRelief" specifies the application of relief shading
      (or "hill shading") to a DEM raster to give it somewhat of a
      three-dimensional effect and to make elevation changes more
      visible.
    </xsd:documentation>
  </xsd:annotation>
  <xsd:complexType>
    <xsd:sequence>
      <xsd:element ref="sld:BrightnessOnly" minOccurs="0"/>
      <xsd:element ref="sld:ReliefFactor" minOccurs="0"/>
    </xsd:sequence>
  </xsd:complexType>
</xsd:element>

```

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

<xsd:element name="ImageOutline">
  <xsd:annotation>
    <xsd:documentation>
      "ImageOutline" specifies how individual source rasters in
      a multi-raster set (such as a set of satellite-image scenes)
      should be outlined to make the individual-image locations visible.
    </xsd:documentation>
  </xsd:annotation>
  <xsd:complexType>
    <xsd:choice>
      <xsd:element ref="sld:LineSymbolizer"/>
      <xsd:element ref="sld:PolygonSymbolizer"/>
    </xsd:choice>
  </xsd:complexType>
</xsd:element>

</xsd:schema>
```

## Annex B: WMS\_DescribeLayerResponse DTD

(Normative)

This annex contains the Describe-Layer-Response Document-Type Definition corresponding to this version of the specification. Comments in the DTD are informative; in case of conflict with the main body of this specification, the main body takes precedence. The DTD definition here (without comments) is normative; in case of conflict with DTD definitions in the main body of this specification, this DTD takes precedence.

```
<!-- WMS_DescribeLayerResponse: the document is returned in response to a
DescribeLayer request made on a WMS. -->

<!ELEMENT WMS_DescribeLayerResponse (LayerDescription*) >
<!ATTLIST WMS_DescribeLayerResponse
    version CDATA #REQUIRED >

<!-- LayerDescription: describes the contents of a NamedLayer, the name of which
is specified in the 'name' attribute. If the NamedLayer is not feature based,
then the LayerDescription has no further contents. If the NamedLayer is feature
based then the 'wfs' attribute gives the URL prefix for the WFS containing the
feature data. Equivalently, the 'owsType' and 'owsURL' attributes can be used to
indicate the OWS type & base URL of a service. The 'wfs' attribute is retained
for greater compatibility with the WFS specification. The presently recognized
valid values for 'owsType' are "WFS" and "WCS", though more may be allowed in the
future.

The LayerDescription contains one or more Query elements that specify the
feature-types present in the NamedLayer. -->

<!ELEMENT LayerDescription (Query*) >
<!ATTLIST LayerDescription
    name CDATA #REQUIRED
    wfs CDATA #IMPLIED
    owsType CDATA #IMPLIED
    owsURL CDATA #IMPLIED >

<!-- Query: a Query uses the 'typeName' attribute to identify a feature/coverage-
type. This is a stripped down version of the Query element used in the WFS. -->

<!ELEMENT Query EMPTY >
<!ATTLIST Query
    typeName CDATA #REQUIRED >
```

## Annex C: Conformance Tests

(Normative)

**NOTE:** A complete Conformance Testing Guideline document for WMS is presently under development as part of the OGC Conformance Testing Program. When complete, the Guideline will include a description and scope of each test suite, test data used in the tests, and documentation of the conformance items that constitute requirements for conformance.

Minimal conformance with this specification requires the following:

1. The Extensible Markup Language (XML) document used encode a **StyledLayerDescriptor** shall be valid against the XML Schema in Annex A. Such validation may be performed using commonly available XML validating tools.

## Annex D: Future Work

(Informative)

Better methods for style-library utilization should be explored.

The capabilities statement for SLD included in the **UserDefinedSymbolization** structure needs to be expanded to include a complete set of all capabilities that may be relevant for a client application to know about a styling service.

Label plotting needs to be overhauled to be more compatible with SVG/CSS and to provide greater capabilities to place labels.

