JTS TOPOLOGY SUITE
TESTRUNNER & TESTBUILDER USER GUIDE

February 3, 2005

Prepared by:

VIVID SOLUTIONS

Document Change Control

REVISION NUMBER	DATE OF ISSUE	AUTHOR(S)	BRIEF DESCRIPTION OF CHANGE
1.0	13-Sep-2001	Jonathan Aquino	Original Draft
1.1	17-Dec-2001	Jonathan Aquino	Added TestBuilder documentation
1.2	30-Jan-2002	Jonathan Aquino	Added Acme Licence

Table of Contents

1. OVERVIEW

This guide describes how to use:

the JTS TestRunner, a Java application that validates the JTS Topology Suite
the JTS TestBuilder, a Java application that creates test files for the TestRunner

The TestRunner and TestBuilder are used by people who are developing or evaluating JTS.

While similarly named, the JTS TestRunner is not the same as the JUnit TestRunner. Both have their place. JUnit is a general Java testing framework that the JTS developer will find useful for testing JTS' various Java classes. However, the JTS TestRunner is easier for testing JTS' computations because, unlike JUnit, it requires no Java coding or compilation – tests are specified using simple text files. These text files can easily be created by hand, or generated by a graphical tool (like the TestBuilder).

The TestRunner and the TestBuilder require one of the following Java products:

Java 2 Runtime Environment (JRE), Standard Edition, v1.3
Java 2 Software Development Kit (SDK), Standard Edition, v 1.3.

Either may be freely downloaded from http://java.sun.com/products/.

This guide is divided into two main sections:

Using The TestRunner
Using The TestBuilder

At the end is an appendix detailing the structure of the XML test files.

For more information on JTS, see the *JTS Technical Specifications*.

2. USING THE TESTRUNNER

This section describes how to use the TestRunner to run a test file that validates the JTS Topology Suite. Test files are created using the TestBuilder (see *3 Using The TestBuilder* on page 11).

Sections 2.1 and 2.2 describe the two ways of using the TestRunner:

Graphical mode. Has file-chooser dialogs that make it easy to build a list of test files.
Text mode. Quicker to start up than graphical mode. Also, the `–files` switch can be used to run all test files in a given directory (see Section 2.4).

Sections 2.3 and 2.4 describe:

how to read the results of the TestRunner
how to configure the TestRunner using switches

2.1 STARTING IN GRAPHICAL MODE

To start the TestRunner in graphical mode, Windows users can simply run testrunner.bat. Users of other operating systems need only:

1. Add to the class path: `jdom.jar`, x`erces.jar`, `jts.jar`, `jts_test.jar` (these files are supplied with JTS)
2. Execute at a command prompt:
   ```
   java com.vividsolutions.jtstest.testrunner.TopologyTestApp -gui
   ```

Figure 2-1 below shows TestRunner’s graphical mode.

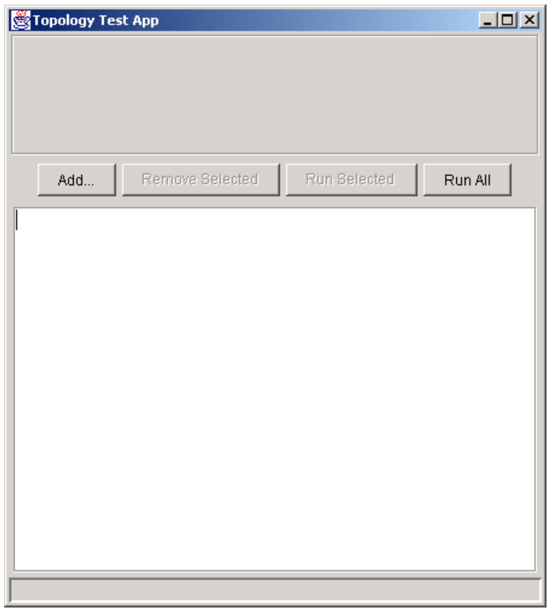
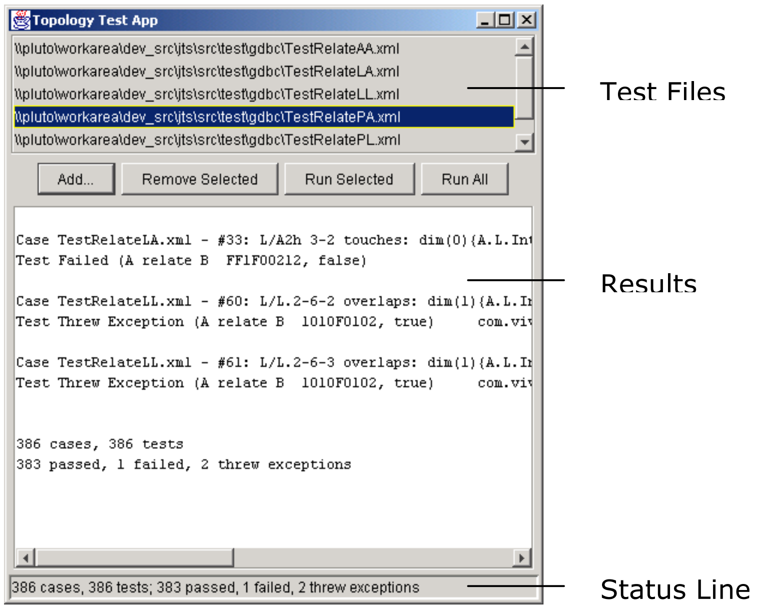


