— **Intersects** (anotherGeometry: Geometry): Integer — Returns 1 (TRUE) if *this* geometric object spatially intersects anotherGeometry.

— **Touches** (anotherGeometry: Geometry): Integer — Returns 1 (TRUE) if *this* geometric object spatially touches anotherGeometry.

— **Crosses** (anotherGeometry: Geometry): Integer — Returns 1 (TRUE) if *this* geometric object spatially crosses'anotherGeometry.

— **Within** (anotherGeometry: Geometry): Integer — Returns 1 (TRUE) if *this* geometric object is spatially within anotherGeometry.

— **Contains** (anotherGeometry: Geometry): Integer — Returns 1 (TRUE) if *this* geometric object spatially contains anotherGeometry.

— **Overlaps** (AnotherGeometry: Geometry) Integer — Returns 1 (TRUE) if *this* geometric object spatially overlaps anotherGeometry.

— **Relate** (anotherGeometry: Geometry, intersectionPatternMatrix: String): Integer — Returns 1 (TRUE) if *this* geometric object is spatially related to anotherGeometry, by testing for intersections between the interior, boundary and exterior of the two geometric objects.

6.2 Annotation Text

Spatially placed text is a common requirement of applications. Many application have stored their text placement information in proprietary manners due lack of a consistent and usable standard. Although the mechanisms for text storage have tended to be compatible, the actual format for exchange has been sufficiently different, and, therefore, non-standardized to interfere with complete data exchange and common usage. To overcome this interoperability gap, this standard, using best engineering practices, defines an implementation of annotation text.

Annotation text is simply placed text that can carry either geographically-related or ad-hoc data and process-related information is displayable text. This text may be used for display in editors or in simpler maps. It is usually lacking in full cartographic quality, but may act as an approximation to such text as needed by any application.

The primary purpose of standardizing this concept is to enable any application using any version of Simple Features data storage or XML to read and write text objects that will describe where and how the text should be displayed. This design ensures that applications that do text placement should have no problem storing their results and that applications that comply with the standard should have no problem exchanging information on text and its placement.

Unlike spatial geometries, text display is very dependent on client text rendering engines and the style and layout attributes applied. The spatial area covered by text is only partially determined by the locating geometry. Style and layout attributes along with the actual text and locating geometry are all needed to display text correctly. Thus, it is critical to have a place to store these attributes in the feature database. While it is impossible to guarantee absolute fidelity of display on all rendering systems, applications can interoperate at a useful level.

The most common perception of text display is for cartographic purposes, for printed maps of high technical and artistic quality. While this is a potential use of placed text, its more every-day use is for identification of features in any display, regardless of the purpose of that display. So both cartographic preprint and data collection edit displays have a requirement for placed-text, albeit at different levels of artistic quality. The purpose is still the same, to aid in the understanding of the “mapped” features, either for map use or feature edit and analysis.

Text can also be used for less precise annotation purposes and more for quick display of text labels that make a display more understandable. The text so placed may not even have any associations to real-world features, but may be used to store information pertinent to the process that the data is undergoing at the moment. Thus, in a data collecting and edit display, a particular placed text may be used to indicate an error in the data that needs to be resolved, such as “sliver,” “gap” and “loop” error in digitization. Here the annotation is placed near the geometric error, but is not necessarily associated to a particular feature, as much as to a portion or portions of feature geometry objects.

Annotation text can include text on maps derived from vector information, or text overlays for imagery for information not discernable from the image, such as place or street names. In most cases, applications that do this have certain rules for creating and re-creating text based on the dynamic view of the mapping application. While this standard is not targeted to those usages, there are some allowances for this type of storage if it is so desired. In particular, it is allowable to store text that does not scale with the map objects but instead has a fixed display size (expressed as “points”, 72 to the inch). However, there are some limitations on this usage particularly with spatial indexing.

6.2.1 Text entities

A text object consists of an ordered list of independently placed text elements, possibly corresponding to individual lines of text in a multiline text display, and an envelope that approximates an outer limit of the text elements when placed. Each element has its own text attributes, but they are not used independently. The first element may set the attribute for all following elements and subsequent elements text attributes are only specified when a change is required. This behavior just extends that of the metadata text attributes to each element of the array

A text object consists of a text string and information about its placement. The most important piece of information is the geometry to which the text is to refer, here referred to as the location geometry. A second geometry may be required to visually connect the placed text and the location geometry, especially where the location geometry is crowded in an area with other close-by features. This other geometry is referred to here as a leader line, and is a displayable curve of no geographic significance. If indexing is used, the envelope or minimum bounding box of the text is a handy piece of information that should be available. In unavailable, the envelope can be calculated from the processes of placing the text. Since this is often cumbersome, precalculating the envelope and storing it is often the most efficient manner to use this information. The other information associated to the annotation text is the various style information, such as the size of the text (usually in units appropriate to the display, such as pixels or points), the font used, characteristics of the font. This is represented in UML in Figure 24.

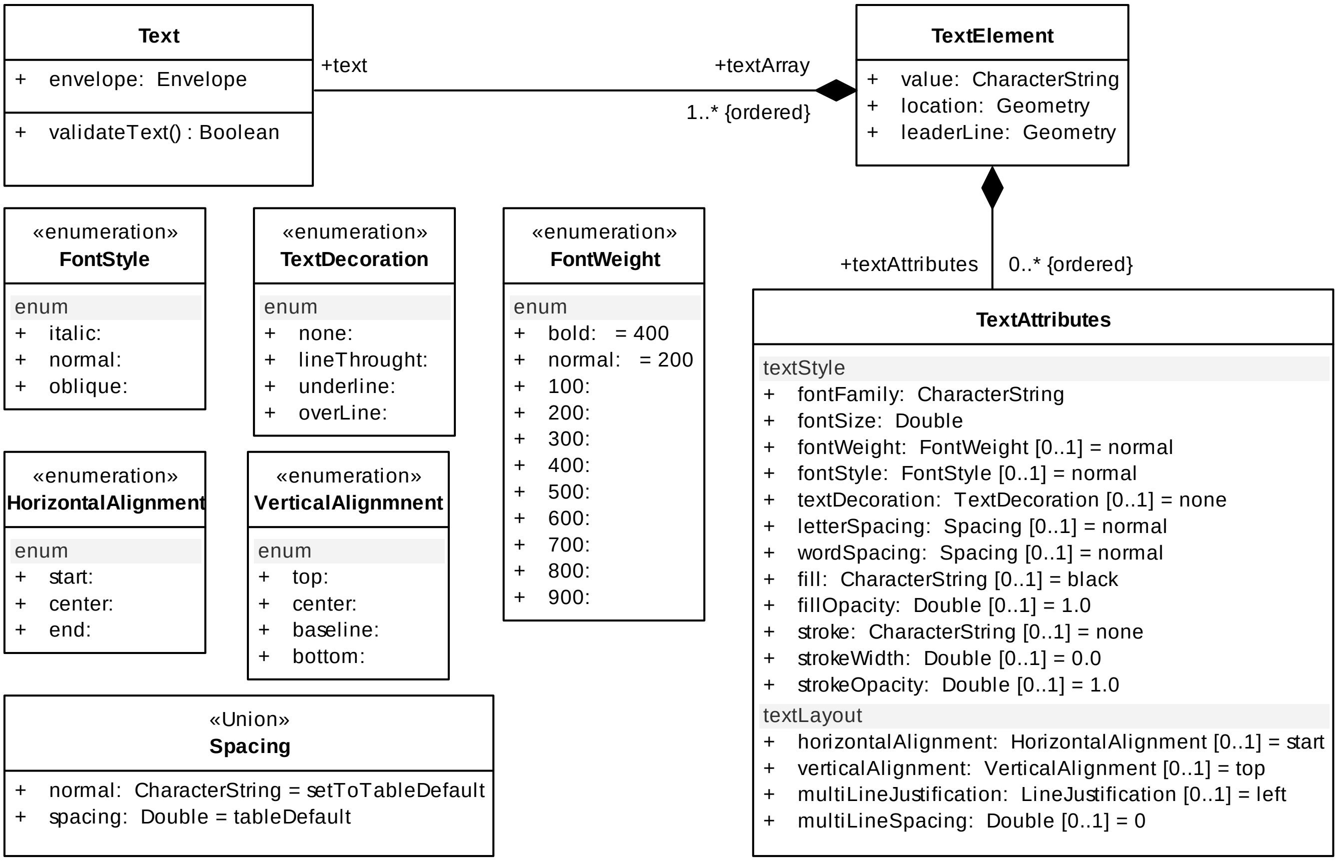


Figure 24: Text classes

Table 2: Fields of the Annotation Text object type

Field Name	Type	Requirements and Defaults
text array	Array of ANNOTATION_TEXT_ELEMENT objects (ANNOTATION_TEXT_ELEMENT object type described in Table 2)	ARRAY must be of least length of 1 or no text is displayed
envelope	GEOMETRY	Required: A geometry envelope used for spatial indexing.

Table 3: Fields of the Annotation Text Element object type

Field Name	Type	Requirements and Defaults
value	VARCHAR2(2000)	Optional –Text to place is first derived from the contents of VALUE in the current element, if VALUE is not null. Otherwise, text is derived from the first non-null preceding element VALUE. If all preceding elements have null VALUE fields, VALUE is derived from the TEXT_EXPRESSION in the metadata table.
location	GEOMETRY	Optional – no text will be displayed if LOCATION is NULL. Locating geometries can be a point or curve type.
text attributes	XML_TYPE (a character string in XML)	Optional – however no text will be displayed if there is not the minimum number of style attributes in either the table metadata (the default values) or the instance.
leader line	GEOMETRY (a curve type)	Optional – if null, there is no leader line.

6.2.2 Text attributes

In addition to the placement information, a set of representation descriptors are needed to properly display the text. These are stored with the text information. These text display attributes or properties include the fields listed in the following tables:

Table 4: Attributes for textstyle

Attribute	Type	Description	Requirements and Defaults
font-family	String	Names such as Arial, Helvetica, Times New Roman. There is no guarantee that the glyphs exist on the client system. These names can be delimited by a semi-colon (;) in SVG and indicate an ordered list of names to use. Ex: Helvetica; Arial	Required to be non-empty string. Server cannot check if valid.
font-size	Float	Size of the text based on the sum of the font ascender, descender and internal leading in points. Note that this value is used in conjunction with a table metadata value indicating the map scale at which this FontSize was determined. In this manner, text that is sized along with the geometry objects is enabled. If the metadata value is null, the text size is fixed. Applications are responsible for calculating the correct size to render the text.	Required positive number.
font-weight	enumeration	Allows for Normal, Bold, or 100, 200, 300, 400, 500, 600, 700, 800 or 900. Normal is the same as 200. Bold is the same as 400.	Defaults to normal.