A simplified mechanism should be included for efficiently encoding thematic styling using solid colors since the user might like to make a thematic or choropleth map. Some common examples are population density maps or temperature maps. These maps typically take a large set of observations on one attribute, break them into ranges, and assign different colors to those ranges. This allows a viewer to quickly see patterns in the data that would otherwise be hidden in the sheer volume of data. This problem can be addressed in a number of ways. The simplest is to return all the observations to the client and let the human or machine define the ranges. The styling document can then simply say, for example, assign all values from 0 to 3 a light red, all values from 4 to 7 a medium red, and all values greater than 7 a dark red.

The problem with this strategy is that there may be millions of observations in the data set. In an Internet environment it would be very cumbersome to transfer all this data to the client just for the purpose of creating color ranges. It is more likely that a client might simply want to specify that the data set should be divided into three ranges with the lower values being a light red, the middle being a medium red, and the highest being dark red. This need is not addressed in this document, but should be a priority of future work.

Once a user goes to all the trouble of creating a style or set of styles, they will probably want to save it for later use, and maybe even tell others about its existence. This paper begins to work these issues by defining how a service can advertise its styles, either wholesale or on a per-layer basis. There is also a preliminary discussion on how clients might store their custom styles on a server for later use. This all hints at the need for an OGC style service, where SLDs can be read, written, updated, permissioned, etc., but a full-fledged SLD Service definition is not defined here.

OGC is closely following the work on ISO 19117 (Portrayal). Concepts being tested in current (OWS-1.2) and upcoming testbeds will attempt to address alignment between OGC SLD and the ISO 19117 standard.



## Annex E: RFC Changes

(Informative)

This annex shows the RFC comments that were received for this document and the responses of the Revision Working Group (RWG). The RWG had the following official members: William Lalonde (CubeWerx) Chair, Jeff Lansing (Polexis), Joshua Lieberman (Syncline), Jerome Sonnet (Ionic Software), and John Vincent (Intergraph).

Two sets of comments were received. Both comments that were accepted and rejected are discussed below, including reasons for rejections. The commenters are presented as being anonymous. The section and page numbers are relative to the version of this document that was used for the RFC procedure, document OGC 01-013r1. It is presently available from: "<http://www.opengis.org/techno/RFC14.pdf>". The commenters' comments are shown in italic text and the RWG response is in normal font.

### E.1. Comments from commenter #1

#### E.1.1. Comment #1

*Specification Section number: [Section 6.2 General HTTP request]*  
*Criticality: [EDITORIAL]*

*There is a reference to OGC web services. It also appears in a couple of other places (page 20,21). This terminology is already used for SOAP based web services and may be misleading in the context of this RFC. The OGC Web Services, or OWS, is the point of the whole exercise; I think the names (and the generic placeholder acronym W\*S for the many kinds of web services) will likely stay, as it is well understood within the consortium.*

#### **RWG decision: REJECT**

The phrase "OGC Web Services (OWS)" is a commonly used phrase within OGC and has specific and distinct meanings from generic web services. Provided that the "OGC" prefix is always present in the phrase, there should be no unnecessary confusion about its meaning.

#### E.1.2. Comment #2

*Specification Section number: [Section Section 7.2 Named Layers]*  
*Criticality: [MAJOR]*

*In the HTTP GET request that follows:*

```

http://yourfavoritesite.com/WMS?
  VERSION=1.1.0&
  REQUEST=GetMap&
  BBOX=0.0,0.0,1.0,1.0&
  LAYERS=Rivers,Roads,Houses&
  STYLES=CenterLine,CenterLine,Outline&
  WIDTH=400&
  HEIGHT=400&
  FORMAT=PNG

```

*The LAYERS and STYLES attributes are meant to say "Rivers should be stylized as centerline, Roads as centerline, and houses as outline." Having these be uncoupled like this means that if, for whatever reason, one or the other changes order, the rendering will be impacted in unexpected ways. The order of LAYERS imparts a Z-order, but changing that Z-order also requires changing the STYLES argument. It would be better if STYLES were self-describing and truly independent, such as*

```

STYLES=Rivers:CenterLine,Roads:CenterLine,Houses:Outline&

```

*Failing to do so will be the source of numerous headaches down the road. Note that the POST or SLD/XML approach does not suffer from the same weakness.*

#### **RWG decision: REJECT**

This is not a comment that should be considered for SLD. It is the WMS specification defines the syntax of the HTTP-GET parameters for the WMS GetMap request, not SLD. However, the RWG does not think that this suggestion provides any improvement, since GET requests are normally generated automatically and since the suggested scheme adds special control characters to the layer-name syntax. CubeWerx Inc., for example, already makes use of the colon character within layer names for a vendor-specific purpose. OGC has benefited from keeping these names as free of arbitrary syntax rules as possible.