Figure 2-1 – Graphical mode, on start up

To add test files to the TestRunner, use the **[Add...]** button. To run the test files, press **[Run All]**. The `test` directory contains sample test files you can try. In Figure 2-2 below, several test files have been added and run.



```
\\pluto\workarea\dev_src\jts\src\test\gdbc\TestRelateAA.xml
\\pluto\workarea\dev_src\jts\src\test\gdbc\TestRelateLA.xml
\\pluto\workarea\dev_src\jts\src\test\gdbc\TestRelateLL.xml
\\pluto\workarea\dev_src\jts\src\test\gdbc\TestRelatePA.xml
\\pluto\workarea\dev_src\jts\src\test\gdbc\TestRelatePL.xml
```

```
Case TestRelateLA.xml - #33: L/A2h 3-2 touches: dim(0){A.L.Int
Test Failed (A relate B  FF1F00212, false)

Case TestRelateLL.xml - #60: L/L.2-6-2 overlaps: dim(1){A.L.In
Test Threw Exception (A relate B  1010F0102, true)     com.viv

Case TestRelateLL.xml - #61: L/L.2-6-3 overlaps: dim(1){A.L.In
Test Threw Exception (A relate B  1010F0102, true)     com.viv


386 cases, 386 tests
383 passed, 1 failed, 2 threw exceptions
```

```
386 cases, 386 tests; 383 passed, 1 failed, 2 threw exceptions
```

Figure 2-2 – Graphical mode, after running several test files

The test files are listed in the upper panel; the results, in the lower panel (results are described in *2.3 Reading The Results* on page 7). Various statistics are displayed in the **status line** at the bottom.

The difference between "cases" and "tests" is described in *Appendix: Test File Format* on page 17.

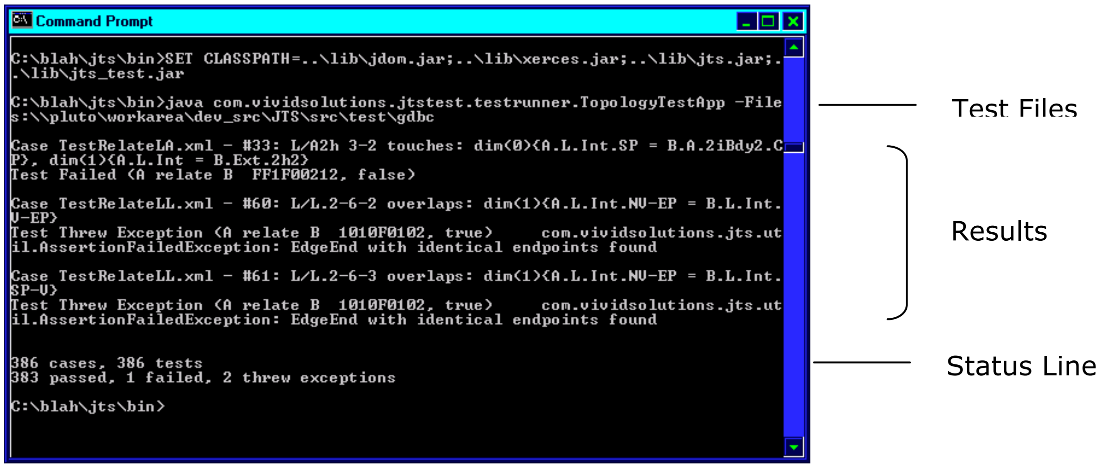
2.2 STARTING IN TEXT MODE

To start the TestRunner in text mode:

