License Agreement

Permission is hereby granted by the Open Geospatial Consortium, ("Licensor"), free of charge and subject to the terms set forth below, to any person obtaining a copy of this Intellectual Property and any associated documentation, to deal in the Intellectual Property without restriction (except as set forth below), including without limitation the rights to implement, use, copy, modify, merge, publish, distribute, and/or sublicense copies of the Intellectual Property, and to permit persons to whom the Intellectual Property is furnished to do so, provided that all copyright notices on the intellectual property are retained intact and that each person to whom the Intellectual Property is furnished agrees to the terms of this Agreement.

If you modify the Intellectual Property, all copies of the modified Intellectual Property must include, in addition to the above copyright notice, a notice that the Intellectual Property includes modifications that have not been approved or adopted by LICENSOR.

THIS LICENSE IS A COPYRIGHT LICENSE ONLY, AND DOES NOT CONVEY ANY RIGHTS UNDER ANY PATENTS THAT MAY BE IN FORCE ANYWHERE IN THE WORLD.

THE INTELLECTUAL PROPERTY IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE, AND NONINFRINGEMENT OF THIRD PARTY RIGHTS. THE COPYRIGHT HOLDER OR HOLDERS INCLUDED IN THIS NOTICE DO NOT WARRANT THAT THE FUNCTIONS CONTAINED IN THE INTELLECTUAL PROPERTY WILL MEET YOUR REQUIREMENTS OR THAT THE OPERATION OF THE INTELLECTUAL PROPERTY WILL BE UNINTERRUPTED OR ERROR FREE. ANY USE OF THE INTELLECTUAL PROPERTY SHALL BE MADE ENTIRELY AT THE USER’S OWN RISK. IN NO EVENT SHALL THE COPYRIGHT HOLDER OR ANY CONTRIBUTOR OF INTELLECTUAL PROPERTY RIGHTS TO THE INTELLECTUAL PROPERTY BE LIABLE FOR ANY CLAIM, OR ANY DIRECT, SPECIAL, INDIRECT OR CONSEQUENTIAL DAMAGES, OR ANY DAMAGES WHATSOEVER RESULTING FROM ANY ALLEGED INFRINGEMENT OR ANY LOSS OF USE, DATA OR PROFITS, WHETHER IN AN ACTION OF CONTRACT, NEGLIGENCE OR UNDER ANY OTHER LEGAL THEORY, ARISING OUT OF OR IN CONNECTION WITH THE IMPLEMENTATION, USE, COMMERCIALIZATION OR PERFORMANCE OF THIS INTELLECTUAL PROPERTY.

This license is effective until terminated. You may terminate it at any time by destroying the Intellectual Property together with all copies in any form. The license will also terminate if you fail to comply with any term or condition of this Agreement. Except as provided in the following sentence, no such termination of this license shall require the termination of any third party end-user sublicense to the Intellectual Property which is in force as of the date of notice of such termination. In addition, should the Intellectual Property, or the operation of the Intellectual Property, infringe, or in LICENSOR’s sole opinion be likely to infringe, any patent, copyright, trademark or other right of a third party, you agree that LICENSOR, in its sole discretion, may terminate this license without any compensation or liability to you, your licensees or any other party. You agree upon termination of any kind to destroy or cause to be destroyed the Intellectual Property together with all copies in any form, whether held by you or by any third party.

Except as contained in this notice, the name of LICENSOR or of any other holder of a copyright in all or part of the Intellectual Property shall not be used in advertising or otherwise to promote the sale, use or other dealings in this Intellectual Property without prior written authorization of LICENSOR or such copyright holder. LICENSOR is and shall at all times be the sole entity that may authorize you or any third party to use certification marks, trademarks or other special designations to indicate compliance with any LICENSOR standards or specifications.

This Agreement is governed by the laws of the Commonwealth of Massachusetts. The application to this Agreement of the United Nations Convention on Contracts for the International Sale of Goods is hereby expressly excluded. In the event any provision of this Agreement shall be deemed unenforceable, void or invalid, such provision shall be modified so as to make it valid and enforceable, and as so modified the entire Agreement shall remain in full force and effect. No decision, action or inaction by LICENSOR shall be construed to be a waiver of any rights or remedies available to it.

None of the Intellectual Property or underlying information or technology may be downloaded or otherwise exported or reexported in violation of U.S. export laws and regulations. In addition, you are responsible for complying with any local laws in your jurisdiction which may impact your right to import, export or use the Intellectual Property, and you represent that you have complied with any regulations or registration procedures required by applicable law to make this license enforceable.
# Contents

<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>Foreword</td>
<td>5</td>
</tr>
<tr>
<td>Introduction</td>
<td>6</td>
</tr>
<tr>
<td>1 Scope</td>
<td>7</td>
</tr>
<tr>
<td>2 Conformance</td>
<td>7</td>
</tr>
<tr>
<td>3 Normative references</td>
<td>7</td>
</tr>
<tr>
<td>4 Terms and definitions</td>
<td>8</td>
</tr>
<tr>
<td>5 Symbols and Abbreviations</td>
<td>12</td>
</tr>
<tr>
<td>5.1 Abbreviations</td>
<td>12</td>
</tr>
<tr>
<td>5.2 Symbols</td>
<td>12</td>
</tr>
<tr>
<td>6 Architecture</td>
<td>13</td>
</tr>
<tr>
<td>6.1 Geometry object model</td>
<td>13</td>
</tr>
<tr>
<td>6.2 Annotation Text</td>
<td>40</td>
</tr>
<tr>
<td>7 Well-known Text Representation for Geometry</td>
<td>51</td>
</tr>
<tr>
<td>7.1 Component overview</td>
<td>51</td>
</tr>
<tr>
<td>7.2 Component description</td>
<td>52</td>
</tr>
<tr>
<td>8 Well-known Binary Representation for Geometry</td>
<td>62</td>
</tr>
<tr>
<td>8.1 Component overview</td>
<td>62</td>
</tr>
<tr>
<td>8.2 Component description</td>
<td>62</td>
</tr>
<tr>
<td>9 Well-known Text Representation of Spatial Reference Systems</td>
<td>73</td>
</tr>
<tr>
<td>9.1 Component overview</td>
<td>73</td>
</tr>
<tr>
<td>9.2 Component description</td>
<td>73</td>
</tr>
<tr>
<td>Annex A (informative) The correspondence of concepts of the common architecture with concepts of the geometry model of ISO 19107</td>
<td>76</td>
</tr>
<tr>
<td>Annex B (informative) Supported spatial reference data</td>
<td>86</td>
</tr>
<tr>
<td>Bibliography</td>
<td>92</td>
</tr>
</tbody>
</table>
Foreword

This standard consists of the following parts, under the general title Geographic information — Simple feature access:

— Part 1: Common architecture
— Part 2: SQL option


Version 1.1 of this standard is a profile of this version in the sense that it is a proper subset of the technology included here, except for some technical corrections and clarification.

This version is a corrigendum, correcting editorial and minor technical issues found version 1.2.0. The relationship to Version 1.1 is unchanged.

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

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

This part of OpenGIS® Simple Features Access (SFA), also called ISO 19125, describes the common architecture for simple feature geometry. The simple feature geometry object model is Distributed Computing Platform neutral and uses UML notation. The base Geometry class has subclasses for Point, Curve, Surface and GeometryCollection. Each geometric object is associated with a Spatial Reference System, which describes the coordinate space in which the geometric object is defined.

The extended Geometry model has specialized 0, 1 and 2-dimensional collection classes named MultiPoint, MultiLineString and MultiPolygon for modeling geometries corresponding to collections of Points, LineStrings and Polygons, respectively. MultiCurve and MultiSurface are introduced as abstract superclasses that generalize the collection interfaces to handle Curves and Surfaces.

The attributes, methods and assertions for each Geometry class are described in Figure 1 in 6.1.1. In describing methods, this is used to refer to the receiver of the method (the object being messaged).

The SFA COM function “signatures” may use a different notation from SFA SQL. COM notation is more familiar for COM programmers. However, UML notation is used throughout this part. There may also be methods used in this Standard that differ from one part to another. Where this is the case, the differences are shown within the part.

This part of OGC Simple Feature Access implements a profile of the spatial schema described in ISO 19107:2003, Geographic information — Spatial schema. Annex A provides a detailed mapping of the schema in this part of SFA with the schema described in ISO 19107:2003.
Geographic information — Simple feature access — Part 1: Common architecture

1 Scope

This standard establishes a common architecture and defines terms to use within the architecture.

This standard does not attempt to standardize and does not depend upon any part of the mechanism by which Types are added and maintained, including the following:

a) syntax and functionality provided for defining types;
b) syntax and functionality provided for defining functions;
c) physical storage of type instances in the database;
d) specific terminology used to refer to User Defined Types, for example UDT.

This standard does standardize names and geometric definitions for Types for Geometry.

This standard does not place any requirements on how to define the Geometry Types in the internal schema nor does it place any requirements on when or how or who defines the Geometry Types.

2 Conformance

In order to conform to this standard, an implementation shall satisfy the requirements of one or more test suites specified in the other parts of ISO 19125.

3 Normative references

The following referenced documents are indispensable for the application of this document. For dated references, only the edition cited applies. For undated references, the latest edition of the referenced document (including any amendments) applies.


[2] ISO 19107, Geographic information — Spatial schema

[3] ISO 19111, Geographic information — Spatial referencing by coordinates

[4] ISO 19133, Geographic information — Location based services — Tracking and navigation

Copyright © 2010 Open Geospatial Consortium, Inc.
4 Terms and definitions

For the purposes of this document, all definitions from Part 1 of this standard and the following terms and definitions apply.

4.1 boundary
set that represents the limit of an entity

NOTE Boundary is most commonly used in the context of geometry, where the set is a collection of points or a collection of objects that represent those points. In other arenas, the term is used metaphorically to describe the transition between an entity and the rest of its domain of discourse.

[ISO 19107]

4.2 buffer
geometric object (4.14) that contains all direct positions (4.7) whose distance from a specified geometric object is less than or equal to a given distance

[ISO 19107]

4.3 coordinate
one of a sequence of \(n\)-numbers designating the position of a point (4.17) in \(n\)-dimensional space

NOTE In a coordinate reference system, the numbers shall be qualified by units.

[adapted from ISO 19111]

4.4 coordinate dimension
number of measurements or axes needed to describe a position in a coordinate system (4.6)

[ISO 19107]

4.5 coordinate reference system
coordinate system (4.6) that is related to the real world by a datum

[adapted from ISO 19111]

4.6 coordinate system
set of mathematical rules for specifying how coordinates (4.3) are to be assigned to each point (4.17)

[ISO 19111]

4.7 curve
topological 1-dimensional geometric primitive (4.15), representing the continuous image of a line

NOTE The boundary of a curve is the set of points at either end of the curve. If the curve is a cycle, the two ends are identical, and the curve (if topologically closed) is considered to not have a boundary. The first point is called the start point, and the last is the end point. Connectivity of the curve is guaranteed by the "continuous image of a line" clause. A topological theorem states that a continuous image of a connected set is connected.
The term “1-dimentional” refers to the topological dimension of the primitive. In this case, it means that each point not on the boundary is an element in a topological open set within the curve which is isomorphic to an open interval \((0, 1)\). For this standard, the coordinate dimension can be 2 (for \(x\) and \(y\)), 3 (with \(z\) or \(m\) added), or 4 (with both \(z\) and \(m\) added). The ordinates \(x\), \(y\) and \(z\) are spatial, and the ordinate \(m\) is a measure.

[ISO 19107]

4.8 direct position
position described by a single set of coordinates (4.3) within a coordinate reference system (4.5)

[ISO 19107]

4.9 end point
last point (4.17) of a curve (4.7)

Note: End and start point are related to the orientation of the curve. Any curve representation can be “flipped” reversing the end and start, without changing the image of the curve as a set of points (direct positions).

[ISO 19107]

4.10 exterior
difference between the universe and the closure

NOTE The concept of exterior is applicable to both topological and geometric complexes.

[ISO 19107]

4.11 feature
abstraction of real world phenomena

NOTE A feature may occur as a type or an instance. Feature type or feature instance is used when only one is meant.

[adapted from ISO 19101]

4.12 feature attribute
characteristic of a feature (4.11)

NOTE A feature attribute has a name, a data type, and a value domain associated to it. A feature attribute for a feature instance also has an attribute value taken from the value domain. No restrictions are implied here as to the type of attributes a feature may have. The “geometries” associated to features are just one type of feature attribute.

[adapted from ISO 19101]
4.13
**geometric complex**
set of disjoint **geometric primitives** (4.15) where the **boundary** (4.1) of each geometric primitive can be represented as the union of other geometric primitives of smaller dimension within the same set

**NOTE** The geometric primitives in the set are disjoint in the sense that no direct position is interior to more than one geometric primitive. The set is closed under boundary operations, meaning that for each element in the geometric complex, there is a collection (also a geometric complex) of geometric primitives that represents the boundary of that element. Recall that the boundary of a point (the only 0D primitive object type in geometry) is empty. Thus, if the largest dimension geometric primitive is a solid (3D), the composition of the boundary operator in this definition terminates after at most 3 steps. It is also the case that the boundary of any object is a cycle.

