GeoAPI changes

TITLE:	**GeoAPI specific changes**
AUTHORS:	**Martin Desruisseaux, Adrian Custer (Geomatys)** **Jody Garnett, Cory Horner,** **Graham Davis (Refractions Research)**
DATE:	**June, 2007**
CATEGORY:	**Change Proposal**

Table of Contents

1. Background

This document tracks the changes submitted since GeoAPI 2.0 release. The changes presented in this proposal are intended to bring the GeoAPI interfaces back into close conformance with the updated OGC Abstract Specifications. The last, and therefore current, release of the GeoAPI library, version 2.0, was made on June 7th 2005. That release brought the GeoAPI interfaces into conformance with OGC Abstract Specifications based on the ISO 19000 series of specifications available at that time. Since then, several OGC Abstract Specifications have been updated. Also in this same interval, several minor issues in the GeoAPI interfaces have been identified. It is therefore desirable to update the GeoAPI interfaces.

The proposal presents two sets of modifications to the GeoAPI 2.0 Java interfaces. These modifications would require small updates to the UML diagrams in the OGC GO-1 Implementation Specification document (03-064r10). These changes do not involve any changes to the OGC Abstract Specifications themselves.

If accepted, the changes would lead to two new releases of the GeoAPI interface library, tentatively numbered 2.1 and 2.2.

- GeoAPI 2.1 would include a series of minor modifications that are upward compatible for interface users.
- GeoAPI 2.2 would include modifications that require changes in the source code of interface users.

Note that in both cases, the changes are incompatible for implementors. It is close to impossible to change an interface in the Java language without causing a compatibility break for implementors. In this document, “compatibility” is always to be understood from the point of view of interfaces users.

Known GeoAPI implementors include the GeoTools and JScience open source projects. The former (GeoTools) already applied the proposed changes since our policy is to test the proposals in at least one (ideally two) implementations before to bring them to OGC. The later (JScience) implements only the “coordinate reference systems” part, which is probably the most stable package in GeoAPI.

1.1 Scope of this document

This document presents changes that are GeoAPI extensions or address issues specific to the Java language. The changes that aims only to bring GeoAPI into conformance with latest specifications are presented in a separated document.

2. Contributors

The following organizations (in alphabetical order) have contributed to GeoAPI development through participation of individual employees:

- Axios Engineering
- Geomatys
- GeoSolutions
- Institut de Recherche pour le développement (IRD)
- Leica Geosystems Geospatial Imaging, LLC (LGGI)
- Refractions Research
- The Open Planning Project (TOPP)
- United States Forest Service (USFS)
- University of Applied Sciences Cologne

3. References

[1] GeoAPI interfaces: http://geoapi.sourceforge.net/2.0/javadoc

[2] ISO-19103: Conceptual schema language, revisions for 2003 and 2005

[3] ISO-19107: Feature geometry

[4] ISO-19111: Spatial referencing by coordinates, revisions 03-073r1 and 04-046r3

[5] ISO-19115: Geographic information – Metadata, revisions for 2003 and 2006

[6] OGC 03-064r10: GO-1 Application Objects

Every change proposed in this document has been applied to the "snapshot" javadoc which can be browsed on-line here:

[7] http://geoapi.sourceforge.net/snapshot/javadoc/

Please note that not all interfaces in the above-cited javadoc are covered in this Request For Changes.

4. Metadata

4.1 Return Collection <? extends Object>

Enable Object-Relational mapping for frameworks like Hibernate by modifying the constraint parameter of the return types from 'Collection <Object>' to 'Collection <? extends Object>'
JIRA task: http://jira.codehaus.org/browse/GEO-94
JIRA task: http://jira.codehaus.org/browse/GEO-54

The use of Java Generics has a number of consequences depending on how and where they are used. Through use we have discovered a number of subtle trade offs when combining Java Collections with type narrowing.

In GeoAPI 2.0 we often made use of a return type of the form `Collection<Identifier>`. Java type narrowing let's us further specify this relationship in a subclass to something more specific - `List<Identifier>`. It does not however let us narrow the types of both the Collection and the Element.

For GeoAPI 3.0 we propose changing the API in several places to be of the form `Collection<? extends Identifier>`. With this syntax subclasses are free to narrow both the Collection type and the element type – `List<ReferenceIdentifier>`.

The ability to type narrow the Collection and Element type is also available for implementations, allowing them to hold their element type fixed if needed. So far implementations have found reason to request this ability for reasons of mutability, persistence and event notification. We must stress that this need resulted in the initial change request, the final decision is based on allowing our interfaces to be as specific as possible.

4.1.1 Affected section(s), table(s), and figure(s)

The changes will not alter any specification documents.

4.1.2 Purposes of the proposed change

This is a compromise for facilitating metadata implementation using some frameworks like Hibernate.

4.1.3 Reasons for change

This is a request from GeoAPI users who tried to implement metadata interfaces using the Hibernate framework.