#### **E.1.3. Comment #3**

*Specification Section number: [Section 9 FeatureTypeStyles]*

*Criticality: [MAJOR]*

*There is a one to many relationship between FeatureTypeStyle and FeatureType. What should be the behavior if a user style contains more than one FeatureTypeStyle for the same FeatureType.*

#### **RWG decision: ACCEPT (clarify spec)**

It is not a "one-to-many" relationship since the specification explicitly states that a FeatureTypeStyle can use only one feature type, whether identified implicitly or explicitly. Assuming that the commenter means "many-to-one", this does not pose a problem.



The common semantic for a **GetMap** request is to render all of the styling elements in the order that they are given, and the same applies to **FeatureTypeStyle** elements even if multiple ones of them refer to the same feature type. Perhaps this needs to be stated more carefully. In contexts other than a **GetMap** request, the semantics are still clear since the multiple **FeatureTypeStyle** elements are contained within a **UserStyle** are merely passed around as an opaque package until they are ultimately used with a **GetMap** request.

A potential advantage of allowing multiple **FeatureTypeStyle** elements to refer to the same feature type is to allow multiple independent **Rule** sets with independent **ElseFilter** conditions, although this may not be used much in practice. SLD is organized to allow multiple **FeatureTypeStyle** elements to refer to the same feature type as part of providing a clean separation between the definition of what features and feature types are included within a **LAYER** and what feature types are handled by a **STYLE**. A clean separation between layers and styles is an important thing to have now and in the future.

#### **E.1.4. Comment #4**

*Specification Section number: [Section 10.2: Scale Selection]*

*Criticality: [MAJOR]*

*Scale directly impact the size of the returned data and the processing and bandwidth required to generate and deliver that data. There should be a provision for the server to decline a request if it determine the result will be too large.*

#### **RWG decision: REJECT**

The server is free to simply return an error. However, for rendered images anyway, the size of the returned data (number of pixels) will always be fixed by the **WIDTH** and **HEIGHT** of the image given in the **GetMap** request. More zoomed-out scales may require more internal processing inside of a rendering server, but it is unclear whether or how this should be exposed to the outside world.

#### **E.1.5. Comment #5**

*Specification Section number: [Section 10.3 Feature Filtering]*

*Criticality: [MINOR]*

*Using Feature Filters to exclude features has certain negative performance implications, since features that get filtered out during stylization still must be retrieved by the WFS and passed to the WMS or intermediate stylization service. Would it not be more appropriate to filter features using WFS capabilities and hence never retrieve features that will not be stylized?*

#### **RWG decision: REJECT**

The SLD design does not impose any performance limitations on fetching features from WFSes. First, filtering may be applied in two places: in the definition of a layer and within style rules. Allowing filtering to be specified in the "layer" definition addresses the above comment directly.

Second, the WMS (or other styling client) is free to analyze requests and to fetch only the features that are necessary for a query, by combining Layer and Rule conditions in the WFS request. The styling client is not artificially limited in how it may construct WFS requests. For example, if a style included only two rules for a Roads feature type with condition "numLanes >= 4" for the first rule and "numLanes >= 6" for the second, the styling client could easily formulate a WFS request that has the condition "numLanes >= 4 OR numLanes >= 6" (using the appropriate condition-representation syntax). In the future, when style and layer definitions are decoupled, this kind of analysis may be necessary anyway.