Attribute	Type	Description	Requirements and Defaults
font-style	enumeration	Normal, Italic or Oblique. Oblique is optional in SVG. It is meant to the opposite angle of italic, slanted left instead of the Italic right. As this has little actual support, the recommendation is that clients just use italic.	Defaults to normal.
text-decoration	enumeration	None, underline, line-through and over-line. Underline is drawn at the baseline, over-line at the baseline + ascent, line-through at baseline + (.5 * ascent). The line is drawn in both the fill and stroke colors, if they exist (see below).	Defaults to none.
letter-spacing	Float and “normal”	SVG allows numbers or “normal”	Defaults to normal.
word-spacing	Float and “normal”	Same as letter spacing but used between words.	Defaults to normal.
fill	String (Fill Type)	This specifies the color of the interior of the glyphs. Colors can be specified in the following manner: 1. Well known SVG font names such as black, blue, red. See http://www.w3.org/TR/SVG/types.html#ColorKeywords . 2. RGB values specified using function syntax such as rgb(255, 0, 255) is a magenta 3. A literal hex value such as #FF00FF which would be the same as the previous RGB example. In general, the Fill should be regarded as the main color of the text. While it should be allowed to render the text with a stroke and no fill, applications that support just a single color should use the fill color.	Defaults to black.
fill-opacity	Float(0-1)	A percentage that specifies the opacity or translucency of the fill. A 0 is fully transparent and 1 is fully opaque.	Defaults to 1
stroke	String (Stroke Type)	This specifies the color of the outline of the glyphs. Stroke allows the same color values as Fill. It is our proposal that we define, contrary to SVG, that the stroke be drawn *before* the fill, which creates a very nice shadow background effect around the text.	Defaults to none.
stroke-width	Float	A width value specifying the stroke width in points.	Defaults to 0. Zero or the lack of this attribute indicates no stroke.
stroke-opacity	Float(0-1)	A percentage that specifies the opacity or translucency of the stroke. A 0 is fully transparent and 1 is fully opaque.	Defaults to 1

Table 5: Attributes for Text Layout

Attribute	Type	Description	Requirements and Defaults
horizontal alignment	Enumeration	3 allowable values which are: "*start", "center", "end"***.** The meaning of these attributes is such that the appropriate part of the text is placed at the point or starting point of the geometry. For example, start means that the first characters of the text is placed there. Note that this means the text is positioned to the right of the geometry.	Optional defaults to "start"
vertical alignment	Enumeration	4 allowable values which are: "top", "center", "baseline" and "bottom". The meaning is similar to that of horizontal alignment. For example, "top" means that the topmost part of the text glyph is placed at the geometry start location.	Optional defaults to "top".
multiline justification	Enumeration	3 allowable values. These are: left, center, and right, The meaning of these attributes is such that each text line is appropriately justified in relation to each other.	Optional as it is not needed in single line text. Defaults to "left"
multiline spacing	Float	A value in points determining the space between lines of text as measured from the bottom of one line to the top of the next.	Optional as it is not needed in single line text. Defaults to 0 which puts each line immediately below the previous one

6.2.3 XML for Text Attributes

The following is a schema for the text attribute XML used as metadata in a text metadata table or object and as text element overrides. It is presented without a namespace. The values for color are as defined in SVG.

```
<?xml version="1.0" encoding="UTF-8"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema" elementFormDefault="qualified"
attributeFormDefault="unqualified">
    <xs:complexType name="textAttributesType">
        <xs:sequence>
            <xs:element ref="textStyle"/>
            <xs:element ref="textlayout"/>
        </xs:sequence>
```

```xml
    </xs:complexType>
    <xs:element name="textAttributes" type="textAttributesType"/>
    <xs:element name="textStyle">
        <xs:annotation>
            <xs:documentation>Text font style attribute</xs:documentation>
        </xs:annotation>
        <xs:complexType>
            <xs:attribute name="font-family" type="xs:string" use="required"/>
            <xs:attribute name="font-size" type="xs:float" use="required"/>
            <xs:attribute name="font-weight" type="fontWeight" use="optional" default="Normal"/>
            <xs:attribute name="font-style" type="fontStyle" use="optional" default="Normal"/>
            <xs:attribute name="text-decoration" type="textDecoration" use="optional"
default="None"/>
            <xs:attribute name="letter-spacing" use="optional" default="Normal"/>
            <xs:attribute name="word-spacing" type="spacing" use="optional" default="Normal"/>
            <xs:attribute name="fill" type="colorType" use="optional" default="black"/>
            <xs:attribute name="fill-opacity" type="opacity" use="optional" default="1.0"/>
            <xs:attribute name="stroke" type="colorType" use="optional" default="black"/>
            <xs:attribute name="stroke-width" type="xs:float" use="optional" default="1.0"/>
            <xs:attribute name="stroke-opacity" type="opacity" use="optional" default="1.0"/>
        </xs:complexType>
    </xs:element>
    <xs:element name="textlayout">
        <xs:annotation>
            <xs:documentation>Text alignment and justification </xs:documentation>
        </xs:annotation>
        <xs:complexType>
            <xs:attribute name="horizontalAlignment" use="optional" default="start">
                <xs:simpleType>
                    <xs:restriction base="xs:string">
                        <xs:enumeration value="start"/>
                        <xs:enumeration value="center"/>
                        <xs:enumeration value="end"/>
                    </xs:restriction>
                </xs:simpleType>
            </xs:attribute>
            <xs:attribute name="verticalAlignment" use="optional" default="top">
                <xs:simpleType>
                    <xs:restriction base="xs:string">
                        <xs:enumeration value="top"/>
                        <xs:enumeration value="center"/>
                        <xs:enumeration value="baseline"/>
                        <xs:enumeration value="bottom"/>
                    </xs:restriction>
                </xs:simpleType>
            </xs:attribute>
            <xs:attribute name="multilineJustification" use="optional" default="left">
                <xs:simpleType>
                    <xs:restriction base="xs:string">
```

```xml
                    <xs:enumeration value="left"/>
                    <xs:enumeration value="center"/>
                    <xs:enumeration value="right"/>
                </xs:restriction>
            </xs:simpleType>
        </xs:attribute>
        <xs:attribute name="multilineSpacing" type="xs:float" use="optional" default="0.0"/>
    </xs:complexType>
</xs:element>
<xs:simpleType name="fontWeight">
    <xs:restriction base="xs:string">
        <xs:enumeration value="Normal"/>
        <xs:enumeration value="Bold"/>
        <xs:enumeration value="100"/>
        <xs:enumeration value="200"/>
        <xs:enumeration value="300"/>
        <xs:enumeration value="400"/>
        <xs:enumeration value="500"/>
        <xs:enumeration value="600"/>
        <xs:enumeration value="700"/>
        <xs:enumeration value="800"/>
        <xs:enumeration value="900"/>
    </xs:restriction>
</xs:simpleType>
<xs:simpleType name="fontStyle">
    <xs:restriction base="xs:string">
        <xs:enumeration value="Normal"/>
        <xs:enumeration value="Italics"/>
        <xs:enumeration value="Oblique"/>
    </xs:restriction>
</xs:simpleType>
<xs:simpleType name="textDecoration">
    <xs:restriction base="xs:string">
        <xs:enumeration value="None"/>
        <xs:enumeration value="Underline"/>
        <xs:enumeration value="LineThrough"/>
        <xs:enumeration value="Overline"/>
    </xs:restriction>
</xs:simpleType>
<xs:simpleType name="spacing">
    <xs:union>
        <xs:simpleType>
            <xs:restriction base="xs:string">
                <xs:enumeration value="Normal"/>
```

```xml
                </xs:restriction>
            </xs:simpleType>
            <xs:simpleType>
                <xs:restriction base="xs:float"/>
            </xs:simpleType>
        </xs:union>
    </xs:simpleType>
    <xs:simpleType name="colorType">
        <xs:union>
            <xs:simpleType>
                <xs:restriction base="xs:string">
                    <xs:pattern value="(rgb\(N,N,N\))"/>
                </xs:restriction>
            </xs:simpleType>
            <xs:simpleType>
                <xs:restriction base="xs:string">
                    <xs:enumeration value="none"/>
                    <xs:enumeration value="aliceblue"/>
                    <xs:enumeration value="antiquewhite"/>
                    <xs:enumeration value="aqua"/>
                    <xs:enumeration value="aquamarine"/>
                    <xs:enumeration value="azure"/>
                    <xs:enumeration value="beige"/>
                    <xs:enumeration value="bisque"/>
                    <xs:enumeration value="black"/>
                    <xs:enumeration value="blanchedalmond"/>
                    <xs:enumeration value="blue"/>
                    <xs:enumeration value="blueviolet"/>
                    <xs:enumeration value="brown"/>
                    <xs:enumeration value="burlywood"/>
                    <xs:enumeration value="cadetblue"/>
                    <xs:enumeration value="chartreuse"/>
                    <xs:enumeration value="chocolate"/>
                    <xs:enumeration value="coral"/>
                    <xs:enumeration value="cornflowerblue"/>
                    <xs:enumeration value="cornsilk"/>
                    <xs:enumeration value="crimson"/>
                    <xs:enumeration value="cyan"/>
                    <xs:enumeration value="darkblue"/>
                    <xs:enumeration value="darkcyan"/>
                    <xs:enumeration value="darkgoldenrod"/>
                    <xs:enumeration value="darkgray"/>
                    <xs:enumeration value="darkgreen"/>
                    <xs:enumeration value="darkgrey"/>
                    <xs:enumeration value="darkkhaki"/>
                    <xs:enumeration value="darkmagenta"/>
                    <xs:enumeration value="darkolivegreen"/>
                    <xs:enumeration value="darkorange"/>
                    <xs:enumeration value="darkorchid"/>
```

```xml
<xs:enumeration value="darkred"/>
<xs:enumeration value="darksalmon"/>
<xs:enumeration value="darkseagreen"/>
<xs:enumeration value="darkslateblue"/>
<xs:enumeration value="darkslategray"/>
<xs:enumeration value="darkslategrey"/>
<xs:enumeration value="darkturquoise"/>
<xs:enumeration value="darkviolet"/>
<xs:enumeration value="deeppink"/>
<xs:enumeration value="deepskyblue"/>
<xs:enumeration value="dimgray"/>
<xs:enumeration value="dimgrey"/>
<xs:enumeration value="dodgerblue"/>
<xs:enumeration value="firebrick"/>
<xs:enumeration value="floralwhite"/>
<xs:enumeration value="forestgreen"/>
<xs:enumeration value="fuchsia"/>
<xs:enumeration value="gainsboro"/>
<xs:enumeration value="ghostwhite"/>
<xs:enumeration value="gold"/>
<xs:enumeration value="goldenrod"/>
<xs:enumeration value="gray"/>
<xs:enumeration value="grey"/>
<xs:enumeration value="green"/>
<xs:enumeration value="greenyellow"/>
<xs:enumeration value="honeydew"/>
<xs:enumeration value="hotpink"/>
<xs:enumeration value="indianred"/>
<xs:enumeration value="indigo"/>
<xs:enumeration value="ivory"/>
<xs:enumeration value="khaki"/>
<xs:enumeration value="lavender"/>
<xs:enumeration value="lavenderblush"/>
<xs:enumeration value="lawngreen"/>
<xs:enumeration value="lemonchiffon"/>
<xs:enumeration value="lightblue"/>
<xs:enumeration value="lightcoral"/>
<xs:enumeration value="lightcyan"/>
<xs:enumeration value="lightgoldenrodyellow"/>
<xs:enumeration value="lightgray"/>
<xs:enumeration value="lightgreen"/>
<xs:enumeration value="lightgrey"/>
<xs:enumeration value="lightpink"/>
<xs:enumeration value="lightsalmon"/>
```

```xml
<xs:enumeration value="lightseagreen"/>
<xs:enumeration value="lightskyblue"/>
<xs:enumeration value="lightslategray"/>
<xs:enumeration value="lightslategrey"/>
<xs:enumeration value="lightsteelblue"/>
<xs:enumeration value="lightyellow"/>
<xs:enumeration value="lime"/>
<xs:enumeration value="limegreen"/>
<xs:enumeration value="linen"/>
<xs:enumeration value="magenta"/>
<xs:enumeration value="maroon"/>
<xs:enumeration value="mediumaquamarine"/>
<xs:enumeration value="mediumblue"/>
<xs:enumeration value="mediumorchid"/>
<xs:enumeration value="mediumpurple"/>
<xs:enumeration value="mediumseagreen"/>
<xs:enumeration value="mediumslateblue"/>
<xs:enumeration value="mediumspringgreen"/>
<xs:enumeration value="mediumturquoise"/>
<xs:enumeration value="mediumvioletred"/>
<xs:enumeration value="midnightblue"/>
<xs:enumeration value="mintcream"/>
<xs:enumeration value="mistyrose"/>
<xs:enumeration value="moccasin"/>
<xs:enumeration value="navajowhite"/>
<xs:enumeration value="navy"/>
<xs:enumeration value="oldlace"/>
<xs:enumeration value="olive"/>
<xs:enumeration value="olivedrab"/>
<xs:enumeration value="orange"/>
<xs:enumeration value="orangered"/>
<xs:enumeration value="orchid"/>
<xs:enumeration value="palegoldenrod"/>
<xs:enumeration value="palegreen"/>
<xs:enumeration value="paleturquoise"/>
<xs:enumeration value="palevioletred"/>
<xs:enumeration value="papayawhip"/>
<xs:enumeration value="peachpuff"/>
<xs:enumeration value="peru"/>
<xs:enumeration value="pink"/>
<xs:enumeration value="plum"/>
<xs:enumeration value="powderblue"/>
<xs:enumeration value="purple"/>
<xs:enumeration value="red"/>
<xs:enumeration value="rosybrown"/>
<xs:enumeration value="royalblue"/>
<xs:enumeration value="saddlebrown"/>
<xs:enumeration value="salmon"/>
<xs:enumeration value="sandybrown"/>
```

```xml
                <xs:enumeration value="seagreen"/>
                <xs:enumeration value="seashell"/>
                <xs:enumeration value="sienna"/>
                <xs:enumeration value="silver"/>
                <xs:enumeration value="skyblue"/>
                <xs:enumeration value="slateblue"/>
                <xs:enumeration value="slategray"/>
                <xs:enumeration value="slategrey"/>
                <xs:enumeration value="snow"/>
                <xs:enumeration value="springgreen"/>
                <xs:enumeration value="steelblue"/>
                <xs:enumeration value="tan"/>
                <xs:enumeration value="teal"/>
                <xs:enumeration value="thistle"/>
                <xs:enumeration value="tomato"/>
                <xs:enumeration value="turquoise"/>
                <xs:enumeration value="violet"/>
                <xs:enumeration value="wheat"/>
                <xs:enumeration value="white"/>
                <xs:enumeration value="whitesmoke"/>
                <xs:enumeration value="yellow"/>
                <xs:enumeration value="yellowgreen"/>
            </xs:restriction>
        </xs:simpleType>
    </xs:union>
  </xs:simpleType>
  <xs:simpleType name="opacity">
    <xs:restriction base="xs:float">
        <xs:minInclusive value="0.0"/>
        <xs:maxInclusive value="1.0"/>
    </xs:restriction>
  </xs:simpleType>
</xs:schema>
```

7 Well-known Text Representation for Geometry

7.1 Component overview

Each Geometry Type has a Well-known Text Representation that can be used both to construct new instances of the type and to convert existing instances to textual form for alphanumeric display.