4.1.4 Specific suggested changes

The constraint parameters in the return types of the following methods were changed from '`Collection <Type>`' to '`Collection < ? extends Type>`'

```
metadata.ExtendedElementInformation.getRationales()
       returns      Collection<? extends InternationalString>
       not          Collection<InternationalString>
metadata.ExtendedElementInformation.getSources()
       returns      Collection<? extends ResponsibleParty>
       not          Collection<ResponsibleParty>
metadata.MetaData.getSpatialRepresentationInfo()
       returns      Collection<? extends SpatialRepresentation>
       not          Collection<SpatialRepresentation>
metadata.MetaData.getReferenceSystemInfo()
       returns      Collection<? extends ReferenceSystem>
       not          Collection<ReferenceSystem>
metadata.MetaData.getMetadataExtensionInfo()
       returns      Collection<? extends MetadataExtensionInformation>
       not          Collection<MetadataExtensionInformation>
metadata.MetaData.getIdentificationInfo()
       returns      Collection<? extends Identification>
       not          Collection<Identification>
metadata.MetaData.getContentInfo()
       returns      Collection<? extends ContentInformation>
       not          Collection<ContentInformation>
metadata.Metadata.getDataQualityInfo()
       returns      Collection<? extends DataQuality>
       not          Collection<DataQuality>
metadata.MetaData.getPortrayalCatalogueInfo()
       returns      Collection<? extends PortrayalCatalogueReference>
       not          Collection<PortrayalCatalogueReference>
metadata.MetaData.getMetadataConstraints()
       returns      Collection<? extends Constraints>
       not          Collection<Constraints>
metadata.MetaData.getApplicationSchemaInfo()
       returns      Collection<? extends ApplicationSchemaInformation>
       not          Collection<ApplicationSchemaInformation>
metadata.MetadataExtensionInformation.getExtendedElementInformation()
       returns      Collection<? extends ExtendedElementInformation>
       not          Collection<ExtendedElementInformation>
metadata.PortrayalCatalogueReference.getPortrayalCatalogueCitations()
       returns      Collection<? extends Citation>
       not          Collection<Citation>
metadata.citation.Citation.getAlternateTitles()
       returns      Collection<? extends InternationalString>
       not          Collection<InternationalString>
metadata.citation.Citation.getDates()
       returns      Collection<? extends CitationDate>
       not          Collection<CitationDate>
metadata.citation.Citation.getIdentifiers()
       returns      Collection<? extends Identifier>
       not          Collection<String>
metadata.constraint.Constraints.getUseLimitation()
       returns      Collection<? extends InternationalString>
       not          Collection<InternationalString>
metadata.content.FeatureCatalogueDescription.getFeatureTypes()
       returns      Collection<? extends GenericName>
       not          Collection<GenericName>
metadata.content.FeatureCatalogueDescription.getFeatureCatalogueCitations()
       returns      Collection<? extends Citation>
       not          Collection<Citation>
metadata.distribution.DigitialTransferOptions.getOnLines()
       returns      Collection<? extends OnLineResource>
       not          Collection<OnLineResource>
```

metadata.distribution.**Distribution.getDistributionFormats()**
	returns	Collection<? extends Format>
	not	Collection<Format>
metadata.distribution.**Distribution.getDistributors()**
	returns	Collection<? extends Distributor>
	not	Collection<Distributor>
metadata.distribution.**Distribution.getTransferOptions()**
	returns	Collection<? extends DigitalTransferOptions>
	not	Collection<DigitalTransferOptions>
metadata.distribution.**Distributor.getDistributionsOrderProcesses()**
	returns	Collection<? extends StandardOrderProcess>
	not	Collection<StandardOrderProcess>
metadata.distribution.**Distributor.getDistributorFormats()**
	returns	Collection<? extends Format>
	not	Collection<Format>
metadata.distribution.**Distributor.getDistributorTransferOptions()**
	returns	Collection<? extends DigitalTransferOptions>
	not	Collection<DigitalTransferOptions>
metadata.distribution.**Format.getFormatDistributors()**
	returns	Collection<? extends Distributor>
	not	Collection<Distributor>
metadata.extent.**BoundingPolygon.getPolygons()**
	returns	Collection<? extends Geometry>
	not	Collection<Geometry>
metadata.extent.**Extent.getGeographicElements()**
	returns	Collection<? extends GeographicExtent>
	not	Collection<GeographicExtent>
metadata.extent.**Extent.getTemporalElements()**
	returns	Collection<? extends TemporalExtent>
	not	Collection<TemporalExtent>
metadata.extent.**Extent.getVerticalElements()**
	returns	Collection<? extends VerticalExtent>
	not	Collection<VerticalExtent>
metadata.extent.**SpatialTemporalExtent.getSpatialElements()**
	returns	Collection<? extends GeographicExtent>
	not	Collection<GeographicExtent>
metadata.identification.**DataIdentification.getSpatialResolution()**
	returns	Collection<? extends Resolution>
	not	Collection<Resolution>
metadata.identification.**DataIdentification.getExtent()**
	returns	Collection<? extends Extent>
	not	Collection<Extent>
metadata.identification.**Identification.getPointOfContacts()**
	returns	Collection<? extends ResponsibleParty>
	not	Collection<ResponsibleParty>
metadata.identification.**Identification.getResourceMaintenance()**
	returns	Collection<? extends MaintenanceInformation>
	not	Collection<MaintenanceInformation>
metadata.identification.**Identification.getGraphicOverviews()**
	returns	Collection<? extends BrowseGraphic>
	not	Collection<BrowseGraphic>
metadata.identification.**Identification.getResourceFormat()**
	returns	Collection<? extends Format>
	not	Collection<Format>
metadata.identification.**Identification.getDescriptiveKeywords()**
	returns	Collection<? extends Keywords>
	not	Collection<Keywords>
metadata.identification.**Identification.getResourceSpecificUsages()**
	returns	Collection<? extends Usage>
	not	Collection<Usage>
metadata.identification.**Identification.getResourceConstraints()**
	returns	Collection<? extends Constraints>
	not	Collection<Constraints>
metadata.identification.**Usage.getUserContactInfo()**
	returns	Collection<? extends ResponsibleParty>

```
       not           Collection<ResponsibleParty>
metadata.lineage.Lineage.getProcessSteps()
       returns       Collection<? extends ProcessStep>
       not           Collection<ProcessStep>
metadata.lineage.Lineage.getSources()
       returns       Collection<? extends Source>
       not           Collection<Source>
metadata.lineage.ProcessStep.getProcessors()
       returns       Collection<? extends ResponsibleParty>
       not           Collection<ResponsibleParty>
metadata.lineage.ProcessStep.getSources()
       returns       Collection<? extends Source>
       not           Collection<Source>
metadata.lineage.Source.getSourceExtents()
       returns       Collection<? extends Extent>
       not           Collection<Extent>
metadata.lineage.Source.getSourceSteps()
       returns       Collection<? extends ProcessStep>
       not           Collection<ProcessStep>
metadata.maintenance.ScopeDescription.getAttributes()
       returns       Set<? extends AttributeType>
       not           Set
metadata.maintenance.ScopeDescription.getFeatures()
       returns       Set<? extends FeatureType>
       not           Set
metadata.maintenance.ScopeDescription.getFeatureInstances()
       returns       Set<? extends FeatureType>
       not           Set
metadata.quality.DataQuality.getReports()
       returns       Collection<? extends Element>
       not           Collection<Element>
metadata.quality.Scope.getLevelDescription()
       returns       Collection<? extends ScopeDescription>
       not           Collection<ScopeDescription>
metadata.spatial.Georeferenceable.getParameterCitation()
       returns       Collection<? extends Citation>
       not           Collection<Citation>
metadata.spatial.GridSpatialRepresentation.getAxisDimensionsProperties()
       returns       List<? extends Dimension>
       not           List<Dimension>
metadata.spatial.VectorSpatialRepresentation.getGeometricObjects()
       returns       Collection<? extends GeometricObjects>
       not           Collection<GeometricObjects>
```