Geometric complexes are often referred to as clean geometry or implicit topology, meaning that the various topological inconsistencies have usually been removed to obtain the “completeness” of the boundary representation.

[ISO 19107]

4.14
**geometric object**
spatial object representing a geometric set

**NOTE** A geometric object consists of a geometric primitive, a collection of geometric primitives, or a geometric complex treated as a single entity. A geometric object may be the spatial representation of an object such as a feature or a significant part of a feature.

Regardless of the representation, the feature is usually assumed to be topologically closed, in that points on the boundary of the feature are assumed to belong to the feature, even though those points may not explicitly be represented in the geometric object. When representing a topological entity, geometric objects are assumed to not contain their boundaries.

[ISO 19107]

4.15
**geometric primitive**
**geometric object** (4.14) representing a single, connected, homogeneous element of space

**NOTE** Geometric primitives are non-decomposed objects that represent information about geometric configuration. They include points, curves, surfaces, and solids. Contrary to common usage, a geometric primitive is open, decomposable (can be broken into smaller objects) because of the inherent continuity of space. Primitive are those things that have not been chosen for such decomposition.

[ISO 19107]

4.16
**interior**
set of all **direct positions** (4.7) that are on a **geometric object** (4.14) but which are not on its **boundary** (4.1)

**NOTE** The interior of a topological object is the continuous image of the interior of any of its geometric realizations. This is not included as a definition because it follows from a theorem of topology. Another way of saying this is that any point on a geometric object is in its interior if it can be placed inside a homeomorphic image of an open set in the Euclidean space of the object’s topological dimension.

[ISO 19107]

4.17
**linear referencing system**
**linear positioning system**
positioning system that measures distance from a reference point along a route (feature)
NOTE The system includes the complete set of procedures for determining and retaining a record of specific points along a linear feature such as the location reference method(s) together with the procedures for storing, maintaining, and retrieving location information about points and segments on the highways. [NCHRP Synthesis 21, 1974]

[ISO 19133]

4.18 point
topological 0-dimensional geometric primitive (4.15), representing a position

NOTE The boundary of a point is the empty set.

[ISO 19107]

4.19 simple feature
feature with all geometric attributes described piecewise by straight line or planar interpolation between sets of points

Note Interpolation is used on curves and surfaces, which by their nature are an infinite set of points and thus not suitable to finite exhaustive representations. Each such geometric entity is decomposed into parts which can be expressed locally as parametric, linear combinations of “control points.” This is described at length in ISO 19107.

For curves, each part (called a “segment” in ISO 19107) has two control points $P_0$ (the “start point”) and $P_1$ (the “end point”). Any other $P$ on the segment can be described using a real number parameter $t$ between 0.0 and 1.0 in the “vector” equation: $P = tP_0 + (1 - t)P_1$.

For surfaces, each part (called a “patch” in ISO 19107) can be viewed as a polygon which can be broken into triangles each with three control points $P_0$, $P_1$, and $P_2$. Any other $P$ in the triangle can be described using 3 non-negative real numbers whose sum is 1.0 (called “barycentric coordinates”) $a, b, c \in \mathbb{R}; a, b, c > 0; a + b + c = 1.0$ in the vector equation: $P = aP_0 + bP_1 + cP_2$.

4.20 start point
first point (4.17) of a curve (4.7)

[ISO 19107]

4.21 surface
topological 2-dimensional geometric primitive (4.15), locally representing a continuous image of a region of a plane

NOTE The boundary of a surface is the set of oriented, closed curves that delineate the limits of the surface.

[adapted from ISO 19107]
5 Symbols and Abbreviations

5.1 Abbreviations

- **API**: Application Program Interface
- **COM**: Component Object Model
- **CORBA**: Common Object Request Broker Architecture
- **DCE**: Distributed Computing Environment
- **DCOM**: Distributed Component Objected Model
- **DE-9IM**: Dimensionally Extended Nine-Intersection Model
- **FID**: Feature ID column in the implementation of feature tables based on predefined data types
- **GID**: Geometry ID column in the implementation of feature tables based on predefined data types
- **IEEE**: Institute of Electrical and Electronics Engineers, Inc.
- **MM**: Multimedia
- **NDR**: Little Endian byte order encoding
- **OLE**: Object Linking and Embedding
- **RPC**: Remote Procedure Call
- **SQL**: Structured query language, not an acronym, pronounced as "sequel"
- **SQL/MM**: SQL Multimedia and Application Packages
- **SRID**: Spatial Reference System Identifier
- **SRTEXT**: Spatial Reference System Well Known Text
- **UDT**: User Defined Type
- **UML**: Unified Modeling Language
- **WKB**: Well-Known Binary (representation for example, geometry)
- **WKTR**: Well-Known Text Representation
- **XDR**: Big Endian byte order encoding

5.2 Symbols

- **nD**: n-Dimensional, where n may be any integer
- **ℜ^n**: n-Dimensional coordinate space, where n may be any integer
- **∅**: empty set, the set having no members
- **∩**: intersection, operation on two or more sets
- **∪**: union, operation on two or more sets
- **−**: difference, operation on two sets
- **∈**: is a member of, relation between an element and a set
- **∉**: is not a member of
- **⊂**: is a proper subset of, i.e. a smaller set not containing all of the larger
- **⊆**: is a subset of
6 Architecture

6.1 Geometry object model

6.1.1 Overview

This subclause describes the object model for simple feature geometry. The simple feature geometry object model is Distributed Computing Platform neutral and uses UML notation. The object model for geometry is shown in Figure 1. The base Geometry class has subclasses for Point, Curve, Surface and GeometryCollection. Each geometric object is associated with a Spatial Reference System, which describes the coordinate space in which the geometric object is defined.
Figure 1: Geometry class hierarchy

Figure 1 is based on an extended Geometry model with specialized 0-, 1- and 2-dimensional collection classes named MultiPoint, MultiLineString and MultiPolygon for modeling geometries corresponding to collections of Points, LineStrings and Polygons, respectively. MultiCurve and MultiSurface are introduced as superclasses that generalize the collection interfaces to handle Curves and Surfaces. Figure 1 shows aggregation lines between the leaf-collection classes and their element classes; the aggregation lines for non-leaf-collection classes are described in the text. Non-homogeneous collections are instances of GeometryCollection.

The attributes, methods and assertions for each Geometry class are described below. In describing methods, this is used to refer to the receiver of the method (the object being messaged).

6.1.2 Geometry

6.1.2.1 Description

Geometry is the root class of the hierarchy. Geometry is an abstract (non-instantiable) class.

The instantiable subclasses of Geometry defined in this Standard are restricted to 0, 1 and 2-dimensional geometric objects that exist in 2, 3 or 4-dimensional coordinate space ($\mathbb{R}^2$, $\mathbb{R}^3$ or $\mathbb{R}^4$). Geometry values in $\mathbb{R}^2$ have points with coordinate values for x and y. Geometry values in $\mathbb{R}^3$ have points with coordinate values for x, y and z or for x, y and m. Geometry values in $\mathbb{R}^4$ have points with coordinate values for x, y, z and m. The interpretation of the coordinates is subject to the coordinate reference systems associated to the point. All coordinates within a geometry object should be in the same coordinate reference systems. Each coordinate shall be unambiguously associated to a coordinate reference system either directly or through its containing geometry.

The z coordinate of a point is typically, but not necessarily, represents altitude or elevation. The m coordinate represents a measurement.

All Geometry classes described in this standard are defined so that instances of Geometry are topologically closed, i.e. all represented geometries include their boundary as point sets. This does not affect their
representation, and open version of the same classes may be used in other circumstances, such as topological representations.

Figure 2: Geometry class operations

### 6.1.2.2 Basic methods on geometric objects

— **Dimension**: Integer — The inherent dimension of this geometric object, which must be less than or equal to the coordinate dimension. In non-homogeneous collections, this will return the largest topological dimension of the contained objects.

— **GeometryType**: String — Returns the name of the instantiable subtype of Geometry of which this geometric object is an instantiable member. The name of the subtype of Geometry is returned as a string.

— **SRID**: Integer — Returns the Spatial Reference System ID for this geometric object. This will normally be a foreign key to an index of reference systems stored in either the same or some other datastore.
— **Envelope** ( ): Geometry — The minimum bounding box for this Geometry, returned as a Geometry. The polygon is defined by the corner points of the bounding box \([\text{MINX}, \text{MINY}], (\text{MAXX}, \text{MINY}), (\text{MAXX}, \text{MAXY}), (\text{MINX}, \text{MAXY}), (\text{MINX}, \text{MINY})]\). Minimums for Z and M may be added. The simplest representation of an Envelope is as two direct positions, one containing all the minimums, and another all the maximums. In some cases, this coordinate will be outside the range of validity for the Spatial Reference System.

— **AsText** ( ): String — Exports this geometric object to a specific Well-known Text Representation of Geometry.

— **AsBinary** ( ): Binary — Exports this geometric object to a specific Well-known Binary Representation of Geometry.

— **IsEmpty** ( ): Integer — Returns 1 (TRUE) if this geometric object is the empty Geometry. If true, then this geometric object represents the empty point set \(\emptyset\) for the coordinate space. The return type is integer, but is interpreted as Boolean, TRUE=1, FALSE=0.

— **IsSimple** ( ): Integer — Returns 1 (TRUE) if this geometric object has no anomalous geometric points, such as self intersection or self tangency. The description of each instantiable geometric class will include the specific conditions that cause an instance of that class to be classified as not simple. The return type is integer, but is interpreted as Boolean, TRUE=1, FALSE=0.

— **Is3D** ( ): Integer — Returns 1 (TRUE) if this geometric object has z coordinate values.

— **IsMeasured** ( ): Integer — Returns 1 (TRUE) if this geometric object has m coordinate values.

— **Boundary** ( ): Geometry — Returns the closure of the combinatorial boundary of this geometric object (Reference [1], section 3.12.2). Because the result of this function is a closure, and hence topologically closed, the resulting boundary can be represented using representational Geometry primitives (Reference [1], section 3.12.2). The return type is integer, but is interpreted as Boolean, TRUE=1, FALSE=0.

### 6.1.2.3 Methods for testing spatial relations between geometric objects

The methods in this subclause are defined and described in more detail following the description of the sub-types of Geometry. For each of the following, the return type is integer, but is interpreted as Boolean, TRUE=1, FALSE=0.

— **Equals** (anotherGeometry: Geometry): Integer — Returns 1 (TRUE) if this geometric object is “spatially equal” to anotherGeometry.

— **Disjoint** (anotherGeometry: Geometry): Integer — Returns 1 (TRUE) if this geometric object is “spatially disjoint” from anotherGeometry.

— **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 as specified by the values in the intersectionPatternMatrix. This returns FALSE if all the tested intersections are empty except exterior (this) intersect exterior (another).

— **LocateAlong** (mValue: Double): Geometry — Returns a derived geometry collection value that matches the specified m coordinate value. See Subclause 6.1.2.6 “Measures on Geometry” for more details.

— **LocateBetween** (mStart: Double, mEnd: Double): Geometry — Returns a derived geometry collection value that matches the specified range of m coordinate values inclusively. See Subclause 6.1.2.6 “Measures on Geometry” for more details.

### 6.1.2.4 Methods that support spatial analysis

All of the following are geometric analysis and depend on the accuracy of the coordinate representations and the limitations of linear interpolation in this standard. The accuracy of the result at a fine level will be limited by these and related issues.

— **Distance** (anotherGeometry: Geometry): Double — Returns the shortest distance between any two Points in the two geometric objects as calculated in the spatial reference system of this geometric object. Because the geometries are closed, it is possible to find a point on each geometric object involved, such that the distance between these 2 points is the returned distance between their geometric objects.

— **Buffer** (distance: Double): Geometry — Returns a geometric object that represents all Points whose distance from this geometric object is less than or equal to distance. Calculations are in the spatial reference system of this geometric object. Because of the limitations of linear interpolation, there will often be some relatively small error in this distance, but it should be near the resolution of the coordinates used.

— **ConvexHull** (): Geometry — Returns a geometric object that represents the convex hull of this geometric object. Convex hulls, being dependent on straight lines, can be accurately represented in linear interpolations for any geometry restricted to linear interpolations.

— **Intersection** (anotherGeometry: Geometry): Geometry — Returns a geometric object that represents the Point set intersection of this geometric object with anotherGeometry.

— **Union** (anotherGeometry: Geometry): Geometry — Returns a geometric object that represents the Point set union of this geometric object with anotherGeometry.

— **Difference** (anotherGeometry: Geometry): Geometry — Returns a geometric object that represents the Point set difference of this geometric object with anotherGeometry.

