INTERNATIONAL STANDARD

ISO 19125-1

First edition
2004-08-01

Corrected version
2004-11-01

Geographic information — Simple feature access —

Part 1: Common architecture

Information géographique — Accès aux entités simples —

Partie 1: Architecture commune

Reference number
ISO 19125-1:2004(E)

ISO copyright office
Case postale 56 • CH-1211 Geneva 20
Tel. + 41 22 749 01 11
Fax + 41 22 749 09 47
E-mail copyright@iso.org
Web www.iso.org

Published in Switzerland

Contents

Page

Foreword

ISO (the International Organization for Standardization) is a worldwide federation of national standards bodies (ISO member bodies). The work of preparing International Standards is normally carried out through ISO technical committees. Each member body interested in a subject for which a technical committee has been established has the right to be represented on that committee. International organizations, governmental and non-governmental, in liaison with ISO, also take part in the work. ISO collaborates closely with the International Electrotechnical Commission (IEC) on all matters of electrotechnical standardization.

International Standards are drafted in accordance with the rules given in the ISO/IEC Directives, Part 2.

The main task of technical committees is to prepare International Standards. Draft International Standards adopted by the technical committees are circulated to the member bodies for voting. Publication as an International Standard requires approval by at least 75 % of the member bodies casting a vote.

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

ISO 19125-1 was prepared by Technical Committee ISO/TC 211, *Geographic information/Geomatics* from a base document supplied by the Open GIS Consortium, Inc.

ISO 19125 consists of the following parts, under the general title *Geographic information — Simple feature access*:

— *Part 1: Common architecture*

— *Part 2: SQL option*

This corrected version of ISO 19125-1:2004 incorporates the following corrections:

— a complete version of Figure 9, which was truncated in the original;

— removal from the Foreword of the reference to ISO 19125-3, which has now been deleted.

Introduction

This part of 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 modelling 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 of ISO 19125. There may also be methods used in this International Standard that differ from one part to another. Where this is the case, the differences are shown within the part.

This part of ISO 19125 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 ISO 19125 with the schema described in ISO 19107:2003.

Geographic information — Simple feature access —

Part 1:
Common architecture

1 Scope

This part of ISO 19125 establishes a common architecture and defines terms to use within the architecture.

This part of ISO 19125 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 part of ISO 19125 does standardize names and geometric definitions for Types for Geometry.

This part of ISO 19125 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 part of ISO 19125, 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.

ISO 19107:2003, *Geographic information — Spatial schema*

ISO 19111:2003, *Geographic information — Spatial referencing by coordinates*

4 Terms and definitions

For the purposes of this document, 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 must 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 **point** (4.17)

[ISO 19111]

4.7
curve
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.

[ISO 19107]

4.7
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)

[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.

[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.

[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.

[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.

[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 homomorphic 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.

[ISO 19107]

4.17
point
0-dimensional **geometric primitive** (4.15), representing a position

NOTE The boundary of a point is the empty set.

[ISO 19107]

4.18
simple feature
feature (4.11) restricted to 2D geometry with linear interpolation between vertices, having both spatial and non spatial attributes

4.19
start point
first **point** (4.17) of a **curve** (4.7)

[ISO 19107]

4.20
surface
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 Abbreviated terms

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
IEEE	Institute of Electrical and Electronics Engineers, Inc.
NDR	Little Endian byte order encoding
OLE	Object Linking and Embedding
RPC	Remote Procedure Call
SQL	Structured Query Language

SRID	Spatial Reference System Identifier
XDR	Big Endian byte order encoding
UDT	User Defined Type
UML	Unified Modeling Language
WKB	Well-Known Binary (representation for example, geometry)

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.