4.1.5 Consequences of the change

Implementors can return collections declaring more specific element types than the ones defined by GeoAPI. In GeoAPI 2.0, it was considered an implementation detail to keep hidden from users' eyes, but users argued that the flexibility to expose their implementation specific classes was important.

Because of the nature of generic type implementation in Java (by “erasure”), this change is invisible to users restricted to Java 1.4 like the Geotools and Geoserver communities. It is an incompatible change for users of Java 1.5 and above like the uDig community, who will need to update their code.

In practice, known users affected by this change have already updated their code on the basis of GeoAPI milestones, because they are the same users that requested the change.

This change has a consequence on the future of metadata interfaces in GeoAPI: collections will be read-only. In GeoAPI 2.0, it was possible to update a metadata element as we see below:

```
citations.getIdentifiers().add("Some title");
```

GeoAPI 2.0 said nothing about whether or not the above was authorized; the door was kept open for future GeoAPI specifications. With the proposed change this door will be closed, and the above will no longer be possible. Some argued this is a good thing, since unmodifiable metadata interfaces have some value. The way to create or update metadata would be left to implementor, or to some future GeoAPI specification.

If this change is approved, it doesn't mean that metadata will never be modifiable in GeoAPI, but that they will not be modifiable in the above way.

4.1.6 Consequences if not approved

The door for writable collections in metadata interfaces would be kept open for future GeoAPI versions, but users of Hibernate or similar frameworks would have difficulties implementing metadata interfaces and may be tempted to abandon GeoAPI.

Users who already took advantage of covariance with GeoAPI milestones would have to revert their changes.

4.2 Return Objects not primitives

The return types of several methods have been altered from primitives (int, boolean) to their object equivalents (Integer, Boolean) and Number return types have been altered to Double.

JIRA task: http://jira.codehaus.org/browse/GEO-103

We suggest altering the return types of `Number` to `Double`, of some `boolean` to `Boolean`, and of `int` to `Integer`.

In GeoAPI 2.0, we were accustomed to expressing mandatory attributes using Java primitive types (e.g. `int`) and optional attributes by Java wrappers (e.g. `Integer`) because only the later allow to return `null` for missing values. Some users considered this policy as a complication compared to an uniform policy (everything as Java wrappers). In addition, it has been argued that even if a missing mandatory attribute is not allowed according to ISO 19115, invalid or incomplete metadata are common in the wild and GeoAPI should not block users and implementors facing such an invalid metadata.

The changes from ISO 19115:2000 to ISO 19115:2003 brings another argument. Some mandatory attributes in the previous ISO specification became optional in the latest one. When using primitive types as in GeoAPI 2.0, such a change in the ISO specification forces us to change the return type from primitive to wrapper, e.g. from `int` to `Integer`, which is an incompatible change. Changing everything to wrappers would allow us to handle future ISO 19115 obligation changes (if any) in a more upward compatible way.

