Apache Spatial Information System
Hiding some of geospatial complexity

martin.desruisseaux@geomatys.com
“We just need latitude and longitude”
Simple, isn't it?
"Referencing by coordinates" in SIS

Total of about 127,000 lines of code
(ignoring about 170,000 lines of comment)
Is Apache SIS a bloated library?

- Depends on desired accuracy
  - Library should tell which accuracy to expect
  - Library should care about domain of validity
- Depends on desired compliance with authorities
  - Library should not misuse EPSG identifiers
  - Library should favour “late-binding” over “early-binding”
- Depends on inter-operability needs
  - Well Known Text (WKT) versions 1 and 2
  - Geographic Markup Language (GML)

Apache SIS referencing module is based on 20 years of experience.
SIS is currently strongly driven by implementation of OGC standards.
What OGC gives to Apache SIS

- Expertise
  - OGC standards are like blueprints
  - OGC meetings help to understand the “why” and to anticipate

- Interoperability
  - Avoid lock-in to a particular software or vendor

- Tests of correctness
  - Complete the software-specific anti-regression tests

Example: “Referencing by coordinates” without guidance from geodesists is rarely done right. Problems are propagated by the port of popular map projections library to various languages with developers ignoring (conscientiousness or not) fundamental issues.
Various kinds of map projections
(only a few are currently implemented in Apache SIS)

- Equirectangular
- Mercator
- Stereographic
- Orthographic
- Lambert azimuthal equal area
- Lambert conic conformal
- American polyconic
- Mollweide

... and many others (> 100), but

- Mercator
- Transverse Mercator
- Lambert Conic Conformal
- Stereographic

cover a vast amount of cases.
Map projections are only part of work

Proportion of map projection code in Apache SIS referencing module
Geodetic datum

Differ in:
- Axis length
- Eccentricity
- Position of ellipsoid centre
- Rotation between ellipsoids
- And more...

Domains of validity

Images by NASA / USGS
How many?
In EPSG dataset 8.9.2:

- **48** ellipsoids
- **471** geodetic datums
- **464** two-dimensional geographic coordinate reference systems
- **4406** projected coordinate reference systems

Latitudes and longitudes without CRS are ambiguous
3 km error in some parts of the world
"I don't need all this complexity"

Let just use the WGS84 datum everywhere

WGS84 = World Geodetic System 1984
This is the datum used by Global Positioning Systems (GPS)
Not exactly what the maps use

Maps in USA and Canada are relative to the North American Datum (NAD)

- NAD83 is tied to North American Plate
- WGS84 is averaged over the world

→ They do not move in the same way

- 30 years ago, NAD83 ≈ WGS84
- Now, differ by about 1.5 metres

<table>
<thead>
<tr>
<th>NAD83</th>
<th>WGS84</th>
</tr>
</thead>
<tbody>
<tr>
<td>(86)</td>
<td>(Transit)</td>
</tr>
<tr>
<td>(HPGN)</td>
<td>(G730)</td>
</tr>
<tr>
<td>(CORS96)</td>
<td>(G873)</td>
</tr>
<tr>
<td>(2007)</td>
<td>(G1150)</td>
</tr>
<tr>
<td>(2011)</td>
<td>(G1674)</td>
</tr>
<tr>
<td>(CSRS)</td>
<td>(G1762)</td>
</tr>
</tbody>
</table>
Legacy systems are still in use

- Some boundaries in USA are still legally defined in NAD27
- Shift varies for each location, can be close to 80 metres
- If the shape is wide enough, transformation causes distortions

Sometime it is more convenient to work with the original shape
Ellipsoidal height oddities

- Difference with geoidal height up to ±100 metres
- Global datum is not necessarily the best match

In summary
If possible, prefer the reference system defined by the mapping agency of the country where the data are located.
“A simple transformation engine”

Can we use WGS84 as the pivot system?
“Early-binding” approach

Well Known Text (WKT) format version 1:

GEOGCS["NTF",
  DATUM["Nouvelle Triangulation Française",
    SPHEROID["Clarke 1880 (IGN)", 6378249.2, 293.466…],
    TOWGS84[-168, -60, 320],
    PRIMEM["Greenwich", 0],
    UNIT["degree", 0.01745…],
    AXIS["Latitude", NORTH],
    AXIS["Longitude", EAST],
    AUTHORITY["EPSG", "4275"]]
“Late-binding” approach

Well Known Text (WKT) format version 2:

```
GEODCRS["NTF",
  DATUM["Nouvelle Triangulation Française",
    ELLIPSOID["Clarke 1880 (IGN)", 6378249.2, 293.466…]],
  PRIMEM["Greenwich", 0],
  CS[ellipsoidal, 2],
  AXIS["Latitude", north],
  AXIS["Longitude", east],
  UNIT["degree", 0.01745…],
  ID["EPSG", 4275]]
```

TOWGS84 is removed in WKT version 2

BOUNDCRS can be used if a replacement is needed.
What Apache SIS 0.7 can do
Introduction to 9 keys Java methods
Next slides use SIS static convenience methods. Most of them can be replaced by GeoAPI methods. GeoAPI 3.0 provides implementation independence.

http://www.opengeospatial.org/standards/geoapi/
Use definitions from a registry

1) Search online: [http://epsg-registry.org/](http://epsg-registry.org/)
2) Give the EPSG code to Apache SIS
3) Verify if the CRS is the expected one

```java
import org.opengis.referencing.crs.CoordinateReferenceSystem;
import org.apache.sis.referencing.CRS;

// Class declaration omitted for brevity

CoordinateReferenceSystem myDataCRS = CRS.forCode("EPSG:3395");

// Verify the definition (WKT 2 syntax)
System.out.println(myDataCRS);
```

Alternatively, parse GML

```xml
<gml:GeodeticCRS gml:id="epsg-crs-4326">
    <gml:identifier codeSpace="IOGP">urn:ogc:def:crs:EPSG::4326</gml:identifier>
    <gml:name>WGS 84</gml:name>
    <gml:scope>Horizontal component of 3D system. </gml:scope>
    <gml:ellipsoidalCS>
        <gml:EllipsoidalCS gml:id="epsg-cs-6422">
            <gml:identifier codeSpace="IOGP">urn:ogc:def:cs:EPSG::6422</gml:identifier>
            <gml:name>Latitude (north), Longitude (east)</gml:name>
            <gml:axis>
                <gml:CoordinateSystemAxis gml:id="epsg-axis-106" uom="urn:ogc:def:uom:EPSG::9122">
                    <gml:identifier codeSpace="IOGP">urn:ogc:def:axis:EPSG::106</gml:identifier>
                    <gml:name>Geodetic latitude</gml:name>
                    <gml:axisAbbrev>φ</gml:axisAbbrev>
                    <gml:axisDirection codeSpace="EPSG">north</gml:axisDirection>
                    <gml:minimumValue>-90</gml:minimumValue>
                    <gml:maximumValue>+90</gml:maximumValue>
                    <gml:rangeMeaning codeSpace="EPSG">exact</gml:rangeMeaning>
                </gml:CoordinateSystemAxis>
            </gml:axis>
            <gml:axis>
                <gml:CoordinateSystemAxis gml:id="epsg-axis-107" uom="urn:ogc:def:uom:EPSG::9122">
                    <gml:identifier codeSpace="IOGP">urn:ogc:def:axis:EPSG::107</gml:identifier>
                    <gml:name>Geodetic longitude</gml:name>
                    <gml:axisAbbrev>λ</gml:axisAbbrev>
                    <gml:axisDirection codeSpace="EPSG">east</gml:axisDirection>
                    <gml:minimumValue>-180</gml:minimumValue>
                    <gml:maximumValue>+180</gml:maximumValue>
                    <gml:rangeMeaning codeSpace="EPSG">wraparound</gml:rangeMeaning>
                </gml:CoordinateSystemAxis>
            </gml:axis>
        </gml:EllipsoidalCS>
    </gml:ellipsoidalCS>
</gml:GeodeticCRS>

<continuing...>
<gml:geodeticDatum>
    <gml:GeodeticDatum gml:id="epsg-datum-6326">
        <gml:identifier codeSpace="IOGP">urn:ogc:def:datum:EPSG::6326</gml:identifier>
        <gml:name>World Geodetic System 1984</gml:name>
        <gml:scope>Satellite navigation. </gml:scope>
        <gml:primeMeridian>
            <gml:PrimeMeridian gml:id = "epsg-meridian-8901">
                <gml:identifier codeSpace="IOGP">urn:ogc:def:meridian:EPSG::8901</gml:identifier>
                <gml:name>Greenwich</gml:name>
                <gml:greenwichLongitude uom="urn:ogc:def:uom:EPSG::9102">0</gml:greenwichLongitude>
            </gml:PrimeMeridian>
        </gml:primeMeridian>
        <gml:ellipsoid>
            <gml:Ellipsoid gml:id = "epsg-ellipsoid-7030">
                <gml:identifier codeSpace="IOGP">urn:ogc:def:ellipsoid:EPSG::7030</gml:identifier>
                <gml:name>WGS 84</gml:name>
                <gml:semiMajorAxis uom="urn:ogc:def:uom:EPSG::9001">6378137.0</gml:semiMajorAxis>
                <gml:secondDefiningParameter>
                    <gml:SecondDefiningParameter>
                        <gml:inverseFlattening uom="urn:ogc:def:uom:EPSG::9201">298.257223563</gml:inverseFlattening>
                    </gml:SecondDefiningParameter>
                </gml:secondDefiningParameter>
            </gml:Ellipsoid>
        </gml:ellipsoid>
    </gml:GeodeticDatum>
</gml:geodeticDatum>
</gml:GeodeticCRS>
```