7.2 Component description

7.2.1 BNF Introduction

The Well-known Text Representation of Geometry is defined below using BNF.

- The notation “{}” denotes an optional token within the braces; the braces do not appear in the output token list.
- The notation "( )" groups a sequence of tokens into a single token; the parentheses do not appear in the output token list.
- The notation "*" after a token denotes the optional use of multiple instances of that token.
- A character string without any modifying symbols denotes an instance of that character string as a single token.
- The notation "|" denotes a choice of two tokens, and do not appear in the output token list,
- The notation "< >" denotes a production defined elsewhere in the list or a basic type.
- The notation ":=" is a production and the grammar on the left may be replaced with the grammar on the right of this symbol. Production is terminated when no undefined production equations are left unresolved.

The text representation of the instantiable Geometry Types implemented shall conform to this grammar. Well known text is case insensitive. Where human readability is important (as in the examples in this standard), an “upper camel-case” where each embedded word is capitalized, should be used.

Note All productions are segregated by coordinate type. This means that any two subelements of any element will always have the same coordinate type, which will be the coordinate type of the larger containing element.

The grammar in this and the following 4 clauses has been designed to support a compact and readable textual representation of geometric objects. The representation of a geometric object that consists of a set of homogeneous components does not include the tags for each embedded component. This first set of productions is to define a double precision literal.

```
<x> ::=                           <signed numeric literal>

<y> ::=                           <signed numeric literal>

<z> ::=                           <signed numeric literal>

<m> ::=                           <signed numeric literal>

<quoted name> ::=                 <double quote> <name> <double quote>

<name> ::=                        <letters>

<letters> ::=                     (<letter>)*

<letter> ::=                      <simple Latin letter>|<digit>|<special>

<simple Latin letter> ::=         <simple Latin upper case letter>
                                     |<simple Latin lower case letter>

<signed numeric literal> ::=      {<sign>}<unsigned numeric literal>

<unsigned numeric literal> ::=    <exact numeric literal>
                                     |<approximate numeric literal>
```

```
<approximate numeric          <mantissa>E<exponent>
   literal> ::=

<mantissa> ::=                <exact numeric literal>

<exponent> ::=                <signed integer>

<exact numeric literal> ::=   <unsigned integer>
                                 {<decimal point>{<unsigned integer>}}
                                 |<decimal point><unsigned integer>

<signed integer> ::=          {<sign>}<unsigned integer>

<unsigned integer> ::=        (<digit>)*

<left delimiter> ::=          <left paren>|<left bracket>
                                 // must match balancing right delimiter

<right delimiter> ::=         <right paren>|<right bracket>
                                 // must match balancing left delimiter

<special> ::=                 <right paren>|<left paren>|<minus sign>
                                 |<underscore>|<period>|<quote>|<space>

<sign> ::=                    <plus sign> | <minus sign>

<decimal point> ::=           <period> | <comma>

<empty set> ::=               EMPTY

<minus sign> ::=              -

<left paren> ::=              (

<right paren> ::=             )

<left bracket> ::=            [

<right bracket> ::=           ]

<period> ::=                  .

<plus sign> ::=               +

<double quote> ::=            "

<quote> ::=                   '

<comma>                       ,

<underscore> ::=              _
```

```
<digit> ::=                         0|1|2|3|4|5|6|7|8|9

<simple Latin lower case            a|b|c|d|e|f|g|h|i|j|k|l|m
   letter> ::=                         |n|o|p|q|r|s|t|u|v|w|x|y|z

<simple Latin upper case            A|B|C|D|E|F|G|H|I|J|K|L|M
   letter> ::=                         |N|O|P|Q|R|S|T|U|V|W|X|Y|Z

<space>=                            " "
                                       // unicode "U+0020" (space)
```

7.2.2 BNF Productions for Two-Dimension Geometry WKT

The following BNF defines two-dimensional geometries in (x, y) coordinate spaces. With the exception of the addition of polyhedral surfaces, these structures are unchanged from earlier editions of this standard.

```
<point> ::=                                 <x> <y>

<geometry tagged text> ::=                  <point tagged text>
                                               | <linestring tagged text>
                                               | <polygon tagged text>
                                               | <triangle tagged text>
                                               | <polyhedralsurface tagged text>
                                               | <tin tagged text>
                                               | <multipoint tagged text>
                                               | <multilinestring tagged text>
                                               | <multipolygon tagged text>
                                               | <geometrycollection tagged text>

<point tagged text> ::=                     point <point text>

<linestring tagged text> ::=                linestring <linestring text>

<polygon tagged text> ::=                   polygon <polygon text>

<polyhedralsurface tagged text> ::=         polyhedralsurface
                                               <polyhedralsurface text>

<triangle tagged text> ::=                  triangle <polygon text>

<tin tagged text>                           tin <polyhedralsurface text>

<multipoint tagged text> ::=                multipoint <multipoint text>

<multilinestring tagged text> ::=           multilinestring <multilinestring text>

<multipolygon tagged text> ::=              multipolygon <multipolygon text>

<geometrycollection tagged text> ::=        geometrycollection
                                               <geometrycollection text>

<point text> ::=                            <empty set> | <left paren> <point> <right
                                               paren>
```

```
<linestring text> ::=                 <empty set> | <left paren>
                                         <point>
                                         {<comma> <point>}*
                                         <right paren>

<polygon text> ::=                    <empty set> | <left paren>
                                         <linestring text>
                                         {<comma> <linestring text>}*
                                         <right paren>

<polyhedralsurface text> ::=          <empty set> | <left paren>
                                         <polygon text>
                                         {<comma> <polygon text>}*
                                         <right paren>

<multipoint text> ::=                 <empty set> | <left paren>
                                         <point text>
                                         {<comma> <point text>}*
                                         <right paren>

<multilinestring text> ::=            <empty set> | <left paren>
                                         <linestring text>
                                         {<comma> <linestring text>}*
                                         <right paren>

<multipolygon text> ::=               <empty set> | <left paren>
                                         <polygon text>
                                         {<comma> <polygon text>}*
                                         <right paren>

<geometrycollection text> ::=         <empty set> | <left paren>
                                         <geometry tagged text>
                                         {<comma> <geometry tagged text>}*
                                         <right paren>
```

7.2.3 BNF Productions for Three-Dimension Geometry WKT

The following BNF defines geometries in 3 dimensional (x, y, z) coordinates.

```
<point z> ::=                         <x> <y> <z>

<geometry z tagged text> ::=          <point z tagged text>
                                         |<linestring z tagged text>
                                         |<polygon z tagged text>
                                         |<polyhedronsurface z tagged text>
                                         |<triangle tagged text>
                                         |<tin tagged text>
                                         |<multipoint z tagged text>
                                         |<multilinestring z tagged text>
                                         |<multipolygon z tagged text>
                                         |<geometrycollection z tagged text>
```

```
<point z tagged text> ::=             point z <point z text>

<linestring z tagged text> ::=        linestring z <linestring z text>

<polygon z tagged text> ::=           polygon z <polygon z text>

<polyhedralsurface z tagged text> ::= polyhedralsurface z
                                         <polyhedralsurface z text>

<triangle z tagged text> ::=          triangle z <polygon z text>

<tin z tagged text>                   tin z <polyhedralsurface z text>

<multipoint z tagged text> ::=        multipoint z <multipoint z text>

<multilinestring z tagged text> ::=   multilinestring z <multilinestring z text>

<multipolygon z tagged text> ::=      multipolygon z <multipolygon z text>

<geometrycollection z tagged          geometrycollection z
   text> ::=                             <geometrycollection z text>

<point z text> ::=                    <empty set> | <left paren> <point z>
                                         <right paren>

<linestring z text> ::=               <empty set> | <left paren> <point z>
                                         {<comma> <point z>}*
                                         <right paren>

<polygon z text> ::=                  <empty set> | <left paren>
                                         <linestring z text>
                                         {<comma> <linestring z text>}*
                                         <right paren>

<polyhedralsurface z text> ::=        <empty set>|<left paren>
                                         <polygon z text>
                                         {<comma> <polygon z text>}*
                                         <right paren>

<multipoint z text> ::=               <empty set> | <left paren>
                                         <point z text>
                                         {<comma> <point z text>}*
                                         <right paren>

<multilinestring z text> ::=          <empty set> | <left paren>
                                         <linestring z text>
                                         {<comma> <linestring z text>}*
                                         <right paren>

<multipolygon z text> ::=             <empty set> | <left paren>
                                         <polygon z text>
                                         {<comma> <polygon z text>}*
                                         <right paren>
```

```
<geometrycollection z text> ::=         <empty set> | <left paren>
                                           <geometry tagged z text>
                                           {<comma> <geometry tagged z text>}*
                                           <right paren>
```

7.2.4 BNF Productions for Two-Dimension Measured Geometry WKT

The following BNF defines two-dimensional geometries in (x, y) coordinate spaces. In addition, each coordinate carries an "m" ordinate value that is part of some linear reference system.

```
<point m> ::=                           <x> <y> <m>

<geometry m tagged text> ::=            <point m tagged text>
                                           |<linestring m tagged text>
                                           |<polygon m tagged text>
                                           |<polyhedralsurface m tagged text>
                                           |<triangle tagged m text>
                                           |<tin tagged m text>
                                           |<multipoint m tagged text>
                                           |<multilinestring m tagged text>
                                           |<multipolygon m tagged text>
                                           |<geometrycollection m tagged text>

<point m tagged text> ::=               point m <point m text>

<linestring m tagged text> ::=          linestring m <linestring m text>

<polygon m tagged text> ::=             polygon m <polygon m text>

<polyhedralsurface m tagged text> ::=   polyhedralsurface m
                                           <polyhedralsurface m text>

<triangle m tagged text> ::=            triangle m <polygon m text>

<tin m tagged text>                     tin m <polyhedralsurface m text>

<multipoint m tagged text> ::=          multipoint m <multipoint m text>

<multilinestring m tagged text> ::=     multilinestring m <multilinestring m text>

<multipolygon m tagged text> ::=        multipolygon m <multipolygon m text>

<geometrycollection m tagged            geometrycollection m
   text> ::=                               <geometrycollection m text>

<point m text> ::=                      <empty set> | <left paren>
                                           <point m>
                                           <right paren>
```

```
<linestring m text> ::=               <empty set> | <left paren>
                                         <point m>
                                         {{<comma> <point m>}+
                                         <right paren>

<polygon m text> ::=                  <empty set> | <left paren>
                                         <linestring m text>
                                         {<comma> <linestring m text>}*
                                         <right paren>

<polyhedralsurface m text> ::=        <empty set> | <left paren>
                                         <polygon m text>
                                         {<comma> <polygon m text>}*
                                         <right paren>

<multipoint m text> ::=               <empty set> | <left paren> <point m text>
                                         {<comma> <point m text>}*
                                         <right paren>

<multilinestring m text> ::=          <empty set> | <left paren>
                                         <linestring m text>
                                         {<comma> <linestring m text>}*
                                         <right paren>

<multipolygon m text> ::=             <empty set> | <left paren>
                                         <polygon m text>
                                         {<comma> <polygon m text>}*
                                         <right paren>

<geometrycollection m text> ::=       <empty set> | <left paren>
                                         <geometry tagged m text>
                                         {<comma> <geometry tagged m text>}*
                                         <right paren>
```