Third, and more abstractly, the RWG does not believe that styling from a remote WFS using a WMS as an intermediate service will be very common in production environments anyway. It is simply too wasteful and slow. Ultimately, either thick clients (e.g., a desktop GIS) or "thick servers" (e.g., a WMS connected directly to the underlying data storage system) will be used instead of transferring gigabytes of data over the internet between various services so that each may incur a large parsing/generating/storage cost in order to apply some small tweak to the data stream. The RWG believes that this sort of thing will only happen in low-volume demonstrations. The one-client/one-server paradigm optimizes the network-transfer and parse/generate penalties.

#### **E.1.6. Comment #6**

*Specification Section number: [Section 11.2 Geometry]*

*Criticality: [MAJOR]*

*Polygons with islands are not covered. What are the rendering rules?*

#### **RWG decision: ACCEPT**

Holes are external to the polygon, so they are not filled, and islands are internal, so they are filled. The borders around both holes and islands are plotted with the specified polygon stroke. The RWG is assuming that the commenter is saying that this should be stated explicitly, though we also assume that the behaviour described here is the norm for rendering polygons/holes/islands.

As a minor point, technically speaking, the RWG does not believe that OGC/Simple-Feature "polygons" are allowed to have islands (only holes), though "multipolygons" are allowed to have islands.

*Why line should be rendered as polygon by joining the end points? The symbolizer should not change the geometry type. May be it should render the line as a line by applying the stroke element and ignore the fill element.*

**RWG decision: REJECT**

If the user wanted that, he could simply use a Line symbolizer. By using a Polygon symbolizer, the user is saying he wants to see all of the selected geometries rendered as polygons. If this is not what the user wants, which likely will only ever happen in the somewhat unusual case of rendering a feature type that has heterogeneous geometry types, filtering can be applied to restrict the features selected to be only of appropriate geometry types. Appropriate Filter functions to identify whether a feature-geometry-property value has a specific geometry type may or may not be commonly available in various systems, but this is an issue for implementing and defining such functions in the OGC Filter specification.

It is generally useful to alter geometry types in some cases for styling purposes, for example, to change a built-up area polygon to a point at certain scales or for labelling.

**E.1.7. Comment #7**

*Specification Section number: [Section 11.3.1 Format]*

*Criticality: [MINOR]*

*The behavior for other type of geometry should be more explicit.*

(see next comment)

**E.1.8. Comment #8**

*Specification Section number: [Section 11.3.2 Graphic]*

*Criticality: [MAJOR]*

*It's not clear how the Rotation applies when the geometry is a line. Should the angle be absolute or relative to the direction of the line. Rotation: We seem to recall the normative definition of rotation is degrees from normal position, or 0 degrees is defined to be the vector parallel to the +X vector. (It might have to be restated explicitly here for clarity, though.)*

**RWG decision: ACCEPT**

The RWG is interpreting this comment to refer to the case of plotting a point graphic relative to a line geometry. It is unclear how often this case will happen in practice, but for the sake of consistency, the translation of the line geometry into a representative point should be considered to be independent of the plotting of the graphic at the point. The rotation to use for the graphic will be the inherent orientation of the source graphic. When a graphic is rotated, it is rotated about its 'inherent' rotation point, which we can

define to be its center point in the absense of any graphic-format-specific information. There is no notion and no need for zero degrees to be defined to be relative to the +X vector, for a Graphic.

The specification has been updated to state more explicitly how all of the geometry-type transformations should be applied, where they are defined

Some of the transformation cases are rather ugly to deal with, but we should define all translations to be valid just for consistency. It is expected that the most common transformations will be the ones that make the most sense.

#### **E.1.9. Comment #9**

Specification Section number: [Section 11.4.3: Font]

Criticality: [MAJOR]

Vendor specific fonts are not discussed. For example how to use vector based fonts? We think the various vendors would be better served bringing their technology closer in line with world-wide typography standards than to have this proposal introduce parochial complexities.

