Open Geospatial Consortium Inc.

Date: 2011-05-28

Reference number of this document: OGC 06-103r4

Version: 1.2.1

Status: Corrigendum

Category: OpenGIS ® Implementation Standard

Editor: John R. Herring

OpenGIS® Implementation Standard for Geographic information - Simple feature access - Part 1: Common architecture

Document type: OpenGIS Implementation Standard
Document subtype: (none)
Document stage: Corrigendum
Document language: English

License Agreement

Contents

Page

Figures

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*

This version supersedes all previous versions of OpenGIS® Simple Features Implementation Standard for SQL, including portions of OGC 99-049 "OpenGIS Simple Features Standard for SQL Rev 1.1", OGC 99-050 "OpenGIS Simple Features Standard For OLE/COM Rev 1.1", OGC 99-054 “OpenGIS Simple Features Standard For CORBA Revision 1.1.”, and OGC 05-126 “OpenGIS Implementation Standard for Geographic information - Simple feature access - Part 1: Common architecture”.

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.

[1] ***ISO/IEC CD 13249-3:2006(E) – Text for FDIS Ballot Information technology – Database languages – SQL Multimedia and Application Packages — Part 3: Spatial, May 15, 2006.***

[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***

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)
WKT	Well-Known Text
WKTR	Well-Known Text Representation
XDR	Big Endian byte order encoding

5.2 Symbols

nD	n-Dimensional, where n may be any integer
$\Re^n$	n-Dimensional coordinate space, where n may be any integer
$\varnothing$	empty set, the set having no members
$\cap$	intersection, operation on two or more sets
$\cup$	union, operation on two or more sets
$-$	difference, operation on two sets
$\in$	is a member of, relation between an element and a set
$\notin$	is not a member of
$\subset$	is a proper subset of, i.e. a smaller set not containing all of the larger
$\subseteq$	is a subset of

$\Leftrightarrow$	if and only if, logical equivalence between statements
$\Rightarrow$	implies, logical implication where the second follows from the first statement
$\exists$	there exists
$\forall$	for all
$\ni$	such that
$f: D \rightarrow R$	Function "f" from domain "D" to range "R"
$\{ X \mid s \}$	set of "X" such that the statement "s" is TRUE
$\wedge$	and, logical intersection
$\vee$	or, logical union
$\neg$	not, logical negation
$=$	equal
$\neq$	not equal
$\leq$	less than or equal to
$<$	less than
$\geq$	greater than or equal to
$>$	greater than
∂	topological boundary operator, mapping a geometric object to its boundary

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.

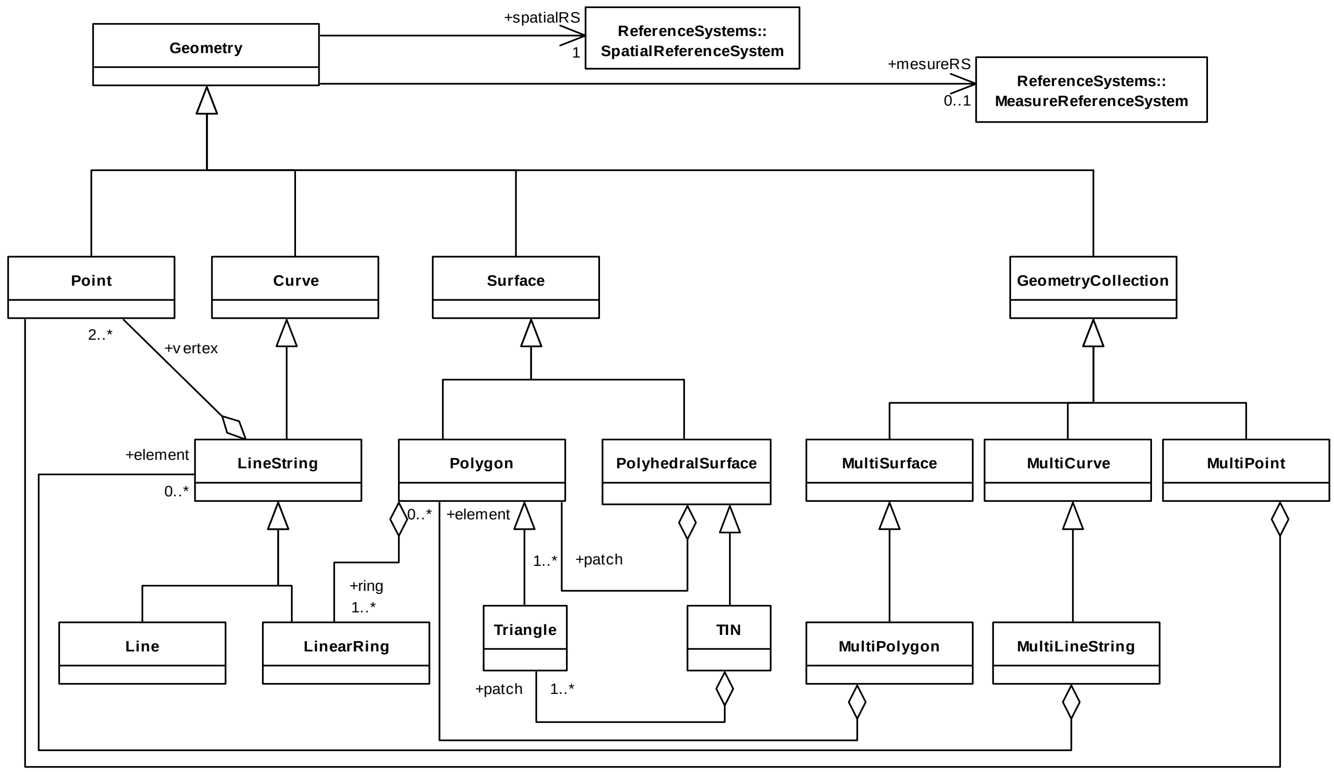


Figure 1: Geometry class hierarchy

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 ($\Re^2$, $\Re^3$ or $\Re^4$). Geometry values in R^2 have points with coordinate values for x and y. Geometry values in R^3 have points with coordinate values for x, y and z or for x, y and m. Geometry values in 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.

