JTS TOPOLOGY SUITE
TEST PLAN (VERSION 1.1)

Prepared by:

Base Mapping and Geomatics Services Branch
BC Ministry of Sustainable Resource Management

VIVID
S O L U T I O N S

April 23, 2003

Document Change Control

REVISION NUMBER	DATE OF ISSUE	AUTHOR(S)	BRIEF DESCRIPTION OF CHANGE
1.0	17-Dec-01	Test Team *Base Mapping and Geomatics* *Vivid Solutions Inc.*	Original Draft

Table of Contents

ACRONYMS AND ABBREVIATIONS

API   Application Programming Interface

DE-9IM   Dimensionally Extended 9 Intersection Model

GDBC   Geographic Data BC (currently the Base Mapping and Geomatics Branch, BC
                Ministry of Sustainable Resource Management)

JTS   JTS Topology Suite

OGC   OpenGIS Consortium

WKT   Well-Known Text

1. INTRODUCTION

1.1 BACKGROUND

The JTS Topology Suite is a Java API that implements a core set of spatial data operations using an explicit precision model and robust geometric algorithms. JTS is also compliant with Open GIS Consortium standards and specifications (OGC 1999, 1996). JTS is intended to be used in the development of applications that support the validation, cleaning, integration and querying of spatial datasets.

1.2 PURPOSE

This document describes the scope of JTS testing, the approach to be taken, required resources (including people and testing configurations), description, compilation of test cases and documentation requirements.

1.3 SCOPE

The test plan only covers unit testing of JTS functions and/or classes for binary predicates and spatial analysis (Table 1-1). There is no specific testing on Well-Known Text (WKT) input/output. However, they are tested every time a geometry in WKT is loaded into a test harness and when there is an output in WKT.

Table 1-1– JTS Components

COMPONENT	VERSION
Basic geometric algorithms and structures	-
Predicate functions	0.6
Spatial analysis operations	0.6
Well-Known Text input/output	0.6

Rigorous testing and validation of basic geometric algorithms and structure are not conducted in this test. The current version of JTS assumes that the input geometries are valid. Invalid geometry may not always be recognised or rejected although it's most likely that JTS would generate an error. Therefore, there are no test cases involving unrepresentable geometries and repeated points in a geometry, e.g., LINESTRING (10 10, 10 10, 20 20, 20 20).

1.4 APPROACH

This plan complies with the standard test cycle of plan, test, document results, log defects, fix defects, and retest until acceptance is reached. Acceptance signifies that JTS meets the specifications as outlined in the JTS RFP and JTS Technical Specifications.

1.5 TEST ENVIRONMENT

The test environment is PC based, typically configured with:

1.   Hardware: Pentium III 355 MHz or faster, with 128 MB RAM or more
2.   Operating systems: Windows NT 4.0 or Windows 2000
3.   Java development environment: JBuilder Foundation 4.0
4.   Java runtime environment: JDK1.3

A test harness was developed by Vivid to facilitate the compilation of test cases, the testing, and the visualisation of test results.

1.6 TEST PLANNING

The first stage of the lifecycle is to identify the approach, logistics, test procedures, and complete the test plan.

The second stage is to generate test cases and to test JTS. Since it is difficult to estimate the minimum number of test cases for testing JTS, a taxonomy is presented in Chapter 2 to assist the identification of geometric components and the organisation of test cases.
A list of potential cases is automatically generated based on the taxonomy and is used as a guide to compile real test cases for testing specifically binary predicates with synthetic data.

Different tests and test data are described in Chapter 3. Chapter 4 defines forms to be used during testing.

1.6.1 Roles and Responsibilities

Table 1-2 – Roles and Responsibilities

ROLE	RESPONSIBILITY
Document test plan	Base Mapping and Geomatics (BMG)
Review test plan	Base Mapping and Geomatics, Vivid Solutions Inc.
Establish test environment	Vivid Solutions Inc.
Create guide/taxonomy of test cases	Base Mapping and Geomatics, Vivid Solutions Inc.
Enter test data	Base Mapping and Geomatics, Vivid Solutions Inc.
Carry out tests	Base Mapping and Geomatics, Vivid Solutions Inc., GeoConnections
Publish test results	Vivid Solutions Inc.

1.6.2 Deliverables

1.   A draft Test Plan shall be delivered to both BMG and Vivid for review and acceptance;
2.   A taxonomy as a guide for generating and compiling test cases shall be created;
3.   A test harness shall be developed to facilitate the test;

4.   All test data shall be compiled and kept as XML files; and
5.   Selected test results shall be published on web site.

1.7 TEST EXECUTION

A test readiness review meeting shall be held to determine if

1.   The systems are installed and configured on the designated test platforms.
2.   All test data has been compiled.
3.   There are no known problems preventing the start of testing.
4.   The test team is ready.

Each time a test case is generated, it will be tested in the test harness. Alternatively, each time a new version of JTS arrived, all compiled test cases will be tested again. Defects are reported immediately to Vivid for fixes. Upon receipt of bug fixes, the appropriate test engineer will run affected test cases again.

Acceptance of JTS will be completed when the JTS Project Manager and Vivid have signed the certificate of acceptance.

1.7.1 Roles and Responsibilities

ROLE	HANDLED BY	RESPONSIBILITY
JTS Project Manager	Mark Sondheim	Review and approve Test Plan and Certificate of Acceptance, issue resolution.
Vivid Solutions Inc. Project Manager	David Ash	Review and approve Test Plan and Certificate of Acceptance, issue resolution
QA Manager	Dave Skea	Assess Problem Severity and Priority, maintain issues list
JTS Topology Suite Test Engineers	Yao Cui Martin Davis Michael Ross Denis Boutin	Compile and execute test cases
JTS Topology Suite Defect Recorder	Martin Davis	Enter and maintain defect reports

1.7.2 Test Deliverables
The following publications will be provided in the course of acceptance testing:

Table 1-3 – Test Deliverables

PUBLICATION	DESCRIPTION	AVAILABILITY
QA Status Meeting Minutes	Minutes from joint project status meetings which include QA status report	Day after meeting
Defect Reports	Defect reports describing defect details	On demand from Defect Tracking System
Test Observations	Bound Observation reports	At final acceptance

1.7.3 Defect Tracking

Vivid will enter and maintain records of defects and bug fixes. Defect severity is defined below:

Table 1-4 – Defect Tracking

SEVERITY	DISCERNING FACTORS
Critical	The entire system or a subsystem will not execute.
High	There is no practical workaround to enable completion of the test block.
Medium	There is a practical workaround but the defect is an inconvenience.
Low	There is a practical workaround but the defect is a minor inconvenience.