7.2.5 BNF Productions for Three-Dimension Measured Geometry WKT

The following BNF defines three-dimensional geometries in (x, y, z) coordinate spaces. In addition, each coordinate carries an "m" ordinate value that is part of some linear reference system.

```
<point zm> ::=                       <x> <y> <z> <m>

<geometry zm tagged text> ::=        <point zm tagged text>
                                        |<linestring zm tagged text>
                                        |<polygon zm tagged text>
                                        |<polyhedralsurface zm tagged text>
                                        |<triangle zm tagged text>
                                        |<tin zm tagged text>
                                        |<multipoint zm tagged text>
                                        |<multilinestring zm tagged text>
                                        |<multipolygon zm tagged text>
                                        |<geometrycollection zm tagged text>

<point zm tagged text> ::=           point zm <point zm text>

<linestring zm tagged text> ::=      linestring zm <linestring zm text>
```

```
<polygon zm tagged text> ::=           polygon zm <polygon zm text>

<polyhedralsurface zm tagged           polyhedralsurface zm
   text> ::=                              <polyhedralsurface zm text>

<triangle zm tagged text> ::=          triangle zm <polygon zm text>

<tin zm tagged text>                   tin zm <polyhedralsurface zm text>

<multipoint zm tagged text> ::=        multipoint zm <multipoint zm text>

<multipoint zm tagged text> ::=        multipoint zm
                                          <multipoint zm text>

<multilinestring zm tagged text> ::=   multilinestring zm
                                          <multilinestring zm text>

<multipolygon zm tagged text> ::=      multipolygon zm
                                          <MultiPolygon zm text>

<geometrycollection zm tagged          geometrycollection zm
   text> ::=                              <geometrycollection zm text>

<point zm text> ::=                    <empty set> | <left paren> <point zm>
                                          <right paren>

<linestring zm text> ::=               <empty set> | <left paren>
                                          <point z>
                                          {<comma> <point z>}*
                                          <right paren>

<polygon zm text> ::=                  <empty set> | <left paren>
                                          <linestring zm text>
                                          {<comma> <linestring zm text>}*
                                          <right paren>

<polyhedralsurface zm text> ::=        <empty set> | <left paren> {
                                          <polygon zm text
                                          {<comma> <polygon zm text>}*)
                                          <right paren>

<multipoint zm text> ::=               <empty set> | <left paren>
                                          <point zm text>
                                          {<comma> <point zm text>}*
                                          <right paren>

<multilinestring zm text> ::=          <empty set> | <left paren>
                                          <linestring zm text>
                                          {<comma> <linestring zm text>}*
                                          <right paren>
```

```
<multipolygon zm text> ::=            <empty set> | <left paren>
                                         <polygon zm text>
                                         {<comma> <polygon zm text>}*
                                         <right paren>

<geometrycollection zm text> ::=      <empty set> | <left paren>
                                         <geometry tagged zm text>
                                         {<comma> <geometry tagged zm text>}*
                                         <right paren>
```

7.2.6 Examples

Examples of textual representations of Geometry are shown in Table 2. The coordinates are shown as integer values; in general they may be any double precision value.

Note The examples of POINTZ, POINTM, and POINTZM at the bottom of Table 6. This same style for distinguishing 2D points from 3D points and from 2D or 3D points with M value can be applied to LINESTRING, POLYGON, MULTIPOINT, MULTILINESTRING, MULTIPOLYGON, and GEOMETRYCOLLECTION types.

Table 6: Example Well-known Text Representation of Geometry

Geometry Type	Text Literal Representation	Comment
Point	Point (10 10)	a Point
LineString	LineString (10 10, 20 20, 30 40)	a LineString with 3 points
Polygon	Polygon ((10 10, 10 20, 20 20, 20 15, 10 10))	a Polygon with 1 exteriorRing and 0 interiorRings
Multipoint	MultiPoint ((10 10), (20 20))	a MultiPoint with 2 points
MultiLineString	MultiLineString ((10 10, 20 20), (15 15, 30 15))	a MultiLineString with 2 linestrings
MultiPolygon	MultiPolygon (((10 10, 10 20, 20 20, 20 15, 10 10)), ((60 60, 70 70, 80 60, 60 60)))	a MultiPolygon with 2 polygons
GeomCollection	GeometryCollection (POINT (10 10), POINT (30 30), LINESTRING (15 15, 20 20))	a GeometryCollection consisting of 2 Point values and a LineString value
Polyhedron	Polyhedron Z (((0 0 0, 0 0 1, 0 1 1, 0 1 0, 0 0 0)), ((0 0 0, 0 1 0, 1 1 0, 1 0 0, 0 0 0)), ((0 0 0, 1 0 0, 1 0 1, 0 0 1, 0 0 0)), ((1 1 0, 1 1 1, 1 0 1, 1 0 0, 1 1 0)), ((0 1 0, 0 1 1, 1 1 1, 1 1 0, 0 1 0)), ((0 0 1, 1 0 1, 1 1 1, 0 1 1. 0 0 1)))	A polyhedron cube, corner at the origin and opposite corner at (1, 1, 1).

Geometry Type	Text Literal Representation	Comment
Tin	Tin Z (((0 0 0, 0 0 1, 0 1 0, 0 0 0)), ((0 0 0, 0 1 0, 1 0 0, 0 0 0)), ((0 0 0, 1 0 0, 0 0 1, 0 0 0)), ((1 0 0, 0 1 0, 0 0 1, 1 0 0)),)	A tetrahedron (4 triangular faces), corner at the origin and each unit coordinate digit.
Point	Point Z (10 10 5)	a 3D Point
Point	Point ZM (10 10 5 40)	the same 3D Point with M value of 40
Point	Point M (10 10 40)	a 2D Point with M value of 40

8 Well-known Binary Representation for Geometry

8.1 Component overview

The Well-known Binary Representation for Geometry (`WKBGeometry`) provides a portable representation of a geometric object as a contiguous stream of bytes. It permits geometric object to be exchanged between an SQL/CLI client and an SQL-implementation in binary form.

8.2 Component description

8.2.1 Introduction

The Well-known Binary Representation for Geometry is obtained by serializing a geometric object as a sequence of numeric types drawn from the set {`Unsigned Integer, Double`} and then serializing each numeric type as a sequence of bytes using one of two well defined, standard, binary representations for numeric types (NDR, XDR). The specific binary encoding (NDR or XDR) used for a geometry representation is described by a one-byte tag that precedes the serialized bytes. The only difference between the two encodings of geometry is one of byte order, the XDR encoding is Big Endian, and the NDR encoding is Little Endian.

8.2.2 Numeric type definitions

An `Unsigned Integer` is a 32-bit (4-byte) data type that encodes a nonnegative integer in the range [0, 4,294,967,295].

A `Double` is a 64-bit (8-byte) double precision datatype that encodes a double precision number using the IEEE 754[18] double precision format.

The above definitions are common to both XDR and NDR.

8.2.3 A common list of codes for geometric types

In this clause and in other places in this multipart standard, geometric types are identified by integer codes. To keep these codes in synchrony and to reserve sections for future use, we define a list here for all geometric object types in this standard or planned for future releases. The shaded codes in the table below are for future use and do not reflect types used here

Table 7: Integer codes for geometric types

Type	Code
Geometry	0
Point	1
LineString	2
Polygon	3
MultiPoint	4
MultiLineString	5
MultiPolygon	6
GeometryCollection	7
CircularString	8
CompoundCurve	9
CurvePolygon	10
MultiCurve	11
MultiSurface	12
Curve	13
Surface	14
PolyhedralSurface	15
TIN	16

Type	Code
Geometry Z	1000
Point Z	1001
LineString Z	1002
Polygon Z	1003
MultiPoint Z	1004
MultiLineString Z	1005
MultiPolygon Z	1006
GeometryCollection Z	1007
CircularString Z	1008
CompoundCurve Z	1009
CurvePolygon Z	1010
MultiCurve Z	1011
MultiSurface Z	1012
Curve Z	1013
Surface Z	1014
PolyhedralSurface Z	1015
TIN Z	1016

Type	Code
`Geometry M`	2000
`Point M`	2001
`LineString M`	2002
`Polygon M`	2003
`MultiPoint M`	2004
`MultiLineString M`	2005
`MultiPolygon M`	2006
`GeometryCollection M`	2007
`CircularString M`	2008
`CompoundCurve M`	2009
`CurvePolygon M`	2010
`MultiCurve M`	2011
`MultiSurface M`	2012
`Curve M`	2013
`Surface M`	2014
`PolyhedralSurface M`	2015
`TIN M`	2016

Type	Code
`Geometry ZM`	3000
`Point ZM`	3001
`LineString ZM`	3002
`Polygon ZM`	3003
`MultiPoint ZM`	3004
`MultiLineString ZM`	3005
`MultiPolygon ZM`	3006
`GeometryCollection ZM`	3007
`CircularString ZM`	3008
`CompoundCurve ZM`	3009
`CurvePolygon ZM`	3010
`MultiCurve ZM`	3011
`MultiSurface ZM`	3012
`Curve ZM`	3013
`Surface ZM`	3014
`PolyhedralSurface ZM`	3015
`TIN ZM`	3016

8.2.4 XDR (Big Endian) encoding of numeric types

The XDR representation of an `Unsigned Integer` is Big Endian (most significant byte first).

The XDR representation of a `Double` is Big Endian (sign bit is first byte).

8.2.5 NDR (Little Endian) encoding of numeric types

The NDR representation of an `Unsigned Integer` is Little Endian (least significant byte first).

The NDR representation of a `Double` is Little Endian (sign bit is last byte).

8.2.6 Conversions between the NDR and XDR representations of WKBGeometry

Conversion between the NDR and XDR data types for `Unsigned Integer` and `Double` numbers is a simple operation involving reversing the order of bytes within each `Unsigned Integer` or `Double` number in the representation.

8.2.7 Relationship to other COM and CORBA data transfer protocols

The XDR representation for `Unsigned Integer` and `Double` numbers described above is also the standard representation for `Unsigned Integer` and for `Double` number in the CORBA Standard Stream Format for Externalized Object Data that is described as part of the CORBA Externalization Service Specification [15].

The NDR representation for `Unsigned Integer` and `Double` number described above is also the standard representation for `Unsigned Integer` and for Double number in the DCOM protocols that is based on DCE RPC and NDR [16].