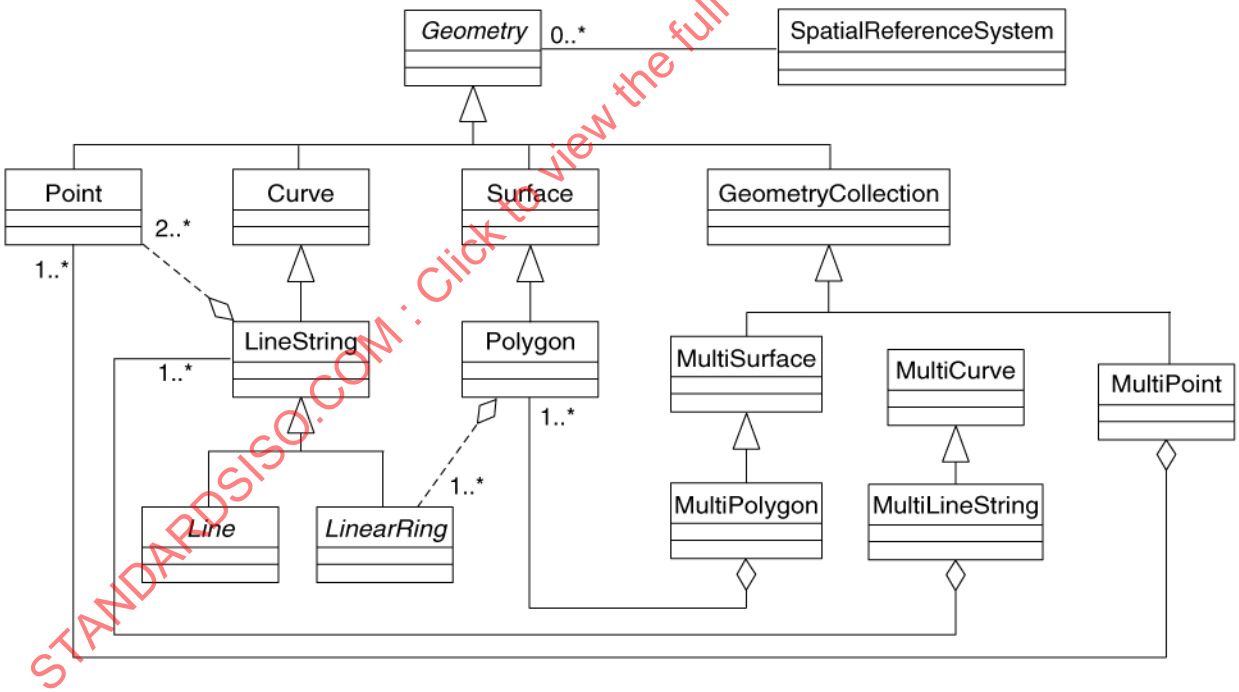


Figure 1 — Geometry class hierarchy

Figure 1 is based on an extended Geometry model with specialized 0-, 1- and 2-dimensional collection classes named MultiPoint, MultiLineString and MultiPolygon for modelling 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. 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.

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 International Standard are restricted to 0, 1 and 2-dimensional geometric objects that exist in 2-dimensional coordinate space ($\Re^2$).

All instantiable Geometry classes described in this part of ISO 19125 are defined so that valid instances of a Geometry class are topologically closed, i.e. all defined geometries include their boundary.

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. This specification is restricted to geometries in 2-dimensional coordinate space.

— **GeometryType** ( ):String — Returns the name of the instantiable subtype of Geometry of which *this* geometric object is a 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.

— **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)].

— **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, $\varnothing$, for the coordinate space.

— **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.

— **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).

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 subtypes of 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 as specified by the values in the intersectionPatternMatrix.

6.1.2.4 Methods that support spatial analysis

— **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.

— **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.

— **ConvexHull**( ):Geometry — Returns a geometric object that represents the convex hull of *this* geometric object.

— **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.

6.1.3 GeometryCollection

6.1.3.1 Description

A GeometryCollection is a geometric object that is a collection of 1 or more geometric objects.

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

6.1.3.2 Methods

— **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 and a y-coordinate value.

The boundary of a Point is the empty set.

6.1.4.2 Methods

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

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

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.

A MultiPoint is simple if no two Points in the MultiPoint are equal (have identical coordinate values).

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 part of ISO 19125 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 $D = [a, b] = \{x \in \Re \mid a \leqslant x \leqslant b\}$ under a mapping $f:[a,b] \rightarrow \Re^2$.

A Curve is simple if it does not pass through the same Point twice (Reference [1], section 3.12.7.3):

$\forall$ c $\in$ Curve, [a, b] = c.Domain,

c.IsSimple$\Leftrightarrow$ ($\forall$ x1, x2 $\in$ (a, b] x1 $\neq$ x2 $\Rightarrow$ f(x1) $\neq$ f (x2)) $\wedge$ ($\forall$ x1, x2 $\in$ [a, b) x1 $\neq$ x2 $\Rightarrow$ f(x1) $\neq$ f(x2))

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

The boundary of a closed Curve is empty.

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.

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).

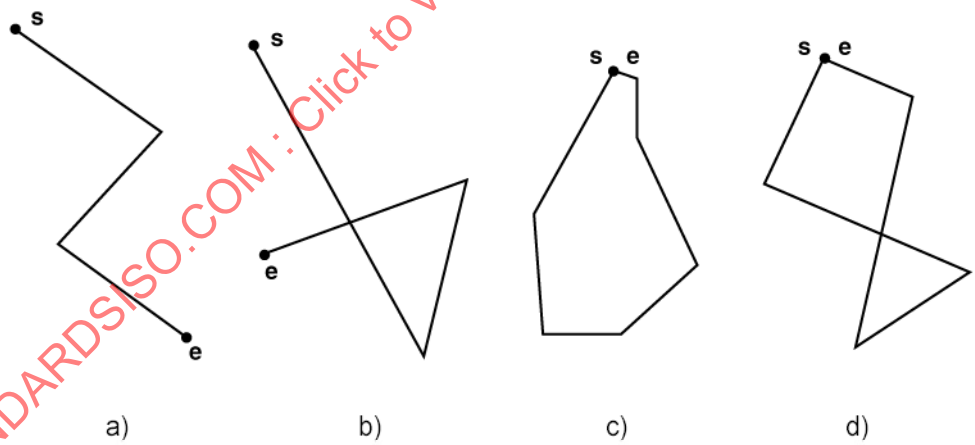
6.1.7 LineString, Line, LinearRing

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 2 — Examples of LineStrings — Simple LineString (a), Non-simple LineString (b), Simple, closed LineString (a LinearRing) (c), Non-simple closed LineString (d)

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 specification; 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.