1.7.4 Escalation Criteria

The responsibility for executing the tests will lie solely with the Test Engineer and will be observed by the QA Manager and Test Observer(s).

If there is contention about the severity of a defect or the pass/fail result of a test, it will be escalated to the issue list. The issue list will be discussed at the weekly summary meetings attended by the JTS project team.

1.7.5 Test Case Execution Order

JTS test cases can be executed in any sequence.

1.7.6 Suspension Criteria and Resumption Requirements

Any test containing a defect that prevents the test procedure from being executed reasonably is grounds for suspension of the test. If the defect is sufficiently large to affect more than 50% of the test cases the entire acceptance testing will be suspended until a resolution can be provided.

Once the defect has been resolved, testing can resume. The resumption point depends on the nature of the defect and will be decided jointly by Base Mapping and Geomatics and Vivid Solutions Inc.

1.8 ACCEPTANCE CRITERIA

A test is *passed* if there are no open defects with a severity of *high* or *critical* logged against it.

JTS Topology Suite is acceptable if:

1.   All tests have been executed.
2.   All tests have passed.

3. All remaining defects reported will be resolved within a mutually agreeable time.

1.9 NAMING CONVENTIONS

Each test case is named/expressed by a taxonomy defined in Chapter 2.

1.10 REFERENCE DOCUMENTS

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

OGC, 1999, OpenGIS Simple Feature Specification for SQL, revision 1.1, OpenGIS Consortium Inc, OpenGIS Project Document 99-049, May 5, 1999.

Vivid, 2001, JTS Topology Suite Technical Specifications. Prepared for Geographic Data BC, BC Ministry of Environment, Lands and Parks. Prepared by Vivid Solutions Inc.

2. TEST CASES

As outlined in section 1.3, cases are created to test binary predicates and spatial analysis functions. The focus is to create normative and representative test cases that are adequate to test all the classes and functions implemented in JTS.

2.1 TESTING JTS BINARY PREDICATES

Binary predicates are Boolean functions that are used to test the existence of a specific topological spatial relationship between two geometric objects (OGC, 1999). Binary predicates include:

- Equals
- Disjoint
- Intersect
- Touches
- Crosses
- Within
- Contains
- Overlaps

The spatial relationship between points, lines and areas, including areas with holes and multiple component lines and areas, is expressed by the Dimensionally Extended Nine-Intersection Model (DE-9IM). In this model, a geometry has Interior, Boundary and Exterior. The test and computation of dimensions of the intersections between the Interiors, Boundaries and Exteriors of the two geometries result an "intersection" matrix pattern.

Theoretically, test cases should consist of all possible combinations of geometries. It is difficult to estimate how many combinations exist, needless to say the enormous amount of work to actually create all these cases without certain level of automation.

One approach is to identify a geometry's unique **intersection components**, the smallest components or parts used in an intersection. By enumerating the combinations of intersection components, it is possible to generate a list of the most basic cases. Using the list of the combinations as a guide, representative test cases in a limited number can be created.

For each test case, a QA engineer will manually code the DE-9IM pattern matrix (e.g., 212101212 for two Polygons which overlap) and enter it as part of a test case. The test harness would return the pattern matrix from JTS. If the pattern matrix from JTS is different from the one coded by the QA engineer, the test harness will give error messages. After verifying the manually coded pattern matrix and examining the nature of the error, a report is sent to the developer at Vivid for a fix. Every time a new release of JTS is received, all the cases are tested.

A taxonomy is introduced to facilitate the compilation, organisation and description of JTS validation cases involving relational operations or intersections. A language is used to describe the intersection components and express intersection.

2.1.1 Geometric Taxonomy of Intersection

A taxonomy of intersection is needed to enumerate all permutations of intersections between two geometric objects. The taxonomy classifies a geometry based on its instantiable type and identifies components used in intersection.

At the top of the taxonomic hierarchy, a geometry is identified by geometry type (Figure 2-1). For a particular geometry type, a subset of instantiable type or geometry subtype is used to construct a test case. For a given geometry subtype, a geometry can be identified by the interior, boundary or exterior. Each of the interior, boundary or exterior components in a different dimension is broken down into smaller parts – the intersection components.

Intersection components are identified for their uniqueness, perceived significance or points of interest in intersection. Identifiable but unnecessary intersection components are not considered.

For testing binary predicates, by pairing two geometries and rotating through the taxonomic hierarchy, it's possible to generate a list of test cases with basic intersection components. For instance, in a simplified version, the list has more than 400 cases involving intersections in zero dimension and nearly 6,000 cases involving intersections in one dimension and line segments in one direction. The list can expand to include more than 60,000 possible cases if we allow the order of the geometries and the direction of line segments to be reversed. Still, the list may not necessarily cover all the possible, interesting and complex test cases. Nevertheless, it can be used as a guide to compile and organise the representative test cases. More than 400 actual test cases are manually created for the validation test suite.

At this point, the taxonomy can be used to identify and organise simple intersection components. The taxonomy can be extended to identify more complicated geometry or intersection components.
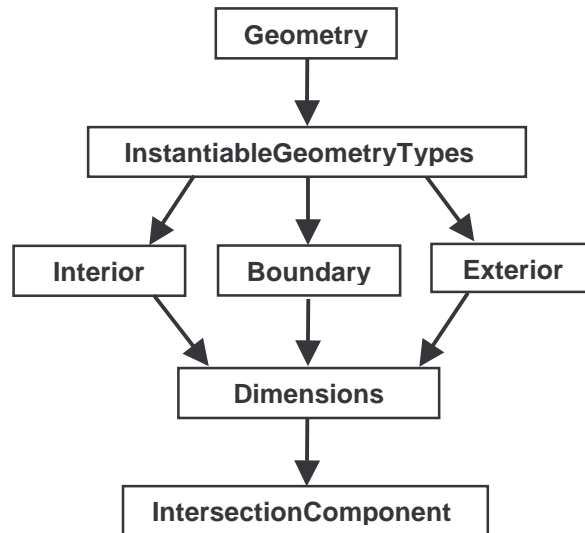
Figure 2-1 – Taxonomic Hierarchy of Geometry

2.1.2 Geometric Types

Geometric types, defined as Simple Feature types in OGC (1999; Figure 2-2), are further elaborated in *JTS Technical Specifications* (Vivid, 2001). Instantiable geometric subtypes are listed in Table 2-1. The current version of JTS doesn't support GeomCollection. However, MultiPoint, MultiLineString and MultiPolygon are supported. In addition, LinearRing and non-simple LineString are included on the list as a subset of LineString.