8.2.8 Description of WKBGeometry representations

The Well-known Binary Representation for Geometry is described below. The basic building block is the representation for a Point, which consists of a number Doubles, depending on the coordinate referece system in use for the geometry. The representations for other geometric objects are built using the representations for geometric objects that have already been defined.

```
// Basic Type definitions
// byte : 1 byte
// uint32 : 32 bit unsigned integer (4 bytes)
// double : double precision number (8 bytes)

// Building Blocks : Coordinate, LinearRing
Point {
  double x;
  double y}

PointZ {
  double x;
  double y;
  double z}

PointM {
  double x;
  double y;
  double m}

PointZM {
  double x;
  double y;
  double z;
  double m}

LinearRing  {
  uint32  numPoints;
  Point   points[numPoints]}

LinearRingZ  {
  uint32  numPoints;
  PointZ  points[numPoints]}

LinearRingM  {
  uint32  numPoints;
  PointM  points[numPoints]}

LinearRingZM  {
  uint32  numPoints;
  PointZM points[numPoints]}
```

```
enum WKBByteOrder {
  wkbXDR = 0,        // Big Endian
  wkbNDR = 1         // Little Endian
}

enum WKBGeometryType {
            wkbPoint                      = 1,
            wkbLineString                 = 2,
            wkbPolygon                    = 3,
            wkbTriangle                   = 17
            wkbMultiPoint                 = 4,
            wkbMultiLineString            = 5,
            wkbMultiPolygon               = 6,
            wkbGeometryCollection         = 7,
            wkbPolyhedralSurface          = 15,
            wkbTIN                        = 16

            wkbPointZ                     = 1001,
            wkbLineStringZ                = 1002,
            wkbPolygonZ                   = 1003,
            wkbTrianglez                  = 1017
            wkbMultiPointZ                = 1004,
            wkbMultiLineStringZ           = 1005,
            wkbMultiPolygonZ              = 1006,
            wkbGeometryCollectionZ        = 1007,
            wkbPolyhedralSurfaceZ         = 1015,
            wkbTINZ                       = 1016

            wkbPointM                     = 2001,
            wkbLineStringM                = 2002,
            wkbPolygonM                   = 2003,
            wkbTriangleM                  = 2017
            wkbMultiPointM                = 2004,
            wkbMultiLineStringM           = 2005,
            wkbMultiPolygonM              = 2006,
            wkbGeometryCollectionM        = 2007,
            wkbPolyhedralSurfaceM         = 2015,
            wkbTINM                       = 2016

            wkbPointZM                    = 3001,
            wkbLineStringZM               = 3002,
            wkbPolygonZM                  = 3003,
            wkbTriangleZM                 = 3017
            wkbMultiPointZM               = 3004,
            wkbMultiLineStringZM          = 3005,
            wkbMultiPolygonZM             = 3006,
            wkbGeometryCollectionZM       = 3007,
            wkbPolyhedralSurfaceZM        = 3015,
            wkbTinZM                      = 3016,
}

WKBPoint {
  byte    byteOrder;
  static  uint32      wkbType = 1;
  Point   point}
```

```
WKBPointZ {
  byte                byteOrder;
  static  uint32      wkbType = 1001;
  PointZ              point}

WKBPointM {
  byte                byteOrder;
  static  uint32      wkbType = 2001;
  PointM              point}

WKBPointZM {
  byte                byteOrder;
  static  uint32      wkbType = 3001;
  PointZM             point}

WKBLineString {
  byte                byteOrder;
  static  uint32      wkbType = 2;
  uint32              numPoints;
  Point               points[numPoints]}

WKBLineStringZ {
  byte                byteOrder;
  static  uint32      wkbType = 1002;
  uint32              numPoints;
  PointZ              points[numPoints]}

WKBLineStringM {
  byte                byteOrder;
  static  uint32      wkbType = 2002;
  uint32              numPoints;
  PointM              points[numPoints]}

WKBLineStringZM {
  byte                byteOrder;
  static  uint32      wkbType = 3002;
  uint32              numPoints;
  PointZM             points[numPoints]}

WKBPolygon {
  byte             byteOrder;
  static uint32    wkbType = 3;
  uint32           numRings;
  LinearRing       rings[numRings]}

WKBPolygonZ {
  byte             byteOrder;
  static uint32    wkbType = 1003;
  uint32           numRings;
  LinearRingZ      rings[numRings]}
```

```
WKBPolygonM {
 byte              byteOrder;
 static uint32     wkbType = 2003;
 uint32            numRings;
 LinearRingM       rings[numRings]}

WKBPolygonZM {
 byte              byteOrder;
 static uint32     wkbType = 3003;
 uint32            numRings;
 LinearRingZM      rings[numRings]}

WKBTriangle {
 byte              byteOrder;
 static uint32     wkbType = 17;
 uint32            numRings;
 LinearRing        rings[numRings]}

WKBTriangleZ {
 byte              byteOrder;
 static uint32     wkbType = 10 17;
 uint32            numRings;
 LinearRingZ       rings[numRings]}

WKBTriangleM {
 byte              byteOrder;
 static uint32     wkbType = 20 17;
 uint32            numRings;
 LinearRingM       rings[numRings]}

WKBTriangleZM {
 byte              byteOrder;
 static uint32     wkbType = 30 17;
 uint32            numRings;
 LinearRingZM      rings[numRings]}

WKBPolyhedralSurface {
 byte              byteOrder;
 static  uint32    wkbType = 15;
 uint32            numPolygons;
 WKBPolygon        polygons[numPolygons]}

WKBPolyhedralSurfaceZ {
 byte              byteOrder;
 static  uint32    wkbType=1015;
 uint32            numPolygons;
 WKBPolygonZ       polygons[numPolygons]}

WKBPolyhedralSurfaceM {
 byte              byteOrder;
 static  uint32    wkbType=2015;
 uint32            numPolygons;
 WKBPolygonM       polygons[numPolygons]}
```

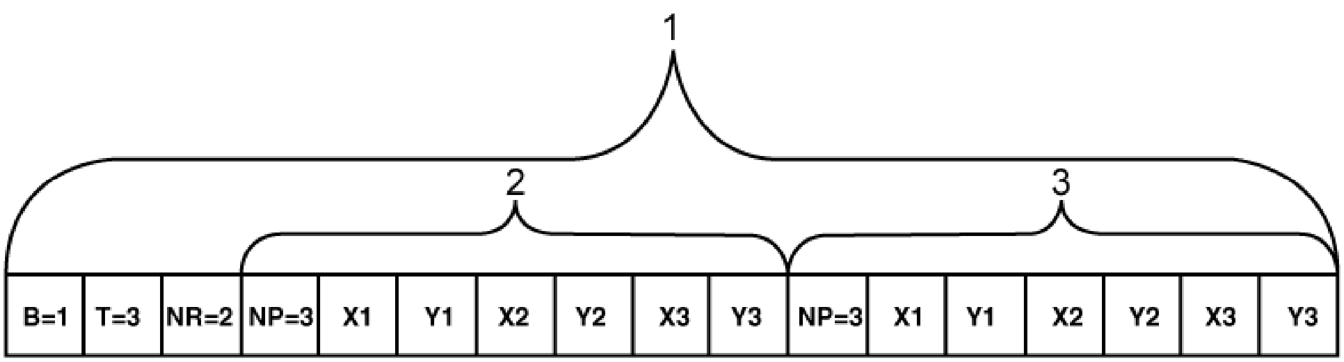
```
WKBPolyhedralSurfaceZM {
  byte             byteOrder;
  static  uint32   wkbType=3015;
  uint32           numPolygons;
  WKBPolygonZM     polygons[numPolygons]}

WKBTIN {
  byte             byteOrder;
  static  uint32   wkbType = 16;
  uint32           numPolygons;
  WKBPolygon       polygons[numPolygons]}

WKBTINZ {
  byte             byteOrder;
  static  uint32   wkbType=1016;
  uint32           numPolygons;
  WKBPolygonZ      polygons[numPolygons]}

WKBTINM {
  byte             byteOrder;
  static  uint32   wkbType=2016;
  uint32           numPolygons;
  WKBPolygonM      polygons[numPolygons]}

WKBTINZM {
  byte             byteOrder;
  static  uint32   wkbType=3016;
  uint32           numPolygons;
  WKBPolygonZM     polygons[numPolygons]}

WKBMultiPoint {
  byte             byteOrder;
  static  uint32   wkbType=4;
  uint32           numPoints;
  WKBPoint         points[numPoints]}

WKBMultiPointZ {
  byte             byteOrder;
  static  uint32   wkbType=1004;
  uint32           numPoints;
  WKBPointZ        points[numPoints]}

WKBMultiPointM {
  byte             byteOrder;
  static  uint32   wkbType=2004;
  uint32           numPoints;
  WKBPointM        points[numPoints]}
```

```
WKBMultiPointZM {
  byte              byteOrder;
  static  uint32    wkbType=3004;
  uint32            numPoints;
  WKBPointZM        points[numPoints]}

WKBMultiLineString {
  byte              byteOrder;
  static  uint32    wkbType = 5;
  uint32            numLineStrings;
  WKBLineString     lineStrings[numLineStrings]}

WKBMultiLineStringZ {
  byte              byteOrder;
  static  uint32    wkbType = 1005;
  uint32            numLineStrings;
  WKBLineStringZ    lineStrings[numLineStrings]}

WKBMultiLineStringM {
  byte              byteOrder;
  static  uint32    wkbType = 2005;
  uint32            numLineStrings;
  WKBLineStringM    lineStrings[numLineStrings]}

WKBMultiLineStringZM {
  byte                 byteOrder;
  static  uint32       wkbType = 3005;
  uint32               numLineStrings;
  WKBLineStringZM      lineStrings[numLineStrings]}

WKBMultiPolygon {
  byte                 byteOrder;
  static  uint32       wkbType = 6;
  uint32               numPolygons;
  WKBPolygon           polygons[numPolygons]}

WKBMultiPolygonZ {
  byte                 byteOrder;
  static  uint32       wkbType = 1006;
  uint32               numPolygons;
  WKBPolygonZ          polygons[numPolygons]}

WKBMultiPolygonM {
  byte              byteOrder;
  static  uint32          wkbType = 2006;
  uint32               numPolygons;
  WKBPolygonM          polygons[numPolygons]}

WKBMultiPolygonZM {
  byte                 byteOrder;
  static  uint32       wkbType = 3006;
  uint32               numPolygons;
  WKBPolygonZM         polygons[numPolygons]}
```

```
WKBGeometryCollection {
  byte                byte_order;
  static  uint32      wkbType = 7;
  uint32              numGeometries;
  WKBGeometry         geometries[numGeometries]}

WKBGeometryCollectionZ {
  byte                byte_order;
  static  uint32      wkbType = 1007;
  uint32              numGeometries;
  WKBGeometryZ        geometries[numGeometries]}

WKBGeometryCollectionM {
  byte                byte_order;
  static  uint32      wkbType = 2007;
  uint32              numGeometries;
  WKBGeometryM        geometries[numGeometries]}

WKBGeometryCollectionZM {
  byte                byte_order;
  static  uint32      wkbType = 3007;
  uint32              numGeometries;
  WKBGeometryZM       geometries[numGeometries]}

WKBGeometry {Union {
       WKBPoint                    point;
       WKBLineString               linestring;
       WKBPolygon                  polygon;
       WKBTriangle                 triangle
       WKBPolyhedralSurface        polyhedralsurface
       WKBTIN                      tin
       WKBMultiPoint               mpoint;
       WKBMultiLineString          mlinestring;
       WKBMultiPolygon             mpolygon;
       WKBGeometryCollection       collection;
  }};

WKBGeometryZ {
  union {
  WKBPointZ               pointz;
  WKBLineStringZ          linestringz;
  WKBPolygonZ             polygonz;
  WKBTriangleZ            trianglez
  WKBPolyhedralSurfaceZ   Polyhedralsurfacez;
  WKBTinZ                 tinz
  WKBMultiPointZ          mpointz;
  WKBMultiLineStringZ     mlinestringz;
  WKBMultiPolygonZ        mpolygonz;
  WKBGeometryCollectionZ  collectionz;
  }};
```

```
WKBGeometryM {Union {
                WKBPointM               pointm;
                WKBLineStringM          linestringm;
                WKBPolygonM             polygonm;
                WKBTriangleM            trianglem
                WKBPolyhedralSurfaceM   Polyhedralsurfacem;
                WKBTinM                 tinm
                WKBMultiPointM          mpointm;
                WKBMultiLineStringM     mlinestringm;
                WKBMultiPolygonM        mpolygonm;
                WKBGeometryCollectionM  collectionm;
  }};

WKBGeometryZM {Union {
               WKBPointZM               pointzm;
               WKBLineStringZM          linestringzm;
               WKBPolygonZM             polygonzm;
               WKBTriangleZM            trianglezm
               WKBPolyhedralSurfaceM    Polyhedralsurfacezm;
               WKBTinZM                 tinzm
               WKBMultiPointZM          mpointzm;
               WKBMultiLineStringZM     mlinestringzm;
               WKBMultiPolygonZ         mpolygonzm;
               WKBGeometryCollectionZM  collectionzm;
  }};
```

Figure 25 shows a pictorial representation of the Well-known Representation for a `Polygon` with one outer ring and one inner ring.



Key

1 WKB Polygon

2 ring 1

3 ring 2

Figure 25: Well-known Binary Representation for a geometric object in NDR format (B = 1) of type Polygon (T = 3) with 2 LinearRings (NR = 2) each LinearRing having 3 points (NP = 3)

8.2.9 Assertions for Well-known Binary Representation for Geometry

The Well-known Binary Representation for Geometry is designed to represent instances of Geometry Types. Any `WKBGeometry` instance shall satisfy the assertions for the type of Geometry that it describes (see 6.1).

9 Well-known Text Representation of Spatial Reference Systems

9.1 Component overview

The Well-known Text Representation of Spatial Reference Systems provides a standard textual representation for spatial reference system information.

9.2 Component description

A Spatial Reference System, also referred to as a coordinate system, is a geographic (latitude-longitude), a projected (X, Y), or a geocentric (X, Y, Z) coordinate system.

The coordinate system is composed of several objects. Each object has a keyword in upper case (for example, DATUM or UNIT) followed by the defining, comma-delimited, parameters of the object in brackets. Some objects are composed of objects so the result is a nested structure. Implementations are free to substitute standard brackets ( ) for square brackets [ ] and should be prepared to read both forms of brackets.