6.1.8.2 Methods

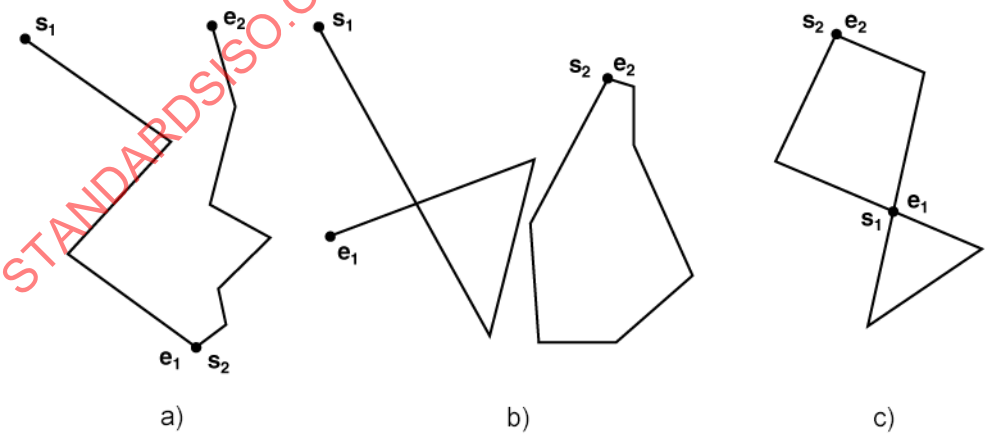
— **IsClosed**( ):Integer — Returns 1 (TRUE) if *this* MultiCurve is closed [StartPoint ( ) = EndPoint ( ) for each Curve in *this* MultiCurve].

— **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 3 are (a)—{s1, e2}, (b)— {s1, e1}, (c)—∅.



Key

s start
e end

Figure 3 — Examples of MultiLineStrings — 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 consists of a single “patch” that is associated with one “exterior boundary” and 0 or more “interior” boundaries. Simple Surfaces in 3-dimensional space are isomorphic to planar Surfaces. Polyhedral Surfaces are formed by “stitching” together simple Surfaces along their boundaries, polyhedral Surfaces in 3-dimensional space may not be planar as a whole (Reference [1], sections 3.12.9.1, 3.12.9.3).

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 subclass of Surface defined in this specification, Polygon, is a simple Surface that is planar.

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

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.

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. $\forall P \in Polygon,\ \forall c1, c2 \in P.Boundary(),\ c1 \neq c2,\ \forall p, q \in Point,\ p, q \in c1,\ p \neq q\ , [p \in c2 \Rightarrow q \notin c2]$;

d) A Polygon may not have cut lines, spikes or punctures e.g.: $\forall P \in Polygon,\ P = Closure(Interior(P))$;

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) make a Polygon a regular closed Point set.

Polygons are simple geometric objects.

Figure 4 shows some examples of Polygons.

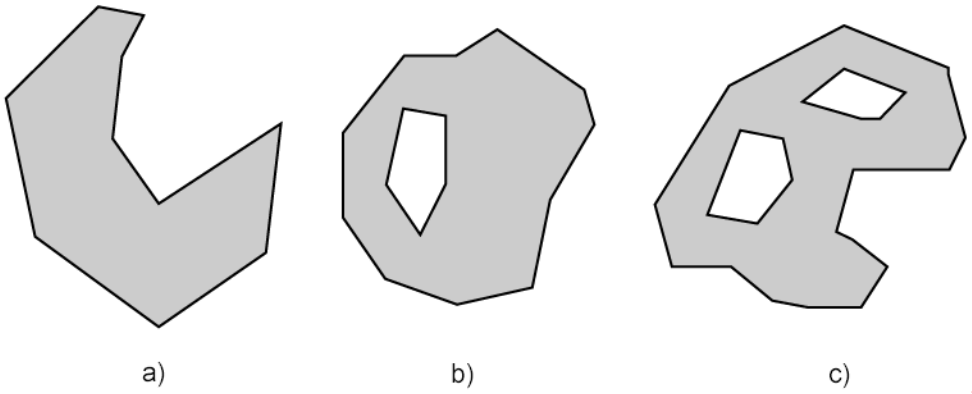


Figure 4 — Examples of Polygons with 1 (a), 2 (b) and 3 (c) Rings, respectively

Figure 5 shows some examples of geometric objects that violate the above assertions and are not representable as single instances of Polygon.

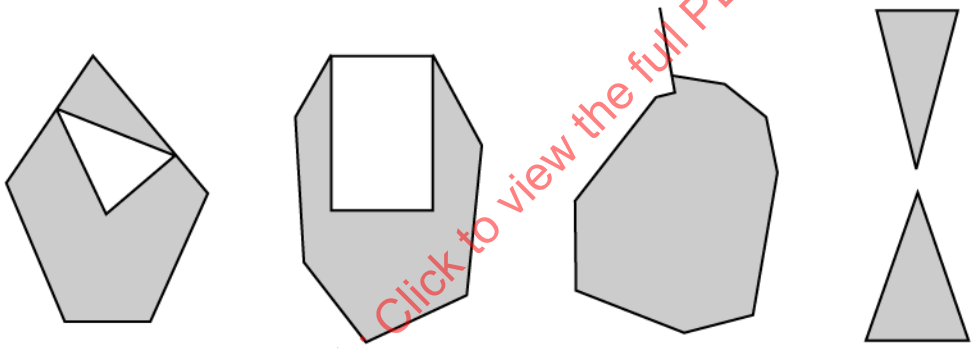


Figure 5 — Examples of objects not representable as a single instance of Polygon

6.1.11.2 Methods

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

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

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

6.1.12 MultiSurface

6.1.12.1 Description

A MultiSurface is a 2-dimensional GeometryCollection whose elements are Surfaces. The interiors of any two Surfaces in a MultiSurface may not intersect. The boundaries of any two elements in a MultiSurface may intersect, at most, at a finite number of Points.