— **SymDifference** (anotherGeometry: Geometry): Geometry — Returns a geometric object that represents the Point set symmetric difference of this geometric object with anotherGeometry.
6.1.2.5 Use of Z and M coordinate values

A Point value may include a z coordinate value. The z coordinate value traditionally represents the third dimension (i.e. 3D). In a Geographic Information System (GIS) this may be height above or below sea level. For example: A map might have point identifying the position of a mountain peak by its location on the earth, with the x and y coordinate values, and the height of the mountain, with the z coordinate value.

A Point value may include an m coordinate value. The m coordinate value allows the application environment to associate some measure with the point values. For example: A stream network may be modeled as multilinestring value with the m coordinate values measuring the distance from the mouth of stream. The method LocateBetween may be used to find all the parts of the stream that are between, for example, 10 and 12 kilometers from the mouth. There are no constraints on the m coordinate values in a Geometry (e.g., the m coordinate values do not have to be continually increasing along a LineString value).

Observer methods returning Point values include z and m coordinate values when they are present.

Spatial operations work in the "map geometry" of the data and will therefore not reflect z or m values in calculations (e.g., Equals, Length) or in generation of new geometry values (e.g., Buffer, ConvexHull, Intersection). This is done by projecting the geometric objects onto the horizontal plane to obtain a "footprint" or "shadow" of the objects for the purposed of map calculations. In other words, it is possible to store and obtain z (and m) coordinate values but they are ignored in all other operations which are based on map geometries. Implementations are free to include true 3D geometric operations, but should be consistent with ISO 19107.

6.1.2.6 Measures on Geometry

The LocateAlong and LocateBetween methods derive MultiPoint or MultiCurve values from the given geometry that match a measure or a specific range of measures from the start measure to the end measure. The LocateAlong method is a variation of the LocateBetween method where the start measure and end measure are equal. (See SQL/MM [1])

6.1.2.6.1 Empty Sets

A null value is returned for empty sets.

6.1.2.6.2 Geometry values without m coordinate values.

An empty set of type Point is returned for geometry values without m coordinate values.

6.1.2.6.3 Zero-dimensional geometry values

Only points in the 0-dimensional geometry values with m coordinate values between $SM$ and $EM$ inclusively are returned as multipoint value. If no matching m coordinate values are found, then an empty set of type Point is returned.

For example:

a) If LocateAlong is invoked with an $M$ value of 4 on a MultiPoint value with well-known text representation:

```
multipoint m(1 0 4, 1 1 1, 1 2 2, 3 1 4, 5 3 4)
```

then the result is the following MultiPoint value with well-known text representation:

```
multipoint m(1 0 4, 3 1 4, 5 3 4)
```

b) If LocateBetween is invoked with an $SM$ value of 2 and an $EM$ value of 4 on a MultiPoint value with well-known text representation:

```
multipoint m(1 0 4, 1 1 1, 1 2 2, 3 1 4, 5 3 5, 9 5 3, 7 6 7)
```

then the result is the following MultiPoint value with well-known text representation:

```
multipoint m(1 0 4, 1 2 2, 3 1 4, 9 5 3)
```

c) If LocateBetween is invoked with an $SM$ value of 1 and an $EM$ value of 4 on a Point value with well-known text representation:

```
point m(7 6 7)
```
then the result is the following MultiPoint value with well-known text representation:
point m empty
d) If LocateBetween is invoked with an SM value of 7 and an EM value of 7 on a Point value with well-known text representation:
point m(7 6 7)
then the result is the following MultiPoint value with well-known text representation:
multipoint m(7 6 7)

6.1.2.6.4 One-dimensional geometry value

Interpolation is used to determine any points on the 1-dimensional geometry with an m coordinate value between mStart and mEnd inclusively. The implementation-defined interpolation algorithm is used to estimate values between measured values, usually using a mathematical function. The interpolation is within a Curve element and not across Curve elements in a MultiCurve. For example, given a measure of 6 and a 2-point LineString where the m coordinate value of start point is 4 and the m coordinate value of the end point is 8, since 6 is halfway between 4 and 8, the interpolation algorithm would be a point on the LineString halfway between the start and end points.

The results are produced in a geometry collection. If there are consecutive points in the 1-dimensional geometry with an m coordinate value between mStart and mEnd inclusively, then a curve value element is added to the geometry collection to represent the curve elements between these consecutive points. Any disconnected points in the 1-dimensional geometry values with m coordinate values between mStart and mEnd inclusively are also added to the geometry collection. If no matching m coordinate values are found, then an empty set of type ST_Point is returned.

For example:
   a) If LocateAlong is invoked with an M value of 4 on a LineString value with well-known text representation:
      LineStringM(1 0 0, 3 1 4, 5 3 4, 5 5 1, 5 6 4, 7 8 4, 9 9 0)
      then the result is the following MultiLineString value with well-known text representation:
      MultiLineStringM((3 1 4, 5 3 4), (5 6 4, 7 8 4))
   b) If LocateBetween is invoked with an mStart value of 2 and an mEnd value of 4 on a LineString value with well-known text representation:
      LineStringM(1 0 0, 1 1 1, 1 2 2, 3 1 3, 5 3 4, 9 5 5, 7 6 6)
      then the result is the following MultiLineString value with well-known text representation:
      MultiLineStringM((1 2 2, 3 1 3, 5 3 4))
   c) If LocateBetween is invoked with an SM value of 6 and an EM value of 9 on a LineString value with well-known text representation:
      LineStringM(1 0 0, 1 1 1, 1 2 2, 3 1 3, 5 3 4, 9 5 5, 7 6 6)
      then the result is the following MultiPoint value with well-known text representation:
      MultiPointM(7 6 6)
   d) If LocateBetween is invoked with an SM value of 2 and an EM value of 4 on a MultiLineString value with well-known text representation:
      MultiLineStringM((1 0 0, 1 1 1, 1 2 2, 3 1 3), (4 5 3, 5 3 4, 9 5 5, 7 6 6))
      then the result is the following MultiLineString value with well-known text representation:
      MultiLineStringM((1 2 2, 3 1 3), (4 5 3, 5 3 4))
   e) If LocateBetween is invoked with an SM value of 1 and an EM value of 3 on a LineString value with well-known text representation:
      LineStringM(0 0 0, 2 2 2, 4 4 4)
      then the result may be the following MultiLineString value with well-known text representation:
      MultiLineStringM((1 1 1, 2 2 2, 3 3 3))
   f) If LocateBetween is invoked with an SM value of 7 and an EM value of 9 on a MultiLineString value with well-known text representation:
      MultiLineStringM((1 0 0, 1 1 1, 1 2 2, 3 1 3), (4 5 3, 5 3 4, 9 5 5, 7 6 6))

Copyright © 2010 Open Geospatial Consortium, Inc.
then the result is the following MultiLineString value with well-known text representation:
PointM empty

6.1.2.6.5 Two-dimensional geometry value

The computation for 2-dimensional geometries is implementation-defined.

6.1.3 GeometryCollection

6.1.3.1 Description

A GeometryCollection is a geometric object that is a collection of some number of geometric objects.

All the elements in a GeometryCollection shall be in the same Spatial Reference System. This is also the Spatial Reference System for the GeometryCollection.

GeometryCollection places no other constraints on its elements. Subclasses of GeometryCollection may restrict membership based on dimension and may also place other constraints on the degree of spatial overlap between elements.

```
Geometry

GeometryCollection

+ numGeometries() : Integer
+ geometryN(n :Integer) : Geometry
```

Figure 3: Geometry collection operations

6.1.3.2 Methods

By the nature of digital representations, collections are inherently ordered by the underlying storage mechanism. Two collections whose difference is only this order are spatially equal and will return equivalent results in any geometric-defined operations.

— NumGeometries ( ): Integer — Returns the number of geometries in this GeometryCollection.


6.1.4 Point

6.1.4.1 Description

A Point is a 0-dimensional geometric object and represents a single location in coordinate space. A Point has an \( x \)-coordinate value, a \( y \)-coordinate value. If called for by the associated Spatial Reference System, it may also have coordinate values for \( z \) and \( m \).

The boundary of a Point is the empty set.
6.1.4.2 Methods

— \( X() : \text{Double} \) — The \( x \)-coordinate value for this Point.

— \( Y() : \text{Double} \) — The \( y \)-coordinate value for this Point.

— \( Z() : \text{Double} \) — The \( z \)-coordinate value for this Point, if it has one. Returns NIL otherwise.

— \( M() : \text{Double} \) — The \( m \)-coordinate value for this Point, if it has one. Returns NIL otherwise.

6.1.5 MultiPoint

A MultiPoint is a 0-dimensional GeometryCollection. The elements of a MultiPoint are restricted to Points. The Points are not connected or ordered in any semantically important way (see the discussion at GeometryCollection).

A MultiPoint is simple if no two Points in the MultiPoint are equal (have identical coordinate values in \( X \) and \( Y \)). Every MultiPoint is spatially equal under the definition in Clause 6.1.15.3 to a simple Multipoint.

The boundary of a MultiPoint is the empty set.

6.1.6 Curve

6.1.6.1 Description

A Curve is a 1-dimensional geometric object usually stored as a sequence of Points, with the subtype of Curve specifying the form of the interpolation between Points. This standard defines only one subclass of Curve, LineString, which uses linear interpolation between Points.

A Curve is a 1-dimensional geometric object that is the homeomorphic image of a real, closed, interval:

\[ D = [a, b] = \{ t \in \mathbb{R} | a \leq t \leq b \} \]

under a mapping
f : [a, b] → ℜ^n

where n is the coordinate dimension of the underlying Spatial Reference System.

A Curve is simple if it does not pass through the same Point twice with the possible exception of the two end points (Reference [1], section 3.12.7.3):

∀ c ∈ Curve, [a, b] = c.Domain, c =: f : [a, b] → ℜ^n

c.IsSimple ⇔ ∀ x₁, x₂ ∈ [a, b]: [ f(x₁)=f(x₂) ∧ x₁<x₂] ⇒ [x₁=a ∧ x₂=b]

A Curve is closed if its start Point is equal to its end Point (Reference [1], section 3.12.7.3).

c.IsClosed ⇔ [f(a) = f(b)]

The boundary of a closed Curve is empty.

The boundary of a non-closed Curve consists of its two end Points (Reference [1], section 3.12.3.2).

A Curve that is simple and closed is a Ring.

A Curve is defined as topologically closed, that is, it contains its endpoints f(a) and f(b).

<table>
<thead>
<tr>
<th>Geometry</th>
<th>LineString</th>
</tr>
</thead>
<tbody>
<tr>
<td>Curve</td>
<td></td>
</tr>
<tr>
<td>+ length() : Double</td>
<td>+ numPoints() : Integer</td>
</tr>
<tr>
<td>+ startPoint() : Point</td>
<td>+ pointN() : Point</td>
</tr>
<tr>
<td>+ endPoint() : Point</td>
<td>+ isClosed() : Boolean</td>
</tr>
<tr>
<td>+ isRing() : Boolean</td>
<td></td>
</tr>
</tbody>
</table>

Figure 5: Curve

6.1.6.2 Methods

— **Length ( )**: Double — The length of this Curve in its associated spatial reference.

— **StartPoint ( )**: Point — The start Point of this Curve.

— **EndPoint ( )**: Point — The end Point of this Curve.

— **IsClosed ( )**: Integer — Returns 1 (TRUE) if this Curve is closed [StartPoint ( ) = EndPoint ( )].

— **IsRing ( )**: Integer — Returns 1 (TRUE) if this Curve is closed [StartPoint ( ) = EndPoint ( )] and this Curve is simple (does not pass through the same Point more than once).

6.1.7 LineString, Line, LinearRing

6.1.7.1 Description

A LineString is a Curve with linear interpolation between Points. Each consecutive pair of Points defines a Line segment.

A Line is a LineString with exactly 2 Points.
A LinearRing is a LineString that is both closed and simple. The Curve in Figure 2, item (c), is a closed LineString that is a LinearRing. The Curve in Figure 2, item (d) is a closed LineString that is not a LinearRing.

![Figure 6: Examples of LineStrings](image)

Key
s start
e end

Figure 6: Examples of LineStrings
Simple LineString (a),
Non-simple LineString (b),
Simple, closed LineString (a LinearRing) (c),
Non-simple closed LineString (d)

<table>
<thead>
<tr>
<th>Curve</th>
</tr>
</thead>
<tbody>
<tr>
<td>LineString</td>
</tr>
<tr>
<td>+ numPoints() : Integer</td>
</tr>
<tr>
<td>+ pointN(Integer) : Point</td>
</tr>
</tbody>
</table>

Figure 7: LineString

6.1.7.2 Methods

— NumPoints ( ) : Integer — The number of Points in this LineString.

— PointN (N: Integer): Point — Returns the specified Point N in this LineString.

6.1.8 MultiCurve

6.1.8.1 Description

A MultiCurve is a 1-dimensional GeometryCollection whose elements are Curves as in Figure 3.
MultiCurve is a non-instantiable class in this standard; it defines a set of methods for its subclasses and is included for reasons of extensibility.

A MultiCurve is simple if and only if all of its elements are simple and the only intersections between any two elements occur at Points that are on the boundaries of both elements.