1. Add to the class path: `jdom.jar`, `xerces.jar`, `jts.jar`, `jts_test.jar` (these files are supplied with JTS)
3. Execute at a command prompt:
   `java com.vividsolutions.jtstest.testrunner.TopologyTestApp -files C:\TestConvexHull.xml` (Note that the `-gui` switch is omitted)

`test.bat` contains an example of starting the TestRunner in text mode.

Figure 2-3 below shows TestRunner's text mode in action.



```
C:\blah\jts\bin>SET CLASSPATH=..\lib\jdom.jar;..\lib\xerces.jar;..\lib\jts.jar;.
.\lib\jts_test.jar

C:\blah\jts\bin>java com.vividsolutions.jtstest.testrunner.TopologyTestApp -File
s:\\pluto\workarea\dev_src\JTS\src\test\gdbc

Case TestRelateLA.xml - #33: L/A2h 3-2 touches: dim(0){A.L.Int.SP = B.A.2iBdy2.C
P}, dim(1){A.L.Int = B.Ext.2h2}
Test Failed (A relate B  FF1F00212, false)

Case TestRelateLL.xml - #60: L/L.2-6-2 overlaps: dim(1){A.L.Int.NV-EP = B.L.Int.
V-EP}
Test Threw Exception (A relate B  1010F0102, true)     com.vividsolutions.jts.ut
il.AssertionFailedException: EdgeEnd with identical endpoints found

Case TestRelateLL.xml - #61: L/L.2-6-3 overlaps: dim(1){A.L.Int.NV-EP = B.L.Int.
SP-V}
Test Threw Exception (A relate B  1010F0102, true)     com.vividsolutions.jts.ut
il.AssertionFailedException: EdgeEnd with identical endpoints found


386 cases, 386 tests
383 passed, 1 failed, 2 threw exceptions

C:\blah\jts\bin>
```

Figure 2-3 – Text mode

Text mode and graphics mode display results in the same format (see *2.3 Reading The Results* on page 7). The status line at the bottom summarizes the results.

2.3 READING THE RESULTS

This section discusses how to read TestRunner output:

the status line
passed tests
failed tests
test exceptions
parse exceptions

2.3.1 Status Line

At the bottom of the results is a status line, such as the following:

```
386 cases, 386 tests
383 passed, 1 failed, 2 threw exceptions
```

Each statistic is described in Table 1 below.

Table 1 – Statistics reported on the status line

STATISTIC	DESCRIPTION
Number of test cases	Each **test case** specifies two geometries to test with e.g. two linestrings
Number of tests	Each **test** specifies an operation to perform on the two geometries e.g. `overlaps`
Number of passed tests	A test **passes** if the operation's actual value matches its expected value
Number of failed tests	A test **fails** if the operation's actual value differs from its expected value
Number of tests throwing exceptions	A **test exception** occurs if an error occurs within the TestRunner or JTS
Number of parsing exceptions (not shown in the example)	A **parse exception** occurs if the test file contains a syntax error (see *XML* on page 17)

2.3.2 Passed Test

Details of passed tests are displayed only if the `–verbose` switch is used (see *Setting Switches* on page 10). Messages like the following are displayed for passed tests:

```
Case TestRelateLA.xml - #38: L/A2h-3-7 touches: dim(0){A.L.Int.V =
                  B.A.2iBdy1.NV}, dim(0){A.L.Int.V = B.2iBdy2.CP}
Test Passed (A relate B  F01FF0212, true)
Test Passed (A intersection B)
Test Passed (A union B)
```

This message indicates that all three tests passed in test case #38 in the test file `TestRelateLA.xml`:

the first test asserted that the statement "A relate B = F01FF0212" is true
the second test involved the `intersection` operation
the third test involved the `union` operation

The name of test case #38 is

```
L/A2h-3-7 touches: dim(0){A.L.Int.V = B.A.2iBdy1.NV}, dim(0){A.L.Int.V
= B.2iBdy2.CP}
```

2.3.3 Failed Test

If tests fail, you will see result messages like the following.

```
Case TestRelateLA.xml - #2: L/A-3-2 touches: dim(0){A.L.Bdy.EP = B.oBdy.CP}
Test Failed (A getBoundary B)
```

```
Case TestRelateLA.xml - #33: L/A2h 3-2 touches: dim(0){A.L.Int.SP =
                  B.A.2iBdy2.CP}, dim(1){A.L.Int = B.Ext.2h2}
Test Failed (A relate B  FF1F00212, true)
```

The first message indicates a failure of a test involving the `getBoundary` operation in test case #2 in the test file `TestRelateLA.xml`.