MultiSurface is a non-instantiable class in this International Standard. It defines a set of methods for its subclasses and is included for reasons of extensibility. The instantiable subclass of MultiSurface is MultiPolygon, corresponding to a collection of Polygons.

6.1.12.2 Methods

— **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.

6.1.13 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 $\in$ MultiPolygon, $\forall$ Pi, Pj $\in$ M.Geometries(), i$\neq$j, Interior(Pi) $\cap$ Interior(Pj) = $\varnothing$;

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 $\in$ MultiPolygon, $\forall$ Pi, Pj $\in$ M.Geometries(), $\forall$ ci $\in$ Pi.Boundaries(), cj $\in$ Pj.Boundaries()
ci $\cap$ cj = {p1, ….., pk | pi $\in$ Point, 1 $\leqslant$ i $\leqslant$ 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 $\in$ MultiPolygon, M = Closure(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 6 shows four examples of valid MultiPolygons with 1, 3, 2 and 2 Polygon elements, respectively.

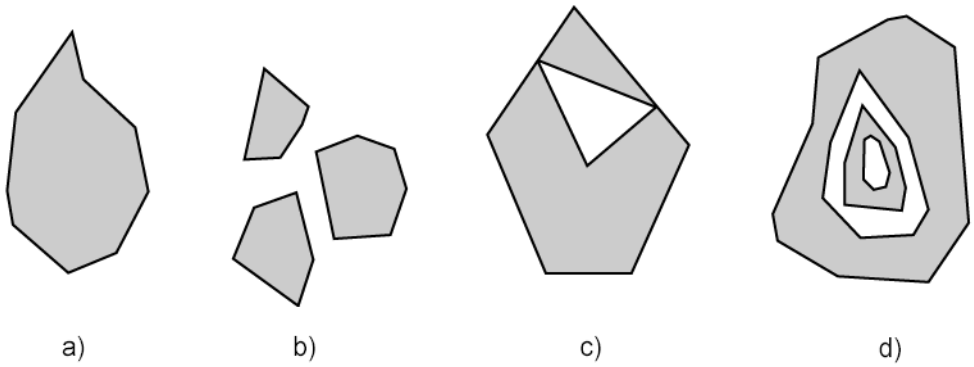


Figure 6 — Examples of MultiPolygons with 1 (a), 3 (b) , 2 (c) and 2 (d) Polygon elements

Figure 7 shows examples of geometric objects not representable as single instances of MultiPolygons.

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

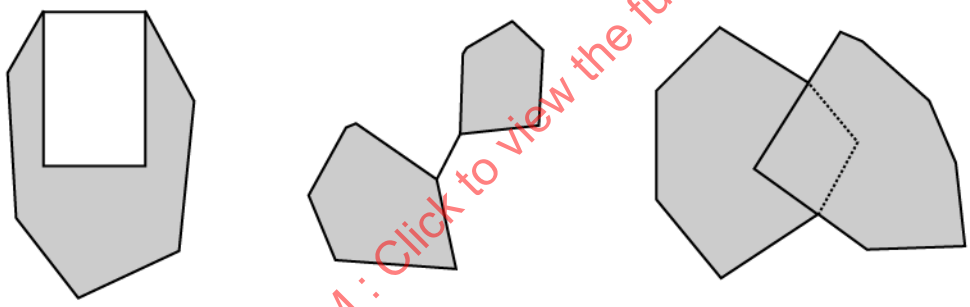


Figure 7 — Geometric objects not representable as a single instance of a MultiPolygon

6.1.14 Relational operators

6.1.14.1 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. Topological spatial relationships between two geometric objects have been a topic of extensive study; see References [4, 5, 6, 7, 8, 9, 10]. The basic approach to comparing two geometric objects is to make pair-wise tests of the intersections between the interiors, boundaries and exteriors of the two geometric objects and to classify the relationship between the two geometric objects based on the entries in the resulting 'intersection' matrix.

The concepts of interior, boundary and exterior are well defined in general topology; see Reference [4]. These concepts can be applied in defining spatial relationships between 2-dimensional objects in 2-dimensional space ($\Re^2$). 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 (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].

6.1.14.2 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 $dim(\varnothing)$.

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.

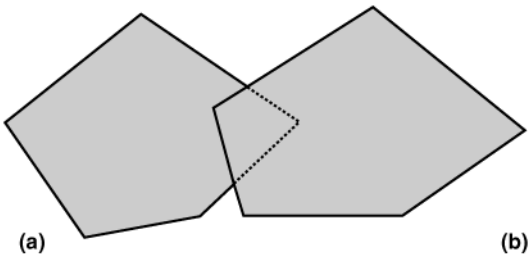
Table 1 shows the general form of the dimensionally extended nine-intersection matrix (DE-9IM).

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 8 — An example instance and its DE-9IM

A spatial relationship predicate can be formulated on two geometric objects 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 *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:

$p = T \geqslant \dim(x) \in \{0, 1, 2\}$, i.e. $x \neq \varnothing$

$p = F \geqslant \dim(x) = -1$, i.e. $x = \varnothing$

$p = * \geqslant \dim(x) \in \{-1, 0, 1, 2\}$, i.e. Don't Care

$p = 0 \geqslant \dim(x) = 0$

$p = 1 \geqslant \dim(x) = 1$

$p = 2 \geqslant \dim(x) = 2$

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.