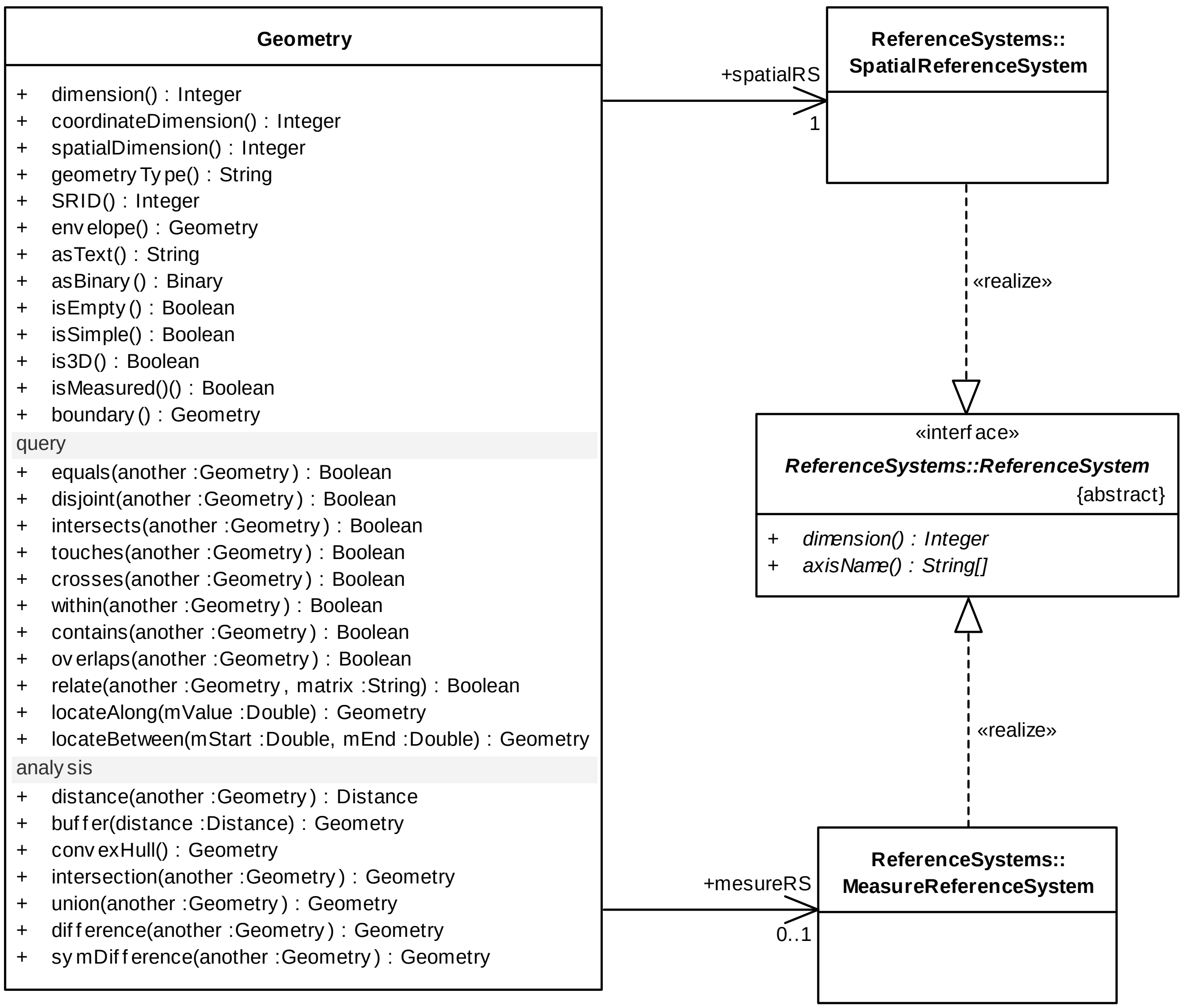


Figure 2: Geometry class operations

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 [(MINX, MINY), (MAXX, MINY), (MAXX, MAXY), (MINX, MAXY), (MINX, 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 ∅ 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))

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.

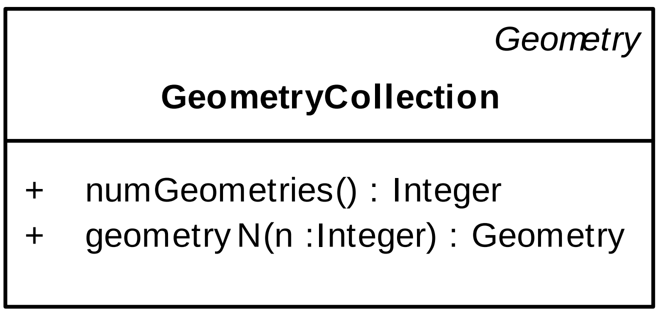


Figure 3: Geometry collection operations

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.

— **GeometryN** (N: integer): Geometry — Returns the Nth geometry 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.

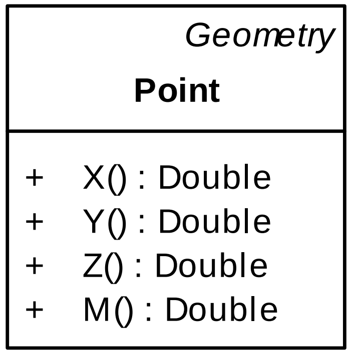


Figure 4: Point

6.1.4.2 Methods

— **X** ( ):Double — The *x*-coordinate value for *this* Point.

— **Y** ( ):Double — The *y*-coordinate value for *this* Point.

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

— **M** ( ):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:

$$D = [a, b] = \{t \in \Re \mid a \le t \le b\}$$

under a mapping

$$f : [a, b] \to \Re^n$$

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):

$$\forall\ c \in \mathrm{Curve},\ [a, b] = c.\mathrm{Domain},\ c =: f : [a, b] \to \Re^n$$
$$c.\mathrm{IsSimple} \Leftrightarrow \forall\ x_1, x_2 \in [a, b]: [\ f(x_1)=f(x_2) \wedge x_1<x_2] \Rightarrow [x_1=a \wedge x_2=b]$$

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

$$c.\mathrm{IsClosed} \Leftrightarrow [f(a) = f(b)]$$

The boundary of a closed Curve is empty.

$$c.\mathrm{IsClosed} \Leftrightarrow [c.\mathrm{boundary} = \varnothing]$$

A Curve that is simple and closed is a Ring.

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

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

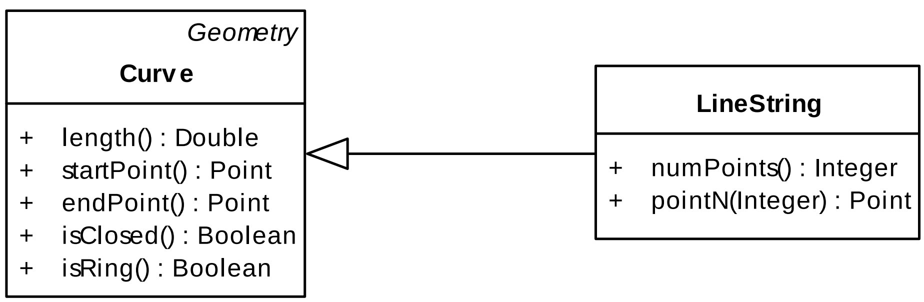


Figure 5: Curve

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).

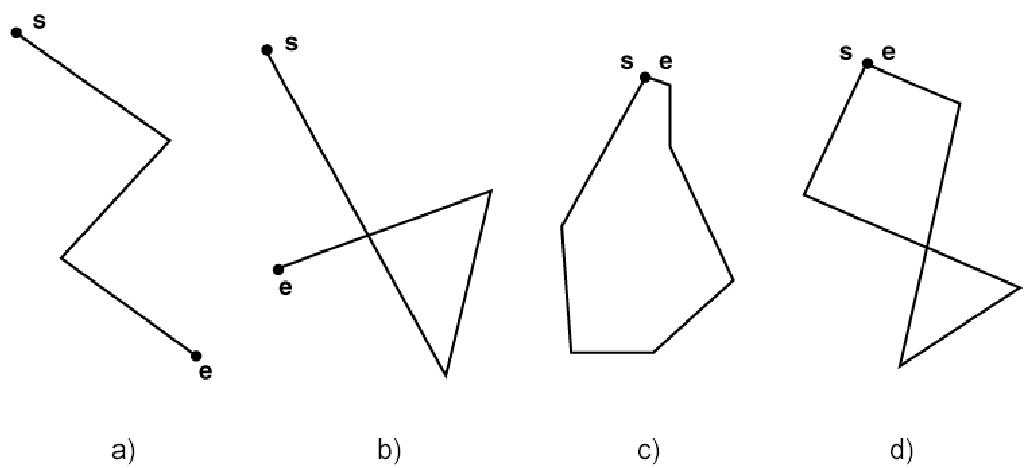
6.1.7 LineString, Line, LinearRing

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.



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)

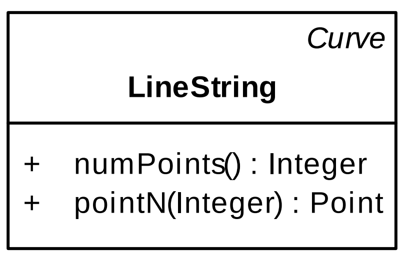


Figure 7: LineString

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.