The second message indicates a failure of a test involving the `relate` operation in test case #33 in the test file `TestRelateLA.xml`. The test asserted that the statement "A relate B = FF1F00212" should be `true`, but evidently that was not so.

If a test fails, either:

fix the test, or
ask the JTS developers to fix the JTS code

2.3.4 Test Exception

If an error occurs within JTS or the TestRunner, you will see a message like the following.

```
Case TestRelateLA.xml - #18: L/A-5-4 within: dim(0){A.L.Bdy.SP =
                  B.A.oBdy.NV}, dim(1){A.L.Bdy.SP-Int.V = B.A.Int},
                  dim(1){A.L.Int.V-Int.V = B.oBdy.NV-oBdy.NV},
                  dim(1){A.L.Int.V-Bdy.EP = B.A.Int}
Test Threw Exception (A relate B  11F00F212, true)
                  java.lang.NullPointerException
```

This message indicates that an `NullPointerException` occurred during a test involving the `relate` operation in test case #18 in the test file `TestRelateLA.xml`.

If you encounter a test exception such as the one above, please inform the JTS developers so that they can fix the code.

2.3.5 Parse Exception

If a test file has a syntax error, you will see a message like the following.

```
An exception occurred while parsing <case> 2 in c:\TestRelateLA.xml:
com.vividsolutions.jts.io.ParseException: Expected word but encountered
number: 150.0
```

For example, this parse exception might have been caused by a missing comma:

```
      LINESTRING (40 230 150 150)
```

instead of

```
      LINESTRING (40 230, 150 150)
```

If you encounter a parse exception such as the one above, find the problem in your test file and fix it.

2.4 SETTING SWITCHES

The TestRunner can be configured using a number of switches, such as `–gui`. They are described in Table 2 below.

Table 2 – TestRunner Switches

SWITCH	DESCRIPTION
-files [.xml files]	Specifies a list of test files and directories containing test files. Separate list items with spaces. Can be used with both graphical mode and text mode.
-properties [.properties file]	Specifies a .properties file, which stores a list of test files. Enables graphical mode to "remember" the last set of files it was working with. If the given .properties file does not exist, it is created. Can be used with the –files switch.
-gui	Runs the TestRunner in graphical mode. Omit this switch to run it in text mode.
-verbose	Displays the results of tests that pass in addition to those that fail. Omit this switch to only show those that fail (recommended).

Any combination of switches is permitted. If no switches are specified, a description of the switches is displayed.

Tip: The easiest way to specify all the files in a directory is to specify the directory using the `–files` switch.

3. USING THE TESTBUILDER

The TestBuilder is an experimental application used to create basic test files. Because of its experimental nature, its GUI has numerous minor defects. However, it has proven to be a useful starting point for creating, visualizing, and running test files.

People who are interested in writing their own test-file generator are directed to *4.2 XML* on page 18.

The TestBuilder has the following useful features:

creating of test geometries either visually or by entering Well-known Text
snapping to a grid
specifying expected values for spatial functions and the intersection matrix
opening and saving XML test files
generating HTML documentation

The TestBuilder has problems with some unusual situations, including:

handling test files with very large numbers
opening/saving test files that with multiple tests for a function

These special tests must be created by hand (see *Appendix: Test File Format*) and run using the TestRunner (see *2 Using The TestRunner*).

3.1 STARTING THE TESTBUILDER

To start the TestBuilder, Windows users can simply run testbuilder.bat. Users of other operating systems need only:

1. Add to the class path: `jdom.jar`, `xerces.jar`, `jts.jar`, `jts_test.jar` (these files are supplied with JTS)
4. Execute at a command prompt:
   `java com.vividsolutions.jtstest.testbuilder.JTSTest`

Figure 3-1 below shows the TestBuilder when first opened.