A spatial relationship predicate can be formulated on two geometric objects 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 *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:

$p = T \geqslant \dim(x) \in \{0, 1, 2\}$, i.e. $x \neq \varnothing$

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.

A spatial relationship predicate can be formulated on two geometric objects 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 *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);
```

6.1.14.3 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).

Disjoint

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

a.Disjoint(b) $\Leftrightarrow$ a $\cap$ b = $\varnothing$

Expressed in terms of the DE-9IM:

a.Disjoint(b) $\Leftrightarrow$ (I(a) $\cap$ I(b) = $\varnothing$) $\wedge$ (I(a)$\cap$B(b) = $\varnothing$) $\wedge$ (B(a) $\cap$ I(b) = $\varnothing$) $\wedge$ (B(a)$\cap$B(b) = $\varnothing$)
$\Leftrightarrow$ 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) $\Leftrightarrow$ (I(a) $\cap$ I(b) = $\varnothing$) $\wedge$ (a $\cap$ b) $\neq$ $\varnothing$

Expressed in terms of the DE-9IM:

a.Touch(b) $\Leftrightarrow$ (I(a) $\cap$ I(b) = $\varnothing$) $\wedge$ ((B(a) $\cap$ I(b) $\neq$ $\varnothing$) $\vee$ (I(a) $\cap$ B(b) $\neq$ $\varnothing$) $\vee$ (B(a) $\cap$ B(b) $\neq$ $\varnothing$))
$\Leftrightarrow$ a.Relate(b, “FT*******”) $\vee$ a.Relate(b, “F**T*****”) $\vee$ a.Relate(b, “F***T****”)

Figure 9 shows some examples of the Touches relationship.

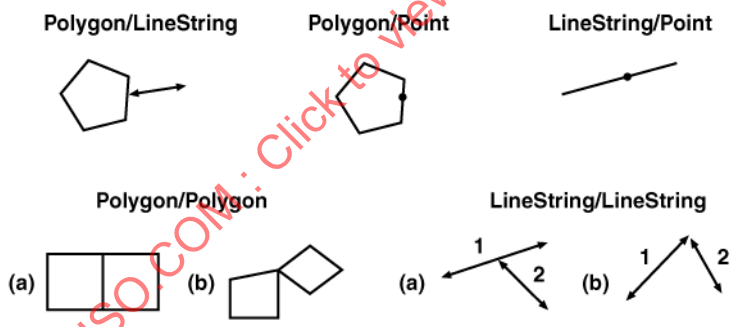


Figure 9 — Examples of the Touches relationship

Crosses

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

a.Cross(b) $\Leftrightarrow$ (dim(I(a) $\cap$ I(b) < max(dim(I(a)), dim(I(b)))) $\wedge$ (a $\cap$ b $\neq$a) $\wedge$ (a $\cap$ b $\neq$b)

Expressed in terms of the DE-9IM:

Case *a* $\in$ *P, b* $\in$ *L* or Case *a* $\in$*P, b* $\in$ *A* or Case *a* $\in$ *L, b* $\in$ *A*:

a.Cross(b) $\Leftrightarrow$ (I(a) $\cap$ I(b) $\neq$ $\varnothing$) $\wedge$ (I(a) $\cap$ E(b) $\neq\varnothing$) $\Leftrightarrow$ a.Relate(b, “T*T******”)

Case *a* $\in$ *L, b* $\in$ *L*:

a.Cross(b) $\Leftrightarrow$ dim(I(a) $\cap$ I(b)) = 0 $\Leftrightarrow$ a.Relate(b, “0********”);

Figure 10 shows some examples of the Crosses relationship.

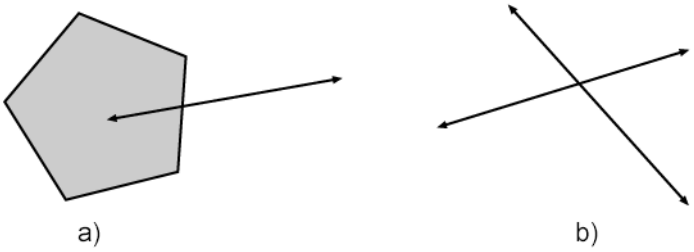


Figure 10 — Examples of the Crosses relationship — Polygon/LineString (a) and LineString/LineString (b)

Within

The Within relationship is defined as

$$a.\text{Within}(b) \Leftrightarrow (a \cap b = a) \wedge (I(a) \cap E(b) \neq \varnothing)$$

Expressed in terms of the DE-9IM:

$$a.\text{Within}(b) \Leftrightarrow (I(a) \cap I(b) \neq \varnothing) \wedge (I(a) \cap E(b) = \varnothing) \wedge (B(a) \cap E(b) = \varnothing)) \Leftrightarrow a.\text{Relate}(b, \text{“TF*F*****”})$$

Figure 11 shows some examples of the Within relationship.

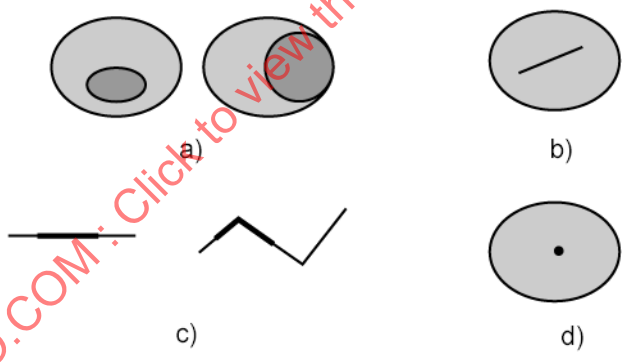


Figure 11 — Examples of the Within relationship — Polygon/Polygon (a), Polygon/LineString (b), LineString/LineString (c), and Polygon/Point (d)

Overlaps

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

It is defined as

$$a.\text{Overlaps}(b) \Leftrightarrow (\dim(I(a)) = \dim(I(b)) = \dim(I(a) \cap I(b))) \wedge (a \cap b \neq a) \wedge (a \cap b \neq b)$$

Expressed in terms of the DE-9IM:

Case $a \in P$, $b \in P$ or Case $a \in A$, $b \in A$:

$$a.\text{Overlaps}(b) \Leftrightarrow (I(a) \cap I(b) \neq \varnothing) \wedge (I(a) \cap E(b) \neq \varnothing) \wedge (E(a) \cap I(b) \neq \varnothing) \Leftrightarrow a.\text{Relate}(b, \text{“T*T***T**”})$$

Case $a \in L$, $b \in L$:

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.

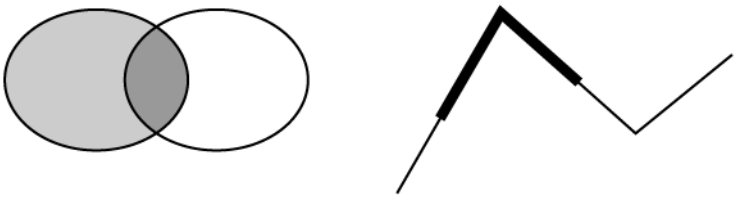


Figure 12 — Examples of the Overlaps relationship — Polygon/LineString (a) and LineString/LineString (b)

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 Well-known Text Representation for Geometry

6.2.1 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.

6.2.2 Language constructs

The Well-known Text Representation of Geometry is defined below; the notation {}* denotes 0 or more repetitions of the tokens within the braces; the braces do not appear in the output token list. The text representation of the instantiable Geometry Types implemented shall conform to this grammar.