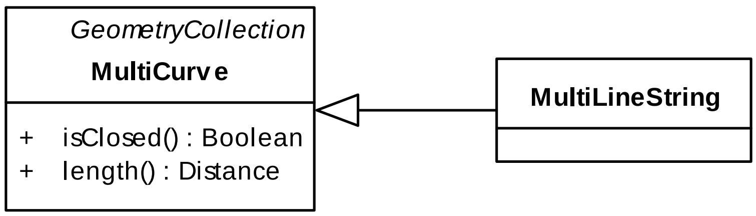


Figure 8: MultiCurve

6.1.8.2 Methods

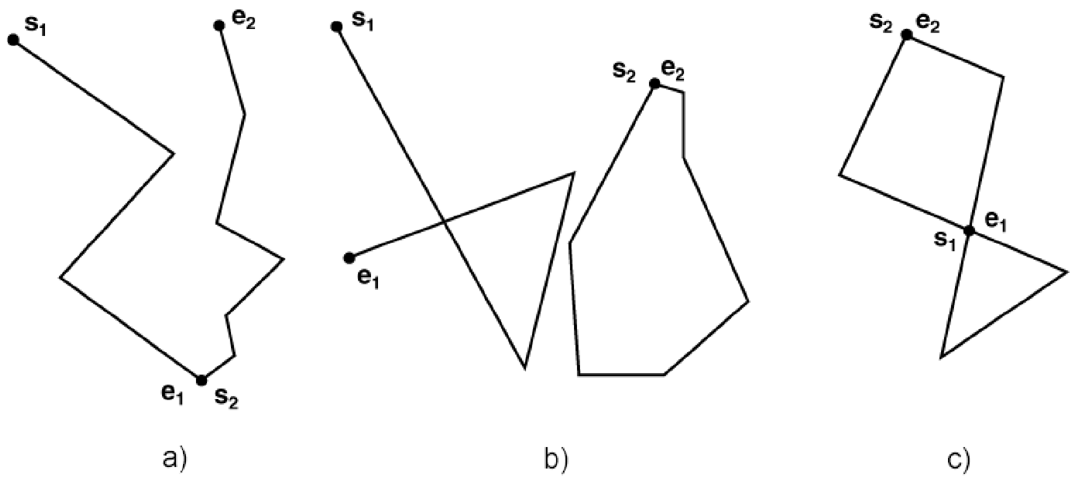
— **IsClosed** ( ): Integer — Returns 1 (TRUE) if *this* MultiCurve is closed [StartPoint ( ) = EndPoint ( ) for each Curve in *this* MultiCurve].

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

6.1.9 MultiLineString

A MultiLineString is a MultiCurve whose elements are LineStrings.

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



Key

s start

e end

Figure 9: Examples of MultiLineStrings

Note: The diagram above contains: Simple MultiLineString (a), Non-simple MultiLineString with 2 elements (b), Non-simple, closed MultiLineString with 2 elements (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.

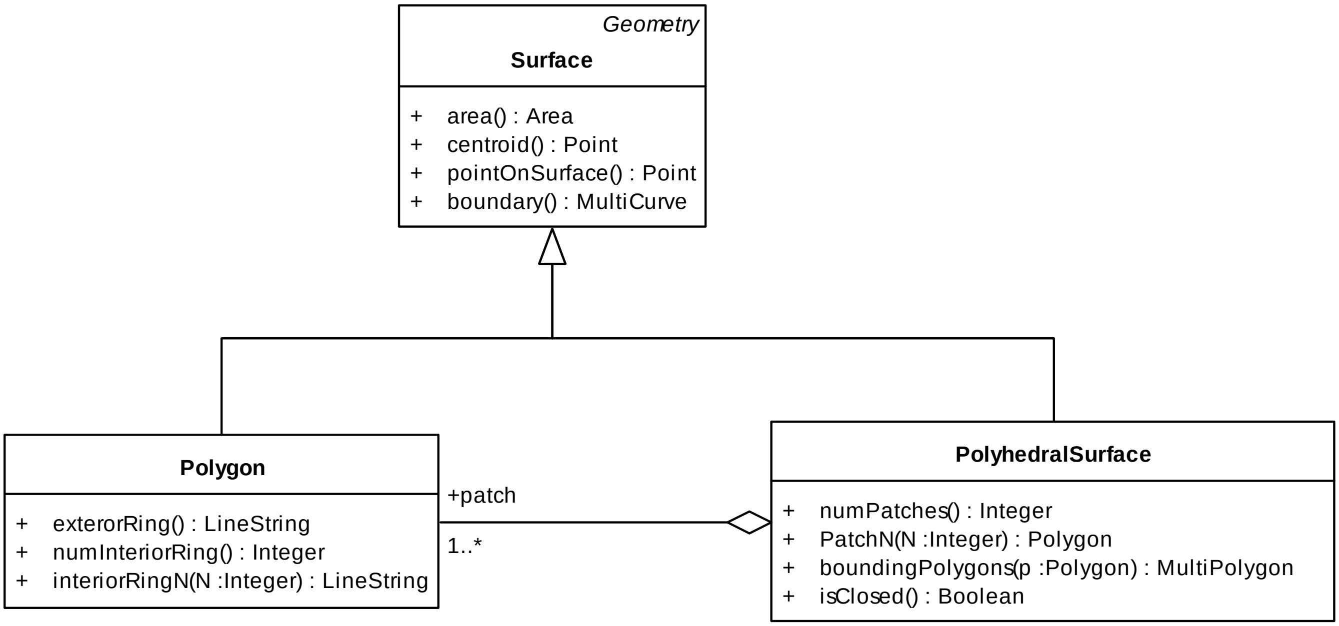


Figure 10: Surface

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.