The boundary of a MultiCurve is obtained by applying the “mod 2” union rule: A Point is in the boundary of a MultiCurve if it is in the boundaries of an odd number of elements of the MultiCurve (Reference [1], section 3.12.3.2).

A MultiCurve is closed if all of its elements are closed. The boundary of a closed MultiCurve is always empty.

A MultiCurve is defined as topologically closed.

![Figure 8: MultiCurve](image)

### 6.1.8.2 Methods

- **IsClosed** (): Integer — Returns 1 (TRUE) if this MultiCurve is closed [StartPoint () = EndPoint () for each Curve in this MultiCurve].

- **Length** (): Double — The Length of this MultiCurve which is equal to the sum of the lengths of the element Curves.

### 6.1.9 MultiLineStyle

A MultiLineStyle is a MultiCurve whose elements are LineStrings.

The boundaries for the MultiLineStyles in Figure 9 are (a)—{s1, e2}, (b)—{s1, e1}, (c)—∅.
6.1.10 Surface

6.1.10.1 Description

A Surface is a 2-dimensional geometric object.

A simple Surface may consists of a single “patch” that is associated with one “exterior boundary” and 0 or more “interior” boundaries. A single such Surface patch in 3-dimensional space is isometric to planar Surfaces, by a simple affine rotation matrix that rotates the patch onto the plane z = 0. If the patch is not vertical, the projection onto the same plane is an isomorphism, and can be represented as a linear transformation, i.e. an affine.

Polyhedral Surfaces are formed by “stitching” together such simple Surfaces patches along their common boundaries. Such polyhedral Surfaces in a 3-dimensional space may not be planar as a whole, depending on the orientation of their planar normals (Reference [1], sections 3.12.9.1, and 3.12.9.3). If all the patches are in alignment (their normals are parallel), then the whole stitched polyhedral surface is co-planar and can be represented as a single patch if it is connected.

The boundary of a simple Surface is the set of closed Curves corresponding to its “exterior” and “interior” boundaries (Reference [1], section 3.12.9.4).

The only instantiable subclasses of Surface defined in this standard are Polygon and PolyhedralSurface. A Polygon is a simple Surface that is planar. A PolyhedralSurface is a simple surface, consisting of some number of Polygon patches or facets. If a PolyhedralSurface is closed, then it bounds a solid. A MultiSurface containing a set of closed PolyhedralSurfaces can be used to represent a Solid object with holes.
6.1.10.2 Methods

— **Area ()**: Double — The area of this Surface, as measured in the spatial reference system of this Surface.

— **Centroid ()**: Point — The mathematical centroid for this Surface as a Point. The result is not guaranteed to be on this Surface.

— **PointOnSurface ()**: Point — A Point guaranteed to be on this Surface.

6.1.11 Polygon, Triangle

6.1.11.1 Description

A Polygon is a planar Surface defined by 1 exterior boundary and 0 or more interior boundaries. Each interior boundary defines a hole in the Polygon. A Triangle is a polygon with 3 distinct, non-collinear vertices and no interior boundary.

The exterior boundary LinearRing defines the “top” of the surface which is the side of the surface from which the exterior boundary appears to traverse the boundary in a counter clockwise direction. The interior LinearRings will have the opposite orientation, and appear as clockwise when viewed from the “top”.

The assertions for Polygons (the rules that define valid Polygons) are as follows:

a) Polygons are topologically closed;

b) The boundary of a Polygon consists of a set of LinearRings that make up its exterior and interior boundaries;

c) No two Rings in the boundary cross and the Rings in the boundary of a Polygon may intersect at a Point but only as a tangent, e.g.
∀ P ∈ Polygon, ∀ c1, c2 ∈ P.Boundary(), c1 ≠ c2,
∀ p, q ∈ Point, p, q ∈ c1, p ≠ q,
[p ∈ c2] ⇒ (∃ δ > 0 ∃ [|p-q|<δ] ⇒ [q ∉ c2]);

Note: This last condition says that at a point common to the two curves, nearby points cannot be common. This forces each common point to be a point of tangency.

d) A Polygon may not have cut lines, spikes or punctures e.g.:

∀ P ∈ Polygon, P = P.Interior.Closure;

e) The interior of every Polygon is a connected point set;

f) The exterior of a Polygon with 1 or more holes is not connected. Each hole defines a connected component of the exterior.

In the above assertions, interior, closure and exterior have the standard topological definitions. The combination of (a) and (c) makes a Polygon a regular closed Point set. Polygons are simple geometric objects. Figure 11 shows some examples of Polygons.

Figure 11: Examples of Polygons with 1 (a), 2 (b) and 3 (c) Rings, respectively

Figure 12 shows some examples of geometric objects that violate the above assertions and are not representable as single instances of Polygon.
6.1.11.2 Methods

— **ExteriorRing ( )**: LineString — Returns the exterior ring of this Polygon.

— **NumInteriorRing ( )**: Integer — Returns the number of interior rings in this Polygon.

— **InteriorRingN (N: Integer)**: LineString — Returns the Nth interior ring for this Polygon as a LineString.

6.1.12 PolyhedralSurface

6.1.12.1 Description

A PolyhedralSurface is a contiguous collection of polygons, which share common boundary segments. For each pair of polygons that "touch", the common boundary shall be expressible as a finite collection of LineStrings. Each...
such LineString shall be part of the boundary of at most 2 Polygon patches. A TIN (triangulated irregular network) is a PolyhedralSurface consisting only of Triangle patches.

For any two polygons that share a common boundary, the “top” of the polygon shall be consistent. This means that when two LinearRings from these two Polygons traverse the common boundary segment, they do so in opposite directions. Since the Polyhedral surface is contiguous, all polygons will be thus consistently oriented. This means that a non-oriented surface (such as Möbius band) shall not have single surface representations. They may be represented by a MultiSurface. Figure 14 shows an example of such a consistently oriented surface (from the top). The arrows indicate the ordering of the linear rings that from the boundary of the polygon in which they are located.

![Figure 14: Polyhedral Surface with consistent orientation](image)

If each such LineString is the boundary of exactly 2 Polygon patches, then the PolyhedralSurface is a simple, closed polyhedron and is topologically isomorphic to the surface of a sphere. By the Jordan Surface Theorem (Jordan’s Theorem for 2-spheres), such polyhedrons enclose a solid topologically isomorphic to the interior of a sphere; the ball. In this case, the “top” of the surface will either point inward or outward of the enclosed finite solid. If outward, the surface is the exterior boundary of the enclosed surface. If inward, the surface is the interior of the infinite complement of the enclosed solid. A Ball with some number of voids (holes) inside can thus be presented as one exterior boundary shell, and some number in interior boundary shells.

### 6.1.12.2 Methods

- **NumPatches () : Integer** — Returns the number of including polygons

- **PatchN (N: Integer): Polygon** — Returns a polygon in this surface, the order is arbitrary.

- **BoundingPolygons (p: Polygon): MultiPolygon** — Returns the collection of polygons in this surface that bounds the given polygon “p” for any polygon “p” in the surface.

- **IsClosed () : Integer** — Returns 1 (True) if the polygon closes on itself, and thus has no boundary and encloses a solid

Copyright © 2010 Open Geospatial Consortium, Inc.
6.1.13 MultiSurface

6.1.13.1 Description

A MultiSurface is a 2-dimensional GeometryCollection whose elements are Surfaces, all using coordinates from the same coordinate reference system. The geometric interiors of any two Surfaces in a MultiSurface may not intersect in the full coordinate system. The boundaries of any two coplanar elements in a MultiSurface may intersect, at most, at a finite number of Points. If they were to meet along a curve, they could be merged into a single surface.

MultiSurface is an instantiable class in this Standard, and may be used to represent heterogeneous surfaces collections of polygons and polyhedral surfaces. It defines a set of methods for its subclasses. The subclass of MultiSurface is MultiPolygon corresponding to a collection of Polygons only. Other collections shall use MultiSurface.

NOTE: The geometric relationships and sets are the common geometric ones in the full coordinate systems. The use of the 2D map operations defined Clause 6.1.15 may classify the elements of a valid 3D MultiSurface as having overlapping interiors in their 2D projections.
6.1.13.2 Methods

MultiSurface inherits operations NumGeometries and GeometryN from GeometryCollection to access its individual component surfaces.

— Area ( ): Double — The area of this MultiSurface, as measured in the spatial reference system of this MultiSurface.

— Centroid ( ): Point — The mathematical centroid for this MultiSurface. The result is not guaranteed to be on this MultiSurface.

— PointOnSurface ( ): Point — A Point guaranteed to be on this MultiSurface.

6.1.14 MultiPolygon

A MultiPolygon is a MultiSurface whose elements are Polygons.

The assertions for MultiPolygons are as follows.

a) The interiors of 2 Polygons that are elements of a MultiPolygon may not intersect.

\[ \forall M \in \text{MultiPolygon}, \forall P_i, P_j \in M.\text{Geometries()}, i \neq j, \]
\[ \text{Interior}(P_i) \cap \text{Interior}(P_j) = \emptyset; \]

b) The boundaries of any 2 Polygons that are elements of a MultiPolygon may not “cross” and may touch at only a finite number of Points.

\[ \forall M \in \text{MultiPolygon}, \forall P_i, P_j \in M.\text{Geometries}(), \]
\[ \forall c_i, c_j \in \text{Curve} \ c_i \in P_i.\text{Boundaries}(), c_j \in P_j.\text{Boundaries}() \]
\[ \exists k \in \text{Integer} \ \exists \ p_1, \ldots, p_k \ | \ p_m \in \text{Point}, 0 < m < k; \]

NOTE Crossing is prevented by assertion (a) above.
c) A MultiPolygon is defined as topologically closed.

d) A MultiPolygon may not have cut lines, spikes or punctures, a MultiPolygon is a regular closed Point set:

\[ \forall M \in \text{MultiPolygon}, M = \text{Closure} (\text{Interior} (M)) \]

e) The interior of a MultiPolygon with more than 1 Polygon is not connected; the number of connected components of the interior of a MultiPolygon is equal to the number of Polygons in the MultiPolygon.

The boundary of a MultiPolygon is a set of closed Curves (LineStrings) corresponding to the boundaries of its element Polygons. Each Curve in the boundary of the MultiPolygon is in the boundary of exactly 1 element Polygon, and every Curve in the boundary of an element Polygon is in the boundary of the MultiPolygon.

The reader is referred to works by Worboys et al. ([13], [14]) and Clementini et al. ([5], [6]) for the definition and specification of MultiPolygons.

Figure 17 shows four examples of valid MultiPolygons with 1, 3, 2 and 2 Polygon elements, respectively.

Figure 18 shows examples of geometric objects not representable as single instances of MultiPolygons.

NOTE The subclass of Surface named Polyhedral Surface as described in Reference [1], is a faceted Surface whose facets are Polygons. A Polyhedral Surface is not a MultiPolygon because it violates the rule for MultiPolygons that the boundaries of the element Polygons intersect only at a finite number of Points.
6.1.15 Relational operators

6.1.15.1 Background

The relational operators are Boolean methods that are used to test for the existence of a specified topological spatial relationship between two geometric objects as they would be represented on a map. Topological spatial relationships between two geometric objects have been a topic of extensive study; see References in the Bibliography numbered [4], [5], [6], [7], [8], [9], and [10]. The basic approach to comparing two geometric objects is to project the objects onto the 2D horizontal coordinate reference system representing the Earth's surface, and then to make pair-wise tests of the intersections between the interiors, boundaries and exteriors of the two projections and to classify the map relationship between the two geometric objects based on the entries in the resulting 3 by 3 'intersection' matrix. The concepts of interior, boundary and exterior are well defined as sets of point geometry, and abstracted in general topology; see Reference [4].

It is important to note that the calculation of the following operations will give equivalent results whether the calculations are done using classical geometric representations or these same calculations are done with algebraic techniques in a well-structured and properly defined equivalent topological structure.

These concepts are applied in this standard for defining spatial relationships between 2-dimensional objects in 2-dimensional space ($\mathbb{R}^2$) by the projection of the objects onto the horizontal surface usually represented in a map. This will give a different result than would be obtained if the full 3D geometry (or its corresponding 3D topology) because of the changes induced in the projection of the objects onto the horizontal map projection. It would be possible to define a full 3D set of operations, but the increase in computational complexity can be prohibitive to most implementations, and is generally not supported in many geographic information systems or other applications dealing with significant volumes of "mapping data." Specification of full 3D operators following this same pattern for higher dimensions is reserved for a future version of this standard.

NOTE It is important to remember that when reading this standard, that when spoken of in the abstract the relationship underlying these operations will refer to the full relationship in the coordinate reference system of the objects being spoken of, unless the operations defined in these clauses is specifically referenced.

In order to apply the concepts of interior, boundary and exterior to 1- and 0-dimensional objects in $\mathbb{R}^2$, a combinatorial topology approach shall be applied (Reference [1], section 3.12.3.2). This approach is based on the accepted definitions of the boundaries, interiors and exteriors for simplicial complexes (see Reference [12]) and yields the following results.