```
<Geometry Tagged Text> :=
            <Point Tagged Text>
          | <LineString Tagged Text>
          | <Polygon 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>
<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 | ( <Point> )
<Point> := <x> <y>
<x> := double precision literal
<y> := double precision literal
<LineString Text> := EMPTY
          | ( <Point >  {, <Point > }* )
<Polygon Text> := EMPTY
          | ( <LineString Text >  {, < LineString Text > }*)
<Multipoint Text> := EMPTY
          | ( <Point Text >  {, <Point Text > }* )
<MultiLineString Text> := EMPTY
          | ( <LineString Text >  {, < LineString Text > }* )
<MultiPolygon Text> := EMPTY
          | ( < Polygon Text >  {, < Polygon Text > }* )
<GeometryCollection Text> := EMPTY
          | ( <Geometry Tagged Text> {, <Geometry Tagged Text> }* )
```

The above grammar 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.

6.2.3 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.

Table 2 — 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

6.3 Well-known Binary Representation for Geometry

6.3.1 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.

6.3.2 Component description

6.3.2.1 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, the NDR encoding is Little Endian.

6.3.2.2 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 data type that encodes a double precision number using the IEEE 754[18] double precision format.

The above definitions are common to both XDR and NDR.

6.3.2.3 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).

6.3.2.4 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).

6.3.2.5 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.

6.3.2.6 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].

6.3.2.7 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 two Double numbers. The representations for other geometric objects are built using the representations for geometric objects that have already been defined.

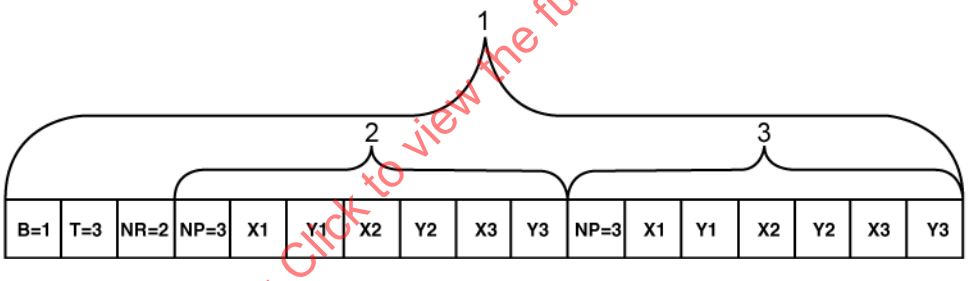
```
// Basic Type definitions
// byte : 1 byte
// uint32 : 32 bit unsigned integer (4 bytes)
// double : double precision number (8 bytes)

// Building Blocks : Point, LinearRing
Point {
        double x;
        double y;
};
LinearRing  {
        uint32  numPoints;
        Point   points[numPoints];
}
```