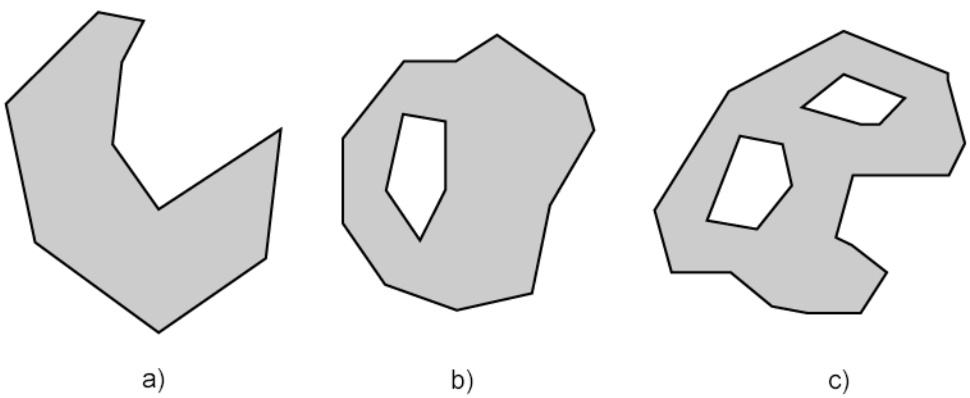


Figure 11: Examples of Polygons with 1 (a), 2 (b) and 3 (c) Rings, respectively

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

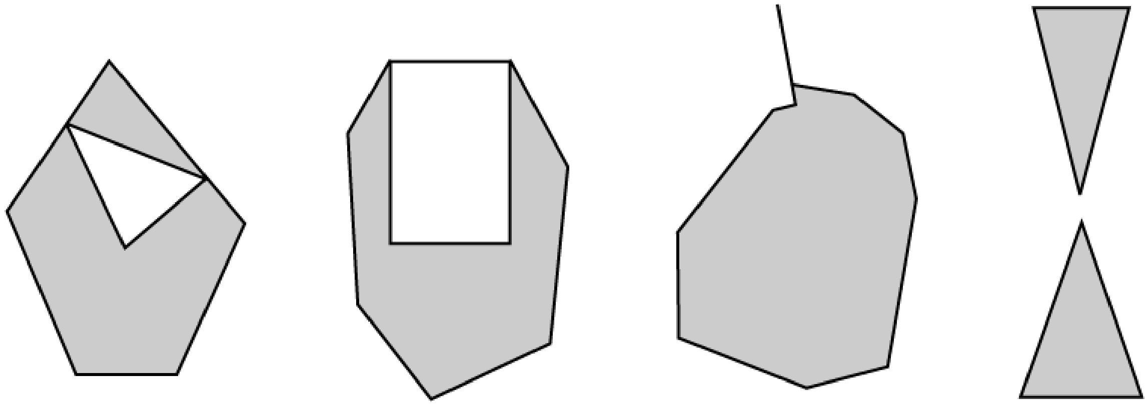


Figure 12: Examples of objects not representable as a single instance of Polygon

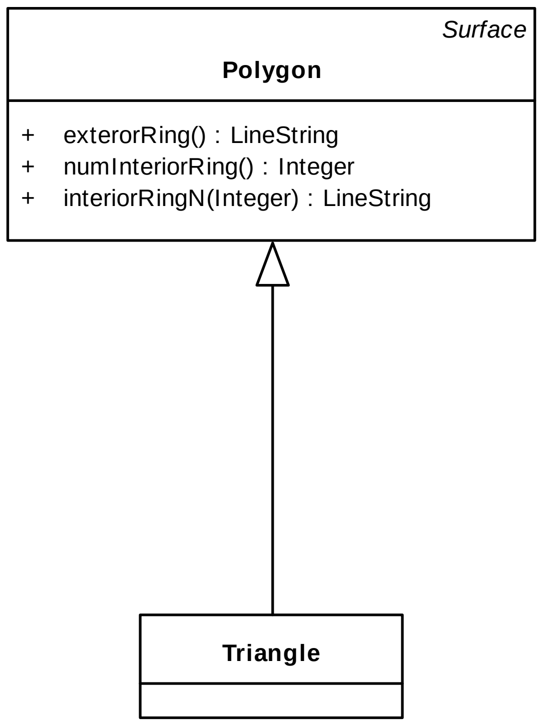


Figure 13: Polygon

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 N^{th} 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.

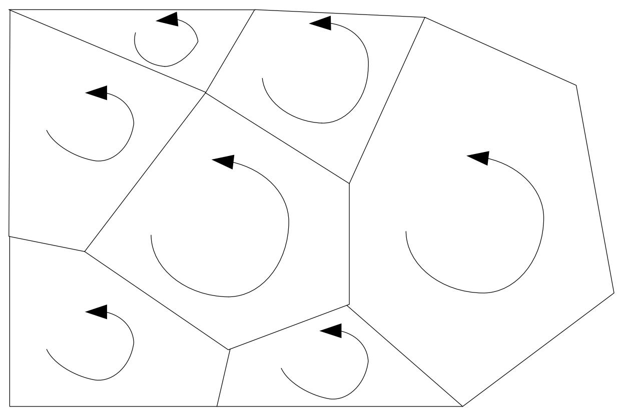


Figure 14: Polyhedral Surface with consistent orientation

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

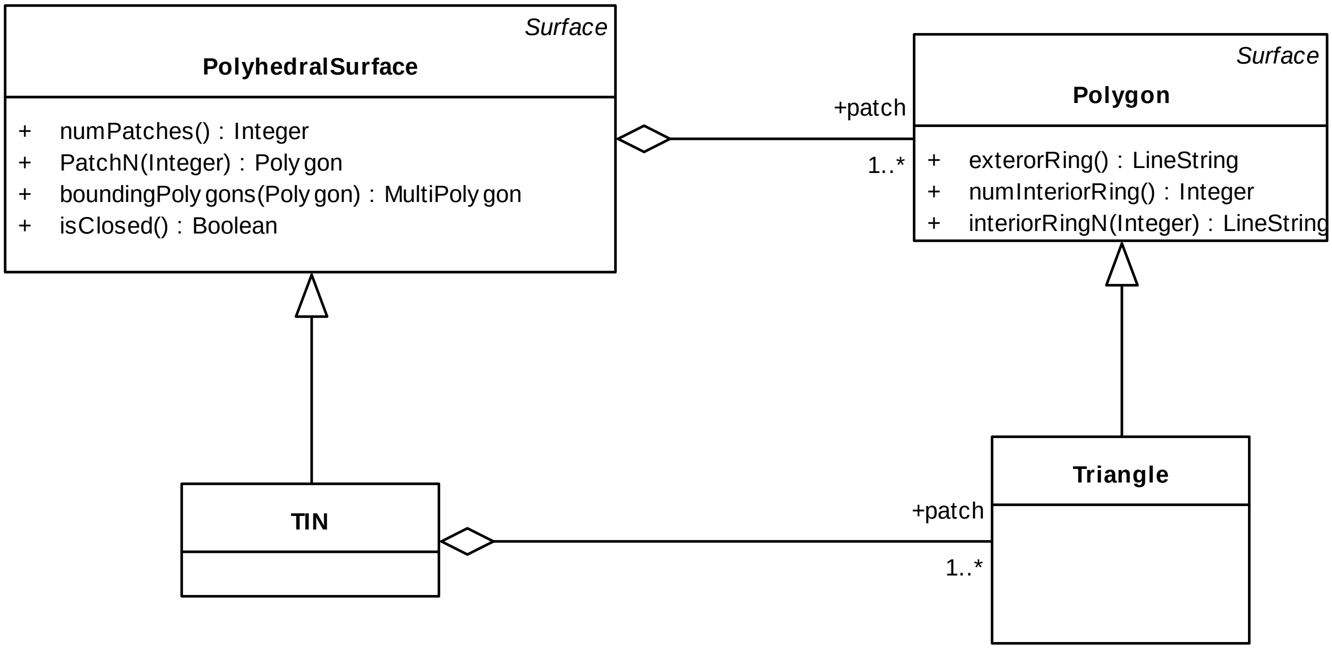


Figure 15: Polyhedral Surface

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.

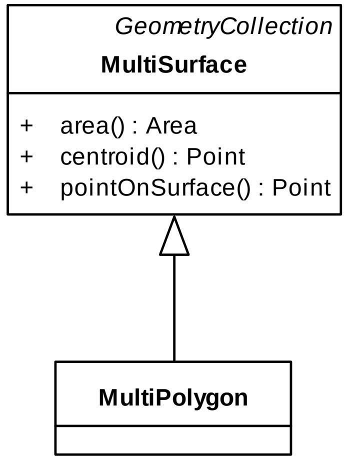


Figure 16: MultiSurface operations

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.

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

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.

$$\forall M \in \text{MultiPolygon},\ \forall P_i, P_j \in M.\text{Geometries}(),\\ \forall c_i, c_j \in \text{Curve}\ c_i \in P_i.\text{Boundaries}(),\ c_j \in P_j.\text{Boundaries}()\\ \exists k \in \text{Integer} \ni c_i \cap c_j = \{p_1, \ldots, p_k \mid p_m \in \text{Point},\ 0<m<k\};$$

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.