CoordinateReferenceSystem myDataCRS = CRS.fromXML("<gml:...>...</gml:>");
Alternatively, parse WKT

```python
CoordinateReferenceSystem myDataCRS = CRS.fromWKT("COMPOUNDCRS[...]");
```
WKT 1 compatibility issue

- Original WKT 1 specification (1999) was unclear about some units
- Newer WKT 1 specification (2001) fixed the ambiguity
  ⚠ Some popular libraries did not followed

Test: ask the library to show “EPSG:4807” definition
(Note: Paris meridian is at about 2.6 grads ≈ 2.3° from Greenwich)

Wrong: PRIMEM["Paris", 2.33722917], UNIT["grad", 0.015707963267948967]
Correct: PRIMEM["Paris", 2.59692130], UNIT["grad", 0.015707963267948967]

- Inconsistent units result in 17 km error at Paris latitude
- Same issue with map projection parameters and feet units
WKT migration compatibility

WKT 1 specified in:
- OGC 99-036
- ISO 19125-1:2004

Interpreted by:
- GDAL

WKT 1 extended in:
- OGC 01-009

Used in:
- GeoPackage 1.0
- GeoAPI 3.0

Compatible under conditions

Compatible with exceptions
- TOWGS84 dropped

Compatible
- unit clarified

Compatible with exceptions under conditions

WKT 2 specified in:
- OGC 12-063
- ISO 19162:2015

Apache SIS 0.6+

- SIS read automatically
- SIS write if asked to

- SIS read automatically
- SIS write if asked to

- SIS read automatically
- SIS write automatically
Find coordinate operation

1) Get two CRS (source and target)
2) Get a coordinate operation from source to target
3) Verify domain of validity and positional accuracy

```java
import org.opengis.referencing.operation.CoordinateOperation;

// Class declaration omitted for brevity

CoordinateReferenceSystem sourceCRS = // any method shown in previous slides
CoordinateReferenceSystem targetCRS = // any method shown in previous slides
CoordinateOperation op = CRS.findOperation(sourceCRS, targetCRS, null);

// Verify domain of validity and accuracy
System.out.println("Valid in " + CRS.getGeographicBoundingBox(op));
System.out.println("Accuracy " + CRS.getLinearAccuracy(op) + " m");
```
“Find operation” detailed