Table 2-1 – Geometry Types

GEOMETRIC SUBTYPES	ACRONYM
Point	P
MultiPoint	mP
Line, LineString	L
closed LineString (LinearRing)	LR
non-simple LineString	nsL
MultiLineString	mL
Polygon	A
MultiPolygon	mA
GeomCollection	C
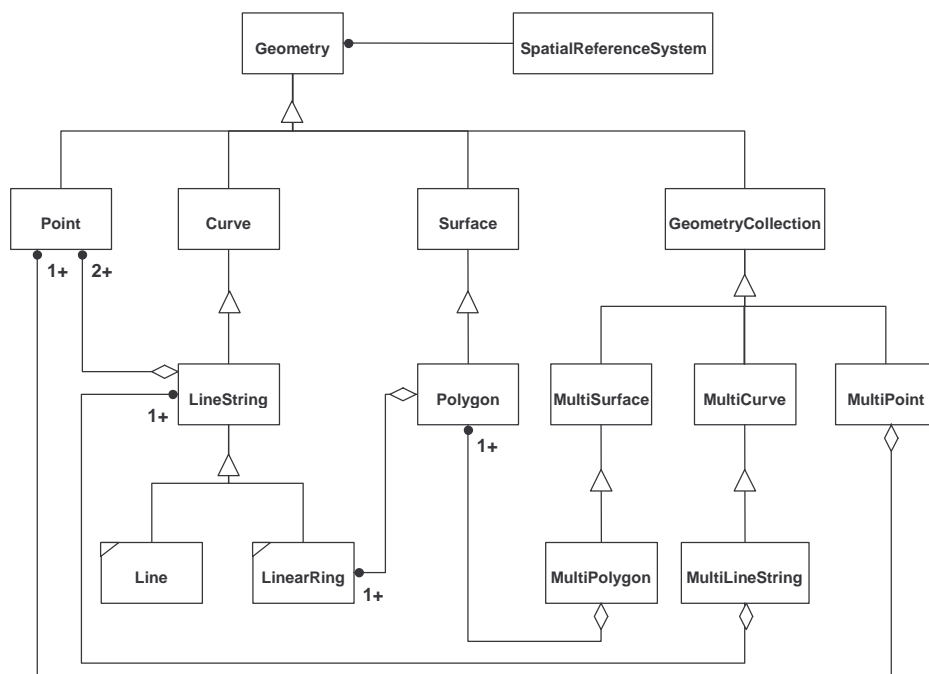
Figure 2-2 – Geometry Class Hierarchy for Simple Features (from OGC, 1999)

In total there are 81 possible combinations of geometric subtypes between a pair of geometries (A and B). Because the test harness can reverse the order of geometry A and B internally, only 36 combinations are needed (Table 2-2).

Table 2-2 – Combinations of Geometric Subtypes between Geometry A and B

		GEOMETRY B								
		P	mP	L	LR	nsL	mL	A	mA	C
GEOMETRY A	P	P/P	P/mP	P/L	P/LR	P/nsL	P/mL	P/A	P/mA	P/C
	mP		mP/mP	mP/L	mP/LR	mP/nsL	mP/mL	mP/A	mP/mA	mP/C
	L			L/L	L/LR	L/nsL	L/mL	L/A	L/mA	L/C
	LR				LR/LR	LR/nsL	LR/mL	LR/A	LR/mA	LR/C
	nsL					nsL/nsL	nsL/mL	nsL/A	nsL/mA	nsL/C
	mL						mL/mL	mL/A	mL/mA	mL/C
	A							A/A	A/mA	A/C
	mA								mA/mA	mA/C
	C									C/C

2.1.3 Interior, Boundary and Exterior of a Geometry

The definitions of Interior, Boundary and Exterior can be found in the *OpenGIS Simple Feature Specification for SQL* (OGC, 1999). A summary of Interior, Boundary and Exterior for each geometric subtype, with minor interpretation, is given in Table 2-3.

Table 2-3 – Interior, Boundary and Exterior of a Geometry

GEOMETRIC SUBTYPES	INTERIOR	BOUNDARY	EXTERIOR
Point, MultiPoint	Point, Points	Empty	Points not in the interior or boundary
Line, LineString	Points that are left when the boundary points are	Two end Points	
LinearRing	All the points along the	Empty	
non-simple LineString	Points that are left when the boundary points are removed	An open end Point or an end Point at a multipassing1 point with an odd number of intersecting line segments	
MultiLineString	Points that are left when the boundary points are	Points in the boundaries of an odd number of its element	
Polygon	Points within the Rings	a set of Rings	
MultiPolygon	Points within the Rings	a set of Rings	

2.1.4 Dimensions of Interior, Boundary and Exterior

Table 2-4 – Dimensions of Interior, Boundary and Exterior

	DIMENSIONS OF INTERSECTION COMPONENTS		
	0	1	2
INTERIOR	Point,Points Line,LineString,LinearRing non-simpleLineString	Line,LineString,LinearRing non-simpleLineString	Polygon MultiPolygon
BOUNDARY	Line,LineString,LinearRing non-simpleLineString Polygon MultiPolygon	Line,LineString, non-simpleLineString Polygon MultiPolygon	
EXTERIOR			Allgeometrytypes

Geometric objects of zero, one or two dimension can be embedded in a two-dimensional space. A geometric object's exterior appears in two dimension. There is no identifiable intersection component in zero or one dimension for Exterior. For Interior and Boundary in a given dimension, only some geometric subtypes have identifiable intersection components (Table 2-4).