Users also requested that existing methods returning `java.lang.Number` should return `java.lang.Double` for more straightforward matching with ISO 19115 and for allowing introspection.

4.2.1 Affected section(s), table(s), and figure(s)

These changes should not have any effect on specification documents but will only alter the realization of the specification in the Java Language.

4.2.2 Purposes of the proposed change

Provides a more uniform API, handle future ISO 19115 obligation changes (if any) without compatibility break in method signature, admit the existence of invalid or incomplete metadata in practice.

4.2.3 Reasons for change

This is a request from GeoAPI users.

4.2.4 Specific suggested changes

metadata.content.Band.getMaxValue()	returns Double not Number.
metadata.content.Band.getMinValue()	returns Double not Number.
metadata.content.Band.getPeakResponse()	returns Double not Number.
metadata.content.Band.getScaleFactor()	returns Double not Number.
metadata.content.Band.getOffset()	returns Double not Number.
metadata.content.FeatureCatalogueDescription.isCompliant()	returns Boolean not boolean.
metadata.content.ImageDescription.getIlluminationElevationAngle()	returns Double not Number.
metadata.content.ImageDescription.getIlluminationAzimuthAngle()	returns Double not Number.
metadata.content.ImageDescription.getCloudCoverPercentage()	returns Double not Number.
metadata.content.ImageDescription.isRadiometricCalibrationDataAvailable()	returns Boolean not boolean.
metadata.content.ImageDescription.isCameraCalibrationInformationAvailable()	returns Boolean not boolean.
metadata.content.ImageDescription.isFilmDistortionInformationAvailable()	returns Boolean not boolean.
metadata.content.ImageDescription.isLensDistortionInformationAvailable()	returns Boolean not boolean.
metadata.distribution.DigitalTransferOptions.getTransferSize()	returns Double not Number.
metadata.distribution.Medium.getDensities()	returns Collection<Double> not Collection<Number>.
metadata.extent.GeographicExtent.getInclusion()	returns Boolean not boolean.
metadata.extent.VerticalExtent.getMinimumValue()	returns Double not double.
metadata.extent.VerticalExtent.getMaximumValue()	returns Double not double.
metadata.identification.Resolution.getDistance()	returns Double not double.
metadata.quality.QuantitativeResult.getValues()	returns Collection<? extends Record> not double[]
metadata.spatial.Dimension.getDimensionSize()	returns Integer not int.
metadata.spatial.Dimension.getResolution()	returns Double not double.
metadata.spatial.GeometricObjects.getGeometricObjectCount()	returns Integer not int.
metadata.spatial.GridSpatialRepresentation.getNumberOfDimensions()	returns Integer not int.

4.2.5 Consequences of the change

This is an incompatible change. However Java 5 users will just need to recompile their code, since autoboxing should handle wrappers transparently. The only exception is `QuantitativeResult.getValues()`, which will requires users to update their code.

Note that known users of GeoAPI metadata interfaces already updated their code on the basis of a GeoAPI milestone.

4.2.6 Consequences if not approved

GeoAPI metadata interfaces will remain fragile to eventual changes in ISO 19115 obligations. Metadata interfaces will not face the reality of invalid metadata, and implementors will start using

magic numbers as placeholders for missing values. Users who already updated their code will need to revert their changes.

5. Geometry (Spatial Schema)

NOTE: *Because the first proposed change affects the names of all the objects in the package and the second proposed change alters the name of a package used in several places, this document adopts a naming scheme as if both of these proposals had been accepted.*

5.1 Drop the `spatialschema` part in package name

The package name `org.opengis.spatialschema.geometry` would be shortened to `org.opengis.geometry`
JIRA task: http://jira.codehaus.org/browse/GEO-110

A proposal has been made to drop the `spatialschema` portion of the package name. It is felt that the `spatialschema` name unnecessarily lengthens the name of the package while providing no tangible benefits.

5.1.1 Affected section(s), table(s), and figure(s)

In [OGC 03-064r10]: section 6.5.2.

5.1.2 Purposes of the proposed change

The original idea for the naming was to group the geometry and the topological parts of ISO 19107 in the same hierarchical group. In the eventuality where GeoAPI would handle other spatial schemata, the `spatialschema` name would enable GeoAPI to group all these alternatives into one hierarchy.

However, the name is exceedingly long and no shorter, suitable alternative has been proposed. This name is felt to be redundant with the "geometry" and "topology" subpackages, which are considered sufficiently self-explaining. Also, new spatial schema other than ISO 19107 are not expected to emerge in GeoAPI. Finally, a parent `spatialschema` package brings no benefit from the Java language point of view since Java packages are not hierarchical. It may bring a benefit from the documentation point of view, but the javadoc tool can performs the same grouping without the need for a common parent package.

The change would benefit both implementors and users of the GeoAPI interfaces by shortening the names of several of the core objects without information lost. It would also bring more consistency to GeoAPI since no other package has a "fooschema" parent.

Only one implementation depending on the old (GeoAPI 2.0) naming scheme is known. This implementation was created by Polexis and donated to the Geotools community, which already applied the proposed name change. An other implementation was recently created by a Ph.D student in Germany, who supported the name change and applied it as well. A third geometry

framework to take into consideration is the Geoxygene project, but this project doesn't seem active anymore and didn't reach the state where they would be implementing GeoAPI interfaces.