Step 1: get EPSG identifiers of both CRS

- Do **not** trust the identifier declared in WKT 1 definition
- Search a matching CRS in EPSG dataset even if no identifier is declared
- Axis order and units may not match (lot of WKT are non-conform)
- Map projection parameters may be declared in different ways

**Example:** Apache SIS can recognise that the following are numerically equivalent:

```
PROJCRS["Scaled Mercator",
  ...
  CONVERSION["Scaled Mercator",
    METHOD["Mercator (variant A)",
    PARAMETER["Scale factor at natural origin", 0.7558...],
    PARAMETER["Longitude of natural origin", 100]],
  ...
]

PROJCRS["WGS 84 / Mercator 41",
  ...
  CONVERSION["Mercator 41",
    METHOD["Mercator (variant B)",
    PARAMETER["Latitude of 1st standard parallel", -41],
    PARAMETER["Longitude of natural origin", 100]],
  ...
  , ID["EPSG", 3994]]
```
“Find operation” detailed

Step 2: find operation paths specified by EPSG

• Use the EPSG identifiers found at step 1
• If EPSG dataset declares more than one operation, then:
  ▪ Choose widest intersection between:
    – area of interest (specified by user)
    – domain of validity (specified by EPSG)
  ▪ If same intersection, choose according accuracy
• Adjust for axis order and units

Work also with three-dimensional geographic CRS
(SIS can expand the search to the two-dimensional domain)
“Find operation” detailed

Step 3: infer operation paths not specified by EPSG

- Separate some CRS components (e.g. temporal)
- Change of axis units (e.g. feet to metres)
- Change of axis directions (e.g. westing to easting)
- Change of axis order (e.g. to right-handed system)
- Change of coordinate system type (e.g. polar to Cartesian)
- Derived CRS ↔ base CRS conversion (e.g. map projection)
- Datum shift using TOWGS84-like information if it exists
"Find operation" example

\[(\varphi, \lambda, t)\]

NAD27

EPSG: 4267

Julian day

\[(x, y, t_2)\]

WGS 84 / World Mercator

EPSG: 3395

Truncated Julian day

"NAD27 to WGS84 (3)" coordinate operation (EPSG: 1172) — for Canada

NAD27

\((\varphi, \lambda)\)

EPSG: 4267

Right-handed NAD27 \((\lambda, \varphi)\)

NAD27 as 3-D

\((\lambda, \varphi, h_1)\)

Geocentric

\((X_1, Y_1, Z_1)\)

Datum shift

\[\Delta X = -10 \text{ m} \]

\[\Delta Y = 158 \text{ m} \]

\[\Delta Z = 187 \text{ m} \]

\[\text{Mercator (variant A)}\]

\[\text{based on WGS84}\]

EPSG: 4326

Truncated Julian day \((t_2)\)

Right-handed WGS84 \((\lambda_2, \varphi_2)\)

WGS84 as 3-D

\((\lambda_2, \varphi_2, h_2)\)