AutoCAD's SHX fonts, for example, should behave in a "reasonable" fashion, say with

```
font-family="txt" font-style="italic" font-weight="bold" font-size="10"
```

the font engine would use txt.shx, interpret italic as oblique, use a heavier (say 2- or 3-pixel) line to depict bold; with the size being based on typographical standard should be adapted to SHX metrics using the inverse of the rule of thirds (character height = 3/4 of the 10 pixel size or 8 pixels after rounding.) The engine would also need to deal gracefully with halos, another stylization that the specification calls for. As such, We don't see how the proposal is deficient in this respect.

#### **RWG decision: ACCEPT**

The RWG suggests that mapping vendor-specific fonts to the SVG/CSS definitions of family, style, weight, and size, is a vendor-specific issue. However, to help to avoid different vendors from going off and doing crazy things, the above example could be included in the SLD spec as general guidance.

#### **E.1.10. Comment #10**

Specification Section number: [Section 11.4.4: - Label Placement]

Criticality: [MAJOR]

Label placement for polygons not discussed. May be at the centroid? No option to place a label along a line. May be the line placement should include more options such as: StartPoint, EndPoint, MidlePoint and may be at a percent distance along the line.

**RWG decision: DEFER** (until after 1.0.0; a real can of worms)

Label placement is a complex issue that is admittedly poorly addressed in this version of SLD. In general, visually pleasing label placement and de-confliction is a very complex (Artificial-Intelligence) subject and many of the details remain implementation-specific in this document.

The RWG is not sure of the degree to which explicit line-placement parameters help, since precise placement options are best suited to definite, fixed geometries that are known in advance, whereas in this environment, it is not known ahead of time which geometries will be processed, how they will intersect, or even at what scale they will be rendered. So, specifying that the label should be drawn at 30% of the distance in to the line has significantly less utility than if a graphic artist was drawing a specific, fixed image.

The RWG recommends that fine-tuning the label placement capabilities should be a "future work" item. The encoding of label-placement parameters should also be made to be more CSS-like in the future.

#### **E.1.11. Comment #11**

Specification Section number: [Section 11.6: Systems with limited Capabilities]

Criticality: [MINOR]

Can the system be interrogated about its capabilities?

**RWG decision: MINOR REVISIONS**

There is only the `<UserDefinedSymbolization>` element for WMS capabilities that defines what styling capabilities are available, and it only indicates the barest minimum, and does not describe any graphical or expression-handling constraints. We should add a list of supported SLD versions to the `<UserDefinedSymbolization>` tag so that SLD versions can be managed. However, describing the graphical capabilities is a can of worms, and can be handled for now on a "best-effort" basis.

#### **E.2. Comments from commenter #2**

##### **E.2.1. Comment #1**

*Section 6.5 Web Map Servers and Web Feature Servers*

*According to the current WFC specification, only OGC simple features and attributes are managed by a WFC. Why WCSs are not mentioned as the source of raster data in this section, while the XML schema for a raster symbolizer is defined in Section 11?*

**RWG recommendation: ACCEPT**

Assuming that "WFC" here means "WFS", this was an oversight. At the time that section was originally written, WCSes didn't exist.

**E.2.2. Comment #2**

*Section 6.6 DescribeLayer Request*

*In this specification, no mechanism is included to manage user-defined layers in a WMS. Since GetStyles and PutStyles are defined for symbology management, should PutLayers operation be supported? (Please read question 5 as well.)*

**RWG recommendation: ACCEPT (make more explicit)**

With the way that SLDs and user-defined layers are handled, a PutStyles operation is implicitly also a put-user-defined-layers operation, since the semantics of PutStyles is to accept the entire SLD that is given and store the contents in some representation inside of the WMS. So, if a WMS indicates that it is capable of handling user-defined layers and that it supports the PutStyles operation, then it logically should also support the declaration and storage of user-defined layers as named layers.

**E.2.3. Comment #3**

*Section 11.3.2 Graphic*

*If an external graphic is used to define a point symbolizer in a user-defined style and the user-defined style is changed to become a named style, should there be a mechanism to copy the external graphic into a WMS to ensure that there will be no broken links in the future?*

**RWG recommendation: REJECT**

How to handle external graphics (reference/cache/store) and how to represent styles inside of a WMS is entirely implementation-dependent. Though, if a WMS stores an external graphic internally, then there could be problems if the external graphic were to change. OTOH, this is a semantic issue which is not addressed: does an external graphic reference refer to the graphic content at the time it is used or only to the external object for all time? It is expected that few external graphics will actually have their content changed as part of its, so the RWG recommends that the specification say that any approach to handling the graphic content is okay.