The proposers feel that it is still possible to apply this name change, because known implementors are just starting their work on the geometry package. This is different than other packages like referencing, which has had a stable implementation for two years. However if this name change is not applied now, next year would probably be too late.

5.1.3 Reasons for change

The changes would make more compact and more readable code. It would eliminate the redundancy provided by the "`spatialschema.geometry`" name juxtaposition and be more consistent with the naming in other parts of GeoAPI project.

5.1.4 Specific suggested changes

The names of all the language elements in the `org.opengis.spatialschema.geometry` namespace would be changed to the `org.opengis.geometry` namespace.

As a consequence:

- The package declaration in all of the files in the package would have to be changed.
- The import statements for all of files relying on interfaces in this package would have to be renamed.
- The `{@link}` and `{@see}` Javadoc tags would have to be altered.

5.1.5 Consequences of the change

Both implementors and users would have to refactor a large portion of their code, although the refactoring is simple and can be performed quickly by a `sed` command or a Ant script. Known implementors and users impacted by this change include: GeoTools, GeoServer, uDig and undisclosed users who gave their opinion on the mailing list. All the above-cited implementors and users applied the change in their development branches.

5.1.6 Consequences if not approved

The old naming scheme would be retained. Above-cited implementors and users would have to revert the changes in their development branch.

5.2 Rename `geometry.geometry` package

Change the package name which previously was `spatialschema.geometry.geometry` to `geometry.coordinate`.

JIRA task: http://jira.codehaus.org/browse/GEO-110

The proposal suggests changing the name of the `geometry.geometry` sub-package to `geometry.coordinate` for eliminating the redundancy and for consistency with the "coordinate geometry package" name of section 6.4 in ISO 19107.

5.2.1 Affected section(s), table(s), and figure(s)

This change has no effect on any specification documents.

5.2.2 Purposes of the proposed change

The "`geometry.coordinate`" name echoes the notion of geometric constructions based on coordinates. It reflects the "*coordinate geometry package*" (6.4) section of ISO 19107.

5.2.3 Reasons for change

The "`geometry.geometry`" nomenclature was felt to be redundant and confusing for users

5.2.4 Specific suggested changes

- Change the "`geometry.geometry`" package name to "`geometry.coordinate`".
- Update all javadoc tags and import statements in the source code.

This change can be done automatically together with the removal of the "`spatialschema`" name with the same Ant script.

5.2.5 Consequences of the change

The changes clarify a possibly confusing "`geometry.geometry`" nomenclature. Impact on implementors and users is the same as for 5.1.5.

5.2.6 Consequences if not approved

The older, possibly confusing, nomenclature will remain. Impact on implementors and users is the same as for 5.1.6.

5.3 Create geometry.PositionFactory

Concise description of the proposed change
JIRA task: http://jira.codehaus.org/browse/GEO-37

GeoAPI 2.0 creates `DirectPosition` objects using `GeometryFactory` but `DirectPosition`s are not geometry objects.

For 2.1, a new factory, `PositionFactory`, is proposed and the two methods which return `DirectPosition`s in `geometry.coordinate.GeometryFactory` have been deprecated.

5.3.1 Affected section(s), table(s), and figure(s)

These changes have no effect on the specification documents.

5.3.2 Purposes of the proposed change

The changes should provide a more straightforward mapping between class hierarchy and the factory to use.

5.3.3 Reasons for change

This is a user request. `GeometryFactory` was felt to be the wrong place to hold creation methods for `DirectPositions`.

5.3.4 Specific suggested changes

Add `PositionFactory` interface in the same package than the one that defines `DirectPosition`. This interface contains the `DirectPosition` constructors previously defined in `GeometryFactory`.

Deprecate `DirectPosition` constructors in `geometry.coordinate.GeometryFactory`.

5.3.5 Consequences of the change

Users will need to update their code. However they can wait as long as the deprecated methods are not removed.

5.3.6 Consequences if not approved

Users would still feel that `DirectPositions` are created in the wrong factory.

5.4 Add `geometry.BoundingBox`

Add a `BoundingBox` interface as a specialization of Envelope in the 2D case.
JIRA task: http://jira.codehaus.org/browse/GEO-113

Add a `BoundingBox` interface as a sub-interface of `Envelope` specialized in the 2D case. This is a convenience interface bringing no new functionalities. This is a request from GeoAPI users wanting a simple equivalence for the `BBOX` WKT construct. This is also useful for alignment with Filter and WFS specifications.

5.4.1 Affected section(s), table(s), and figure(s)

none.

5.4.2 Purposes of the proposed change

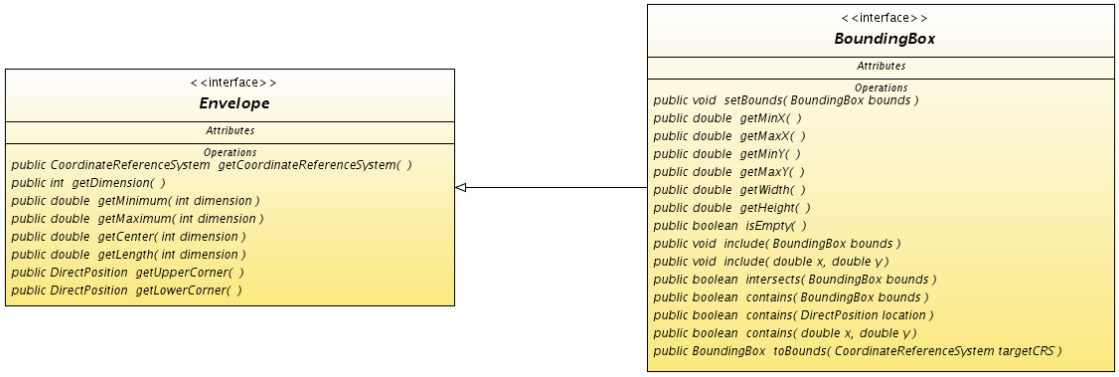
Provide a simple front end to `Envelope` in the 2D case, similar to `BBOX`.