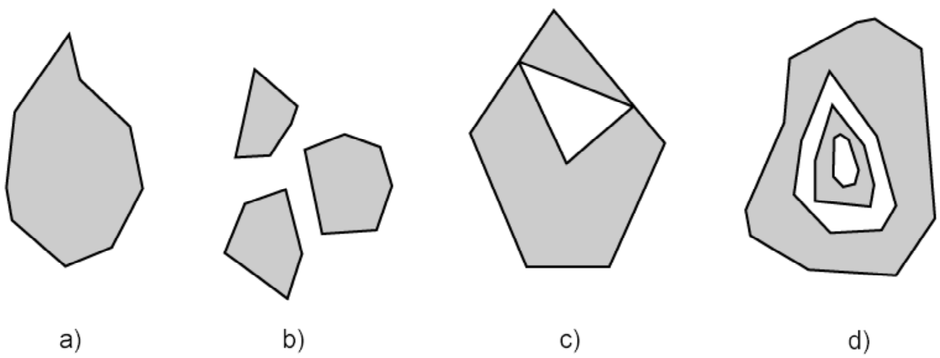


Figure 17: Examples of MultiPolygons with 1 (a), 3 (b) , 2 (c) and 2 (d) Polygon elements

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

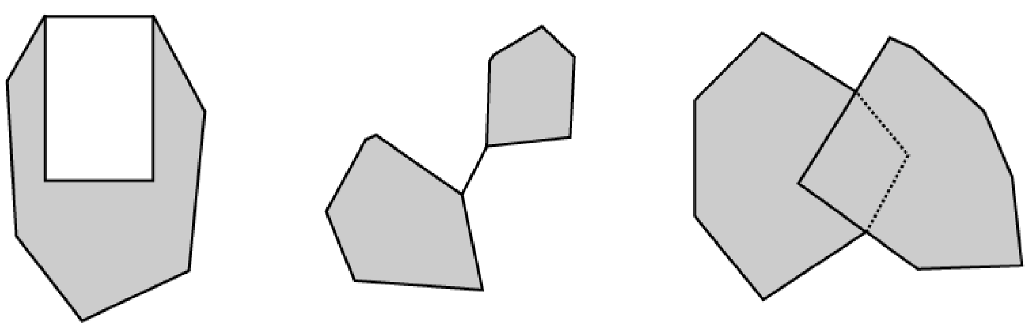


Figure 18: Geometric objects not representable as a single instance of a MultiPolygon

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 ($\Re^2$) 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 $\Re^2$, 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 $\Re^1$ and $\Re^2$ 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.

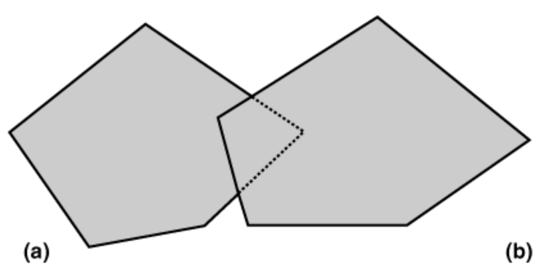
Table 1 shows the general form of the dimensionally extended nine-intersection matrix (DE-9IM).

Table 1: The DE-9IM

	Interior	**Boundary**	**Exterior**
Interior	$dim(I(a) \cap I(b))$	$dim(I(a) \cap B(b))$	$dim(I(a) \cap E(b))$
Boundary	$dim(B(a) \cap I(b))$	$dim(B(a) \cap B(b))$	$dim(B(a) \cap E(b))$
Exterior	$dim(E(a) \cap I(b))$	$dim(E(a) \cap B(b))$	$dim(E(a) \cap E(b))$

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.



	Interior	Boundary	Exterior
Interior	2	1	2
Boundary	1	0	1
Exterior	2	1	2

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:

```
p = T ⇒ dim(x) ∈ {0, 1, 2}, i.e. x ≠ ∅
p = F ⇒ dim(x) = -1, i.e. x = ∅
p = * ⇒ dim(x) ∈ {-1, 0, 1, 2}, i.e. Don’t Care
p = 0 ⇒ dim(x) = 0
p = 1 ⇒ dim(x) = 1
p = 2 ⇒ dim(x) = 2
```

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:

```
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.Equals(b) ⇔ a ⊆ b ∧ b ⊆ a
```

Expressed in terms of the DE-9IM:

```
a.Equals(b)  ⇔ [  (I(a) ∩ I(b) ≠ ∅) ∧
```

```
                  (I(a) ∩ B(b) = ∅) ∧
                  (I(a) ∩ E(b) = ∅) ∧
                  (B(a) ∩ I(b) = ∅) ∧
                  (B(a) ∩ B(b) ≠ ∅) ∧
                  (B(a) ∩ E(b) = ∅) ∧
                  (E(a) ∩ I(b) = ∅) ^
                  (E(a) ∩ B(b) = ∅) ^
                  (E(a) ∩ E(b) ≠ ∅) ]
             ⇔  a.Relate(b, "TFFFTFFFT")
```

Disjoint

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

```
a.Disjoint(b) ⇔ a ∩ b = ∅
```

Expressed in terms of the DE-9IM:

```
a.Disjoint(b) ⇔ [  (I(a) ∩ I(b) = ∅) ∧
                   (I(a) ∩ B(b) = ∅) ∧
                   (B(a) ∩ I(b) = ∅) ∧
                   (B(a) ∩ B(b) = ∅) ]
             ⇔  a.Relate(b, "FF*FF****")
```

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.Touch(b) ⇔ (I(a)∩I(b)=∅)∧(a∩b)≠∅
```

Expressed in terms of the DE-9IM:

```
[a.Touch(b) ⇔ [    (I(a) ∩ I(b) = ∅) ∧
                   [   (B(a) ∩ I(b) ≠ ∅) ∨
                       (I(a) ∩ B(b) ≠ ∅) ∨
                       (B(a) ∩ B(b) ≠ ∅) ] ]
             ⇔ [  a.Relate(b, "FT*******") ∨
                  a.Relate(b, "F**T*****") ∨
                  a.Relate(b, "F***T****") ]
```

Figure 9 shows some examples of the Touches relationship.

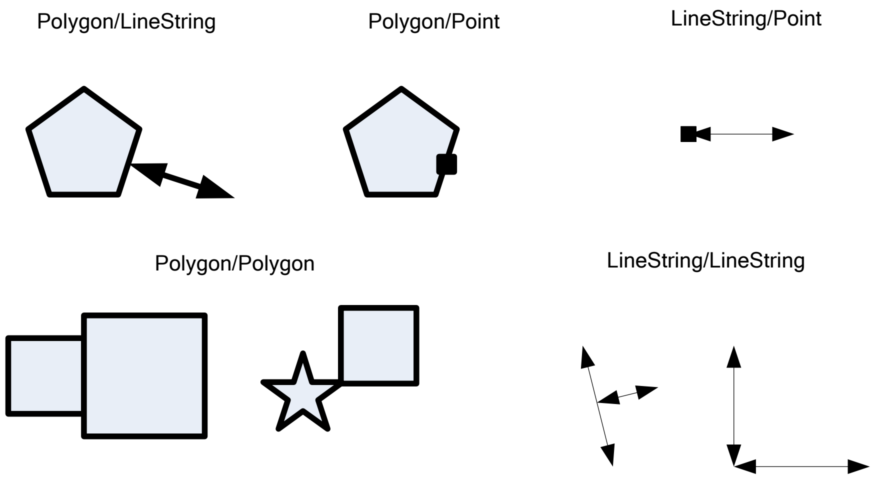


Figure 20: Examples of the Touches relationship

Crosses

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

```
a.Cross(b) ⇔ [I(a)∩I(b)≠∅ ∧ (a ∩ b ≠a) ∧ (a ∩ b ≠b)]
```

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

Expressed in terms of the DE-9IM:

```
Case a∈P, b∈L or
Case a∈P, b∈A or
Case a∈L, b∈A:
  a.Cross(b)  ⇔ [ I(a)∩I(b)≠∅ ∧ I(a)∩E(b)≠∅ ]
                ⇔ a.Relate(b, "T*T******")