**E.2.4. Comment #4***Section 11.3 Point Symbolizer*

*The GetStyles operation could provide information about named styles in named layers. Is there a way to find all the well-known names of marks in a WMS?*

**RWG recommendation: DEFER** (until after 1.0.0)

Presently, no. The RWG views Marks as a fallback mechanism in case external graphics are unavailable or unsupported, so it has lower importance than perhaps some other content. Though, as noted elsewhere, SLD could use greater capability-reporting capabilities in the future. Reporting which well-known marks are available for use seems like more of a 'capabilities' issue than a 'get-data' issue.

**E.2.5. Comment #5***Section 13.2 PutStyles*

*It is not clear to me if the PutSytyles operation allows a user to create a named style from a user-defined layer and a user-defined style specified in an SLD. As I understand a user-defined style is associated with a layer. Does the PutStyles operation also make a user-defined layer become a named layer?*

**RWG recommendation: ACCEPT**

(addressed above in comment #2 of this set)

**E.2.6. Comment #6***Security, Sharing and Access Control*

*I am assuming that all named styles and all named layers are accessible to all WMS clients.*

*Should the PutStyles operation be reserved to privileged user only? Should the replacement and deletion of a user created named style be prohibited? If not, a client application needs to verify the availability and/or the definition of each named style before using it.*

**RWG recommendation: REJECT**

Access controls aren't considered in this specification. It seems like a much more general problem for OGC services, and so should be addressed there first. WFS, for example, allows the manipulation of features inside of a feature server, which is analogous to manipulating styles inside of a WMS in terms of access control.

The availability of named styles can be verified in the WMS capabilities document. Handling the case that an object (such as a style) disappears from or changes inside of a WMS after the capabilities document has been fetched is addressed by the 'updateSequence' mechanism that is available in the WMS capabilities and request interface. For this reason, a WMS should update its updateSequence value appropriately if any internal styles or layer definitions are changed.

**E.2.7. Comment #7**

*Introduction Page ix, 3rd paragraph*

*There are three basic ways to style a data set. I can find two but not three ways.*

**RWG recommendation: ACCEPT**

This should be changed to "two basic ways". There used to be a paragraph that described a "choropleth" method, but then complained that it's not implemented. (Though it could be simulated with rules.)

**E.2.8. Comment #8**

*Section 7.2 Named Layers, Page 17, next to the last paragraph*

*A named styled layer can include a any => include any*

**RWG recommendation: ACCEPT**

**E.2.9. Comment #9**

*Section 8 User-Defined Styles, Page 22, first paragraph*

*A user-defined allows => A user-defined style allows*

**RWG recommendation: ACCEPT**

**E.2.10. Comment #10**

*Section 9 FeatureTypeStyle, Page 24, first paragraph*

*of a layer) => of a layer.*

**RWG recommendation: ACCEPT**

**E.2.11. Comment #11**

*Section 10 Rules, page 25, third paragraph*

*a FeatureTypeStyleStyle => a FeatureTypeStyle*



**RWG recommendation: ACCEPT****E.2.12. Comment #12**

*Section 11.3.2 Graphic, page 42, last paragraph*

*and marks it allows a style => and marks. It allows a style*

**RWG recommendation: ACCEPT** (restate sentence)

The sentence may be awkward as-is, but the RWG does not think that it is incorrect. Putting a period where indicated makes the second clause of the sentence that the period ends incomplete.

**E.2.13. Comment #13**

*Section 11.5.2 Parameters, page 50*

*a pallet-type =>? a pallete-type*

**RWG recommendation: ACCEPT** (use consistent spelling)

There appear to be multiple spellings for the word, but "palette" appears to be the 'most' correct one, so this spelling should be used throughout the document. Additionally, an informal Google.com search shows that this is the most common spelling on the Web.