5.4.3 Reasons for change

This is a user request.

5.4.4 Specific suggested changes

We suggest to add an interface as the one in the diagram below, as an `Envelope` sub-interface. Note that every methods in the proposed `BoundingBox` match exactly the equivalent method in `java.awt.geom.Rectangle2D`, so an other way to see `BoundingBox` is as a junction point between OGC `Envelope` and J2SE `Rectangle2D`.



5.4.5 Consequences of the change

This is a compatible change.

5.4.6 Consequences if not approved

GeoAPI users will continue to use `Envelope`, which is slightly less convenient in the 2D case. They are likely to write their own `BoundingBox` interface.

5.5 Add `geometry.Precision` and `geometry.PrecisionType`

Geometry implementors requested a way to specify the underlying precision of a geometry.
JIRA task: http://jira.codehaus.org/browse/GEO-106

Implementation of topological operations need to know the precision of coordinate points stored in a geometry. GeoAPI exposes every coordinate as a IEEE 754 double precision floating point number. If a geometry use single precision floating point numbers, a widening conversion is performed (note: this is the same approach used by Java2D). However, some topological

operations may produce erroneous results if they performed their calculation in double precision while the precision in the underlying geometry was simple precision.

5.5.1 Affected section(s), table(s), and figure(s)

None.

5.5.2 Purposes of the proposed change

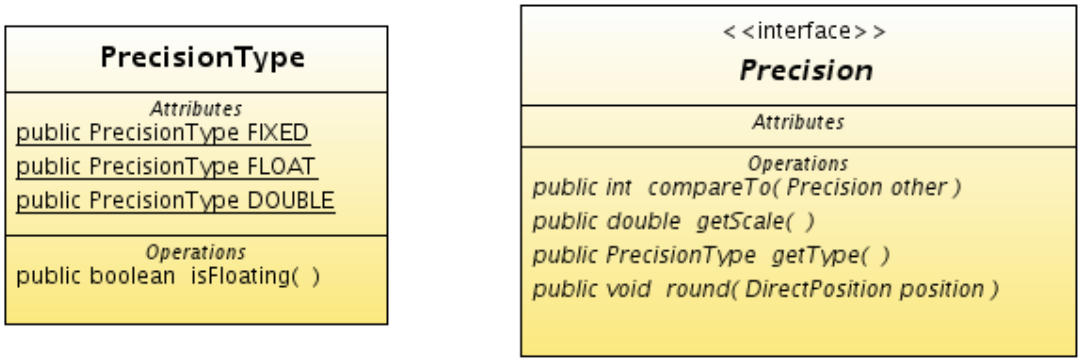
Allows a more reliable implementation of topological operations.

5.5.3 Reasons for change

This is a requirement from geometry implementors. This suggestion is based on the experience of JTS (Java Topology Suite) implementors.

5.5.4 Specific suggested changes

Add the following code list and interface (note that `PrecisionType` extends `CodeList` and consequently implements `Comparable`):



5.5.5 Consequences of the change

This is a compatible change.

5.5.6 Consequences if not approved

Geometry implementors may miss an important piece of information for implementing topological operations.

5.6 Add `Geometry.getPrecision()`

Add the method `getPrecision()` to the interface `geometry.Geometry`
JIRA task: http://jira.codehaus.org/browse/GEO-107

Geometries need a precision model to guide the accuracy of topology operations. In order to easily and publicly get that model, a `getPrecison()` method is needed.

5.6.1 Affected section(s), table(s), and figure(s)

In [OGC 03-064r10]: section 6.4, Figure 13

5.6.2 Purposes of the proposed change

The purpose of this change is to provide a method for developers to get the precision model used for a geometry's topology operations.

5.6.3 Reasons for change

The is currently no way for developers to publicly get the precision model for a `Geometry`.

5.6.4 Specific suggested changes

The proposed changes for `Geometry` are:

- make method `getPrecision()`

5.6.5 Consequences of the change

This change will not break any existing code as it is only adding functionality.

5.6.6 Consequences if not approved

If not approved, developers will have no public method for fetching the precision model of a `Geometry`.

5.7 Create `geometry.aggregate.AggregateFactory`

Add a factory for the aggregate interfaces.
JIRA task: http://jira.codehaus.org/browse/GEO-114

GeoAPI 2.0 defines some aggregate interfaces from ISO 19107, but did not provide any constructors for them. The `AggregateFactory` interface would provide these constructors.

5.7.1 Affected section(s), table(s), and figure(s)

None.

5.7.2 Purposes of the proposed change

Provides constructors for the aggregate interfaces.