Case a∈L, b∈L:
  a.Cross(b)  ⇔ dim(I(a)∩I(b)) = 0
                   ⇔ a.Relate(b, "0********");
```

Figure 10 shows some examples of the Crosses relationship.

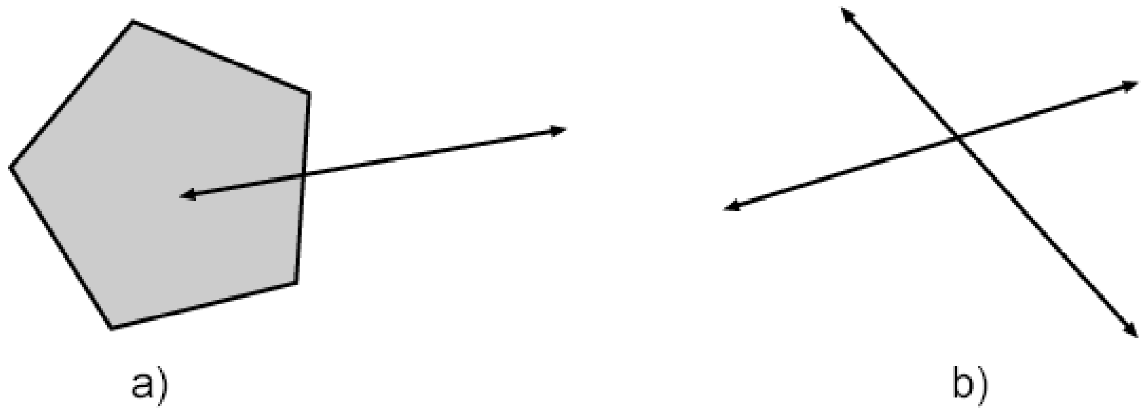


Figure 21: Examples of the Crosses relationship
Polygon/LineString (a)
and LineString/LineString (b)

Within

The Within relationship is defined as

```
a.Within(b) ⇔ (a∩b=a) ∧ (I(a)∩E(b)=∅)
```

Expressed in terms of the DE-9IM:

```
a.Within(b)  ⇔ [ I(a)∩I(b)≠∅ ∧ I(a)∩E(b)=∅ ∧ B(a)∩E(b)=∅ ]
 ⇔ a.Relate(b, "T*F**F***")
```

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

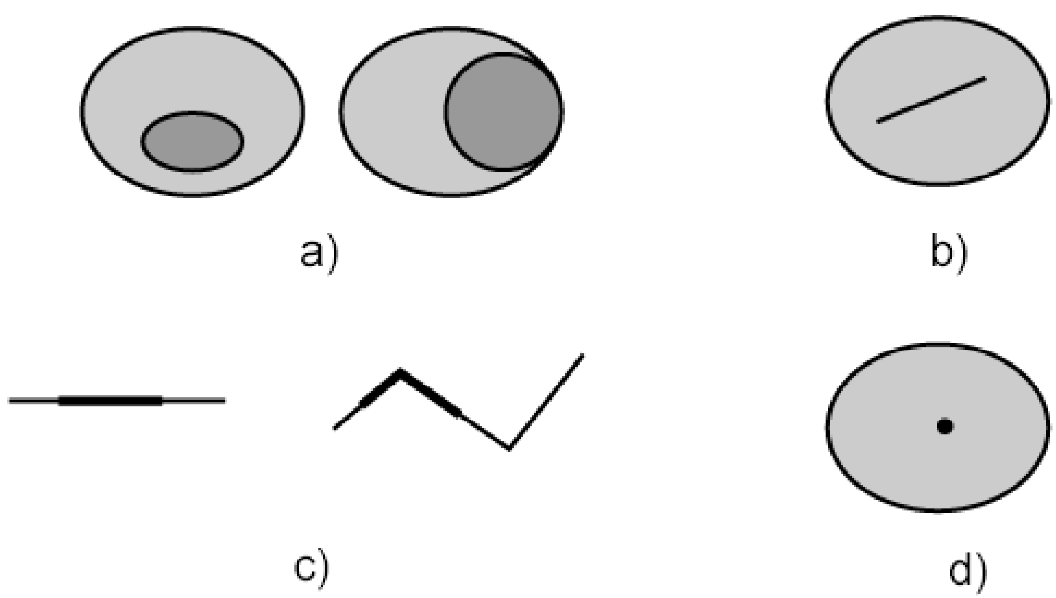


Figure 22: Examples of the “Within” relationship
Polygon/Polygon (a), Polygon/LineString (b), LineString/LineString (c), and Polygon/Point (d)

Overlaps

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

It is defined as

```
a.Overlaps(b) ⇔ ( dim(I(a)) = dim(I(b)) = dim(I(a) ∩ I(b)))
                    ∧ (a ∩ b ≠ a) ∧ (a ∩ b ≠ b)
```

Expressed in terms of the DE-9IM:

```
Case a ∈ P, b ∈ P or Case a ∈ A, b ∈ A:
   a.Overlaps(b)  ⇔     (I(a) ∩ I(b) ≠ ∅) ∧
                        (I(a) ∩ E(b) ≠ ∅) ∧
                        (E(a) ∩ I(b) ≠ ∅)
                  ⇔ a.Relate(b, "T*T***T**")
Case a ∈ L, b ∈ L:
   a.Overlaps(b) ⇔ (dim(I(a) ∩ I(b) = 1) ∧ (I(a) ∩ E(b) ≠ ∅) ∧ (E(a) ∩ I(b) ≠
      ∅) ⇔ a.Relate(b, "1*T***T**")
```

Figure 12 shows some examples of the Overlaps relationship.

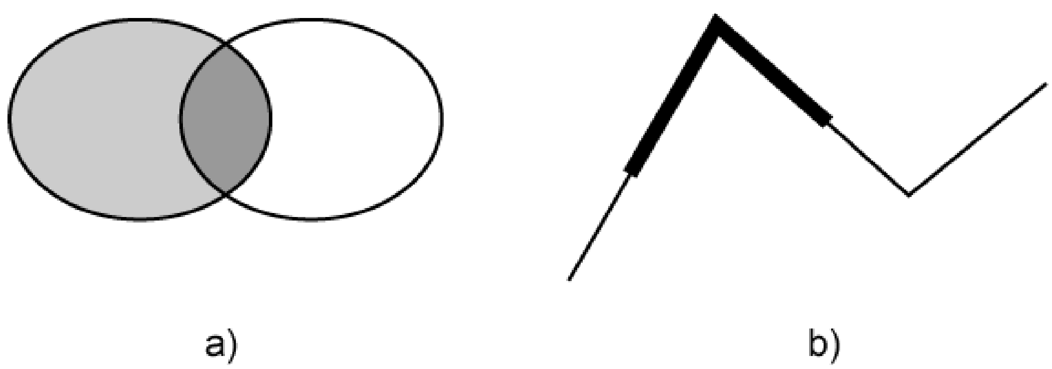


Figure 23: Examples of the Overlaps relationship
Polygon/LineString (a)
and LineString/LineString (b)

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

Contains

```
a.Contains(b) ⇔ b.Within(a)
```

Intersects

```
a.Intersects(b) ⇔ ! a.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.