Informative Annex B provides a non-exhaustive list of Geodetic Codes and Parameters for defining the objects in the Well-Known Text Representation for spatial reference information.

The Extended Backus Naur Form (EBNF) definition for the string representation of a coordinate system is as follows, using square brackets. Some definitions for numbers and names are taken from the Geometry WKT.

```
<spatial reference system> ::=      <projected cs> |
                                       <geographic cs> |
                                       <geocentric cs>

<projected cs> ::=                  PROJCS <left delimiter>
                                       <csname>
                                       <comma> <geographic cs>
                                       <comma> <projection>
                                       (<comma> <parameter> )*
                                       <comma> <linear unit>
                                       <right delimiter>

<geographic cs> ::=                 GEOGCS <left delimiter> <csname>
                                       <comma> <datum>
                                       <comma> <prime meridian>
                                       <comma> <angular unit>
                                       (<comma> <linear unit> )
                                       <right delimiter>

<geocentric cs> ::=                 GEOCCS <left delimiter>
                                       <name>
                                       <comma> <datum>
                                       <comma> <prime meridian>
                                       <comma> <linear unit>
                                       <right delimiter>

<datum> ::=                         DATUM <left delimiter> <datum name>
                                       <comma> <spheroid>
                                       <right delimiter>
```

```
<projection> ::=                    PROJECTION <left delimiter>
                                       <projection name>
                                       <right delimiter>

<parameter> ::=                     PARAMETER <left delimiter>
                                       <parameter name>
                                       <comma> <value>
                                       <right delimiter>

<spheroid> ::=                      SPHEROID <left delimiter>
                                       <spheroid name>
                                       <comma> <semi-major axis>
                                       <comma> <inverse flattening>
                                       <right delimiter>

<prime meridian> ::=                PRIMEM <left delimiter>
                                       <prime meridian name>
                                       <comma> <longitude>
                                       <right delimiter>

<linear unit> ::=                   <unit>

<angular unit> ::=                  <unit>

<unit> ::=                          UNIT <left delimiter>
                                       <unit name>
                                       <comma> <conversion factor>
                                       <right delimiter>

<value> ::=                         <signed numeric literal>

<semi-major axis> ::=               <signed numeric literal>

<longitude> ::=                     <signed numeric literal>

<inverse flattening> ::=            <signed numeric literal>

<conversion factor> ::=             <signed numeric literal>

<unit name> ::=                     <quoted name>

<spheroid name> ::=                 <quoted name>

<projection name> ::=               <quoted name>

<prime meridian name> ::=           <quoted name>

<parameter name> ::=                <quoted name>

<datum name> ::=                    <quoted name>

<csname> ::=                        <quoted name>
```

NOTE: The semi-major axis is measured in meters and shall be> 0.

NOTE Conversion factor specifies number of meters (for a linear unit) or number of radians (for an angular unit) per unit and shall be greater than zero.

A data set's coordinate system is identified by the `PROJCS` keyword if the data are in projected coordinates, by `GEOGCS` if in geographic coordinates, or by `GEOCCS` if in geocentric coordinates.

The `PROJCS` keyword is followed by all of the “pieces” which define the projected coordinate system. The first piece of any object is always the name. Several objects follow the projected coordinate system name: the geographic coordinate system, the map projection, 0 or more parameters, and the linear unit of measure. All projected coordinate systems are based upon a geographic coordinate system, so the pieces specific to a projected coordinate system shall be described first.

EXAMPLE 1 UTM zone 10N on the NAD83 datum is defined as

```
PROJCS["NAD_1983_UTM_Zone_10N",
   <geographic cs>,
   PROJECTION["Transverse_Mercator"],
   PARAMETER["False_Easting",500000.0],
   PARAMETER["False_Northing",0.0],
   PARAMETER["Central_Meridian",-123.0],
   PARAMETER["Scale_Factor",0.9996],
   PARAMETER["Latitude_of_Origin",0.0],
   UNIT["Meter",1.0]]
```

The name and several objects define the geographic coordinate system object in turn: the datum, the ellipsoid, the prime meridian, and the angular unit of measure.

EXAMPLE 2 The geographic coordinate system string for UTM zone 10 on NAD83 is

```
GEOGCS["GCS_North_American_1983",
   DATUM["D_North_American_1983",
   ELLIPSOID["GRS_1980",6378137,298.257222101]],
   PRIMEM["Greenwich",0],
   UNIT["Degree",0.0174532925199433]]
```

EXAMPLE 3 The full string representation of UTM Zone 10N is

```
PROJCS["NAD_1983_UTM_Zone_10N",
   GEOGCS["GCS_North_American_1983",
   DATUM[   "D_North_American_1983",ELLIPSOID["GRS_1980",6378137,298.257222101]],
   PRIMEM["Greenwich",0],UNIT["Degree",0.0174532925199433]],
   PROJECTION["Transverse_Mercator"],PARAMETER["False_Easting",500000.0],
   PARAMETER["False_Northing",0.0],PARAMETER["Central_Meridian",-123.0],
   PARAMETER["Scale_Factor",0.9996],PARAMETER["Latitude_of_Origin",0.0],
   UNIT["Meter",1.0]]
```

Annex A
(informative)

The correspondence of concepts of the common architecture with concepts of the geometry model of ISO 19107

A.1 Introduction

This informative annex identifies similarities and differences between the geometric concepts this Standard, with respect to the geometry model of the ISO 19107. These are referred to throughout this annex as the SFA-CA and the Spatial schema, respectively.

A.2 Geometry model

A.2.1 Geometry model of SFA-CA

Figure 1 shows the geometry model and the contents of SFA-CA. For a full detailed description, the interested reader is referred to 6.1.

A.2.2 Parts of geometry model of Spatial schema

Figure A.1 shows the root class in the geometry part of Spatial schema. Figure A.2 shows more details for the inheritance hierarchy. For a full detailed description, the interested reader is referred to ISO 19107.

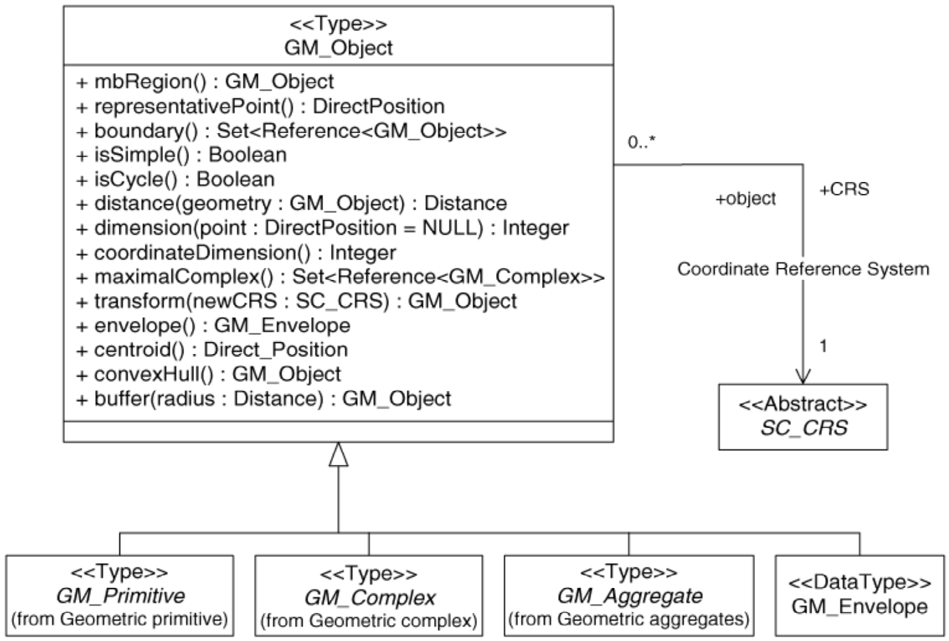


Figure A.1: The root type and subordinates of the Spatial schema

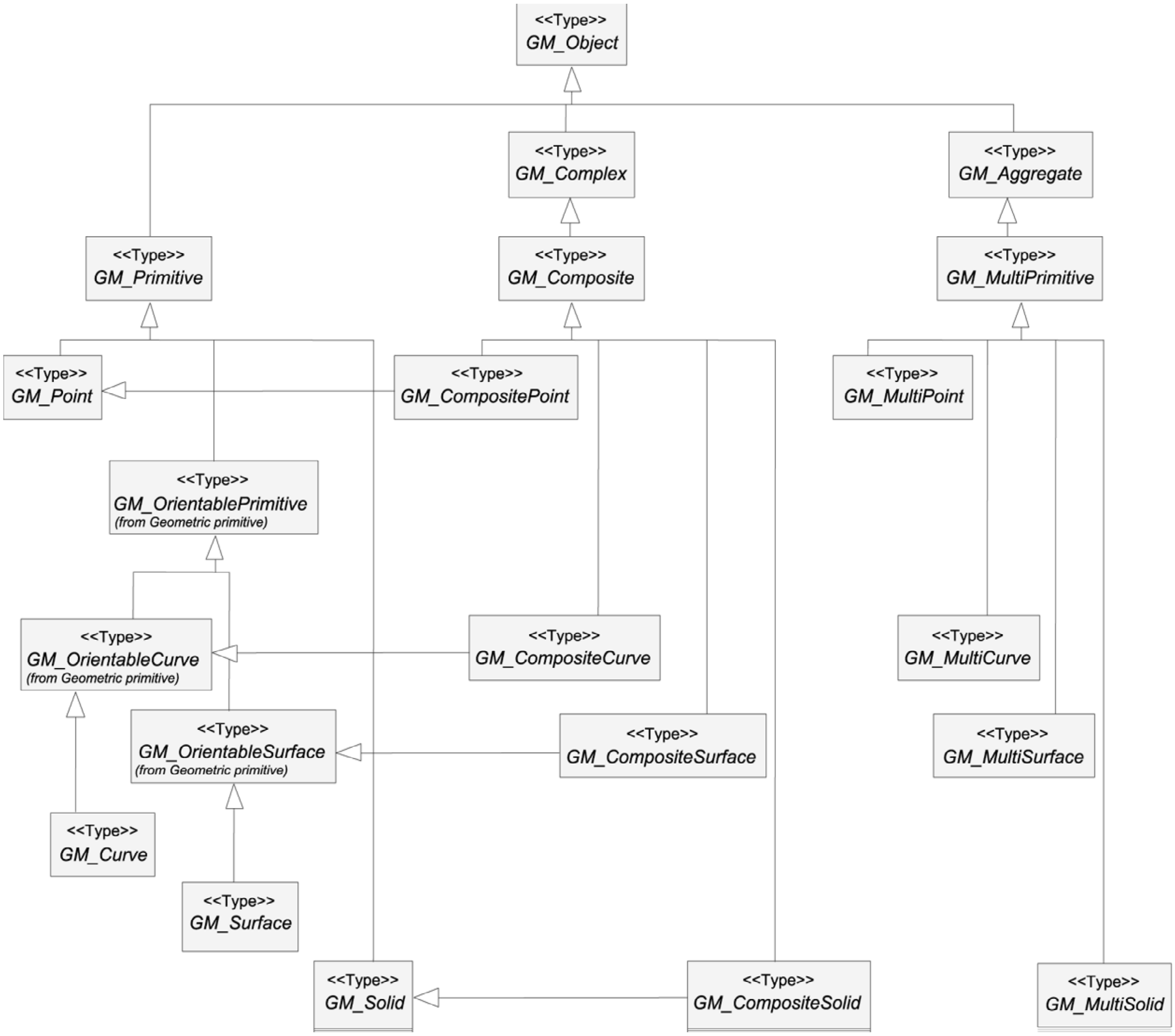


Figure A.2: The GM_Object hierarchy

A.3 Correspondence

A.3.1 Overview

The geometric concepts of the SFA-CA and their respective correspondences to concepts of Spatial schema are described as follows.

— The SFA-CA deals only with at most 2-dimensional geometric objects, whereas the Spatial schema handles up to 3-dimensional geometric objects.

— The Geometry Type of SFA-CA corresponds to the GM_Object of Spatial schema.

— Individual subtypes of the Geometry Type of SFA-CA correspond to one or more subtypes of the geometry model of Spatial schema.

— The GeometryCollection type of SFA-CA corresponds to a more restrictive type of the GM_Aggregate of the Spatial schema.