```
enum wkbGeometryType {
        wkbPoint = 1,
        wkbLineString = 2,
        wkbPolygon = 3,
        wkbMultiPoint = 4,
        wkbMultiLineString = 5,
        wkbMultiPolygon = 6,
        wkbGeometryCollection = 7
};
enum wkbByteOrder {
        wkbXDR = 0,            // Big Endian
        wkbNDR = 1             // Little Endian
};
WKBPoint {
        byte       byteOrder;
        uint32     wkbType;                          // 1
        Point      point;
}
WKBLineString {
        byte       byteOrder;
        uint32     wkbType;                          // 2
        uint32     numPoints;
        Point      points[numPoints];
}
WKBPolygon {
        byte       byteOrder;
        uint32     wkbType;                          // 3
        uint32     numRings;
        LinearRing     rings[numRings];
}
WKBMultiPoint {
        byte       byteOrder;
        uint32     wkbType;                      // 4
        uint32     num_wkbPoints;
        WKBPoint   WKBPoints[num_wkbPoints];
}
WKBMultiLineString {
        byte       byteOrder;
        uint32     wkbType;                          // 5
        uint32     num_wkbLineStrings;
        WKBLineString  WKBLineStrings[num_wkbLineStrings];
}
wkbMultiPolygon {
        byte       byteOrder;
        uint32     wkbType;                          // 6
        uint32     num_wkbPolygons;
        WKBPolygon     wkbPolygons[num_wkbPolygons];
}
```

```
WKBGeometry {
        union {
            WKBPoint                point;
            WKBLineString           linestring;
            WKBPolygon              polygon;
            WKBGeometryCollection   collection;
            WKBMultiPoint           mpoint;
            WKBMultiLineString      mlinestring;
            WKBMultiPolygon         mpolygon;
        }
};
WKBGeometryCollection {
        byte        byte_order;
        uint32      wkbType;                        // 7
        uint32      num_wkbGeometries;
        WKBGeometry     wkbGeometries[num_wkbGeometries];
}
```

Figure 13 shows a pictorial representation of the Well-known Representation for a `Polygon` with one outerRing and one innerRing.



Key

1 WKB Polygon
2 ring 1
3 ring 2

Figure 13 — 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)

6.3.2.8 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).

6.4 Well-known Text Representation of Spatial Reference Systems

6.4.1 Component overview

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

6.4.2 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.

```
<coordinate system> = <projected cs> | <geographic cs> | <geocentric cs>
<projected cs> = PROJCS["<name>", <geographic cs>, <projection>, {<parameter>,}* <linear unit>]
<projection> = PROJECTION["<name>"]
<parameter> = PARAMETER["<name>", <value>]
<value> = <number>
```

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.

```
<geographic cs> = GEOGCS["<name>", <datum>, <prime meridian>, <angular unit>]
<datum> = DATUM["<name>", <ellipsoid>]
<ellipsoid> = ELLIPSOID["<name>", <semi-major axis>, <inverse flattening>]
<semi-major axis> = <number>   NOTE: semi-major axis is measured in meters and must be > 0.
<inverse flattening> = <number>
<prime meridian> = PRIMEM["<name>", <longitude>]
<longitude> = <number>
<angular unit> = <unit>
<linear unit> = <unit>
<unit> = UNIT["<name>", <conversion factor>]
<conversion factor> = <number>
```

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.

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

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

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

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

A geocentric coordinate system is similar to a geographic coordinate system. It is represented by