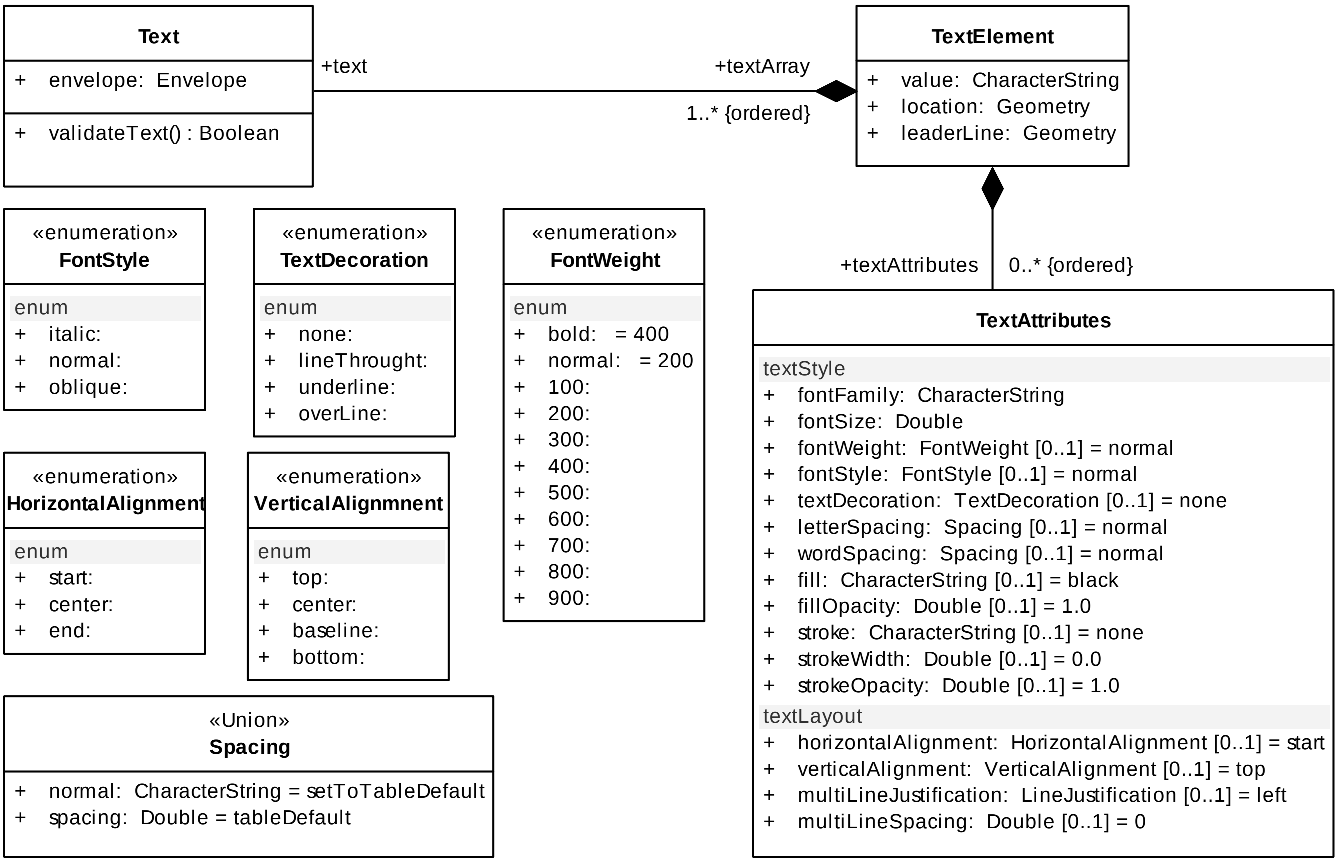


Figure 24: Text classes

Table 2: Fields of the Annotation Text object type

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

Table 3: Fields of the Annotation Text Element object type

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

6.2.2 Text attributes

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

Table 4: Attributes for textstyle

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

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

Table 5: Attributes for Text Layout

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

6.2.3 XML for Text Attributes

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

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

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

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

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

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

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

7 Well-known Text Representation for Geometry

7.1 Component overview

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

7.2 Component description

7.2.1 BNF Introduction

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

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

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

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

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

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

<y> ::=                           <signed numeric literal>

<z> ::=                           <signed numeric literal>

<m> ::=                           <signed numeric literal>

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

<name> ::=                        <letters>

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

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

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

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

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



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

<mantissa> ::=                      <exact numeric literal>

<exponent> ::=                      <signed integer>

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

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

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

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

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

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

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

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

<empty set> ::=                     EMPTY

<minus sign> ::=                    -

<left paren> ::=                    (

<right paren> ::=                   )

<left bracket> ::=                  [

<right bracket> ::=                 ]

<period> ::=                        .

<plus sign> ::=                     +

<double quote> ::=                  "

<quote> ::=                         '

<comma>                             ,

<underscore> ::=                    _
```

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

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

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

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

7.2.2 BNF Productions for Two-Dimension Geometry WKT

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

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

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

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

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

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

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

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

<tin tagged text>                          tin <polyhedralsurface text>

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

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

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

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

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

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

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

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

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

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

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

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

7.2.3 BNF Productions for Three-Dimension Geometry WKT

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

7.2.4 BNF Productions for Two-Dimension Measured Geometry WKT

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

7.2.5 BNF Productions for Three-Dimension Measured Geometry WKT

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

7.2.6 Examples

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

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

Table 6: Example Well-known Text Representation of Geometry

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

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

8 Well-known Binary Representation for Geometry

8.1 Component overview

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

8.2 Component description

8.2.1 Introduction

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

8.2.2 Numeric type definitions

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

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

The above definitions are common to both XDR and NDR.

8.2.3 A common list of codes for geometric types

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

Table 7: Integer codes for geometric types

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

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

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

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

8.2.4 XDR (Big Endian) encoding of numeric types

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

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

8.2.5 NDR (Little Endian) encoding of numeric types

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

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

8.2.6 Conversions between the NDR and XDR representations of WKBGeometry

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

8.2.7 Relationship to other COM and CORBA data transfer protocols

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

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

8.2.8 Description of WKBGeometry representations

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

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

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

PointZ {
  double x;
  double y;
  double z}

PointM {
  double x;
  double y;
  double m}

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

LinearRing  {
  uint32  numPoints;
  Point   points[numPoints]}

LinearRingZ  {
  uint32  numPoints;
  PointZ  points[numPoints]}

LinearRingM  {
  uint32  numPoints;
  PointM  points[numPoints]}

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

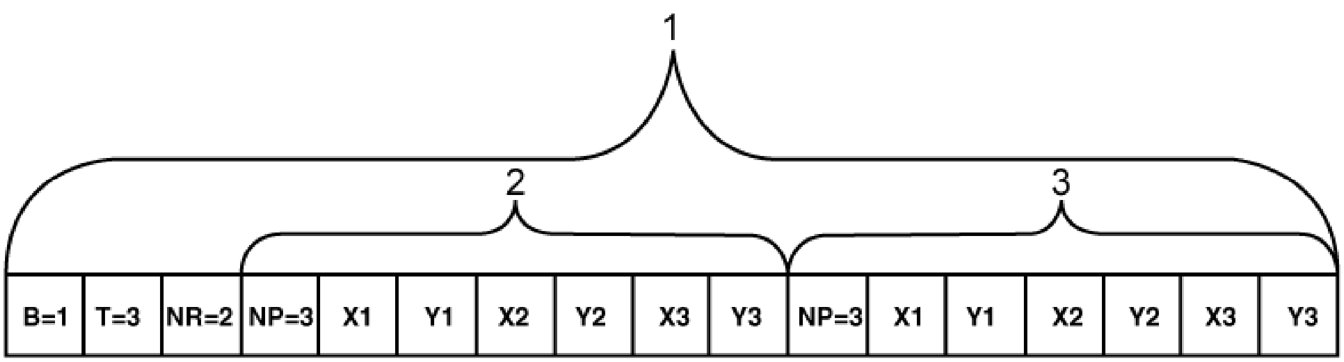
```
WKBPolyhedralSurfaceZM {
  byte               byteOrder;
  static  uint32     wkbType=3015;
  uint32             numPolygons;
  WKBPolygonZM       polygons[numPolygons]}

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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



Key

1 WKB Polygon

2 ring 1

3 ring 2

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

8.2.9 Assertions for Well-known Binary Representation for Geometry

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

9 Well-known Text Representation of Spatial Reference Systems

9.1 Component overview

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

9.2 Component description

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

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

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

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

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

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

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

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

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

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

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

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

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

<linear unit> ::=                     <unit>

<angular unit> ::=                    <unit>

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

<value> ::=                           <signed numeric literal>

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

<longitude> ::=                       <signed numeric literal>

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

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

<unit name> ::=                       <quoted name>

<spheroid name> ::=                   <quoted name>

<projection name> ::=                 <quoted name>

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

<parameter name> ::=                  <quoted name>

<datum name> ::=                      <quoted name>

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

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

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

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

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

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

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

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

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

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

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

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

Annex A
(informative)