2.1.5 Intersection Components

Intersection components are the smallest identifiable parts in a geometry that are used in an intersection. As stated earlier, the intersection components are parts of Interior, Boundary and Exterior that are broken down based on their dimensions, uniqueness, and perceived significance or points of interest in an intersection. For instance, the LineString in

---

[1] Multipassing is at a point that a line passes through it more than once (excluding end points which are identical). Multipassing could be self-intersecting in points (crossing) or line segments (overlapping) or both.

Figure 2-3 has the following unique intersection components in zero dimension for boundary and interior: an end point, a vertex and a non-vertex. All the three shaded points are considered as unique and significant intersection components.

Figure 2-3 – Example of Intersection Components in Zero Dimension for a LineString

Except for cases that involve the whole geometry (i.e., from start point to end point), unique and significant intersection components in one dimension are line segments defined by two intersection components in zero dimension.

To identify and properly describe most of the simple intersection components, a language is introduced based on the test case taxonomy and is discussed in the next section.

2.1.6 A Language for Expressing Intersection

A grammar that defines a language to express intersection and to identify intersection components is defined by the Backus Naur Form (BNF) notation (Listing 2-1). BNF has been seen in different forms with various flavours and enhancements. The following general meta-symbols and definitions of BNF are used in this document:

```
::=     definition; meaning "is defined as"
|       separates alternatives; meaning "or"
<>      encloses nonterminal
[]      encloses optional item
()      encloses a group of alternatives
```

Nonterminals: grammatical symbols that need to be defined
Terminals: symbols that need no further definition

Terminals of only one character are surrounded by quotes (" ") to distinguish them from meta-symbols.

Listing 2-1 – Test Cases and Intersection Components Defined by BNF

```
<intersection_expression>  ::=  <result_dimensionality> <open_brace> <intersection_term>
                                <intersection_operator> <intersection_term> <close_brace>
  <result_dimensionality>  ::=  <dimension> <open_paren> <result_dimensions> <close_paren>
      <result_dimensions>  ::=  <whole_number_constant>
       <intersection_term>  ::=  <target> <separator> <geometry> <separator>
                                <intersection_component>
               <geometry>  ::=  <Point>
                                | <LineString>
                                | <linearRing>
                                | <non_simple_LineString>
                                | <Polygon>
                                | <MultiPoint>
                                | <MultiLineString>
```

```
                                 | <MultiPolygon>
<intersection_component>  ::=  <interior_component>
                                 | <boundary_component>
                                 | <exterior_component>
   <interior_component>  ::=  <interior_component_0d>
                                 | <interior_component_1d>
                                 | <interior_component_1d> <separator> <reverse>
                                 | <interior>
 <interior_component_0d>  ::=  <interior> <separator> <close_point>
                                 | <interior> <separator> <vertex_choice>
                                 | <interior> <separator> <close_point> <multipass>
                                 | <interior> <separator> <vertex_choice> <multipass>
 <interior_component_1d>  ::=  <interior> <separator> <start_point> <to> <end_point>
                                 | <interior> <separator> <end_point> <to> <vertex_choice>
                                 | <interior> <separator> <vertex> <to> <nonVertex>
                                 | <interior> <separator> <end_point> <to> <vertex_choice>
                                     <multipass>
                                 | <interior> <separator> <vertex> <to> <nonVertex> <multipass>
                                 | <interior> <separator> <end_point> <multipass> <to>
                                     <vertex_choice>
                                 | <interior> <separator> <vertex> <multipass> <to> <nonVertex>
                                 | <interior> <separator> <end_point> <multipass> <to>
                                     <vertex_choice> <multipass>
                                 | <interior> <separator> <vertex> <multipass> <to> <nonVertex>
                                     <multipass>
         <vertex_choice>  ::=  <nonVertex> | <vertex>
             <multipass>  ::=  <crossing>
                                 | <overlapping>
                                 | <both>         ; both crossing and overlapping
   <boundary_component>  ::=  <line_boundary> | <polygon_boundary>
        <line_boundary>  ::=  <boundary> <separator> <end_point>
                                 | <boundary> <separator> <end_point> <multipass>
     <polygon_boundary>  ::=  <polygon_boundary_0d>
                                 | <polygon_boundary_1d>
                                 | <polygon_boundary_1d> <separator> <reverse>
  <polygon_boundary_0d>  ::=  <polygon_boundary_types> <separator> <close_point>
                               <polygon_boundary_types> <separator> <vertex_choice>
                               <polygon_boundary_types> <separator> <touch_point>
                               <boundary> <separator> <close_point>
                               <boundary> <separator> <vertex_choice>
                               <boundary> <separator> <touch_point>
                                         ; touch_point: at the connecting point between
                                             polygon boundaries
  <polygon_boundary_1d>  ::=  <polygon_boundary_types> <separator> <start_point> <to>
                                     <end_point>
                                 | <polygon_boundary_types> <separator> <end_point> <to>
                                     <vertex_choice>
                                 | <polygon_boundary_types> <separator> <end_point> <to>
                                     <touch_point>
                                 | <polygon_boundary_types> <separator> <vertex> <to> <nonVertex>
```

```
                                 | <polygon_boundary_types> <separator> <vertex_choice> <to>
                                    touch_point>
                                 | <boundary> <separator> <start_point> <to> <end_point>
                                 | <boundary> <separator> <end_point> <to> <vertex_choice>
                                 | <boundary> <separator> <end_point> <to> <touch_point>
                                 | <boundary> <separator> <vertex> <to> <nonVertex>
                                 | <boundary> <separator> <vertex_choice> <to> touch_point>
<polygon_boundary_types> ::= <outer_boundary> | <inner_boundary>
    <exterior_component> ::= <exterior> | <exterior> <separator> <hole>


    Terminal Definitions
            <boundary> ::= Bdy
                <both> ::= "b"
         <close_brace> ::= "}"
         <close_paren> ::= ")"
         <close_point> ::= CP
            <crossing> ::= "x"
           <end_point> ::= EP
            <exterior> ::= Ext
                <hole> ::= "h"
      <inner_boundary> ::= iBdy
            <interior> ::= Int
 <intersection_operator> ::= "="
           <linearRing> ::= LR
           <LineString> ::= "L"
           <MultiPoint> ::= mP
      <MultiLineString> ::= mL
         <MultiPolygon> ::= mA
  <non_simple_LineString> ::= nsL
           <nonVertex> ::= NV
          <open_brace> ::= "{"
          <open_paren> ::= "("
      <outer_boundary> ::= oBdy
         <overlapping> ::= "o"
               <Point> ::= "P"
             <Polygon> ::= "A"
             <reverse> ::= Rev
           <separator> ::= "."
         <start_point> ::= SP
              <target> ::= "A" | "B"
                  <to> ::= "-"
         <touch_point> ::= "TP"
              <vertex> ::= "V"
```
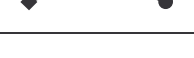
The following extensions are used in validation test suite to identify a particular geometry in
MultiPoint, MultiLineString or MultiPolygon:

```
        <geometry>  ::=  <number_of_geometries> <MultiGeometry> <geometry_id>
```

```
<number_of_geometries> ::=  <whole number constant>
        <MultiGeometry> ::=  <MultiPoint>
                             | <MultiLineString>
                             | <MultiPolygon>
            <geometry_id> ::=  <whole number constant>
```