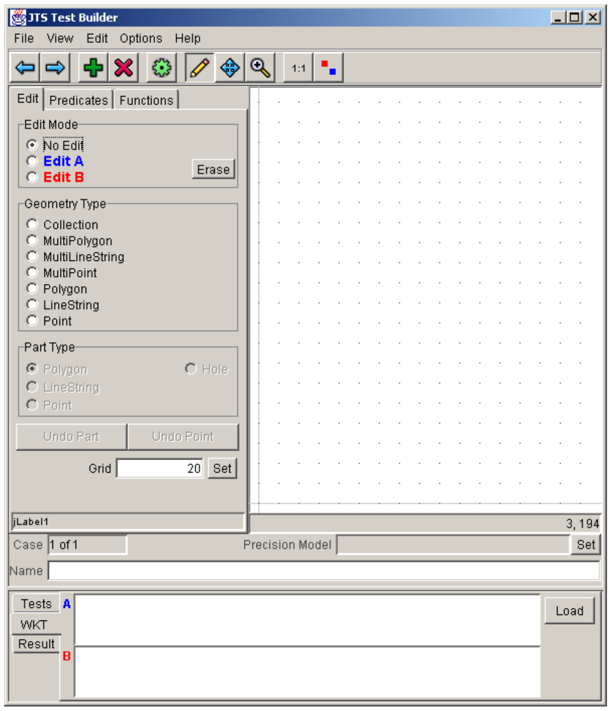
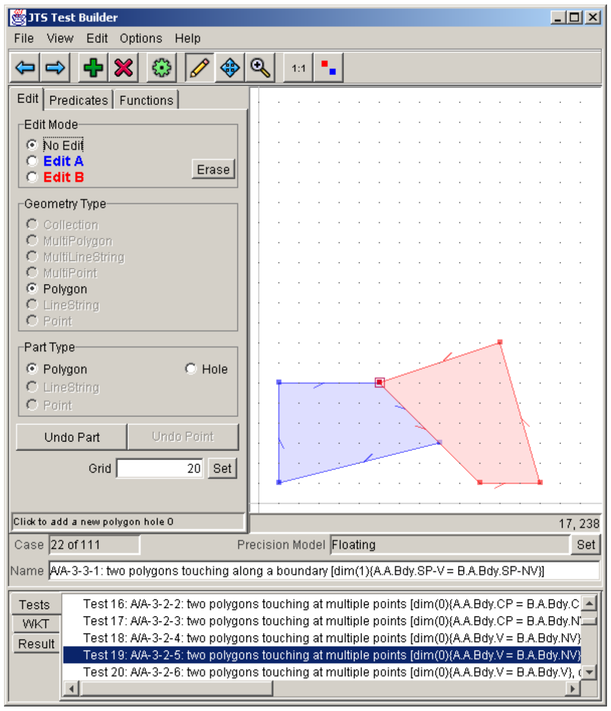


Figure 3-1 – TestBuilder on startup

3.2 OPENING A TEST FILE

To open an existing test XML file, click File / Open XML File(s). The `test` directory contains sample files that you can try opening. In Figure 3-2 below, the TestBuilder has been used to open `TestRelateAA.xml`.



Grid 20

Click to add a new polygon hole 0

17, 238

Case 22 of 111

Precision Model Floating

Name A/A-3-3-1: two polygons touching along a boundary [dim(1){A.A.Bdy.SP-V = B.A.Bdy.SP-NV}]

Test 16: A/A-3-2-2: two polygons touching at multiple points [dim(0){A.A.Bdy.CP = B.A.Bdy.C
Test 17: A/A-3-2-3: two polygons touching at multiple points [dim(0){A.A.Bdy.CP = B.A.Bdy.N
Test 18: A/A-3-2-4: two polygons touching at multiple points [dim(0){A.A.Bdy.V = B.A.Bdy.NV}
Test 19: A/A-3-2-5: two polygons touching at multiple points [dim(0){A.A.Bdy.V = B.A.Bdy.NV}
Test 20: A/A-3-2-6: two polygons touching at multiple points [dim(0){A.A.Bdy.V = B.A.Bdy.V}, (

Figure 3-2 – The TestBuilder, after opening a file

The TestBuilder displays one test case at a time. A **test case** consists of a pair of geometries and their tests. A **test** is an operation and its expected result, when performed on the two geometries. Click on the Previous button or Next button to cycle between test cases. To jump to a particular test, click it in the **Tests** tab at the bottom of the TestBuilder.