— The concepts of GM_Complex and GM_Composite of the Spatial schema denote the notions of 'manifolds'. These notions are not provided by the SFA-CA.

— The SFA-CA does not support the notions of topology, which is explicitly modelled by the topology model provided by the Spatial schema.

We are only concerned with the second, third and fourth items of the above list when describing the correspondences. However, there are some main modelling principles which have to be mentioned. That is, the level of abstraction between the SFA-CA and the Spatial schema is a predominant concern throughout this correspondence description, and is summarized mainly by the following facts.

a) SFA-CA is an implementation and platform dependent specification;

b) Spatial schema is an abstract and non-platform dependent specification.

Hence, all practical correspondence, e.g., by implementing interoperability, between systems based on the SFA-CA standard with systems based solely on the Spatial schema specification shall take into account concrete representations and concrete data types of the systems. This is especially important when an SFA-CA database server should support multiple Spatial-schema-based applications.

EXAMPLE 1 The x- and y-coordinates in SFA-CA are explicitly defined as of the type Double. In the Spatial schema, the corresponding coordinates are only given as of the type Number, i.e., an abstract datatype.

EXAMPLE 2 All Boolean operations in SFA-CA return “1” when true, otherwise it is interpreted as false, i.e., in either case an integer return type. A similar operation in the Spatial schema denotes an explicit Boolean value.

Finally, attributes of the Spatial schema are abstracts in the sense that they may be given in terms of access and mutator operators, or as concrete representational attributes, by an implementation. Details on any of these matters are not commented further in this document.

Most of the correspondences in the following are given on a tabular form, i.e., named concepts and signature descriptions of SFA-CA are shown in the first column, and corresponding named concepts and signature description of the Spatial schema are given in the second column. Wherever we need to emphasize the correspondence, we give a comment in the third column. Hence, we emphasize the correspondence from concepts of the SFA-CA to concepts of the Spatial schema, and not the other way around. Thus, SFA-CA needs to be contained by the Spatial schema to be regarded as part of the ISO 19100 series of standards.

A.3.2 Geometry type

A.3.2.1 Overview

In most respects the Geometry type of SFA-CA corresponds to the definition of GM_Object of the Spatial schema. We pinpoint all the definitions of the Geometry type with the corresponding definitions of the GM_Object type. Here we follow the structure of this Standard, and divide the correspondence descriptions into three subclauses, given next.

A.3.2.2 Basic methods on geometry

SFA-CA	Spatial schema	Comment
Geometry .Dimension ( ):Integer	GM_Object::dimension(): Integer	—
Geometry.GeometryType ( ):String	*Not defined*	Defined by an application schema
Geometry.SRID ( ):Integer	GM_Object::CRS : CRS	—
Geometry.Envelope( ):Geometry	GM_Object::envelope(): GM_Envelope GM_Object::mbRegion(): GM_Object	An application has to decide which operator to deploy
Geometry.AsText( ):String	*Not defined*	Defined by an application schema
Geometry.AsBinary( ):Binary	*Not defined*	Defined by an application schema
Geometry.IsEmpty( ):Integer	TransfiniteSet<DirectPosition>.isEmpty	Test for the empty set
Geometry.IsSimple( ):Integer	GM_Object::isSimple(): Boolean	—
Geometry.Boundary( ):Geometry	GM_Object::boundary(): Set<Reference<GM_Object>>	The signature changes in the subtypes of GM_Object.

A.3.2.3 Methods for testing spatial relations between geometric objects

In SFA-CA, the set of Egenhofer and Clementini operators is defined directly on the Geometry type. However, in the spatial schema, the full set of these operators is not defined as explicit behavioral properties of the GM_Object. but as free functions on pairs of geometric or topological objects (ISO 19107, Clause 8). The GM_Object implements set operations relations from the interface template (a parameterized classifier in ISO 19107) TransfiniteSet<DirectPosition>. Spatial operations can be derived from ISO 19107 from the free functions defined in Clause 8: Derived topological relations.

SFA-CA	Spatial schema	Comment
Geometry.Equals(anotherGeometry: Geometry):Integer	GM_Object::equals(pointSet: GM_Object): Boolean	—
Geometry.Intersects(anotherGeometry: Geometry):Integer	GM_Object::intersects(pointSet: GM_Object): Boolean	Intersects is a derived operator.
Geometry.Contains(anotherGeometry: Geometry):Integer	GM_Object::contains(pointSet: GM_Object): Boolean	—

For the other operators of the Geometry type, i.e., Disjoint, Touches, Crosses, Within, Overlaps and Relate, the Spatial schema outlines in ISO 19107:2003 (cf. Clause 8) how to define the corresponding methods in the Spatial schema. Note that this outline refers to all three GM_Object, GM_Primitive, and GM_Composite, as the geometric object types. The GM_Aggregate type will derive such relations from its respective GM_Primitives type, which comprises the element type of an aggregate.

A.3.2.4 Methods that support spatial analysis

SFA-CA	Spatial schema	Comment
Geometry.Distance(anotherGeometry: Geometry):Double	GM_Object::distance(): Distance	—
Geometry.Buffer(distance:Double): Geometry	GM_Object::buffer(radius: Distance): GM_Object	Note the difference in parameters.
Geometry.ConvexHull( ):Geometry	GM_Object::convexHull(): GM_Object	—
Geometry.Intersection(AnotherGeometry:Geometry):Geometry	GM_Object::Intersection(pointSet: GM_Object): GM_Object	In principle, this method is used to define the spatial relations above.
Geometry.Union(anotherGeometry: Geometry):Geometry	GM_Object::union(pointSet: GM_Object): GM_Object	—
Geometry.Difference(anotherGeometry: Geometry):Geometry	GM_Object::difference(pointSet: GM_Object): GM_Object	—
Geometry.SymDifference(AnotherGeometry:Geometry):Geometry	GM_Object::symmetricDifference(pointSet: GM_Object): GM_Object	—

Both the SFA-CA and the Spatial schema sets of set-theoretic (i.e., set-geometric) operations, i.e., the last four rows above, explain the semantics in terms of some implicit point-sets. Theoretically, this is correct, but it is not verified explicitly that these point-set assumptions are valid for the types of geometric values given by these two geometry models.

A.3.3 “Atomic” subtypes of the Geometry type

A.3.3.1 Overview

The structure of the subtype hierarchies of SFA-CA and the Spatial schema above differ in many respects. However, this subclause will outline the possible correspondence between the two hierarchies of “atomic” subtypes. That is, the term 'atomic subtype' refers to a type which is not a collection, composite, complex, or aggregate type. In the following we also include all the operators.

A.3.3.2 Point

SFA-CA	Spatial schema	Comment
Point	GM_Point DirectPosition	Both alternatives are valid. DirectPosition defines the ordinates, i.e., the sequence of numeric coordinates denoting a Point.
Point.X( ):Double	GM_Point::position.ordinate[1] DirectPosition::ordinate[1]	Either of these two, depending on the definition of an application schema
Point.Y( ):Double	GM_Point::position.ordinate[2] DirectPosition::ordinate[2]	See the previous comment.

A.3.3.3 Curve

SFA-CA	Spatial schema	Comment
Curve	GM_Curve GM_GenericCurve GM_CurveSegment GM_LineString GM_LineSegment	The notion of a curve in SFA-SQL may correspond to a number of definitions in Spatial schema.
Curve.Length( ):Double	GM_GenericCurve::length():Length	Operation length is defined with different parameters depending on whether the whole or a part of the curve length is computed.
Curve.StartPoint( ):Point	GM_GenericCurve::startPoint() : DirectPosition	—
Curve.EndPoint( ):Point	GM_GenericCurve:: endPoint() : DirectPosition	—
Curve.IsClosed( ):Integer	GM_Object.isCycle() : Boolean	Given by startPoint() = endPoint(); may be similar as the GM_Object::isSimple:Boolean
Curve.IsRing( ):Integer	GM_Object.isCycle() : Boolean AND GM_Object.isSimple() : Boolean	Given by both closed and simple properties

A.3.3.4 LineString

SFA-CA	Spatial schema	Comment
LineString	GM_LineString	—
LinearString.NumPoints( ):Integer	GM_LineString::controlPoints.count	May be derived
LinearString.PointN(N:Integer):Point	GM_LineString::controlPoints(N)	May be derived

A.3.3.5 LinearRing and LineSegment

These two types are represented as restricted cases of LineString instances in SFA-CA, i.e., both are of type LineString with additional constraints. They are non-instantiable types in the SFA-CA, and correspond to GM_Ring and GM_LineSegment in the Spatial schema, respectively. Note, however, that the SFA-CA implementation standard assumes that a system handles these two types by means of added functionality that is not defined by the SFA-SQL.

A.3.3.6 Surface

The Surface type of the SFA-CA standard is not an instantiable type. The only surface instantiable by SFA-CA is the planer and simple 2D surface given by the Polygon type given in the next subclause.

A.3.3.7 Polygon

SFA_CA	Spatial schema	Comment
Polygon	GM_GenericSurface GM_Surface GM_SurfacePatch GM_Polygon	GM_Polygon and GM_SurfacePatch is not shown in Figure A.3, and the correspondences in this case are more involved, cf. these matters in Reference [1].
Surface.Area( ):Double	GM_GenericSurface::area() : Area	—
Surface.Centroid( ):Point	GM_Object::centroid : DirectPosition	—
Surface.PointOnSurface( ):Point	GM_Object:: representativePoint() : DirectPosition	—
Polygon.ExteriorRing( ): LineString	GM_Polygon::exterior : GM_GenericCurve	The exterior attribute is defined also as zero or more curves in Reference [1].
Polygon.InteriorRingN (N:Integer): LineString	*Not defined*	May be calculated, e.g. from the interior attribute of GM_Polygon
Polygon.NumInteriorRing( ):Integer	*Not defined*	May be calculated, e.g. from the interior attribute of GM_Polygon

A.3.3.8 PolyhedralSurface

A PolyhedralSurface is a contiguous collection of polygons, which share common boundary segments and which as a unit have the topological attributes of a surface. All surface functions are inherited by PolyhedralSurface.

SFA-CA	Spatial schema	Comment
PolyhedralSurface	GM_PolyhedralSurface as a subtype of GM_Surface	—
PolyhedralSurface.NumPatches() : Integer	GM_PolyhedralSurface.patch.count : Integer	Size of the “patch” association role
PolyhedralSurface.PatchN (N: Integer): Polygon	GM_PolyhedralSurface.patch.getAt(N) : GM_Polygon	Retrieve a particular offset in the “patch” association role
PolyhedralSurface.BoundingPolygons (p: Polygon): MultiPolygon		Query against “patch” for polygons that share boundary with “p”
IsClosed (): Integer	GM_Object.isCycle() : Boolean	

A.3.4 Collection subtypes of the Geometry type

A.3.4.1 Overview

This subclause describes the correspondence between the constructs of collections in SFA-CA and aggregates in Spatial schema. The Spatial schema also provides the notions of manifolds, in terms of a structured geometric type as a collection of geometric composites, i.e., each composite comprised by composites on a lower level and dimension. However, these notions are not supported by SFA-CA and have to handled by other means in an SFA-CA based database.

A.3.4.2 GeometryCollection

This is the root type of other more specialized collection types, which are collections of what we above termed atomic geometric types.

SFA-CA	Spatial schema	Comment
GeometryCollection	GM_Aggregate and its subtype GM_MultiPrimitive	—
GeometryCollection :: NumGeometries( ) : Integer	GM_Aggregate.element.count : Integer	May be calculated as the count of the "element" association role of GM_Aggregate
GeometryCollection :: GeometryN(N : Integer) : Geometry	GM_Aggregate.element.getAt(N) : GM_Geometry	May be calculated, e.g. from the "element" association role of GM_Aggregate

The subtypes of GeometryCollection, to be presented next, shall ensure the following constraints, which are not automatically ensured by aggregates of the Spatial schema. These constraints are summarized as follows.

a) For every element in a GeometryCollection, its interior shall be disjoint to the interior of every other, but distinct element of the same GeometryCollection.

b) For every boundary of an element in a GeometryCollection, it may only intersect with a boundary of another, but distinct element at most in a finite number of points.

Moreover, the aggregates of the spatial schema referred to below have not defined any explicit methods. It is assumed that methods applied to aggregates as geometric objects are derived from existing methods defined for the GM_Primitives, which comprises the aggregates.

A.3.4.3 MultiPoint

SFA-CA	Spatial schema	Comment
MultiPoint	GM_MultiPoint	—

MultiPoint in SFA-CA corresponds to GM_MultiPoint in the Spatial schema. No additional methods are defined for MultiPoint.