Geocentric

\((X_2, Y_2, Z_2)\)
```java
System.out.println(op);
```

## Well Known Text (WKT) version 2

```
CoordinateOperation["NAD27 to WGS 84 (3)",
SourceCRS[GeodeticCRS["NAD27",
  Datum["North American Datum 1927",
    Ellipsoid["Clarke 1866", 6378206.4, 294.9786982138982]],
  CS[ellipsoidal, 2],
  Axis["Geodetic latitude (Lat)", north],
  Axis["Geodetic longitude (Lon)", east],
  Unit["degree", 0.017453292519943295],
  Id["EPSG", 4267, "8.9"],
TargetCRS[... definition omitted for brevity ...],
Method["Geocentric translations (geog2D domain)",
  Parameter["X-axis translation", -10, Unit["metre", 1]],
  Parameter["Y-axis translation", 158, Unit["metre", 1]],
  Parameter["Z-axis translation", 187, Unit["metre", 1]],
OperationAccuracy[20],
Scope["Accuracy 15m, 11m and 6m in X, Y and Z axes.",
Area["Canada - onshore and offshore.",
BBox[40.04, -141.01, 86.46, -47.74],
Id["EPSG", 1172, "8.9"],
URI["urn:ogc:def:coordinateOperation:EPSG:8.9:1172"],
Remark["Derived at 112 stations."])]
```

... map projection part omitted ...
Coordinate operation execution

MathTransform mt =
  op.getMathTransform();
System.out.println(mt);

Axis order change from (φ, λ) to right-handed

\[
\begin{bmatrix}
0 & 1 & 0 \\
1 & 0 & 0 \\
0 & 0 & 1
\end{bmatrix}
\]

Well Known Text (WKT) version 1

Concat_MT[
  Param_MT["Affine parametric transformation",
    Parameter["A0", 0.0],
    Parameter["A1", 1.0],
    Parameter["B0", 1.0],
    Parameter["B1", 0.0]],
  Inverse_MT[Param_MT["Geographic3D to 2D conversion"],
    Param_MT["Geographic/geocentric conversions",
      Parameter["semi_major", 6378206.4, Unit["metre", 1]],
      Parameter["semi_minor", 6356583.8, Unit["metre", 1]]],
  Param_MT["Geocentric translations (geocentric domain)",
    Parameter["X-axis translation", -10.0, Unit["metre", 1]],
    Parameter["Y-axis translation", 158.0, Unit["metre", 1]],
    Parameter["Z-axis translation", 187.0, Unit["metre", 1]]],
  Param_MT["Geocentric_To_Ellipsoid",
    Parameter["semi_major", 6378137.0, Unit["metre", 1]],
    Parameter["semi_minor", 6356752.314245179, Unit["metre", 1]]],
  Param_MT["Geographic3D to 2D conversion"],
  Param_MT["Mercator (variant A)",
    Parameter["semi_major", 6378137.0, Unit["metre", 1]],
    Parameter["semi_minor", 6356752.314245179, Unit["metre", 1]]]]
Execution internal (what SIS really does)

$$\begin{pmatrix}
\pi/180 & 0 & 0 \\
0 & \pi/180 & 0 \\
0 & 0 & 1 \\
0 & 0 & 1
\end{pmatrix}
\times
\begin{pmatrix}
1 & 0 & 0 \\
0 & 1 & 0 \\
0 & 0 & h_1 \\
0 & 0 & 1
\end{pmatrix}
\times
\begin{pmatrix}
0 & 1 & 0 \\
1 & 0 & 0 \\
0 & 0 & 1
\end{pmatrix}
= \begin{pmatrix}
0 & \pi/180 & 0 \\
\pi/180 & 0 & 0 \\
0 & 0 & h_1 \\
0 & 0 & 1
\end{pmatrix}$$

Pseudo-WKT 1

```
Concat_MT[
  Param_MT["Affine parametric transformation",
    Parameter["A0", 0.0],
    Parameter["A1", 0.017453292519943295],
    Parameter["B0", 0.017453292519943295],
    Parameter["B1", 0.0]],
  Param_MT["Ellipsoid (radians domain) to centric",
    Parameter["eccentricity", 0.0822718542230039]],
  Param_MT["Affine",
    Parameter["num_row", 4],
    Parameter["num_col", 4],
    Parameter["elt_0_0", 1.0000108809202437],
    Parameter["elt_0_3", -1.567855942887398E-6],
    Parameter["elt_1_1", 1.0000108809202437],
    Parameter["elt_1_3", 2.4772123897620886E-5],
    Parameter["elt_2_2", 1.0000108809202437],
    Parameter["elt_2_3", 2.931890613199434E-5]],
  Param_MT["Centric to ellipsoid (radians domain)",
    Parameter["eccentricity", 0.08181919084262157]],
  Param_MT["Mercator (radians domain)",
    Parameter["eccentricity", 0.0818191908426215]],
  Param_MT["Affine parametric transformation",
    Parameter["A0", 6378137],
    Parameter["B1", 6378137]]]
```
Transformation outputs