```
<geocentric cs> = GEOCCS["<name>", <datum>, <prime meridian>, <linear unit>]
```

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 International 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.

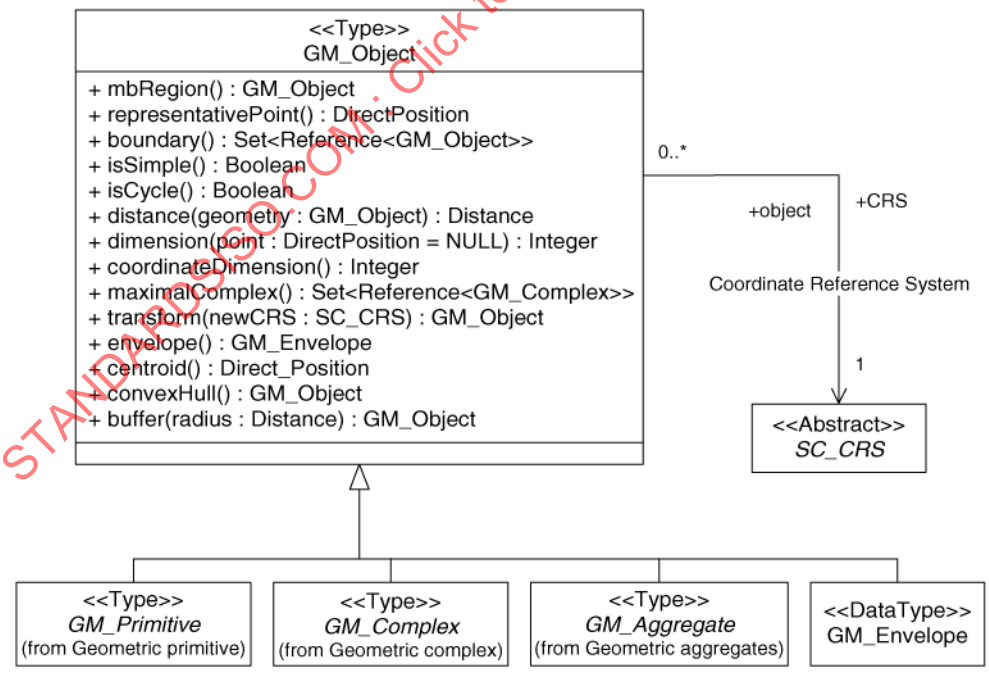


Figure A.1 — The root type and subordinates of the Spatial schema

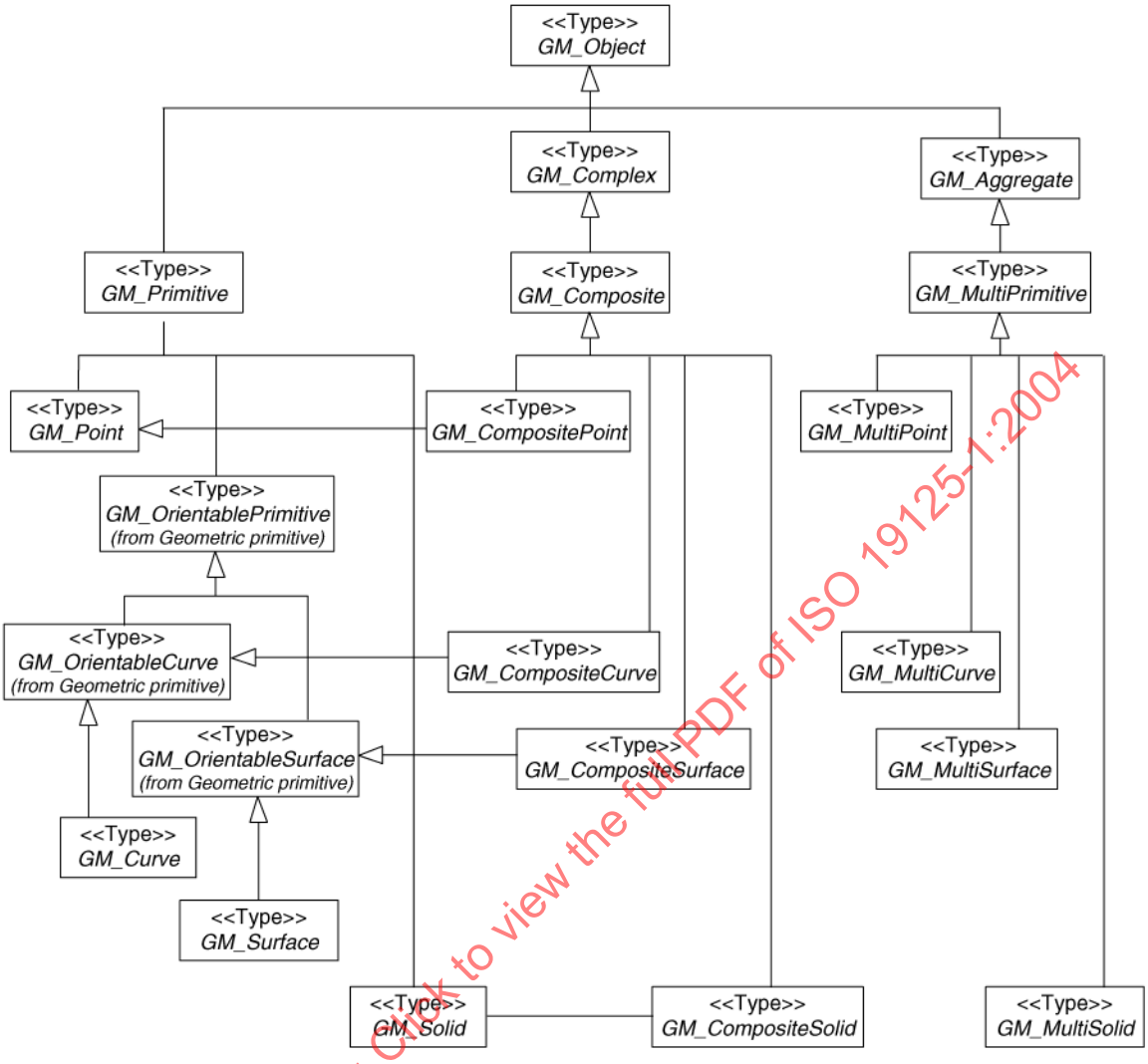


Figure A.2 — The GM_Object hierarchy

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 specification with systems based solely on the Spatial schema specification must 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 International 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	*Not defined*	Defined by an application schema
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 behavioural properties of the GM_Object. Still, the GM_Object inherits spatial relations from the interface type TransfiniteSet.

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 two ordinates, i.e., the 2D coordinate 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	*Not defined*	Given by startPoint() = endPoint(); may be similar as the GM_Object::isSimple:Boolean
Curve.IsRing( ):Integer	*Not defined*	Given by both closed and simple properties, but may be similar to the GM_Object::isCycle:Boolean

A.3.3.4 LineString

SFA-CA	Spatial schema	Comment
LineString	GM_LineString	—
LinearString.NumPoints( ):Integer	*Not defined*	May be calculated
LinearString.PointN(N:Integer):Point	*Not defined*	May be derived

A.3.3.5 LinearRing and Line

These two types are only derived types 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 specification 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.3.4 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 GM_MultiPrimitive	—
GeometryCollection:: NumGeometries( ):Integer	*Not defined*	May be calculated, e.g. from the elements attribute of GM_Aggregate
GeometryCollection:: GeometryN(N:Integer):Geometry	*Not defined*	May be calculated, e.g. from the elements attribute of GM_Aggregate

The subtypes of GeometryCollections, to be presented next, must 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 must 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.