The boundary of a geometric object is a set of geometric objects of the next lower dimension. The boundary of a Point or a MultiPoint is the empty set. The boundary of a non-closed Curve consists of its two end Points; the boundary of a closed Curve is empty. The boundary of a MultiCurve consists of those Points that are in the boundaries of an odd number of its element Curves. The boundary of a Polygon consists of its set of Rings. The boundary of a MultiPolygon consists of the set of Rings of its Polygons. The boundary of an arbitrary collection of geometric objects whose interiors are disjoint consists of geometric objects drawn from the boundaries of the element geometric objects by application of the "mod 2" union rule (Bibliographic Reference [1], section 3.12.3.2).

The domain of geometric objects considered is those that are topologically closed. The interior of a geometric object consists of those Points that are left when the boundary Points are removed. The exterior of a geometric object consists of Points not in the interior or boundary.

Studies on the relationships between two geometric objects both of maximal dimension in $\mathbb{R}^1$ and $\mathbb{R}^2$ considered pair-wise intersections between the interior and boundary sets and led to the definition of a four-intersection
model; see Reference [8]. The model was extended to consider the exterior of the input geometric objects, resulting in a nine-intersection model (see Reference [11]) and further extended to include information on the dimension of the results of the pair-wise intersections resulting in a dimensionally extended nine-intersection model; see Reference [5]. These extensions allow the model to express spatial relationships between points, lines and areas, including areas with holes and multi-component lines and areas; see Reference [6].

6.1.15.2 The Dimensionally Extended Nine-Intersection Model (DE-9IM)

Given a geometric object \( a \), let \( I(a) \), \( B(a) \) and \( E(a) \) represent the interior, boundary and exterior of \( a \), respectively.

Let \( \dim(x) \) return the maximum dimension (-1, 0, 1, or 2) of the geometric objects in \( x \), with a numeric value of -1 corresponding to \( \dim(\emptyset) \).

The intersection of any two of \( I(a) \), \( B(a) \) and \( E(a) \) can result in a set of geometric objects, \( x \), of mixed dimension. For example, the intersection of the boundaries of two Polygons may consist of a point and a line.

Table 1 shows the general form of the dimensionally extended nine-intersection matrix (DE-9IM).

<table>
<thead>
<tr>
<th></th>
<th>Interior</th>
<th>Boundary</th>
<th>Exterior</th>
</tr>
</thead>
<tbody>
<tr>
<td>Interior</td>
<td>( \dim(I(a) \cap I(b)) )</td>
<td>( \dim(I(a) \cap B(b)) )</td>
<td>( \dim(I(a) \cap E(b)) )</td>
</tr>
<tr>
<td>Boundary</td>
<td>( \dim(B(a) \cap I(b)) )</td>
<td>( \dim(B(a) \cap B(b)) )</td>
<td>( \dim(B(a) \cap E(b)) )</td>
</tr>
<tr>
<td>Exterior</td>
<td>( \dim(E(a) \cap I(b)) )</td>
<td>( \dim(E(a) \cap B(b)) )</td>
<td>( \dim(E(a) \cap E(b)) )</td>
</tr>
</tbody>
</table>

For regular, topologically closed input geometric objects, computing the dimension of the intersection of the interior, boundary and exterior sets does not have, as a prerequisite, the explicit computation and representation of these sets. To compute if the interiors of two regular closed Polygons intersect, and to ascertain the dimension of this intersection, it is not necessary to explicitly represent the interior of the two Polygons, which are topologically open sets, as separate geometric objects. In most cases, the dimension of the intersection value at a cell is highly constrained, given the type of the two geometric objects. In the Line-Area case, the only possible values for the interior-interior cell are drawn from \{-1, 1\} and in the Area-Area case, the only possible values for the interior-interior cell are drawn from \{-1, 2\}. In such cases, no work beyond detecting the intersection is required.

Figure 8 shows an example DE-9IM for the case where \( a \) and \( b \) are two Polygons that overlap.
Figure 19: An example instance and its DE-9IM

On two geometric objects, a spatial relationship predicate can be expressed as a formula that takes as input a pattern matrix representing the set of acceptable values for the DE-9IM for the two geometric objects. If the spatial relationship between the two geometric objects corresponds to one of the acceptable values as represented by the pattern matrix, then the predicate returns TRUE.

The pattern matrix consists of a set of nine pattern-values, one for each cell in the matrix. The possible pattern-values of $p$ are \{T, F, *, 0, 1, 2\} and their meanings for any cell where $x$ is the intersection set for the cell are as follows:

\[
\begin{align*}
p = T &\Rightarrow \dim(x) \in \{0, 1, 2\}, \text{ i.e. } x \neq \emptyset \\
p = F &\Rightarrow \dim(x) = -1, \text{ i.e. } x = \emptyset \\
p = * &\Rightarrow \dim(x) \in \{-1, 0, 1, 2\}, \text{ i.e. Don’t Care} \\
p = 0 &\Rightarrow \dim(x) = 0 \\
p = 1 &\Rightarrow \dim(x) = 1 \\
p = 2 &\Rightarrow \dim(x) = 2
\end{align*}
\]

The pattern matrix can be represented as an array or list of nine characters in row major order. As an example, the following code fragment could be used to test for "Overlap" between two areas:

```c
char * overlapMatrix = "T*T***T**";
Geometry* a, b;
Boolean b = a->Relate(b, overlapMatrix);
```

6.1.15.3 Named spatial relationship predicates based on the DE-9IM

The Relate predicate based on the pattern matrix has the advantage that clients can test for a large number of spatial relationships and fine tune the particular relationship being tested. It has the disadvantage that it is a lower-level building block and does not have a corresponding natural language equivalent. Users of the proposed system include IT developers using the COM API from a language such as Visual Basic, and interactive SQL users who may wish, for example, to select all features ‘spatially within’ a query Polygon, in addition to more spatially "sophisticated" GIS developers.

To address the needs of such users, a set of named spatial relationship predicates has been defined for the DE-9IM; see References [5, 6]. The five predicates are named Disjoint, Touches, Crosses, Within and Overlaps. The definition of these predicates (see References [5, 6]) is given below. In these definitions, the term P is used to refer to 0-dimensional geometries (Points and MultiPoints), L is used to refer to 1-dimensional geometries (LineStrings and MultiLineStrings) and A is used to refer to 2-dimensional geometries (Polygons and MultiPolygons).

**Equals**

Given two (topologically closed) geometric objects “a” and “b”:

\[
a.\text{Equals}(b) \iff a \subseteq b \land b \subseteq a
\]

Expressed in terms of the DE-9IM:

\[
a.\text{Equals}(b) \iff \left( I(a) \cap I(b) \neq \emptyset \right) \land
\]

Copyright © 2010 Open Geospatial Consortium, Inc.
Disjoint

Given two (topologically closed) geometric objects "a" and "b":

\[ a \text{ Disjoint (b)} \iff a \cap b = \emptyset \]

Expressed in terms of the DE-9IM:

\[ a \text{ Disjoint (b)} \iff [ (I(a) \cap I(b) = \emptyset) \land \\
(I(a) \cap B(b) = \emptyset) \land \\
(B(a) \cap I(b) = \emptyset) \land \\
(B(a) \cap B(b) = \emptyset) ] \\
\iff a \text{ Relate(b, "TFFTTFFFT")} \]

Touches

The Touches relationship between two geometric objects “a” and “b” applies to the A/A, L/L, L/A, P/A and P/L groups of relationships but not to the P/P group. It is defined as

\[ a \text{ Touch (b)} \iff (I(a) \cap I(b) = \emptyset) \land \\
(a \cap b) \neq \emptyset \]

Expressed in terms of the DE-9IM:

\[ a \text{ Touch (b)} \iff [ (I(a) \cap I(b) = \emptyset) \land \\
[ (B(a) \cap I(b) \neq \emptyset) \lor \\
(I(a) \cap B(b) \neq \emptyset) \lor \\
(B(a) \cap B(b) \neq \emptyset) ] ] \\
\iff [ a \text{ Relate(b, "FF*FF****")} \lor \\
a \text{ Relate(b, "F**T*****")} \lor \\
a \text{ Relate(b, "F***T****")} ]

Figure 9 shows some examples of the Touches relationship.
Crosses

The Crosses relationship applies to P/L, P/A, L/L and L/A situations. It is defined as

\[
a.\text{Cross}(b) \iff [I(a) \cap I(b) \neq \emptyset \land (a \cap b \neq a) \land (a \cap b \neq b)]
\]

Note: Previous definition had an unnecessary statement on dimension which was always true.

Expressed in terms of the DE-9IM:

Case \(a \in P, b \in L\) or \(a \in P, b \in A\) or \(a \in L, b \in A\):

\[
a.\text{Cross}(b) \iff [I(a) \cap I(b) \neq \emptyset \land I(a) \cap E(b) \neq \emptyset ]
\]

\[
\iff a.\text{Relate}(b, "T*T*****")
\]

Case \(a \in L, b \in L\):

\[
a.\text{Cross}(b) \iff \dim(I(a) \cap I(b)) = 0
\]

\[
\iff a.\text{Relate}(b, "0******")
\]

Figure 10 shows some examples of the Crosses relationship.
Within

The Within relationship is defined as

\[ a.\text{Within}(b) \iff (a \cap b = a) \land (I(a) \cap E(b) = \emptyset) \]

Expressed in terms of the DE-9IM:

\[ a.\text{Within}(b) \iff [ I(a) \cap I(b) \neq \emptyset \land I(a) \cap E(b) = \emptyset \land B(a) \cap E(b) = \emptyset ] \]
\[ \iff a.\text{Relate}(b, \ "T**F***") \]

Figure 11 shows some examples of the “Within” relationship.

Overlaps

The Overlaps relationship is defined for A/A, L/L and P/P situations.
It is defined as

\[ a.\text{Overlaps}(b) \iff (\dim(I(a)) = \dim(I(b)) = \dim(I(a) \cap I(b))) \]
\[ \land (a \cap b \neq a) \land (a \cap b \neq b) \]

Expressed in terms of the DE-9IM:

Case \( a \in P, b \in P \) or Case \( a \in A, b \in A \):
\[ a.\text{Overlaps}(b) \iff (I(a) \cap I(b) \neq \emptyset) \land \\
(I(a) \cap E(b) \neq \emptyset) \land \\
(E(a) \cap I(b) \neq \emptyset) \iff a.\text{Relate}(b, \text{"T*T***T**"}) \]

Case \( a \in L, b \in L \):
\[ a.\text{Overlaps}(b) \iff (\dim(I(a) \cap I(b) = 1) \land (I(a) \cap E(b) \neq \emptyset) \land (E(a) \cap I(b) \neq \emptyset) \iff a.\text{Relate}(b, \text{"1*T***T**"}) \]

Figure 12 shows some examples of the Overlaps relationship.

![Figure 23: Examples of the Overlaps relationship
Polygon/LineString (a)
and LineString/LineString (b)](image)

The following additional named predicates are also defined for user convenience:

**Contains**

\[ a.\text{Contains}(b) \iff b.\text{Within}(a) \]

**Intersects**

\[ a.\text{Intersects}(b) \iff \neg a.\text{Disjoint}(b) \]

Based on the above operators the following methods are defined on Geometry:

---

**Equals** (anotherGeometry: Geometry): Integer — Returns 1 (TRUE) if this geometric object is spatially equal to anotherGeometry.