For a case with multiple intersections, it may be necessary to have more than one expression to describe the case.

In each dimension of intersection, representative examples of intersection components are summarised in Table 2-5 and Table 2-6.

Note: In Table 2-5 and Table 2-6, only line segments in one direction for one-dimensional intersection components are listed. If the direction of line segments is reversed and non-simple LineString is considered, there are a large number of possible intersection components in one dimension.

Table 2-5 – Intersection Components in Different Dimensions

		DIMENSIONS OF INTERSECTION COMPONENTS						
		0			**1**		**2**	
	Point	Line	Polygon	Line	Polygon	Point	Line	Polygon
INTERIOR	Int	Int.CP Int.V Int.NV Int.CPx Int.CPo Int.CPb Int.Vx Int.Vo Int.Vb Int.NVx Int.NVo Int.NVb		Int.EP-SP Int.EP-V Int.EP-NV Int.V-NV Int.CP-V Int.CP-NV				Int
BOUNDARY		Bdy.EP Bdy.EPx Bdy.EPo Bdy.EPb	Bdy.CP Bdy.V Bdy.NV Bdy.TP oBdy.CP oBdy.V oBdy.NV oBdy.TP iBdy.CP iBdy.V iBdy.NV iBdy.TP		Bdy.EP-SP Bdy.EP-V Bdy.EP-NV Bdy.EP-TP Bdy.V-NV Bdy.TP-EP Bdy.TP-V Bdy.TP-NV oBdy.EP-SP oBdy.EP-V oBdy.EP-NV oBdy.EP-TP oBdy.V-NV oBdy.TP-V oBdy.TP-NV iBdy.EP-SP iBdy.EP-V iBdy.EP-NV iBdy.EP-TP iBdy.V-NV iBdy.TP-V iBdy.TP-NV			

EXTERIOR						Ext	Ext	Ext Ext.h
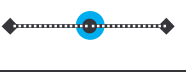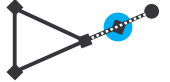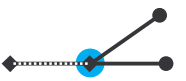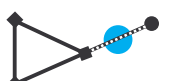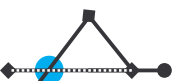
Table 2-6 – Examples of Intersection Components in One Dimension for Non-simple LineString
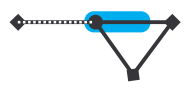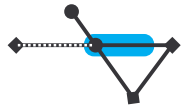
Int.EP-V	Int.EPb-Vb	Int.CPx-NVb
Int.EP-NV	Int.EPb-NVx	Int.CPo-Vx
Int.EP-Vx	Int.EPb-NVo	Int.CPo-Vo
Int.EP-Vo	Int.EPb-NVb	Int.CPo-Vb
Int.EP-Vb	Int.V-NV	Int.CPo-NVx
Int.EP-NVx	Int.V-CPx	Int.CPo-NVo
Int.EP-NVo	Int.V-CPo	Int.CPo-NVb
Int.EP-NVb	Int.V-CPb	Int.CPb-Vx
Int.EPx-V	Int.V-Vx	Int.CPb-Vo
Int.EPx-NV	Int.V-Vo	Int.CPb-Vb
Int.EPx-Vx	Int.V-Vb	Int.CPb-NVx
Int.EPx-Vo	Int.V-NVx	Int.CPb-NVo
Int.EPx-Vb	Int.V-NVo	Int.CPb-NVb
Int.EPx-NVx	Int.V-NVb	Int.Vx-Vo
Int.EPx-NVo	Int.NV-CPx	Int.Vx-Vb
Int.EPx-NVb	Int.NV-CPo	Int.Vx-NVx
Int.EPo-V	Int.NV-CPb	Int.Vx-NVo
Int.EPo-NV	Int.NV-Vx	Int.Vx-NVb
Int.EPo-Vx	Int.NV-Vo	Int.Vo-Vb
Int.EPo-Vo	Int.NV-Vb	Int.Vo-NVx
Int.EPo-Vb	Int.NV-NVx	Int.Vo-NVo
Int.EPo-NVx	Int.NV-NVo	Int.Vo-NVb
Int.EPo-NVo	Int.NV-NVb	Int.Vb-NVx
Int.EPo-NVb	Int.CPx-Vx	Int.Vb-NVo
Int.EPb-V	Int.CPx-Vo	Int.Vb-NVb
Int.EPb-NV	Int.CPx-Vb	Int.NVx-NVo
Int.EPb-Vx	Int.CPx-NVx	Int.NVx-NVb
Int.EPb-Vo	Int.CPx-NVo	Int.NVo-NVb
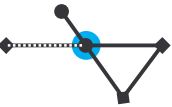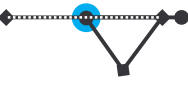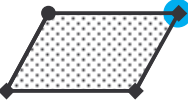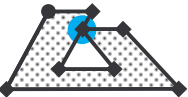
Examples of intersection components of a geometry, expressed by the defined syntax rules, with graphics and descriptions, are shown in Table 2-7.
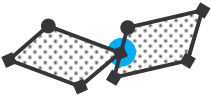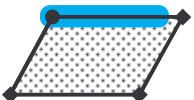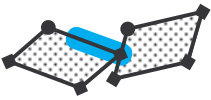
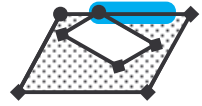**Table 2-7 – Examples of Intersection Components in Different Dimensions**
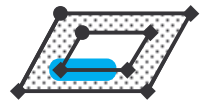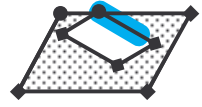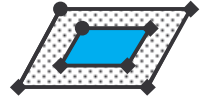
Geometry	Expression	Description
Interiorintersectioncomponentsinzerodimension		
●	`A.P.Int`	Apoint:ataPoint