Use the Pan , Zoom In/Out , Zoom 1:1 and Zoom to Full Extent tools to examine the geometries. To see the Well-known Text for the geometries, click the **WKT** tab at the bottom of the TestBuilder.

3.3 SETTING THE PRECISION MODEL

A test file's precision model is set using Edit / Precision Model. The default setting is Floating. For more information on precision models, see the *JTS Technical Specifications* supplied with JTS.

3.4 CREATING GEOMETRIES

To create a new test case:

On the **Edit** tab, click on **Edit A** in the **Edit Mode** panel, then select a geometry type from the **Geometry Type** panel
Click on the **Edit** button on the tool bar to start editing geometry A in the display window
Continue with **Edit B** and edit geometry B

Geometry A is shown in blue; geometry B, in red. The vector direction is indicated on each line segment. For a polygon, the interior area is shaded.

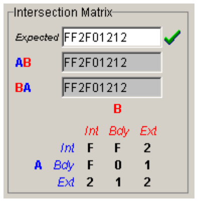
The grid to which points snap can be set by entering a new number and clicking **Set**. An entire geometry can be erased by clicking **Erase**. The Undo Point and Undo Part buttons are used to undo a point or part of a geometry.

You can also create or edit geometries by entering text in the WKT tab, at the bottom of the TestBuilder.

A test case can be deleted (permanently) using the ✖ button.

3.5 SPECIFYING EXPECTED INTERSECTION MATRIX VALUES

To set the expected intersection matrix, click the **Predicates** tab. Type the expected intersection matrix in the **Expected** edit box (you can leave it blank if you merely wish to view the actual intersection matrix). Click on **Run** to display the actual intersection matrix (see Figure 3-3 below).



Expected: FF2F01212 ✓

AB: FF2F01212

BA: FF2F01212

		B		
		Int	Bdy	Ext
	Int	F	F	2
A	Bdy	F	0	1
	Ext	2	1	2

Figure 3-3 – The Intersection Matrix panel

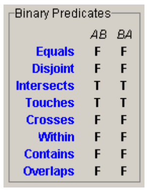
If the actual intersection matrix matches the expected value, a green tick will appear; otherwise, a red cross will appear. The expected value will be saved when you save the test file.

The intersection matrix is displayed in three different ways:

A with respect to B, as a string of characters
B with respect to A, as a string of characters
A with respect to B, as a matrix

For more information on the intersection matrix see the *OpenGIS® Simple Features Specification for SQL* (http://www.opengis.org/techno/specs.htm).

Below the Intersection Matrix panel is the Binary Predicates panel, which shows the values of various binary predicates of A with respect to B, and B with respect to A (see Figure 3-4 below).



Binary Predicates

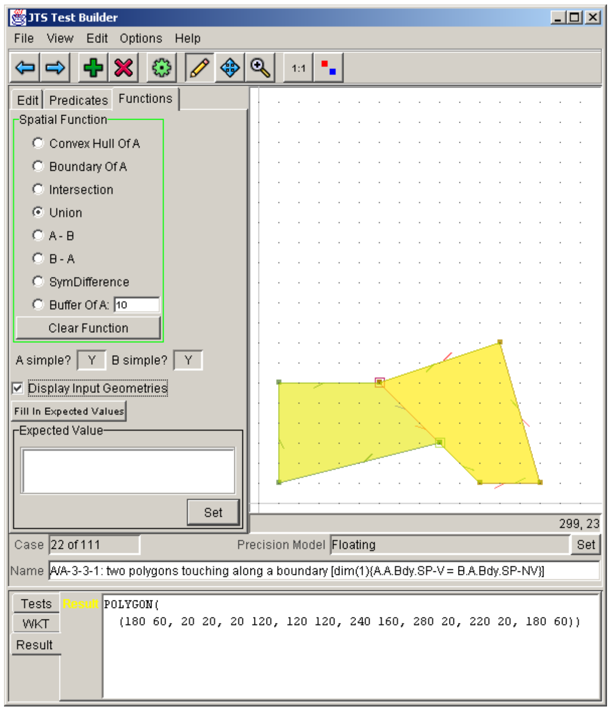
	AB	BA
Equals	F	F
Disjoint	F	F
Intersects	T	T
Touches	T	T
Crosses	F	F
Within	F	F
Contains	F	F
Overlaps	F	F

Figure 3-4 – The Binary Predicates panel

The TestBuilder cannot set the expected values of binary predicates; they must be set by editing the test file (see the example in *Appendix: Test File Format* on page 17)

3.6 SPECIFYING EXPECTED SPATIAL FUNCTION VALUES

To set the expected values or view the actual values for spatial functions, begin by clicking the **Functions** tab. Pick one of the items in the **Spatial Functions** panel e.g. Union. The actual value of the function will be displayed in yellow (see Figure 3-5 below).