**Disjoint** (anotherGeometry: Geometry): Integer — Returns 1 (TRUE) if this geometric object is spatially disjoint from anotherGeometry.
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

<table>
<thead>
<tr>
<th>Field Name</th>
<th>Type</th>
<th>Requirements and Defaults</th>
</tr>
</thead>
<tbody>
<tr>
<td>text array</td>
<td>Array of ANNOTATION_TEXT_ELEMENT objects (ANNOTATION_TEXT_ELEMENT object type described in Table 2)</td>
<td>ARRAY must be of least length of 1 or no text is displayed</td>
</tr>
<tr>
<td>envelope</td>
<td>GEOMETRY</td>
<td>Required: A geometry envelope used for spatial indexing.</td>
</tr>
</tbody>
</table>
Table 3: Fields of the Annotation Text Element object type

<table>
<thead>
<tr>
<th>Field Name</th>
<th>Type</th>
<th>Requirements and Defaults</th>
</tr>
</thead>
<tbody>
<tr>
<td>value</td>
<td>VARCHAR2(2000)</td>
<td>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.</td>
</tr>
<tr>
<td>location</td>
<td>GEOMETRY</td>
<td>Optional – no text will be displayed if LOCATION is NULL. Locating geometries can be a point or curve type.</td>
</tr>
<tr>
<td>text attributes</td>
<td>XML_TYPE (a character string in XML)</td>
<td>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.</td>
</tr>
<tr>
<td>leader line</td>
<td>GEOMETRY (a curve type)</td>
<td>Optional – if null, there is no leader line.</td>
</tr>
</tbody>
</table>

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

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>font-family</td>
<td>String</td>
<td>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</td>
</tr>
<tr>
<td>font-size</td>
<td>Float</td>
<td>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.</td>
</tr>
<tr>
<td>font-weight</td>
<td>enumeration</td>
<td>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.</td>
</tr>
</tbody>
</table>

Copyright © 2010 Open Geospatial Consortium, Inc.
<table>
<thead>
<tr>
<th>Attribute</th>
<th>Type</th>
<th>Description</th>
<th>Requirements and Defaults</th>
</tr>
</thead>
<tbody>
<tr>
<td>font-style</td>
<td>enumeration</td>
<td>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.</td>
<td>Defaults to normal.</td>
</tr>
<tr>
<td>text-decoration</td>
<td>enumeration</td>
<td>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).</td>
<td>Defaults to none.</td>
</tr>
<tr>
<td>letter-spacing</td>
<td>Float and &quot;normal&quot;</td>
<td>SVG allows numbers or &quot;normal&quot;</td>
<td>Defaults to normal.</td>
</tr>
<tr>
<td>word-spacing</td>
<td>Float and &quot;normal&quot;</td>
<td>Same as letter spacing but used between words.</td>
<td>Defaults to normal.</td>
</tr>
<tr>
<td>fill</td>
<td>String (Fill Type)</td>
<td>This specifies the color of the interior of the glyphs. Colors can be specified in the following manner:</td>
<td>Defaults to black.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>1. Well known SVG font names such as black, blue, red. See <a href="http://www.w3.org/TR/SVG/types.html#ColorKeywords">http://www.w3.org/TR/SVG/types.html#ColorKeywords</a>.</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>2. RGB values specified using function syntax such as rgb(255, 0, 255) is a magenta</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>3. A literal hex value such as #FF00FF which would be the same as the previous RGB example.</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>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.</td>
<td></td>
</tr>
<tr>
<td>fill-opacity</td>
<td>Float(0-1)</td>
<td>A percentage that specifies the opacity or translucency of the fill. A 0 is fully transparent and 1 is fully opaque.</td>
<td>Defaults to 1</td>
</tr>
<tr>
<td>stroke</td>
<td>String (Stroke Type)</td>
<td>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.</td>
<td>Defaults to none.</td>
</tr>
<tr>
<td>stroke-width</td>
<td>Float</td>
<td>A width value specifying the stroke width in points.</td>
<td>Defaults to 0. Zero or the lack of this attribute indicates no stroke.</td>
</tr>
<tr>
<td>stroke-opacity</td>
<td>Float(0-1)</td>
<td>A percentage that specifies the opacity or translucency of the stroke. A 0 is fully transparent and 1 is fully opaque.</td>
<td>Defaults to 1</td>
</tr>
</tbody>
</table>
Table 5: Attributes for Text Layout

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Type</th>
<th>Description</th>
<th>Requirements and Defaults</th>
</tr>
</thead>
<tbody>
<tr>
<td>horizontal alignment</td>
<td>Enumeration</td>
<td>3 allowable values which are: &quot;start&quot;, &quot;center&quot;, &quot;end&quot;. 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.</td>
<td>Optional defaults to &quot;start&quot;</td>
</tr>
<tr>
<td>vertical alignment</td>
<td>Enumeration</td>
<td>4 allowable values which are: &quot;top&quot;, &quot;center&quot;, &quot;baseline&quot; and &quot;bottom&quot;. 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.</td>
<td>Optional defaults to &quot;top&quot;</td>
</tr>
<tr>
<td>multiline justification</td>
<td>Enumeration</td>
<td>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.</td>
<td>Optional as it is not needed in single line text. Defaults to &quot;left&quot;</td>
</tr>
<tr>
<td>multiline spacing</td>
<td>Float</td>
<td>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.</td>
<td>Optional as it is not needed in single line text. Defaults to 0 which puts each line immediately below the previous one</td>
</tr>
</tbody>
</table>

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
<?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>
  </xs:complexType>
</xs:schema>
```

Copyright © 2010 Open Geospatial Consortium, Inc.
<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">
            <xs:enumeration value="left"/>
            <xs:enumeration value="center"/>
            <xs:enumeration value="right"/>
            <xs:enumeration value="justify"/>
          </xs:restriction>
        </xs:simpleType>
      </xs:attribute>
    </xs:complexType>
  </xs:element>
</xs:complexType>
<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:element>
<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:element>
<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:element>
<xs:simpleType name="spacing">
  <xs:union>
    <xs:simpleType>
      <xs:restriction base="xs:string">
        <xs:enumeration value="Normal"/>
      </xs:restriction>
    </xs:simpleType>
  </xs:union>
</xs:simpleType>
<xs:simpleType>
  <xs:restriction base="xs:float"/>
</xs:simpleType>
</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"/>
<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"/>
<xs:enumeration value="lightseagreen"/>
<xs:enumeration value="lightskyblue"/>
<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="mediummagenta"/>
<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"/>
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.

\[
\begin{align*}
\langle x \rangle & ::= \langle \text{signed numeric literal} \rangle \\
\langle y \rangle & ::= \langle \text{signed numeric literal} \rangle \\
\langle z \rangle & ::= \langle \text{signed numeric literal} \rangle \\
\langle m \rangle & ::= \langle \text{signed numeric literal} \rangle \\
\langle \text{quoted name} \rangle & ::= \langle \text{double quote} \rangle \langle \text{name} \rangle \langle \text{double quote} \rangle \\
\langle \text{name} \rangle & ::= \langle \text{letters} \rangle \\
\langle \text{letters} \rangle & ::= (\langle \text{letter} \rangle)^* \\
\langle \text{letter} \rangle & ::= \langle \text{simple Latin letter} \rangle | \langle \text{digit} \rangle | \langle \text{special} \rangle \\
\langle \text{simple Latin letter} \rangle & ::= \langle \text{simple Latin upper case letter} \rangle \\
& \quad | \langle \text{simple Latin lower case letter} \rangle \\
\langle \text{signed numeric literal} \rangle & ::= \{\langle \text{sign} \rangle\}\langle \text{unsigned numeric literal} \rangle \\
\langle \text{unsigned numeric literal} \rangle & ::= \langle \text{exact numeric literal} \rangle \\
& \quad | \langle \text{approximate numeric literal} \rangle
\end{align*}
\]
<approximate numeric literal> ::= <mantissa>E<exponent>

<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> ::= _
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> ::= \langle empty set \rangle | \langle left paren \rangle
\langle point \rangle
\langle comma \rangle \langle point \rangle \langle comma \rangle *
\langle right paren \rangle

<polygon text> ::= \langle empty set \rangle | \langle left paren \rangle
\langle linestring text \rangle
\langle comma \rangle \langle linestring text \rangle \langle comma \rangle *
\langle right paren \rangle

<polyhedralsurface text> ::= \langle empty set \rangle | \langle left paren \rangle
\langle polygon text \rangle
\langle comma \rangle \langle polygon text \rangle \langle comma \rangle *
\langle right paren \rangle

<multipoint text> ::= \langle empty set \rangle | \langle left paren \rangle
\langle point text \rangle
\langle comma \rangle \langle point text \rangle \langle comma \rangle *
\langle right paren \rangle

<multilinestring text> ::= \langle empty set \rangle | \langle left paren \rangle
\langle linestring text \rangle
\langle comma \rangle \langle linestring text \rangle \langle comma \rangle *
\langle right paren \rangle

<multipolygon text> ::= \langle empty set \rangle | \langle left paren \rangle
\langle polygon text \rangle
\langle comma \rangle \langle polygon text \rangle \langle comma \rangle *
\langle right paren \rangle

<geometrycollection text> ::= \langle empty set \rangle | \langle left paren \rangle
\langle geometry tagged text \rangle
\langle comma \rangle \langle geometry tagged text \rangle \langle comma \rangle *
\langle right paren \rangle

7.2.3 BNF Productions for Three-Dimension Geometry WKT

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

<point z> ::= \langle x \rangle \langle y \rangle \langle z \rangle

<geometry z tagged text> ::= \langle point z tagged text \rangle
| \langle linestring z tagged text \rangle
| \langle polygon z tagged text \rangle
| \langle polyhedralsurface z tagged text \rangle
| \langle triangle tagged text \rangle
| \langle tin tagged text \rangle
| \langle multipoint z tagged text \rangle
| \langle multilinestring z tagged text \rangle
| \langle multipolygon z tagged text \rangle
| \langle geometrycollection z tagged text \rangle
<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 text> ::= geometrycollection z 
                                      <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>
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.

\[
\begin{align*}
\langle \text{point m} \rangle & ::= \langle x \rangle \langle y \rangle \langle m \rangle \\
\langle \text{geometry m tagged text} \rangle & ::= \langle \text{point m tagged text} \rangle \\
& \quad | \langle \text{linestring m tagged text} \rangle \\
& \quad | \langle \text{polygon m tagged text} \rangle \\
& \quad | \langle \text{polyhedralsurface m tagged text} \rangle \\
& \quad | \langle \text{triangle tagged m text} \rangle \\
& \quad | \langle \text{tin tagged m text} \rangle \\
& \quad | \langle \text{multipoint m tagged text} \rangle \\
& \quad | \langle \text{multilinestring m tagged text} \rangle \\
& \quad | \langle \text{multipolygon m tagged text} \rangle \\
& \quad | \langle \text{geometrycollection m tagged text} \rangle \\
\end{align*}
\]

\[
\begin{align*}
\langle \text{point m tagged text} \rangle & ::= \langle \text{point m text} \rangle \\
\langle \text{linestring m tagged text} \rangle & ::= \langle \text{linestring m text} \rangle \\
\langle \text{polygon m tagged text} \rangle & ::= \langle \text{polygon m text} \rangle \\
\langle \text{polyhedralsurface m tagged text} \rangle & ::= \langle \text{polyhedralsurface m text} \rangle \\
\langle \text{triangle m tagged text} \rangle & ::= \langle \text{triangle m text} \rangle \\
\langle \text{tin m tagged text} \rangle & ::= \langle \text{tin m text} \rangle \\
\langle \text{multipoint m tagged text} \rangle & ::= \langle \text{multipoint m text} \rangle \\
\langle \text{multilinestring m tagged text} \rangle & ::= \langle \text{multilinestring m text} \rangle \\
\langle \text{multipolygon m tagged text} \rangle & ::= \langle \text{multipolygon m text} \rangle \\
\langle \text{geometrycollection m tagged text} \rangle & ::= \langle \text{geometrycollection m text} \rangle \\
\langle \text{point text} \rangle & ::= \langle \text{point text} \rangle \\
\end{align*}
\]
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 text> ::= polyhedralsurface zm <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 text> ::= geometrycollection zm <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>
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

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

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

<table>
<thead>
<tr>
<th>Type</th>
<th>Code</th>
</tr>
</thead>
<tbody>
<tr>
<td>Geometry Z</td>
<td>1000</td>
</tr>
<tr>
<td>Point Z</td>
<td>1001</td>
</tr>
<tr>
<td>LineString Z</td>
<td>1002</td>
</tr>
<tr>
<td>Polygon Z</td>
<td>1003</td>
</tr>
<tr>
<td>MultiPoint Z</td>
<td>1004</td>
</tr>
<tr>
<td>MultiLineString Z</td>
<td>1005</td>
</tr>
<tr>
<td>MultiPolygon Z</td>
<td>1006</td>
</tr>
<tr>
<td>GeometryCollection Z</td>
<td>1007</td>
</tr>
<tr>
<td>CircularString Z</td>
<td>1008</td>
</tr>
<tr>
<td>CompoundCurve Z</td>
<td>1009</td>
</tr>
<tr>
<td>CurvePolygon Z</td>
<td>1010</td>
</tr>
<tr>
<td>MultiCurve Z</td>
<td>1011</td>
</tr>
<tr>
<td>MultiSurface Z</td>
<td>1012</td>
</tr>
<tr>
<td>Curve Z</td>
<td>1013</td>
</tr>
<tr>
<td>Surface Z</td>
<td>1014</td>
</tr>
<tr>
<td>PolyhedralSurface Z</td>
<td>1015</td>
</tr>
<tr>
<td>TIN Z</td>
<td>1016</td>
</tr>
</tbody>
</table>
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 reference 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.