Example for a two-dimensional map projection:

\[
\text{transform}(\phi, \lambda) : \begin{pmatrix} x \\ y \end{pmatrix}
\]

\[
\text{derivative}(\phi, \lambda) : \begin{pmatrix} \frac{\partial x}{\partial \phi} & \frac{\partial x}{\partial \lambda} \\ \frac{\partial y}{\partial \phi} & \frac{\partial y}{\partial \lambda} \end{pmatrix}
\]

Number of rows or columns depend on the number of dimensions
Transforming points

- **MathTransform** is the type doing the actual work
  - Unmodifiable and thread-safe
  - Same `transform(...)` methods than `java.awt.geom.AffineTransform`

```java
import org.opengis.referencing.operation.MathTransform;

// Class declaration omitted for brevity

CoordinateOperation op = CRS.findOperation(sourceCRS, targetCRS, null);
MathTransform mt = op.getMathTransform();

double[] sourcePts = {φ₁, λ₁, φ₂, λ₂, φ₃, λ₃, φ₄, λ₄, …};
double[] targetPts = new double[2 * numPts];
mt.transform(sourcePts, 0, targetPts, 0, numPts);
```
Transforming envelopes

• Need smallest box containing the transformed shape
  - Transforming the 4 corners is not sufficient
  - SIS uses transform derivatives (Jacobian)

```java
import org.opengis.geometry.Envelope;
import org.apache.sis.geometry.Envelopes;
import org.apache.sis.geometry.Envelope2D;

// Class declaration omitted for brevety
Envelope envelope = new Envelope2D(xmin, ymin, width, height);
Envelope transformed = Envelopes.transform(op, envelope);
```
Transforming envelopes

- Need smallest box containing the transformed shape
  - Transforming the 4 corners is not sufficient
  - SIS uses transform derivatives (Jacobian)
Summary

• Complexity depends on accuracy needs:
  • For an accuracy better than 1 kilometre, datum cannot be ignored.
  • For an accuracy of a few metres, local systems may be preferable.

• Some libraries / applications do not know their accuracy.

• Transforming everything to a "pivot" system is not always desirable:
  • The pivot may be a moving target, while direct and better defined paths exist.
  • Working with data in their original (legacy) system can avoid geometric distortion.

• Envelope transformations need to take curvature in account.

• If still unsure about whether SIS is worth its weight, consider using GeoAPI.

• If using WKT 1 format, safer to restrict to metre and degree units
  • Unless the library is known to be OGC 01-009 compliant.
Some SIS key static methods

- Get your `CoordinateReferenceSystem` instances:

  ```java
  CoordinateReferenceSystem myDataCRS = CRS.forCode("EPSG:3395");
  CoordinateReferenceSystem myDataCRS = CRS.fromWKT("PROJCRS[...]");
  ```

- Get your `CoordinateOperation` instance:

  ```java
  CoordinateOperation op = CRS.findOperation(source, target, aoi);
  ```

- Is the operation suitable to your needs?

  ```java
  double positionalAccuracy = CRS.getLinearAccuracy(op);
  GeographicBoundingBox domain = CRS.getGeographicBoundingBox(op);
  ```

- Transform your data

  ```java
  Envelope transformed = Envelopes.transform(op, envelope);
  ```
Thanks!
ありがとう
Merci!

http://www.opengeospatial.org/
http://www.geoapi.org/
http://sis.apache.org