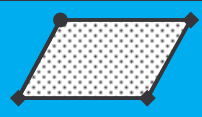|		
	`A.LR.Int.CP`	AclosedLineString:attheclosingpoint
	`A.L.Int.V`	ALineString:atavertex
	`A.L.Int.NV`	ALine:atanon-vertex
	`A.nsL.Int.CPx`	Anon-simpleLineString:attheclosingpointwithcrossing multipass
	`A.nsL.Int.CPo`	Anon-simpleLineString:attheclosingpointwith overlappinglinesegments
	`A.nsL.Int.CPo`	Anon-simpleLineString:attheclosingpointwith overlappinglinesegments
	`A.nsL.Int.CPb`	Anon-simpleLineString:attheclosingpointwithboth crossingandoverlappinglinesegments
	`A.nsL.Int.Vx`	Anon-simpleLineString:atavertexwithcrossingline segments
	`A.nsL.Int.Vo`	Anon-simpleLineString:atavertexwithoverlappingline segments
	`A.nsL.Int.Vb`	Anon-simpleLineString:atavertexwithbothcrossingand overlappinglinesegments
	`A.nsL.Int.NVx`	Anon-simpleLineString:atanon-vertexwithcrossingline segments
	`A.nsL.Int.NVo`	Anon-simpleLineString:atanon-vertexwithoverlapping linesegments
	`A.nsL.Int.NVb`	Anon-simpleLineString:atanon-vertexwithbothcrossing andoverlappinglinesegments

Interiorintersectioncomponentsinonedimension

	`A.L.Int.SP-EP`	ALine:fromthestartpointtotheendpoint
	`A.L.Int.EP-V`	ALineString:fromtheendpointtoavertex
	`A.L.Int.EP-NV`	ALineString:fromtheendpointtoanon-vertex

	`A.LR.Int.EP-NV`	AclosedLineString:fromtheclosingpointtoanon-vertex
	`A.L.Int.V-NV`	ALineString:fromavertextoanon-vertex
	`A.L.Int.NV-NV`	ALineString:fromanon-vertextoanon-vertex
	`A.nsL.Int.EPb-NV`	Anon-simpleLineString:fromtheclosingpointwithboth crossingandoverlappinglinesegmentstoanon-vertex
	`A.nsL.Int.EPb-NV`	Anon-simpleLineString:fromendpointwithbothcrossing andoverlappinglinesegmentstoanon-vertex
	`A.nsL.Int.EPo-NVo`	Anon-simpleLineString:fromtheclosingpointwith overlappinglinesegmentstoanon-vertexwithoverlapping linesegments
	`A.nsL.Int.EPo-NV`	Anon-simpleLineString:fromendpointwithoverlapping linesegmentstoanon-vertex
	`A.nsL.Int.EPo-NVo`	Anon-simpleLineString:fromendpointwithoverlapping linesegmentstoanon-vertexwithoverlappinglinesegments
	`A.nsL.Int.EPx-NV`	Anon-simpleLineString:fromtheendpointwithcrossing linesegmentstoanon-vertex
	`A.nsL.Int.EPx-NV`	Anon-simpleLineString:fromtheclosingpointwithcrossing linesegmentstoanon-vertex
	`A.nsL.Int.NVx-EP`	Anon-simpleLineString:fromanon-vertexwithcrossingline segmentstotheendpoint
	`A.nsL.Int.NVo-NVo`	Anon-simpleLineString:fromanon-vertexwithoverlapping linesegmentstoanon-vertexwithoverlappinglinesegments
	`A.nsL.Int.NVb-Vo`	Anon-simpleLineString:fromanon-vertexwithboth crossingandoverlappinglinesegmentstoavertexwith overlappinglinesegments
	`A.nsL.Int.Vb-NV`	Anon-simpleLineString:fromavertexwithoverlappingline segmentstoanon-vertex
	`A.nsL.Int.Vb-NVo`	Anon-simpleLineString:fromavertexwithbothcrossing andoverlappinglinesegmentstoanon-vertexwith overlappinglinesegments

	`A.nsL.Int.Vo-NVo`	Anon-simpleLineString:fromavertexwithoverlappingline segmentstoanon-vertexwithoverlappinglinesegments
	`A.nsL.Int.Vx-NV`	Anon-simpleLineString:fromavertexwithcrossingline segmentstoanon-vertex

Interiorintersectioncomponentintwodimension

	`A.A.Int`	APolygon:intheinterior

Boundaryintersectioncomponentsinzerodimension

	`A.L.Bdy.EP`	ALine:attheendpoint
	`A.nsL.Bdy.EPb`	Anon-simpleLineString:attheendpointwithbothcrossing andoverlappinglinesegments
	`A.nsL.Bdy.EPb`	Anon-simpleLineString:attheendpointwithbothcrossing andoverlappinglinesegments
	`A.nsL.Bdy.EPb`	Anon-simpleLineString:attheendpointwithbothcrossing andoverlappinglinesegments
	`A.nsL.Bdy.EPo`	Anon-simpleLineString:attheendpointwithoverlapping linesegments
	`A.nsL.Bdy.EPx`	Anon-simpleLineString:attheendpointwithcrossingline segments
	`A.A.Bdy.CP`	APolygon:attheclosingpoint
	`A.A.Bdy.V`	APolygon:atavertexontheboundary
	`A.A.Bdy.NV`	APolygon:atanon-vertexontheboundary
	`A.A.Bdy.TP`	APolygon:atatouchingpointontheboundary
	`A.A.Bdy.TP`	APolygonwithahole:atthetouchingpointbetweenouter andinnerboundaries