```cpp
// 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 WKBBByteOrder {
    wkbXDR = 0,       // Big Endian
    wkbNDR = 1        // Little Endian
}

enum WKBGeometryType {
    wkbPoint = 1,
    wkbLineString = 2,
    wkbPolygon = 3,
    wkbTriangle = 4,
    wkbMultiPoint = 5,
    wkbMultiLineString = 6,
    wkbMultiPolygon = 7,
    wkbGeometryCollection = 8,
    wkbPolyhedralSurface = 9,
    wkbTIN = 10

    wkbPointZ = 1001,
    wkbLineStringZ = 1002,
    wkbPolygonZ = 1003,
    wkbTriangleZ = 1004,
    wkbMultiPointZ = 1005,
    wkbMultiLineStringZ = 1006,
    wkbMultiPolygonZ = 1007,
    wkbGeometryCollectionZ = 1008,
    wkbPolyhedralSurfaceZ = 1009,
    wkbTINZ = 1010

    wkbPointM = 2001,
    wkbLineStringM = 2002,
    wkbPolygonM = 2003,
    wkbTriangleM = 2004,
    wkbMultiPointM = 2005,
    wkbMultiLineStringM = 2006,
    wkbMultiPolygonM = 2007,
    wkbGeometryCollectionM = 2008,
    wkbPolyhedralSurfaceM = 2009,
    wkbTINM = 2010

    wkbPointZM = 3001,
    wkbLineStringZM = 3002,
    wkbPolygonZM = 3003,
    wkbTriangleZM = 3004,
    wkbMultiPointZM = 3005,
    wkbMultiLineStringZM = 3006,
    wkbMultiPolygonZM = 3007,
    wkbGeometryCollectionZM = 3008,
    wkbPolyhedralSurfaceZM = 3009,
    wkbTINZM = 3010
}

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
}

WKBPoint2M {
    byte     byteOrder;
    static   uint32  wkbType = 3001;
    Point2M  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]
}

WKBLineString2M {
    byte     byteOrder;
    static   uint32  wkbType = 3002;
    uint32    numPoints;
    Point2M   points[numPoints]
}

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

WKBPolygonZ {
    byte     byteOrder;
    static   uint32  wkbType = 1003;
    uint32    numRings;
    LinearRing2 rings[numRings]
}

Copyright © 2010 Open Geospatial Consortium, Inc.
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]}

Copyright © 2010 Open Geospatial Consortium, Inc.
WKBMultipointZM {
  byte               byteOrder;
  static  uint32    wkbType=3004;
  uint32            numPoints;
  WKBPoinZM         points[numPoints]}

WKBMultipLineString {
  byte               byteOrder;
  static  uint32    wkbType = 5;
  uint32            numLineStrings;
  WKBLinestring     lineStrings[numLineStrings]}

WKBMultipLineStringZ {
  byte               byteOrder;
  static  uint32    wkbType = 1005;
  uint32            numLineStrings;
  WKBLinestringZ    lineStrings[numLineStrings]}

WKBMultipLineStringM {
  byte               byteOrder;
  static  uint32    wkbType = 2005;
  uint32            numLineStrings;
  WKBLinestringM    lineStrings[numLineStrings]}

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

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.

\[
\text{<spatial reference system> ::= <projected cs> | <geographic cs> | <geocentric cs>}
\]

\[
\text{<projected cs> ::= PROJCS <left delimiter> <csname> <comma> <geographic cs> <comma> <projection> (<comma> <parameter> )* <comma> <linear unit> <right delimiter>}
\]

\[
\text{<geographic cs> ::= GEOGCS <left delimiter> <csname> <comma> <datum> <comma> <prime meridian> <comma> <angular unit> (<comma> <linear unit> ) <right delimiter>}
\]

\[
\text{<geocentric cs> ::= GEOCCS <left delimiter> <name> <comma> <datum> <comma> <prime meridian> <comma> <linear unit> <right delimiter>}
\]

