

ISO Geometry Research

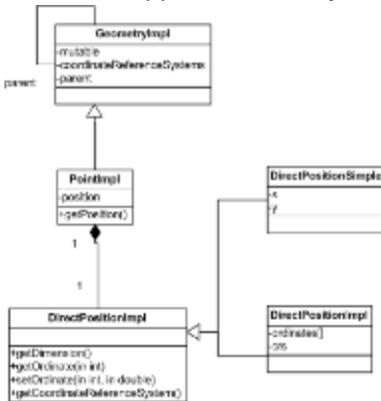
There are two implementations available on trunk:

- [Geometry](#) - an experimental module implementing ISO 19107 (a.k.a. Topic 1 - Feature Geometry). This implementation is a project of Prof. Dr. Jackson Roehrig and Sanjay Dominik Jena of the University of Applied Sciences Cologne, Germany (Fachhochschule Köln).
- [JTS Wrapper](#) - JTS Wrapper provides mission critical implementations of ISO 19107 GeoAPI interfaces backed by the Java Topology Suite, empowering other GeoTools modules to migrate off of direct dependency on JTS.

Implementation Options for ISO 19107 Geometry

Direct Implementation - Approach used by [Geometry](#) module

This is the approach used by the unsupported/geometry module.

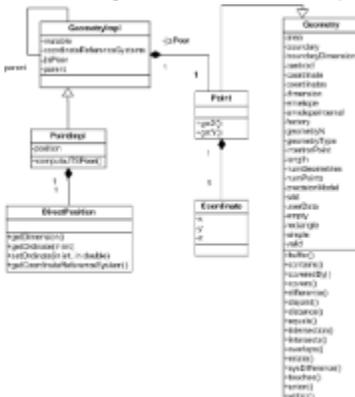


There is a lot to be said for an easy to understand implementation - it is simple sweet and direct. No duplication, no JTS at all. There are ways not to throw away the years of work placed into JTS (The team appears to have ported some of the JTS code over to the new geometry abstraction, although the the testing framework would be a bigger win in terms of trust).

This is a **lot of work** but represents a nice long term solution if a committed team is around for years to come, we can cheat time by stealing the JTS test cases (ie high value), but this approach has the great risk but **will be done eventually**.

Delegate implementation to JTS - Approach used by [JTS Wrapper](#)

This design is used in the implementation donated by SYS Technologies.

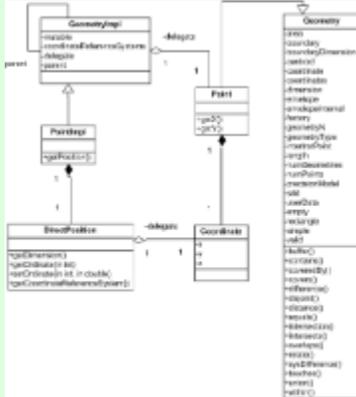


This design is a direct wrapper around JTS **but** JTS is not the final authority on topological information. Instead a JTS object is produced as needed from the information held in the wrapper object - this may result in greater performance (but does have the danger of getting out of sync).

This approach is **good** and is also used by the gvSig team (where they put off creation of the JTS object until (and only if) a complex topological function needs to be called. This is **fast** when done right.

✓ **Alternative**

This is only presented as a reference point, no implementation exists.



A simple and quick way to get an implementation out the door quickly would be to wrap JTS and delegate all the topological data (ie the coordinates) to the JTS classes for safe keeping and stability. Additional metadata required by ISO Geometry (for example CoordinateReferenceSystem) can be held by the wrapper class.

The based location for this code would be as part of the JTS codebase, the downside is that this model goes beyond the "pure topo" mandate of JTS and it does not have a path forward for curve and surface support.

This is **safe** when done right, and is an option for the jts wrapper module if synchronization issues become a problem.

Planning

Making an ISO Geometry Implementation Supported for for 2.4.x

As part of preparing for the roll out of ISO Feature interfaces we will need to set up GeoTools with an implementation of ISO Geometry (the subject of this page).

Criteria for evaluation:

- community - will people be around and interested in this codebase next year?
- complete - is the implementation complete
- quality - is the model accurately represented, how stable are the calculations and so on

There are two implementations:

- [Geometry](#) - looks like the community support is there, the implementation is complete for initial Point, Line, Polygon work ... however MultiGeometry and 3D remains open, the implementation appears accurate and easy to debug but the stability of calculations is not measured

- [JTS Wrapper](#) - this currently represents abandonware (no community), the implementation is complete with respect to SFSQL and has been used in a commercial setting, the module is accurate although has no path for being complete. On the bright side the stability of calculations is the responsibility of another project ... JTS.

ISO Geometry Integration (for [2.5.x](#))

There are two aspects of Geometry that must be accounted for when integrating into the GeoTools codebase:

Filter

GeoTools 2.4 has paved the way here with the design used here for separating out property access from the implementation used. In short an extension point is provided which integrators can use to teach GeoTools about additional geometry formats.

This task is tracked as part of [Expression Improvements](#).

Converter from ISO Geometry to Shape

In order to have the renderer function the existing ExpressionValue construct (and Converter extension point) can be used to morph data between formats according to context. A straight forward application of this idea would be to evaluate the ISO Geometry into a Java 2D Shape construct; we may need to make a special effort to provide the required decimation level to control the process.

Deep Integration with DataStore

We also have the **need for speed**, especially in the reading to renderer department. In an ideal world we would make several 'GeometryFactory' interfaces which the DataStores could make use of as needed.

- SFSQLGeometryFactory - would produce JTS implementations matching the SFSQL specification
- ISOGeometryFactory - would produce ISO Geometry implementations matching the GeoAPI interfaces / TC211 abstractions
- ShapeGeometryFactory - would directly produce Java 2D Shapes at the required level of decimation

It should be noted that this GeometryFactory API will need to:

- Construct the usual suspects (Point, LineString, Polygon)
- Their plurals (MultiPoint, MultiLineString, MultiPolygon, GeometryCollection)
- The additional ISO constructs (Curve, Surface, etc...)
- Be able to directly work with double arrays (as this is the internal format used by many things like Oracle SDO Structs, and Shapefiles)

We may need to produce our own flavour of CoordinateSequence to act as a bridge between what DataStore "Reader" can provide and what GeometryFactory can produce.