	`A.mA.Bdy.TP`	TwoPolygons:atthetouchingpointofboundaries
	`A.A.oBdy.CP`	APolygonwithahole:attheclosingpointontheouter boundary
	`A.A.oBdy.V`	APolygonwithahole:atavertexontheouterboundary
	`A.A.oBdy.NV`	APolygonwithahole:atanon-vertexontheouterboundary
	`A.A.iBdy.CP`	APolygonwithahole:attheclosingpointofinnerboundary
	`A.A.iBdy.V`	APolygonwithahole:atavertexontheinnerboundary
	`A.A.iBdy.NV`	APolygonwithahole:atanon-vertexontheinnerboundary
	`A.A.iBdy.TP`	APolygonwithtwoholes:atthetouchingpointofinner boundaries
Boundaryintersectioncomponentsinonedimension		
	`A.A.Bdy.SP-EP`	APolygon:fromthestartpointtotheendpointonthe boundary
	`A.A.Bdy.EP-V`	APolygon:fromtheendpointtoavertexontheboundary
	`A.A.Bdy.V-NV`	APolygon:fromavertextoanon-vertexontheboundary
	`A.A.Bdy.NV-NV`	APolygon:fromanon-vertextoanon-vertex
	`A.mA.Bdy.TP-NV`	TwoPolygons:fromthetouchingpointoftwoPolygonstoa non-vertex
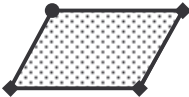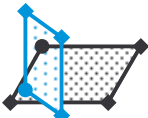
	`A.A.oBdy.V-NV`	APolygonwithahole:fromtheavertextoanon-vertexon theouterboundary
	`A.A.oBdy.NV-NV`	APolygonwithahole:fromanon-vertextoanon-vertex
	`A.A.oBdy.TP-NV`	APolygonwithahole:fromthetouchingpointtoanon-vertex
	`A.A.iBdy.CP-NV`	APolygonwithahole:fromtheclosingpointtoanon-vertex ontheinnerboundary
	`A.A.iBdy.V-NV`	APolygonwithahole:fromavertextoanon-vertexonthe innerboundary
	`A.A.iBdy.NV-NV`	APolygonwithahole:fromanon-vertextoanon-vertexon theinnerboundary
	`A.A.iBdy.TP-NV`	APolygonwithahole:fromthetouchingpointtoanon-vertexontheinnerboundary
	`A.A.iBdy.TP-NV`	APolygonwithtwoholes:fromthetouchingpointoftwo holestoanon-vertexontheinnerboundary

Exteriorintersectioncomponentsintwodimension

	`A.P.Ext`	Apoint:exterior
	`A.L.Ext`	ALine:exterior
	`A.L.Ext`	ALineString:exterior
	`A.LR.Ext`	AclosedLineString:exterior
	`A.nsL.Ext`	Anon-simpleLineString:exterior
	`A.A.Ext`	APolygon:exterior
	`A.A.Ext.h`	APolygonwithahole:exteriorinthehole

VIVID
SOLUTIONS

> **Note:**
> Solid circle: a Point, an end point or closing point
> Diamond: explicit vertex
> Shaded circle: intersection component in zero dimension
> Shaded line: intersection component in one dimension
> Shaded area: intersection component in two dimension

Examples of relate intersections are shown in Table 2-8.

Table 2-8 – Examples of Intersection Expression

Geometry		Intersection	Expression
A	**B**		
●	●	●	`dim(0){A.P.Int = B.P.Int}`
●	•——————•	●——————•	`dim(0){A.P.Int = B.L.Bdy.EP}`
(line)	(line)	(crossing lines)	`dim(0){A.L.Int.NV = B.L.Int.VN}`
●	(square with hole)	(square with hole and dot)	`dim(0){A.P.Int = B.A.Ext.h}`
●——●	(line bent)	(line bent)	`dim(1){A.L.Int.EP-SP = B.L.Int.EP-NV}`
●——●	(shaded parallelogram)	(shaded parallelogram with line)	`dim(1){A.L.Int.SP-EP = B.A.Bdy.NV-NV}`
(shaded shape)	(shaded parallelogram)	(overlapping shapes)	`dim(2){A.A.Int = B.A.Int}`

> **Note:**
> Solid circle: a Point, an end point or closing point
> Diamond: explicit vertex
> Shaded area: intersection component in two dimension
> Shaded geometry: Geometry A

2.2 TESTING SPATIAL ANALYSIS FUNCTIONS

Spatial analysis functions include:

- Buffer
- Convex Hull

- Intersection
- Union
- Difference
- Symmetrical Difference

While set-theoretic operations in spatial functions (i.e., intersection, union, difference and symmetrical difference) require a pair of geometries in a test case, buffer and convex hull are constructive operations which need the input of one or more geometries.
Similar to test cases for binary predicates, there are an infinite number of possible combinations of geometries or variations of a geometry. The key is to create normative and representative cases that are adequate to validate the results of JTS spatial functions.
While all the test cases for predicates can be modified to test spatial functions, some additional cases are created as well.

For each test case, a QA engineer will determine the expected result from each function where applicable and enter it as part of the test case. If there is a difference between the expected result and the result returned from a JTS spatial function, the error is flagged.

When using test cases from predicates, the expected result can be entered in the test harness and the result that is returned from a JTS spatial function can be visually verified.

3. TEST TYPES AND TEST DATA

3.1 TACTICAL TEST SUITE WITH SYNTHETIC DATA

A small, tactical test suite with less than one hundred cases was compiled by the developers to test the implementation of binary predicates and spatial analysis functions. These test cases may not necessarily cover representative geometries and geometric combinations, yet they are critical in verifying the results returned from JTS binary predicates and spatial functions. These cases are included in the test harness.