**E.2.14. Comment #14**

*Section 13.1 GetStyles, page 63*

*application/vnd.ogc.sld+xml =>? application/vnd.ogc.sld\_xml*

*Section 13.2 PutStyles, page 64*

*application/vnd.ogc.success+xml =>? application/vnd.ogc.sld\_xml*

**RWG recommendation: REJECT**

This needs to be synchronized with the WMS spec. At one point it was decided that the WMS spec should use the "+" notation since this is the norm for XML documents. The change has been slowly being adopted by various OWS specifications, including the latest versions of WMS, though apparently not WMS 1.1.1. (An underscore was previously used instead of the plus sign in older WMS specs.)

Also, since the "application/vnd.ogc.sld+xml" and "application/vnd.ogc.success+xml" are defined here and in no other document, it isn't strictly an error to define them in the standard 'XML' way.

## Annex F: OGC SLD and ISO 19117

### Introduction

It is the intent to make SLD as compliant as feasible with ISO 19117 Geographic information — Portrayal, clauses A.1 Portrayal schema, A.3 Priority attribute, A.5 External function.

Developments needed to achieve this goal are listed in the remainder of this annex. This annex also describes issues related to ISO 19117.

### ISO 19117 Clause A.1 Portrayal schema

- a) Test purpose: To verify conformance to the portrayal schema.
- b) Test method: verify that the portrayal specifications and portrayal rules are not part of the dataset, and that the portrayal specifications are stored separately and referenced from the portrayal rules. The portrayal specifications may be stored externally and referenced using a universal reference standard like URL. Verify that the portrayal rules are stored in a portrayal catalogue, and that the portrayal rules are specified for the feature classes they will be applied on. Verify that the portrayal rules are expressed using the OCL query language.
- c) Reference: ISO 19117, 8.
- d) Test type: basic test.

### Discussion of Clause A.1 relative to SLD

Allowing the pieces to be separable is a desirable quality and is provided by the SLD extensions being developed for the Style Management System of the OWS-1.2 project, hereafter referred to as "SLD 0.7.3". REQUIRING them to be separated imposes practical problems and disallows the direct usage of simple styling without the installation and population of styling catalogs. The analogous components to portrayal rules and portrayal specifications in SLD 0.7.3 are <FeatureStyle> and <Symbol> elements, respectively. (<Symbol> is an abstract element that has many derived sub classes that provide different types of symbols). SLD 0.7.3 allows separate repositories of <FeatureStyle> and <Symbol> fragments with cross-referencing using URLs.

Clause A.1 states "The portrayal specifications may be stored externally and referenced using a universal reference standard like URL." SLD 0.7.3 provides this.

Clause A.1 states: "Verify that portrayal rules are stored in a portrayal catalogue, and that portrayal rules are specified for the feature classes that they will be applied to." SLD 0.7.3 provides this but also allows <FeatureStyles> to be provided in-line with a

rendering request. The <StyledLayerDescriptor> element is used to encode rendering requests.

Clause A.1 states “Verify that the portrayal rules are expressed using the OCL query language.” Using any different query language poses numerous practical problems, but this issue in isolation is not relevant to the SLD specification itself. SLD re-uses the OGC Filter specification, which was created in parallel with the OGC WFS specification, so the issue of query languages must be resolved there.

### **ISO 19117 A.3 Priority attribute**

- a) Test purpose: To verify correct use of priority attributes.
- b) Test method: Verify that all the portrayal rules have a priority attribute if priority attributes are used. Verify that If two portrayal rules returning TRUE have the same priority value, then the application shall decide which one takes precedence. Verify that all the portrayal rules have a priority attribute if priority attributes are used.
- c) Reference: ISO 19117, 7.2.
- d) Test type: basic test.

### **Discussion of Clause A.3 relative to SLD**

[sic—the last sentence in b) appears to be repeated twice].

SLD uses a different model for evaluating portrayal rules in that it allows the selection of multiple overlapping rules for a single feature. It is unclear whether this is more or less desirable in practice than the 19117 approach.