A.3.4.4 MultiLineString

A MultiLineString is a subtype of the non-instantiable type MultiCurve. Note the use of MultiCurve in the references to the methods of MultiLineString in the table below. That is, the MultiLineString geometric type does not have any methods defined on its own.

SFA-CA	Spatial schema	Comment
MultiLineString	GM_MultiCurve GM_MultiLineString	—
MultiCurve.IsClosed( ):Integer	GM_Object.isCycle() : Boolean	May be derived by testing the start and end points of every GM_Primitive in the aggregate
MultiCurve.Length( ):Double	GM_MultiCurve::length : Length	—

A.3.4.5 MultiPolygon

A MultiPolygon is a subtype of the non-instantiable type MultiSurface. Note the use of MultiSurface in the references to the methods of the MultiPolygon in the table below. That is, the MultiPolygon geometric type does not have any methods defined on its own.

SFA-CA	Spatial schema	Comment
MultiPolygon	GM_MultiSurface	This correspondence is unclear and precaution should be taken, cf. also the correspondence for Polygon above.
MultiSurface.Area () : Double	GM_MultiSurface::area : Area	—
MultiSurface.PointOnSurface( ) : Point	GM_Object:: representativePoint() : DirectPosition	—
MultiSurface.Centroid( ):Point	GM_Object::centroid() : DirectPosition	—

Annex B
(informative)

Supported spatial reference data

B.1 Purpose of this annex

This informative annex provides a non-exhaustive list of Geodetic Codes and Parameters for specifying spatial references. This annex is provided for illustrative purposes when referring to 6.4. This annex may be replaced by a formal catalogue of Geodetic Codes and Parameters as part of ISO 19127 in the future.

B.2 Linear units

Table B - 1 — Linear units

Name	Value
Metre	1,0
International Foot	0,304 8
U.S. Foot	12/39,37
Modified American Foot	12,000 458 4/39,37
Clarke's Foot	12/39,370 432
Indian Foot	12/39,370 141
Link	7,92/39,370 432
Link (Benoit)	7,92/39,370 113
Link (Sears)	7,92/39,370 147
Chain (Benoit)	792/39,370 113
Chain (Sears)	792/39,370 147
Yard (Indian)	36/39,370 141
Yard (Sears)	36/39,370 147
Fathom	1,828 8
Nautical Mile	1 852,0
South African Cape Foot	0,314 855 575 16
South African Geodetic Foot	0,304 797 265 4
German Legal Meter	1,000 013 596 5

B.3 Angular units

Table B - 2 — Angular units

Name	Value
Radian	1,0
Decimal Degree	$\pi/180$
Decimal Minute	$(\pi/180)/60$
Decimal Second	$(\pi/180)/3\,600$

Gon	π/200
Grad	π/200

B.4 Ellipsoids and spheres

Table B - 3 — Ellipsoids and spheres

Name	Semi-major axis	Inverse flattening
Airy	6 377 563,396	299,324 964 6
Modified Airy	6 377 340,189	299,324 964 6
Australian	6 378 160	298,25
Bessel	6 377 397,155	299,152 812 8
Modified Bessel	6 377 492,018	299,152 812 8
Bessel (Namibia)	6 377 483,865	299,152 812 8
Clarke 1866	6 378 206,4	294,978 698 2
Clarke 1866 (Michigan)	6 378 693,704	294,978 684 677
Clarke 1880 (Arc)	6 378 249,145	293,466 307 656
Clarke 1880 (Benoit)	6 378 300,79	293,466 234 571
Clarke 1880 (IGN)	6 378 249,2	293,466 02
Clarke 1880 (Modified)	6 378 249,145	293,466 315 8
Clarke 1880 (RGS)	6 378 249,145	293,465
Clarke 1880 (SGA)	6 378 249,2	293,465 98
Everest 1830	6 377 276,345	300,801 7
Everest 1975	6 377 301,243	300,801 7
Everest (Sarawak and Sabah)	6 377 298,556	300,801 7
Modified Everest 1948	6 377 304,063	300,801 7
GEM10C	6 378 137	298,257 222 101
GRS 1980	6 378 137	298,257 222 101
Helmert 1906	6 378 200	298,3
International 1924	6 378 388	297,0
Krasovsky	6 378 245	298,3
NWL9D	6 378 145	298,25
OSU_86F	6 378 136,2	298,257 22
OSU_91A	6 378 136,3	298,257 22
Plessis 1817	6 376 523	308,64
Sphere (radius = 1.0)	1	0
Sphere (radius = 6 371 000 m)	6 371 000	0

Struve 1860	6 378 297	294,73
War Office	6 378 300,583	296
WGS 1984	6 378 137	298,257 223 563

B.5 Geodetic datums

Table B - 4— Geodetic datums

Name	Name
Adindan	Liberia 1964
Afgooye	Lisbon
Agadez	Loma Quintana
Australian Geodetic Datum 1966	Lome
Australian Geodetic Datum 1984	Luzon 1911
Ain el Abd 1970	Mahe 1971
Amersfoort	Makassar
Aratu	Malongo 1987
Arc 1950	Manoca
Arc 1960	Massawa
Ancienne Triangulation Française	Merchich
Barbados	Militar-Geographische Institute
Batavia	Mhast
Beduaram	Minna
Beijing 1954	Monte Mario
Reseau National Belge 1950	M'poraloko
Reseau National Belge 1972	NAD Michigan
Bermuda 1957	North American Datum 1927
Bern 1898	North American Datum 1983
Bern 1938	Nahrwan 1967
Bogota	Naparima 1972
Bukit Rimpah	Nord de Guerre
Camacupa	NGO 1948
Campo Inchauspe	Nord Sahara 1959
Cape	NSWC 9Z-2
Carthage	Nouvelle Triangulation Française
Chua	New Zealand Geodetic Datum 1949
Conakry 1905	OS (SN) 1980
Corrego Alegre	OSGB 1936
Côte d'Ivoire	OSGB 1970 (SN)

Datum 73	Padang 1884
Deir ez Zor	Palestine 1923
Deutsche Hauptdreiecksnetz	Pointe Noire
Douala	Provisional South American Datum 1956
European Datum 1950	Pulkovo 1942
European Datum 1987	Qatar
Egypt 1907	Qatar 1948
European Reference System 1989	Qornoq
Fahud	RT38
Gandajika 1970	South American Datum 1969
Garoua	Sapper Hill 1943
Geocentric Datum of Australia 1994	Schwarzeck
Guyane Française	Segora
Hartebeeshoek(WGS84) South African	Serindung
Herat North	Stockholm 1938
Hito XVIII 1963	Sudan
Hu Tzu Shan	Tananarive 1925
Hungarian Datum 1972	Timbalai 1948
Indian 1954	TM65
Indian 1975	TM75
Indonesian Datum 1974	Tokyo
Jamaica 1875	Trinidad 1903
Jamaica 1969	Trucial Coast 1948
Japanese Geodetic Datum 2000	Voirol 1875
Kalianpur	Voirol Unifie 1960
Kandawala	WGS 1972
Kertau	WGS 1972 Transit Broadcast Ephemeris
Kuwait Oil Company	WGS 1984
La Canoa	Yacare
Lake	Yoff
Leigon	Zanderij

B.6 Prime meridians

Table B - 5 — Prime meridians

Name	Value
Greenwich	0° 0' 0"
Bern	7° 26' 22,5" E
Bogota	74° 4' 51,3" W
Brussels	4° 22' 4,71" E
Ferro	17° 40' 0" W
Jakarta	106° 48' 27,79" E
Lisbon	9° 7' 54,862" W
Madrid	3° 41' 16,58" W
Paris	2° 20' 14,025" E
Rome	12° 27' 8,4" E
Stockholm	18° 3' 29" E

B.7 Map projections

Table B - 6 — Map projections

Cylindrical projections	Conic projections
Cassini	Albers conic equal-area
Gauss-Kruger	Lambert conformal conic
Mercator	Azimuthal or Planar Projections
Oblique Mercator (Hotine)	Polar Stereographic
Transverse Mercator	Stereographic

B.8 Map projection parameters

Table B - 7 — Map projection parameters

Name	Description
central_meridian	the line of longitude chosen as the origin of x-coordinates
scale_factor	multiplier for reducing a distance obtained from a map to the actual distance on the datum of the map
standard_parallel_1	a line of latitude along which there is no distortion of distance. Also called ‘latitude of true scale’
standard_parallel_2	a line of latitude along which there is no distortion of distance
longitude_of_center	the longitude which defines the center point of the map projection
latitude_of_center	the latitude which defines the center point of the map projection
latitude_of_origin	the latitude chosen as the origin of y-coordinates
false_easting	added to x-coordinates; used to give positive values
false_northing	added to y-coordinates; used to give positive values
azimuth	the angle east of north which defines the center line of an oblique projection
longitude_of_point_1	the longitude of the first point needed for a map projection
latitude_of_point_1	the latitude of the first point needed for a map projection
longitude_of_point_2	the longitude of the second point needed for a map projection
latitude_of_point_2	the latitude of the second point needed for a map projection

Bibliography

[1] *The OpenGIS Abstract Specification: An Object Model for Interoperable Geoprocessing*, Revision 1, OpenGIS Consortium, Inc, OpenGIS Project Document Number 96-015R1, 1996

[2] *OpenGIS Project Document 96-025: Geodetic Reference Systems*, OpenGIS Consortium, Inc., October 14, 1996

[3] Petrotechnical Open Software Consortium (POSC) *Epicentre Model*, available at: <ftp://posc.org/Epicentre/>, July 1995

[4] Clementini, E., Di Felice, P., van Oostrom, P. *A Small Set of Formal Topological Relationships Suitable for End-User Interaction, in D. Abel and B. C. Ooi (Ed.), Advances in Spatial Databases* — Third International Symposium. SSD 1993. LNCS **692**, pp. 277-295. Springer Verlag. Singapore (1993)

[5] Clementini E. and Di Felice P. *A Comparison of Methods for Representing Topological Relationships*, Information Sciences **80** (1994), pp. 1-34

[6] Clementini, E. and Di Felice, P. A *Model for Representing Topological Relationships Between Complex Geometric Features in Spatial Databases*, Information Sciences **90(1-4)** (1996), pp. 121-136

[7] Clementini E., Di Felice P and Califano, G. *Composite Regions in Topological Queries, Information Systems*, **20(6)** (1995), pp. 33-48

[8] Egenhofer, M.J. and Franzosa, R. *Point Set Topological Spatial Relations*, International Journal of Geographical Information Systems, **5(2)** (1991), pp. 161-174

[9] Egenhofer, M.J., Clementini, E. and Di Felice, P. *Topological relations between regions with holes*, International Journal of Geographical Information Systems, **8(2)** (1994), pp. 129-142

[10] Egenhofer, M.J. and Herring, J. *A mathematical framework for the definition of topological relationships*. Proceedings of the Fourth International Symposium on Spatial Data Handling, Columbus, OH, pp. 803-813

[11] Egenhofer, M.J. and Herring, J. *Categorizing binary topological relationships between regions, lines and points in geographic databases*, Tech. Report 91-7, National Center for Geographic Information and Analysis, Santa Barbara, CA (1991)

[12] Egenhofer, M.J. and Sharma, J. *Topological Relations between regions in* $\Re^2$ *and* Z^2, Advances in Spatial Databases — Third International Symposium, SSD 1993, **692**, Lecture Notes in Computer Science, pp. 36-52, Springer Verlag, Singapore (1993)

[13] Worboys, M.F. and Bofakos, P. *A Canonical model for a class of areal spatial objects*, Advances in Spatial Databases — Third International Symposium, SSD 1993, **692**, Lecture Notes in Computer Science, pp. 36-52, Springer Verlag, Singapore (1993).

[14] Worboys, M.F. *A generic model for planar geographical objects*, International Journal of Geographical Information Systems (1992) **6(5)**, pp. 353-372

[15] *CORBA services: Common Object Services Specification*, Ch 8. Externalization Service Specification, OMG. Available at <http://www.omg.org/technology/documents/corba_spec_catalog.htm>

[16] *Distributed Component Object Model* — DCOM 1.0, Microsoft Corporation. Available at <http://www.microsoft.com/com/tech/DCOM.asp>

[17] ISO 19101:2002, *Geographic information — Reference model*

[18] IEEE 754, *IEEE Standard for binary Floating-Point Arithmetic*