\[
\text{<datum> ::= DATUM <left delimiter> <datum name> <comma> <spheroid> <right delimiter>}
\]
<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]]
```
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
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.

Copyright © 2010 Open Geospatial Consortium, Inc.
### A.3.2.2 Basic methods on geometry

<table>
<thead>
<tr>
<th>SFA-CA</th>
<th>Spatial schema</th>
<th>Comment</th>
</tr>
</thead>
<tbody>
<tr>
<td>Geometry.Dimension ( )::Integer</td>
<td>GM_Object::dimension(): Integer</td>
<td></td>
</tr>
<tr>
<td>Geometry.GeometryType ( )::String</td>
<td>Not defined</td>
<td>Defined by an application schema</td>
</tr>
<tr>
<td>Geometry.SRID ( )::Integer</td>
<td>GM_Object::CRS : CRS</td>
<td></td>
</tr>
<tr>
<td>Geometry.Envelope( )::Geometry</td>
<td>GM_Object::envelope(): GM_Envelope</td>
<td>An application has to decide which operator to deploy</td>
</tr>
<tr>
<td>Geometry.AsText( )::String</td>
<td>Not defined</td>
<td>Defined by an application schema</td>
</tr>
<tr>
<td>Geometry.AsBinary( )::Binary</td>
<td>Not defined</td>
<td>Defined by an application schema</td>
</tr>
<tr>
<td>Geometry.IsEmpty( )::Integer</td>
<td>TransfiniteSet&lt;DirectPosition&gt;::isEmpty</td>
<td>Test for the empty set</td>
</tr>
<tr>
<td>Geometry.IsSimple( )::Integer</td>
<td>GM_Object::isSimple(): Boolean</td>
<td></td>
</tr>
</tbody>
</table>

### 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.

<table>
<thead>
<tr>
<th>SFA-CA</th>
<th>Spatial schema</th>
<th>Comment</th>
</tr>
</thead>
</table>
A.3.2.4 Methods that support spatial analysis

<table>
<thead>
<tr>
<th>SFA-CA</th>
<th>Spatial schema</th>
<th>Comment</th>
</tr>
</thead>
<tbody>
<tr>
<td>Geometry.Distance(anotherGeometry: Geometry): Double</td>
<td>GM_Object::distance(): Distance</td>
<td>—</td>
</tr>
<tr>
<td>Geometry.ConvexHull( ):Geometry</td>
<td>GM_Object::convexHull(): GM_Object</td>
<td>—</td>
</tr>
<tr>
<td>Geometry.Intersection(AnotherGeometry:Geometry):Geometry</td>
<td>GM_Object::Intersection(pointSet: GM_Object): GM_Object</td>
<td>In principle, this method is used to define the spatial relations above.</td>
</tr>
</tbody>
</table>

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

<table>
<thead>
<tr>
<th>SFA-CA</th>
<th>Spatial schema</th>
<th>Comment</th>
</tr>
</thead>
<tbody>
<tr>
<td>Point</td>
<td>GM_Point DirectPosition</td>
<td>Both alternatives are valid. DirectPosition defines the ordinates, i.e., the sequence of numeric coordinates denoting a Point.</td>
</tr>
<tr>
<td>Point.X( ):Double</td>
<td>GM_Point::position.ordinate[1] DirectPosition::ordinate[1]</td>
<td>Either of these two, depending on the definition of an application schema</td>
</tr>
</tbody>
</table>
A.3.3.3 Curve

<table>
<thead>
<tr>
<th>SFA-CA</th>
<th>Spatial schema</th>
<th>Comment</th>
</tr>
</thead>
<tbody>
<tr>
<td>Curve</td>
<td>GM_Curve</td>
<td>The notion of a curve in SFA-SQL may correspond to a number of definitions in Spatial schema.</td>
</tr>
<tr>
<td></td>
<td>GM_GenericCurve</td>
<td></td>
</tr>
<tr>
<td></td>
<td>GM_CurveSegment</td>
<td></td>
</tr>
<tr>
<td></td>
<td>GM_LineString</td>
<td></td>
</tr>
<tr>
<td></td>
<td>GM_LineSegment</td>
<td></td>
</tr>
</tbody>
</table>

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

<table>
<thead>
<tr>
<th>SFA-CA</th>
<th>Spatial schema</th>
<th>Comment</th>
</tr>
</thead>
<tbody>
<tr>
<td>LineString</td>
<td>GM_LineString</td>
<td>—</td>
</tr>
<tr>
<td>LinearString.NumPoints( ):Integer</td>
<td>GM_LineString::controlPoints.count</td>
<td>May be derived</td>
</tr>
<tr>
<td>LinearString.PointN(N:Integer):Point</td>
<td>GM_LineString::controlPoints(N)</td>
<td>May be derived</td>
</tr>
</tbody>
</table>

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

<table>
<thead>
<tr>
<th>SFA_CA</th>
<th>Spatial schema</th>
<th>Comment</th>
</tr>
</thead>
<tbody>
<tr>
<td>Polygon</td>
<td>GM_GenericSurface</td>
<td>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].</td>
</tr>
<tr>
<td>Surface.Area( ) : Double</td>
<td>GM_GenericSurface::area() : Area</td>
<td>—</td>
</tr>
<tr>
<td>Surface.Centroid( ) : Point</td>
<td>GM_Object::centroid : DirectPosition</td>
<td>—</td>
</tr>
<tr>
<td>Surface.PointOnSurface( ) : Point</td>
<td>GM_Object::representativePoint() : DirectPosition</td>
<td>—</td>
</tr>
<tr>
<td>Polygon.ExteriorRing( ) : LineString</td>
<td>GM_Polygon::exterior : GM_GenericCurve</td>
<td>The exterior attribute is defined also as zero or more curves in Reference [1].</td>
</tr>
<tr>
<td>Polygon.InteriorRingN (N: Integer) : LineString</td>
<td>Not defined</td>
<td>May be calculated, e.g. from the interior attribute of GM_Polygon</td>
</tr>
<tr>
<td>Polygon.NumInteriorRing( ) : Integer</td>
<td>Not defined</td>
<td>May be calculated, e.g. from the interior attribute of GM_Polygon</td>
</tr>
</tbody>
</table>

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.

<table>
<thead>
<tr>
<th>SFA-CA</th>
<th>Spatial schema</th>
<th>Comment</th>
</tr>
</thead>
<tbody>
<tr>
<td>PolyhedralSurface</td>
<td>GM_PolyhedralSurface as a subtype of GM_Surface</td>
<td>—</td>
</tr>
<tr>
<td>PolyhedralSurface.PatchN (N: Integer) : Polygon</td>
<td>GM_PolyhedralSurface.patch.getAt(N) : GM_Polygon</td>
<td>Retrieve a particular offset in the &quot;patch&quot; association role</td>
</tr>
<tr>
<td>PolyhedralSurface.BoundingPolygons (p: Polygon) : MultiPolygon</td>
<td>---</td>
<td>Query against &quot;patch&quot; for polygons that share boundary with &quot;p&quot;</td>
</tr>
<tr>
<td>IsClosed( ) : Integer</td>
<td>GM_Object.isCycle() : Boolean</td>
<td>—</td>
</tr>
</tbody>
</table>

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.

Copyright © 2010 Open Geospatial Consortium, Inc.
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.

<table>
<thead>
<tr>
<th>SFA-CA</th>
<th>Spatial schema</th>
<th>Comment</th>
</tr>
</thead>
<tbody>
<tr>
<td>GeometryCollection</td>
<td>GM_Aggregate and its subtype</td>
<td></td>
</tr>
<tr>
<td></td>
<td>GM_MultiPrimitive</td>
<td></td>
</tr>
<tr>
<td>GeometryCollection :</td>
<td></td>
<td></td>
</tr>
<tr>
<td>NumGeometries( ) :</td>
<td>GM_Aggregate.element.count : Integer</td>
<td>May be calculated as the count of the</td>
</tr>
<tr>
<td>Integer</td>
<td></td>
<td>&quot;element&quot; association role of GM_Aggregate</td>
</tr>
<tr>
<td>GeometryCollection :</td>
<td></td>
<td></td>
</tr>
<tr>
<td>GeometryN( N : Integer ) : Geometry</td>
<td>GM_Aggregate.element.getAt(N) : GM_Geometry</td>
<td>May be calculated, e.g. from the</td>
</tr>
<tr>
<td></td>
<td></td>
<td>&quot;element&quot; association role of GM_Aggregate</td>
</tr>
</tbody>
</table>

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

<table>
<thead>
<tr>
<th>SFA-CA</th>
<th>Spatial schema</th>
<th>Comment</th>
</tr>
</thead>
<tbody>
<tr>
<td>MultiPoint</td>
<td>GM_MultiPoint</td>
<td>—</td>
</tr>
</tbody>
</table>

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.
<table>
<thead>
<tr>
<th>SFA-CA</th>
<th>Spatial schema</th>
<th>Comment</th>
</tr>
</thead>
</table>
| 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.

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

<table>
<thead>
<tr>
<th>Name</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Metre</td>
<td>1.0</td>
</tr>
<tr>
<td>International Foot</td>
<td>0.304 8</td>
</tr>
<tr>
<td>U.S. Foot</td>
<td>12/39,37</td>
</tr>
<tr>
<td>Modified American Foot</td>
<td>12,000 458 4/39,37</td>
</tr>
<tr>
<td>Clarke's Foot</td>
<td>12/39,370 432</td>
</tr>
<tr>
<td>Indian Foot</td>
<td>12/39,370 141</td>
</tr>
<tr>
<td>Link</td>
<td>7,92/39,370 113</td>
</tr>
<tr>
<td>Link (Benoit)</td>
<td>7,92/39,370 147</td>
</tr>
<tr>
<td>Link (Sears)</td>
<td>7,92/39,370 147</td>
</tr>
<tr>
<td>Chain (Benoit)</td>
<td>792/39,370 113</td>
</tr>
<tr>
<td>Chain (Sears)</td>
<td>792/39,370 147</td>
</tr>
<tr>
<td>Yard (Indian)</td>
<td>36/39,370 141</td>
</tr>
<tr>
<td>Yard (Sears)</td>
<td>36/39,370 147</td>
</tr>
<tr>
<td>Fathom</td>
<td>1,828 8</td>
</tr>
<tr>
<td>Nautical Mile</td>
<td>1 852,0</td>
</tr>
<tr>
<td>South African Cape Foot</td>
<td>0,314 855 575 16</td>
</tr>
<tr>
<td>South African Geodetic Foot</td>
<td>0,304 797 265 4</td>
</tr>
<tr>
<td>German Legal Meter</td>
<td>1,000 013 596 5</td>
</tr>
</tbody>
</table>

B.3 Angular units

Table B - 2 — Angular units

<table>
<thead>
<tr>
<th>Name</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Radian</td>
<td>1.0</td>
</tr>
<tr>
<td>Decimal Degree</td>
<td>π/180</td>
</tr>
<tr>
<td>Decimal Minute</td>
<td>(π/180)/60</td>
</tr>
<tr>
<td>Decimal Second</td>
<td>(π/180)/3 600</td>
</tr>
</tbody>
</table>
### B.4 Ellipsoids and spheres

Table B - 3 — Ellipsoids and spheres

<table>
<thead>
<tr>
<th>Name</th>
<th>Semi-major axis</th>
<th>Inverse flattening</th>
</tr>
</thead>
<tbody>
<tr>
<td>Airy</td>
<td>6 377 563,396</td>
<td>299,324 964 6</td>
</tr>
<tr>
<td>Modified Airy</td>
<td>6 377 340,189</td>
<td>299,324 964 6</td>
</tr>
<tr>
<td>Australian</td>
<td>6 378 160</td>
<td>298,25</td>
</tr>
<tr>
<td>Bessel</td>
<td>6 377 397,155</td>
<td>299,152 812 8</td>
</tr>
<tr>
<td>Modified Bessel</td>
<td>6 377 492,018</td>
<td>299,152 812 8</td>
</tr>
<tr>
<td>Bessel (Namibia)</td>
<td>6 377 483,865</td>
<td>299,152 812 8</td>
</tr>
<tr>
<td>Clarke 1866</td>
<td>6 378 206,4</td>
<td>294,978 698 2</td>
</tr>
<tr>
<td>Clarke 1866 (Michigan)</td>
<td>6 378 693,704</td>
<td>294,978 684 677</td>
</tr>
<tr>
<td>Clarke 1880 (Arc)</td>
<td>6 378 249,145</td>
<td>293,466 307 656</td>
</tr>
<tr>
<td>Clarke 1880 (Benoit)</td>
<td>6 378 300,79</td>
<td>293,466 234 571</td>
</tr>
<tr>
<td>Clarke 1880 (IGN)</td>
<td>6 378 249,2</td>
<td>293,466 02</td>
</tr>
<tr>
<td>Clarke 1880 (Modified)</td>
<td>6 378 249,145</td>
<td>293,466 315 8</td>
</tr>
<tr>
<td>Clarke 1880 (RGS)</td>
<td>6 378 249,145</td>
<td>293,465</td>
</tr>
<tr>
<td>Clarke 1880 (SGA)</td>
<td>6 378 249,2</td>
<td>293,465 98</td>
</tr>
<tr>
<td>Everest 1830</td>
<td>6 377 276,345</td>
<td>300,801 7</td>
</tr>
<tr>
<td>Everest 1975</td>
<td>6 377 301,243</td>
<td>300,801 7</td>
</tr>
<tr>
<td>Everest (Sarawak and Sabah)</td>
<td>6 377 298,556</td>
<td>300,801 7</td>
</tr>
<tr>
<td>Modified Everest 1948</td>
<td>6 377 304,063</td>
<td>300,801 7</td>
</tr>
<tr>
<td>GEM10C</td>
<td>6 378 137</td>
<td>298,257 222 101</td>
</tr>
<tr>
<td>GRS 1980</td>
<td>6 378 137</td>
<td>298,257 222 101</td>
</tr>
<tr>
<td>Helmert 1906</td>
<td>6 378 200</td>
<td>298,3</td>
</tr>
<tr>
<td>International 1924</td>
<td>6 378 388</td>
<td>297,0</td>
</tr>
<tr>
<td>Krasovsky</td>
<td>6 378 245</td>
<td>298,3</td>
</tr>
<tr>
<td>NWL9D</td>
<td>6 378 145</td>
<td>298,25</td>
</tr>
<tr>
<td>OSU_86F</td>
<td>6 378 136,2</td>
<td>298,257 22</td>
</tr>
<tr>
<td>OSU_91A</td>
<td>6 378 136,3</td>
<td>298,257 22</td>
</tr>
<tr>
<td>Plessis 1817</td>
<td>6 376 523</td>
<td>308,64</td>
</tr>
<tr>
<td>Sphere (radius = 1.0)</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>Sphere (radius = 6 371 000 m)</td>
<td>6 371 000</td>
<td>0</td>
</tr>
</tbody>
</table>
### B.5 Geodetic datums

#### Table B - 4— Geodetic datums

<table>
<thead>
<tr>
<th>Name</th>
<th>Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>Adindan</td>
<td>Liberia 1964</td>
</tr>
<tr>
<td>Afgooye</td>
<td>Lisbon</td>
</tr>
<tr>
<td>Agadez</td>
<td>Loma Quintana</td>
</tr>
<tr>
<td>Australian Geodetic Datum 1966</td>
<td>Lome</td>
</tr>
<tr>
<td>Australian Geodetic Datum 1984</td>
<td>Luzon 1911</td>
</tr>
<tr>
<td>Ain el Abd 1970</td>
<td>Mahe 1971</td>
</tr>
<tr>
<td>Amersfoort</td>
<td>Makassar</td>
</tr>
<tr>
<td>Aratu</td>
<td>Malongo 1987</td>
</tr>
<tr>
<td>Arc 1950</td>
<td>Manoca</td>
</tr>
<tr>
<td>Arc 1960</td>
<td>Massawa</td>
</tr>
<tr>
<td>Ancienne Triangulation Française</td>
<td>Merchich</td>
</tr>
<tr>
<td>Barbados</td>
<td>Militar-Geographische Institute</td>
</tr>
<tr>
<td>Batavia</td>
<td>Mhast</td>
</tr>
<tr>
<td>Beduaram</td>
<td>Minna</td>
</tr>
<tr>
<td>Beijing 1954</td>
<td>Monte Mario</td>
</tr>
<tr>
<td>Reseau National Belge 1950</td>
<td>M'poraloko</td>
</tr>
<tr>
<td>Reseau National Belge 1972</td>
<td>NAD Michigan</td>
</tr>
<tr>
<td>Bermuda 1957</td>
<td>North American Datum 1927</td>
</tr>
<tr>
<td>Bern 1898</td>
<td>North American Datum 1983</td>
</tr>
<tr>
<td>Bern 1938</td>
<td>Nahrwan 1967</td>
</tr>
<tr>
<td>Bogota</td>
<td>Naparima 1972</td>
</tr>
<tr>
<td>Bukit Rimpah</td>
<td>Nord de Guerre</td>
</tr>
<tr>
<td>Camacupa</td>
<td>NGO 1948</td>
</tr>
<tr>
<td>Campo Inchauspe</td>
<td>Nord Sahara 1959</td>
</tr>
<tr>
<td>Cape</td>
<td>NSWC 9Z-2</td>
</tr>
<tr>
<td>Carthage</td>
<td>Nouvelle Triangulation Française</td>
</tr>
<tr>
<td>Chua</td>
<td>New Zealand Geodetic Datum 1949</td>
</tr>
<tr>
<td>Conakry 1905</td>
<td>OS (SN) 1980</td>
</tr>
<tr>
<td>Corrego Alegre</td>
<td>OSGB 1936</td>
</tr>
<tr>
<td>Côte d'Ivoire</td>
<td>OSGB 1970 (SN)</td>
</tr>
<tr>
<td>Datum 73</td>
<td>Padang 1884</td>
</tr>
<tr>
<td>-------------------------</td>
<td>-----------------------------------</td>
</tr>
<tr>
<td>Deir ez Zor</td>
<td>Palestine 1923</td>
</tr>
<tr>
<td>Deutsche Hauptdreiecksnetz</td>
<td>Pointe Noire</td>
</tr>
<tr>
<td>Douala</td>
<td>Provisional South American Datum 1956</td>
</tr>
<tr>
<td>European Datum 1950</td>
<td>Pulkovo 1942</td>
</tr>
<tr>
<td>European Datum 1987</td>
<td>Qatar</td>
</tr>
<tr>
<td>Egypt 1907</td>
<td>Qatar 1948</td>
</tr>
<tr>
<td>European Reference System 1989</td>
<td>Oronoq</td>
</tr>
<tr>
<td>Fahud</td>
<td>RT38</td>
</tr>
<tr>
<td>Gandajika 1970</td>
<td>South American Datum 1969</td>
</tr>
<tr>
<td>Garoua</td>
<td>Sapper Hill 1943</td>
</tr>
<tr>
<td>Geocentric Datum of Australia 1994</td>
<td>Schwarzeck</td>
</tr>
<tr>
<td>Guyane Française</td>
<td>Segora</td>
</tr>
<tr>
<td>Hartebeeshoek(WGS84) South African</td>
<td>Serindung</td>
</tr>
<tr>
<td>Herat North</td>
<td>Stockholm 1938</td>
</tr>
<tr>
<td>Hito XVIII 1963</td>
<td>Sudan</td>
</tr>
<tr>
<td>Hu Tzu Shan</td>
<td>Tananarive 1925</td>
</tr>
<tr>
<td>Hungarian Datum 1972</td>
<td>Timbalai 1948</td>
</tr>
<tr>
<td>Indian 1954</td>
<td>TM65</td>
</tr>
<tr>
<td>Indian 1975</td>
<td>TM75</td>
</tr>
<tr>
<td>Indonesian Datum 1974</td>
<td>Tokyo</td>
</tr>
<tr>
<td>Jamaica 1875</td>
<td>Trinidad 1903</td>
</tr>
<tr>
<td>Jamaica 1969</td>
<td>Trucial Coast 1948</td>
</tr>
<tr>
<td>Japanese Geodetic Datum 2000</td>
<td>Voirol 1875</td>
</tr>
<tr>
<td>Kalianpur</td>
<td>Voirol Unifie 1960</td>
</tr>
<tr>
<td>Kandawala</td>
<td>WGS 1972</td>
</tr>
<tr>
<td>Kertau</td>
<td>WGS 1972 Transit Broadcast Ephemeris</td>
</tr>
<tr>
<td>Kuwait Oil Company</td>
<td>WGS 1984</td>
</tr>
<tr>
<td>La Canoa</td>
<td>Yacare</td>
</tr>
<tr>
<td>Lake</td>
<td>Yoff</td>
</tr>
<tr>
<td>Leigon</td>
<td>Zanderij</td>
</tr>
</tbody>
</table>
B.6 Prime meridians

<table>
<thead>
<tr>
<th>Name</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Greenwich</td>
<td>0° 0' 0&quot;</td>
</tr>
<tr>
<td>Bern</td>
<td>7° 26' 22.5&quot; E</td>
</tr>
<tr>
<td>Bogota</td>
<td>74° 4' 51.3&quot; W</td>
</tr>
<tr>
<td>Brussels</td>
<td>4° 22' 4.71&quot; E</td>
</tr>
<tr>
<td>Ferro</td>
<td>17° 40' 0&quot; W</td>
</tr>
<tr>
<td>Jakarta</td>
<td>106° 48' 27.79&quot; E</td>
</tr>
<tr>
<td>Lisbon</td>
<td>9° 7' 54.862&quot; W</td>
</tr>
<tr>
<td>Madrid</td>
<td>3° 41' 16.58&quot; W</td>
</tr>
<tr>
<td>Paris</td>
<td>2° 20' 14.025&quot; E</td>
</tr>
<tr>
<td>Rome</td>
<td>12° 27' 8.4&quot; E</td>
</tr>
<tr>
<td>Stockholm</td>
<td>18° 3' 29&quot; E</td>
</tr>
</tbody>
</table>

B.7 Map projections

<table>
<thead>
<tr>
<th>Cylindrical projections</th>
<th>Conic projections</th>
</tr>
</thead>
<tbody>
<tr>
<td>Cassini</td>
<td>Albers conic equal-area</td>
</tr>
<tr>
<td>Gauss-Kruger</td>
<td>Lambert conformal conic</td>
</tr>
<tr>
<td>Mercator</td>
<td>Azimuthal or Planar Projections</td>
</tr>
<tr>
<td>Oblique Mercator (Hotine)</td>
<td>Polar Stereographic</td>
</tr>
<tr>
<td>Transverse Mercator</td>
<td>Stereographic</td>
</tr>
</tbody>
</table>
### B.8 Map projection parameters

**Table B - 7 — Map projection parameters**

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