5.7.3 Reasons for change

There is no constructor for aggregate interfaces in GeoAPI 2.0.

5.7.4 Specific suggested changes

Add `AggregateFactory` interface in the same package as the one that defines `Aggregate`. This interface contains methods that create and return `Aggregates` such as `MultiPrimitive`, `MultiPoint`, `MultiCurve`, and `MultiSurface`.

All aggregates created through this interface will use the `getCoordinateReferenceSystem` factory's coordinate reference system.

5.7.5 Consequences of the change

This is a compatible change.

5.7.6 Consequences if not approved

Users will have no way to create aggregate objects without relying to implementation-specific code.

5.8 Create `geometry.complex.ComplexFactory`

Add a factory for the complex interfaces.
JIRA task: http://jira.codehaus.org/browse/GEO-115

GeoAPI 2.0 defines some complex interfaces from ISO 19107, but did not provide any constructors for them. The `ComplexFactory` interface would provide these constructors.

5.8.1 Affected section(s), table(s), and figure(s)

None.

5.8.2 Purposes of the proposed change

Provides constructors for the complex interfaces.

5.8.3 Reasons for change

There is no constructor for complex interfaces in GeoAPI 2.0.

5.8.4 Specific suggested changes

Add `ComplexFactory` interface in the same package as the one that defines `Complex`. This interface contains methods that create and return `Complexes` such as `CompositeCurve`, `CompositePoint`, and `CompositeSurface`.

All complexes created through this interface will use the `getCoordinateReferenceSystem()` factory's coordinate reference system.

5.8.5 Consequences of the change

This is a compatible change.

5.8.6 Consequences if not approved

Users will have no way to create complex objects without relying to implementation-specific code.

5.9 Add `Envelope.getCoordinateReferenceSystem()`

Add a `getCoordinateReferenceSystem()` convenience method in `Envelope`

JIRA task: http://jira.codehaus.org/browse/GEO-116

Experience in the Geotools project suggests that fetching an `Envelope` CRS is a common operation. This information is currently available indirectly through the direct position returned by `getLowerCorner()` or `getUpperCorner()`. There is no unique or privileged path. Paranoiac users check both the lower and upper corners CRS for making sure that they agree. On some implementations, indirect paths may have the cost of a temporary `DirectPosition` object creation. We suggest to add a `getCoordinateReferenceSystem()` convenience method in `Envelope` for providing a direct way to fetch this frequently used information.

5.9.1 Affected section(s), table(s), and figure(s)

In [OGC 03-064r10]: figure 19.

5.9.2 Purposes of the proposed change

Mostly convenience. The proposed change add no new functionality.

5.9.3 Reasons for change

Experience in the Geotools project suggests that working with envelopes frequently involve fetching the envelope CRS.

5.9.4 Specific suggested changes

We propose to add the `getCoordinateReferenceSystem()` method in the `Envelope` interface.

5.9.5 Consequences of the change

GeoAPI implementors would be required to add the above-cited method in their implementations (even if they choose to return a null value). No action needed for GeoAPI users.

5.9.6 Consequences if not approved

No blocker consequences since the proposed addition is only a convenience method. Paranoid users will continue to check both the lower and upper corner CRS.

5.10 Modify `geometry.coordinate.PointArray`

> **The `PointArray` interface should be made to extend `List<Position>` and the access rules to several methods should be changed.**
> **JIRA task:** http://jira.codehaus.org/browse/GEO-108

Some users want that the interface `geometry.coordinate.PointArray` extends `List<Position>` directly rather than providing a method to return a `List<Position>` often backed by itself. The methods `get(int,DirectPosition)` and `set(int,DirectPosition)` would be renamed `getDirect(…)` and `setDirect(…)` respectively in order to avoid confusion with the `get` and `set(int,Position)` method inherited from `List<Position>`.

5.10.1 Affected section(s), table(s), and figure(s)

The changes will have no effect on the specification documents.

5.10.2 Purposes of the proposed change

The purpose of the change is to allow `PointArray` to implement the Java Collection interfaces in a more direct way.

5.10.3 Reasons for change

Ease of use for developers familiar with the Java collections API. This is a user request.

5.10.4 Specific suggested changes

The changes proposed in `PointArray` are:

- Make `PointArray` a `List<Position>`

- rename method `get(…)` to `getDirectPosition(…)`. Deprecate the old method.
- rename method `set(…)` to `setDirectPosition(…)`. Deprecate the old method.

- deprecate method `length()`
- deprecate method `positions()`

- un-deprecate method `getDimension()`

- `declare method setDirect() to throw UnsupportedOperationException`

5.10.5 Consequences of the change

This is a compatible change.

5.10.6 Consequences if not approved

If not approved, client code will continue to use Java collection in a more indirect way through the `PointArray.positions` method.

5.11 Modify return type of `Aggregate.getElements()`

Modify the return type of `geometry.aggregate.Aggregate.getElements()`
JIRA task: http://jira.codehaus.org/browse/GEO-117

The method `getElements()` in `geometry.aggregate.Aggregate` should return `Set<? extends Geometry>` rather than `Set<Geometry>`.

5.11.1 Affected section(s), table(s), and figure(s)

None.

5.11.2 Purposes of the proposed change

Java 5 allows method return types to be collections of any type that is extended by the same interface. The purpose of this change is to allow aggregates to expose in their API the restrictions that a particular aggregate may impose on `Geometry` types.