Union selected; Buffer Of A: 10

A simple? Y B simple? Y

☑ Display Input Geometries

299, 23

Case 22 of 111 Precision Model Floating Set

Name A/A-3-3-1: two polygons touching along a boundary [dim(1){A.A.Bdy.SP-V = B.A.Bdy.SP-NV}]

Result

```
POLYGON(
  (180 60, 20 20, 20 120, 120 120, 240 160, 280 20, 220 20, 180 60))
```

Figure 3-5 – Union of two geometries displayed in yellow

To set the expected value of the spatial function, enter its well known text in the **Expected Value** edit box, and press **Set**. If the expected value matches the actual value, a green tick will appear beside the spatial function name; otherwise, a red cross will appear. The expected values you enter will be saved when you save the test file.

To quickly set the expected values of the spatial functions to their actual values, press **Fill In Expected Values**. You should check that the generated values are correct.

To hide the blue A and the red B geometries while spatial functions are being displayed, uncheck the **Display Input Geometries** checkbox.

To view the Well-known Text of the spatial function's actual value, click the **Result** tab at the bottom of the TestBuilder.

3.7 SAVING A TEST FILE

To save the current tests, simply click File / Save As XML, and enter a filename.

3.8 GENERATING HTML

The TestBuilder can generate HTML documentation for your test cases. Simply load all your test files into the TestBuilder, click File / Save As HTML, and pick an empty directory. HTML generation can take 10 minutes for 300 test cases, creating hundreds of .gif and .html files. The top-level file is `index.html`.

You will be prompted with the question, "Would you like the spatial function images to show the A and B geometries?" If you answer Yes, the A and B geometries will appear on images of spatial functions. Typically one would choose No.

The only predicates and spatial functions that will be documented are those for which you have specified expected results in your test files. However, the HTML will display actual results, not expected results.

To display the HTML files properly, place the `jts.css` file in the parent directory of the directory containing the HTML files. The `jts.css` file is included with JTS.

For an example of HTML generated by the TestBuilder, see http://www.vividsolutions.com/jts/tests/index.html.

4. APPENDIX: TEST FILE FORMAT

The section describes the format of test files read by the TestRunner. Test files are XML files that can be edited using any text editor. The easiest way to create a test file is to use the TestBuilder (see *3 Using The TestBuilder* on page 11).

A test file contains one or more test cases, each of which contains one or more tests. A **test case** specifies two geometries to analyze e.g. two polygons. Each **test** specifies (1) an operation to perform on the geometries and (2) its expected result e.g. that the `intersects` operation returns `true`.

Section 4.1 provides an example of a simple test file. Section 4.2 defines the XML format of a test file.

4.1 EXAMPLE

Below is a simple example of a test file.

Listing 4-1 – A simple JTS test file

```
<run>
  <desc>example</desc>
  <precisionModel type="FIXED" scale="1" offsetx="0" offsety="0" />
  <case>
    <desc>point in a polygon</desc>
    <a>POLYGON ((0 0, 10 0, 10 10, 0 10, 0 0))</a>
    <b>POINT (5 5)</b>
    <test>
      <op name="contains">true</op>
    </test>
    <test>
      <op name="contains" arg1="b" arg2="a">false</op>
    </test>
  </case>
  <case>
    <desc>two lines that cross</desc>
    <a>LINESTRING (0 0, 10 10)</a>
    <b>LINESTRING (0 10, 10 0)</b>
    <test>
      <op name="crosses">true</op>
    </test>
  </case>
</run>
```

This test file has two test cases:

point in a polygon
two lines that cross

The first test case has two tests:

that polygon A contains point B
that point B does not contain polygon A

The second test case has one test:

that line A and line B cross

Not shown here are **spatial function tests**, which return a geometry (e.g. `POINT (5 5)`) instead of `true` or `false`. For examples of spatial function tests, see the test files in the `test` directory.