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

A.1 Introduction

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

A.2 Geometry model

A.2.1 Geometry model of SFA-CA

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

A.2.2 Parts of geometry model of Spatial schema

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

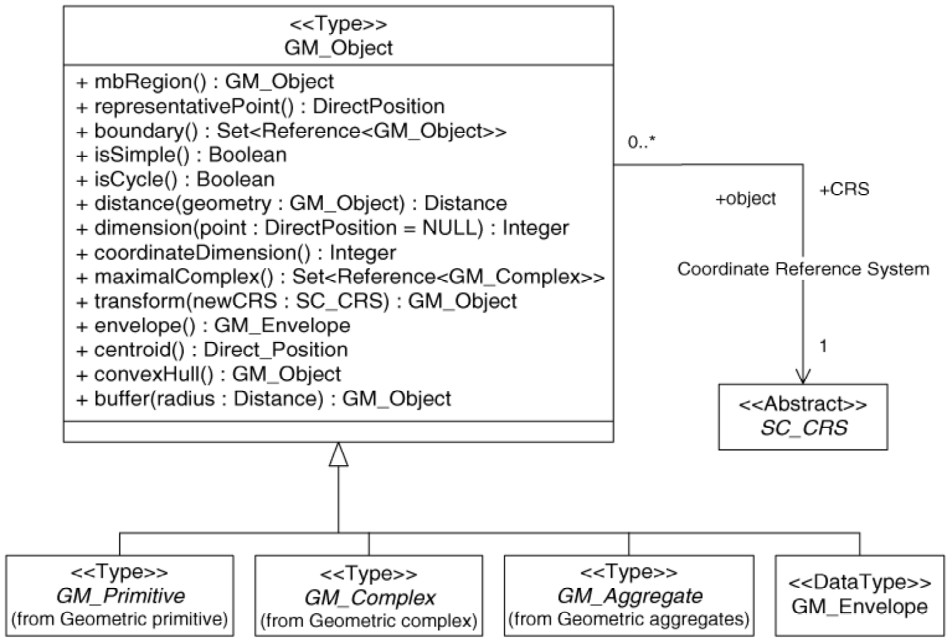


Figure A.1: The root type and subordinates of the Spatial schema

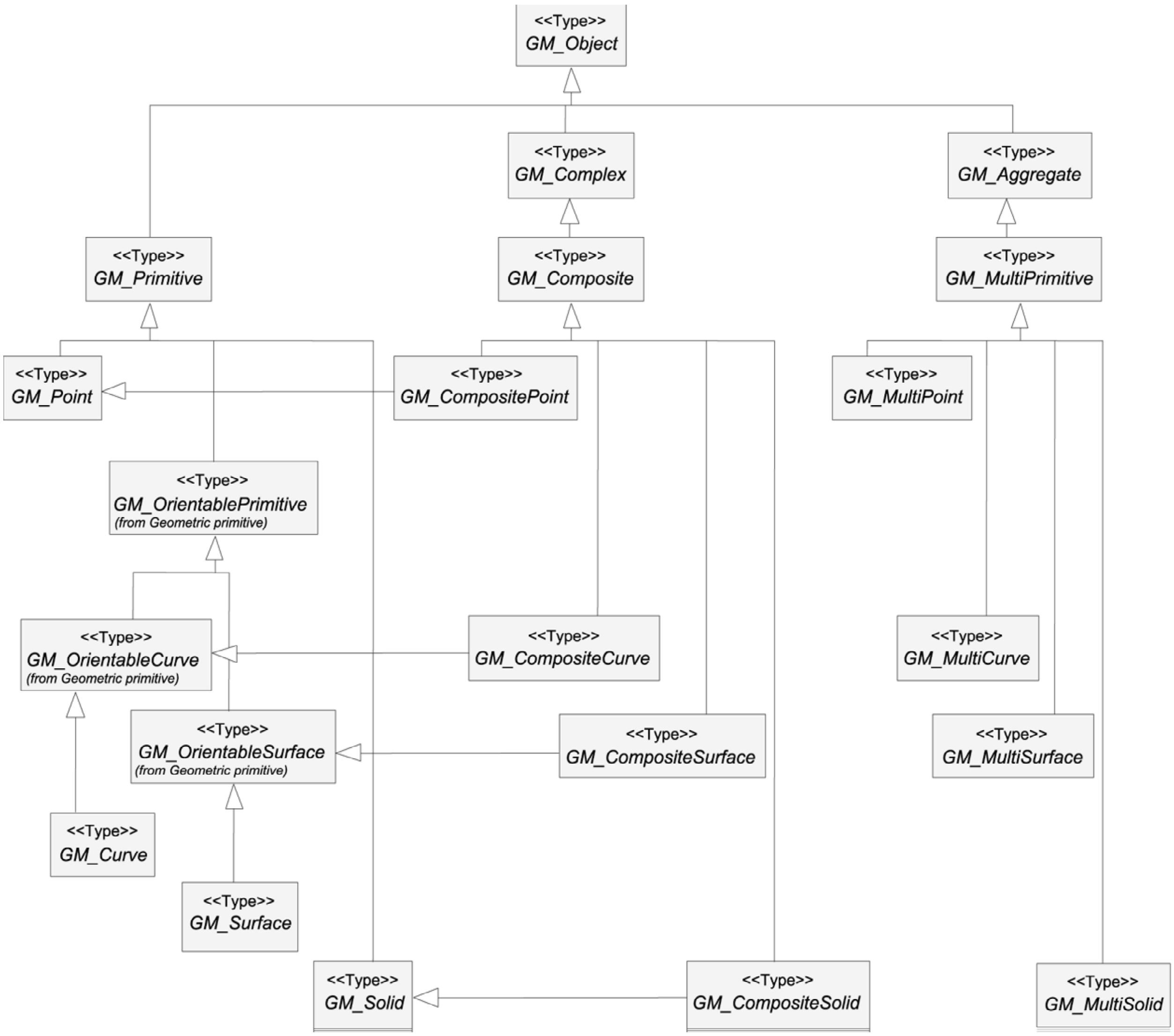


Figure A.2: The GM_Object hierarchy

A.3 Correspondence

A.3.1 Overview

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

A.3.2 Geometry type

A.3.2.1 Overview

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

A.3.2.2 Basic methods on geometry

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

A.3.2.3 Methods for testing spatial relations between geometric objects

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

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

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

A.3.2.4 Methods that support spatial analysis

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

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

A.3.3 “Atomic” subtypes of the Geometry type

A.3.3.1 Overview

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

A.3.3.2 Point

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

A.3.3.3 Curve

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

A.3.3.4 LineString

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

A.3.3.5 LinearRing and LineSegment

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

A.3.3.6 Surface

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

A.3.3.7 Polygon

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

A.3.3.8 PolyhedralSurface

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

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

A.3.4 Collection subtypes of the Geometry type

A.3.4.1 Overview

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

A.3.4.2 GeometryCollection

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

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

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

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

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

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

A.3.4.3 MultiPoint

SFA-CA	Spatial schema	Comment
MultiPoint	GM_MultiPoint	—

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

A.3.4.4 MultiLineString

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

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

A.3.4.5 MultiPolygon

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

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

Annex B
(informative)

Supported spatial reference data

B.1 Purpose of this annex

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

B.2 Linear units

Table B - 1 — Linear units

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

B.3 Angular units

Table B - 2 — Angular units

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

Gon	π/200
Grad	π/200

B.4 Ellipsoids and spheres

Table B - 3 — Ellipsoids and spheres

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

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

B.5 Geodetic datums

Table B - 4— Geodetic datums

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

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

B.6 Prime meridians

Table B - 5 — Prime meridians

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

B.7 Map projections

Table B - 6 — Map projections

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

B.8 Map projection parameters

Table B - 7 — Map projection parameters

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

Bibliography

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