3.2 VALIDATION SUITE WITH SYNTHETIC DATA

Representative test cases with synthetic data and for validating binary predicates were compiled. Initially, the cases were compiled for testing a specific predicate and as such were grouped as blocks for each of the predicate functions (e.g., 1 for Equals, 2 for Disjoint, 3 for Touches, 4 for Crosses, 5 for Contains or Within, and 6 for Overlaps). The validation suite was expanded with the guide from a list of potential test cases generated by the geometric taxonomy. Attempt has been made to organise the test cases by following the enumerated list from the geometric taxonomy, at least within each block of test cases. While many similar or insignificant test cases generated from the geometric taxonomy are not included in the suite, interesting cases not on the list were compiled and added to the suite. All the validation test cases can be adopted to test spatial analysis functions as well.

3.3 STRESS/VOLUME TEST

Stress testing determines the performance and reliability of JTS when a large volume of data is used. Data for stress test consists of Polygon geometries from existing mapping projects. Datasets are arbitrarily grouped as small, medium, large and very large geometries in terms of the number of vertices and number of holes in a Polygon (Table 3-1).

Table 3-1 – Stress Test Datasets

GROUP	NUMBER OF VERTICES	NUMBER OF HOLES IN A POLYGON
Small	< 1,000	< 10
Medium	1,000 – 10,000	10 – 100
Large	10,000 – 100,000	100 – 1,000
Very Large	> 100,000	> 1,000

Geometries with different sizes of areas in Polygons or lengths of LineStrings will be selected and tested as well.

Test data is in WKT or embedded in XML and can be obtained from the Corporate Projects Unit of the Base Mapping and Geomatics Branch.

3.4 ROBUSTNESS TEST

For a given precision model, incorrect answers could result from round-off errors in non-robust algorithms. Efficient robust algorithms are available to deal with this problem. Both non-robust and robust algorithms are implemented in JTS. To test the robust algorithms, a test case should result in an incorrect answer in the non-robust version of JTS but the robust code should give the correct result.

As an example, a case is created to test binary predicate Intersects. The case is a line that passes very close to the origin or a point at origin, but does not intersect it (Figure 3-1).
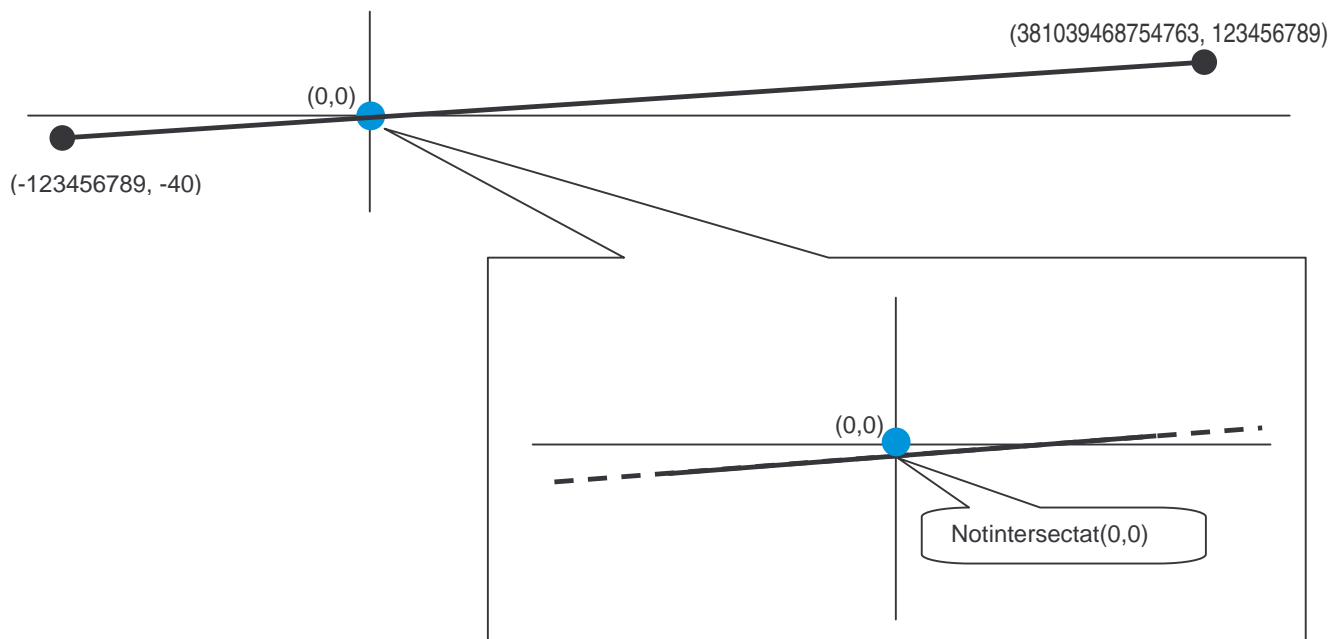
Figure 3-1 – Testing of Robustness

The line is LINESTRING (-123456789 -40, 381039468754763 123456789).

For a line ($x1\ y1,\ x2\ y2$) that does pass through the origin, $x1/\ y1 = x2\ /\ y2$. For this test case:

$$-123456789 / -40 = 3086419.725$$

However,

$$381039468754763 / 123456789 =$$
$$3086419.724999999797499998157249983230974847401871111357027 1133489467315$$

This indicates that the line passes below the origin and does not intersect it. Using non-robust line intersection code, JTS erroneously reports that the line does intersect the origin. It fails when non-robust line intersection code is used.

3.5 CASES FROM GEOCONNECTIONS

Geometries from spatial databases managed by GeoConnections will be used to test JTS. Data will be extracted and provided by GeoConnecitons.

4. ACCEPTANCE TEST FORMS

The following forms will be used during acceptance testing:

- JTS Topology Suite Observation Report; and

- Acceptance Test Certificate.

4.1 JTS TOPOLOGY SUITE OBSERVATION REPORT

Test Suite:		Date:		Observer:		
Test Case		**Observation/Verification**				**Ref#**
Completion Status:		[Pass / Fail]		**Verified By:**		
Re-run Required:		[Yes / No]		**Re-run Date:**		

Page _____ of _____

VIVID
SOLUTIONS

4.2 ACCEPTANCE TEST CERTIFICATE

JTS Topology Suite Acceptance

This document certifies that the JTS Topology Suite has performed satisfactorily and according to the technical specifications.

.... Date: ....

Mark Sondheim, Base Mapping and Geomatics

.... Date: ....

David Ash, Vivid Solutions Inc.