4.2 XML SPECIFICATION

Listing 4-2 below specifies the syntax of a JTS test file. The notation used is an extension of Backus Naur Form.

Listing 4-2 – XML format of a JTS test file

```
testFile  ::=
     <run>
          [ <desc> text </desc> ]
          [ <workspace dir="directory"/> ]
          [ <tolerance> double </tolerance> ]
          <precisionModel
               type = "type" [scale="double" offsetx="double" offsetY="double"] />
          { caseText }
     </run>

caseText  ::=
     <case>
          [ <desc> text </desc> ]
          <a> well-known-text </a> | <a file="filename"/>
          <b> well-known-text </b> | <b file="filename"/>
          { testText }
     </case>

testText  ::=
     <test>
          [ <desc> text </desc> ]
          <op name="opName"
                    arg1="geometry-index"
                    [ arg2="arg" ]
                    [ arg3 | pattern ="arg" ] >
               result
          </op>
     </test>

opName  ::=
    equals | disjoint | intersects | touches | crosses |
    within | contains | overlaps   | relate  | isSimple  |
    convexHull | intersection | union | difference | symDifference |
    getBoundary| buffer | distance | getArea | isEmpty | isValid |
    isWithinDistance | getEnvelope | distance | getLength
    getDimension | getBoundaryDimension | getNumPoints | getSRID

result ::= boolean | integer | double | well-known-text

boolean ::= true | false

arg ::= geometry-index | null | integer | double | string

geometry-index ::= A | B

type ::= FIXED | FLOATING
```

A test file begins and ends with `<run>` and `</run>` tags. A **run** contains one or more **test cases**. A **test case** contains one or more **tests**. Runs, test cases and tests can have a description, enclosed in `<desc>` tags.

A run must specify a precision model. For more information on precision models, see the *JTS Technical Specifications* supplied with JTS. The precision model can be either fixed precision or floating-point precision. If fixed-precision, the precision model must specify a scale and offset.

A test case specifies one or two geometries (e.g. `POINT (5 5)` and `POINT (10 20)`). Most of the operations (e.g. `overlaps`) require two geometries. Some operations (`isSimple`, `convexhull`, `getBoundary`) need only one geometry.

A test specifies an operation to validate (e.g. `crosses`) and its expected value (e.g. `true` or `POINT (5 10)`). You can also specify the geometry to perform the operation on i.e. `A` or `B` (in `arg1`), as well as any parameters to pass to the operation (in `arg2` and `arg3`). The default values for `arg1` and `arg2` are `A` and `B`.

"POINT (10 10)" is an example of **Well-known Text**. For more information on the Well-known Text format, see *SQL Textual Representation of Geometry* in the *OpenGIS® Simple Features Specification for SQL*[1].

If desired, the Well-known Text for A or B may be stored in a separate file. The filename is specified in the `file` attribute. The file's directory may also be specified in the file attribute, or in the `workspace` tag. If no directory is specified, it is taken to be the XML file's directory.

The `relate` operation's `arg3` must be a pattern e.g. `1010F0102`. For backwards compatibility, if the operation is `relate`, you can use the keyword `pattern` as an alternative to the keyword `arg3`. For more information on the relate operation, see *The Dimensionally Extended Nine-Intersection Model* in the *OpenGIS® Simple Features Specification for SQL*.

[1] http://www.opengis.org/techno/specs.htm

The following operations return `true` or `false`: `equals`, `disjoint`, `intersects`, `touches`, `crosses`, `within`, `contains`, `overlaps`, `relate`, `isSimple`, `isEmpty`, `isValid`, `isWithinDistance`.

The following operations return a geometry: `convexHull`, `intersection`, `union`, `difference`, `symDifference`, `getBoundary`, `buffer`, `getEnvelope`. `buffer`'s `arg3` must be a distance.

The following operations return a double: `distance`, `getArea`, `getLength`.

The following operations return an integer: `getBoundaryDimension`, `getDimension`, `getNumPoints`, `getSRID`.

The optional `<tolerance>` tag specifies the tolerance used when comparing integer, double, and geometry results to their expected values. A geometry result is considered to match its expected value if the corresponding vertices are no farther than the tolerance.

5. APPENDIX: ACME LICENCE

Note: The TestBuilder uses the Acme GifEncoder, which requires that the licence below be included in this documentation.

Visit the ACME Labs Java page for up-to-date versions of this and other fine Java utilities: http://www.acme.com/java/