### **ISO 19117 A.5 External function**

- a) Test purpose: To verify correct use of external functions.
- b) Test method: Verify that the portrayal catalogue lists the external functions used, including the parameters and returned values.
- c) Reference: ISO 19117, 8.3.5.
- d) Test type: basic test.

### **Discussion of Clause A.5 relative to SLD**

SLD 0.7.3 includes an analogous concept by allowing a block of <EnvironmentVariables> definitions within the rendering request. Rules and graphical-styling parameters are permitted to reference these runtime-supplied values in the same way as feature attributes. This fulfills the same purpose as 19117 external functions for

static values, and the OGC-Filter specification also provides a capability for a portrayal service to declare custom functions for use in expressions. However, there are practical problems with the 19117 approach and with OGC-Filter custom functions when they are used in globally-accessible expression environment. ISO 19117 does not appear to address the issue of actually calling an external function, and OGC-Filter custom functions are specific to individual server instances. Also, if ISO 19117 functions are meant to be used as remote procedure calls, they would require a separate remote procedure call to some external server for EVERY feature portrayed (since function side effects are not disallowed for calls with the same parameter values). This would be ENORMOUSLY wasteful, especially by comparison if the external function returns a static value. If the external functions are assumed to be well-known and implemented by every portrayal service, the selection of these functions is severely limited and the specific list of required functions is not addressed by the 19117 specification. If the 19117 functions are server-instance-specific, then they are unusable in an interoperable environment. Using a variable-value block in SLD seems to be a very practical approach by comparison. Rendering scales are treated specially in SLD in that scale selection is incorporated directly into the syntax of the <Rule> element. It is unclear whether this is more or less desirable in practice than the 19117 approach of using the same mechanism as for user-supplied runtime parameters.

### **Additional Comments**

The "Abstract test suite" annex doesn't seem to address the encoding of the portrayal specifications. Portrayal representation and encoding is the greatest weakness of the 19117 specification, because the approach used is not feasible for interoperability and the choice of UML as the encoding language is particularly strange. In essence, 19117 does not actually attempt to address the issue of interoperable graphical styling. The purpose of the 19117 portrayal specification appears to be only to let the styling designer declare whatever styling operations he wishes to have exist and then supply the instances of styling-parameter values in terms of these declarations. However, the declaring of the interfaces of these styling operations does not magically make them pop into existence, though the ISO-19117 specification stops at this point. For automatic processing, this is analogous to using WSDL and assuming that computers will automatically understand interface semantics, which is provably impossible. ISO 19117 also does not cleanly separate multiple instances of declared parameter values for multiple invocations of a styling operation. An example of this problem is present in the "ps21" ParameterSet definition in clause B.4.2. For interoperable styling, there needs to be a set of well-known styling parameters available that the portrayal service can understand and map to its own internal styling language. SLD declares a subset of well-known SVG styling parameters and extras, which is sufficient, but 19117 does nothing. ISO 19117 is not a workable self-complete design for interoperable portrayal. SLD is. Immediate implementations of interoperable portrayal are required for OGC projects.

## Bibliography

1. *Web Map Server Interface Implementation Specification, Version 1.0.0*, OpenGIS Project Document 00-028, Alan Doyle (International Interfaces, Inc.) Editor, April 2000, <http://www.opengis.org/techno/specs/00-028.pdf>.
2. *Web Map Service Implementation Specification, Version 1.1.0*, OpenGIS Project Document 01-047r2, Jeff de La Beaujardière (NASA) Editor, June 2001, <http://www.opengis.org/techno/specs/01-047r2.pdf>.
3. Cox, S., Cuthbert, A., Lake, R., and Martell, R. (eds.), "OpenGIS Recommendation - Geography Markup Language 2.0," February 2000, <http://www.opengis.org/techno/specs/>.
4. Vretanos, P. (ed.), "OpenGIS Discussion Paper #01-023: Web Feature Service Draft Candidate Implementation Specification 0.0.12," January 2001, <http://www.opengis.org/techno/discussions.htm>