5.11.3 Reasons for change

The reason for this change is to allow developers to create their own implementations of objects that extend `Geometry` and expose them in their `Aggregates` API. This change also allows

developers to properly type narrow. Developers will be able to properly specify the type of geometry when sub typing.

5.11.4 Specific suggested changes

The changes proposed in `Aggregate` are:

- change return type of method `getElements()` from `Set<Geometry>` to `Set<? extends Geometry>`

5.11.5 Consequences of the change

This change will not affect Java 4 users. Java 5 users may need to update their code.

5.11.6 Consequences if not approved

If this change is not approved, developers will not be able to narrow the geometry type allowed for a particular Aggregates.

5.12 Modify return type of `Complex.getElements()`

Modify the return type of the method `geometry.complex.Complex.getElements()`
JIRA task: http://jira.codehaus.org/browse/GEO-118

The method `getElements()` in `geometry.aggregate.Aggregate` should return `Collection<? extends Primitive>` rather than `Set<Primitive>`.

5.12.1 Affected section(s), table(s), and figure(s)

None.

5.12.2 Purposes of the proposed change

Java 5 allows method return types to be collections of any type that is extended by the same interface. The purpose of this change is to allow aggregates to expose in their API the restrictions that a particular complex may impose on `Primitive` types.

5.12.3 Reasons for change

The reason for this change is to allow developers to create their own implementations of objects and expose them in their `Complexes` API. This change also allows developers to properly type narrow. Developers will be able to properly specify the type of primitive when sub typing.

5.12.4 Specific suggested changes

The changes proposed in `Complex` are:

- change return type of method `getElements()` from `Set<Primitive>` to `Collection<? extends Primitive>`

5.12.5 Consequences of the change

This change will force older code to be updated to accept the new return type of a `Collection` instead of a `Set`.

5.12.6 Consequences if not approved

If this change is not approved, developers will not be able to narrow the primitive type allowed for a particular Complexes.

5.13 Define `equals(Object)` and `hashCode()` in `DirectPosition`

Define `equals(Object)` and `hashCode()` methods in `DirectPosition`.
JIRA task: http://jira.codehaus.org/browse/GEO-119

Document how the `equals(Object)` method in `DirectPosition` should perform the test for equality, in order to allow comparisons of direct positions from different implementations. Document how the `hashCode()` method should compute its value for consistency with the `equals` definition.

The `equals(Object)` and `hashCode()` methods are inherited for all Java objects, including interfaces. Consequently the addition of those methods in the `DirectPosition` interface make no difference on an API point of view. The only purpose is to document how those methods should perform their tests. If such documentation is not provided, implementors are free to implement those methods as they wish. This works well as long as users compare only `DirectPosition` instances backed by the same implementation, but leads to problems when comparing instances from different implementations: incomparable objects at best, violation of `Object.equals` contract (symmetry, transitivity...) at worst.

The Java Collection framework face the same issue and address it through detailed specifications of `equals(Object)` and `hashCode()` in `Collection`, `Set` and `List` interfaces. We propose a similar approach for `DirectPosition`.

5.13.1 Affected section(s), table(s), and figure(s)

In [OGC 03-064r10]: section 6.4.1.1, Figure 14

5.13.2 Purposes of the proposed change

Allow `DirectPosition` comparison to work properly across different implementations.

5.13.3 Reasons for change

In GeoAPI 2.0, developers can not assume that two `DirectPosition`s are comparable if they are not backed by the same implementation. Experience suggests that users need to handle various implementations of `DirectPosition` in the same application.

5.13.4 Specific suggested changes

Document `equals(Object)` as below:

> Compares this direct position with the specified object for equality. Two direct positions are considered equal if the following conditions are meet:
>
> - `object` is non-null and is an instance of `DirectPosition`
> - Both direct positions have the same number of dimension
> - Both direct positions have the same or equal coordinate reference system
> - For all dimension *i*, the ordinate value of both direct positions at that dimension are equals in the sense of `java.lang.Double.equals(Object)`. In other words, `java.util.Arrays.equals(getCoordinates(), object.getCoordinates())` returns `true`.

Document `hashCode()` as below:

> Returns a hash code value for this direct position. This method should returns the same value as: `java.util.Arrays.hashCode(getCoordinates()) + getCoordinateReferenceSystem().hashCode()` where the right hand side of the addition is omitted if the coordinate reference system is null.

5.13.5 Consequences of the change

This is a compatible change since it doesn't bring any API change. Implementors will need to review their `equals(Object)` and `hashCode()` implementations.

5.13.6 Consequences if not approved

If not approved, `DirectPosition` are not likely to be comparable between different implementations. Users that need to compare different implementations of `DirectPosition` while need to write their own code.

5.14 Deprecate `DirectPosition.clone()`

> **The `clone()` method in `geometry.DirectPosition` should be deprecated.**
> **JIRA task:** http://jira.codehaus.org/browse/GEO-109

The `clone()` method should be removed and the `Cloneable` status of the interface should be left to implementors. Users should use `PositionFactory` instead.

5.14.1 Affected section(s), table(s), and figure(s)

In [OGC 03-064r10]: section 6.4.1.1, Figure 14

5.14.2 Purposes of the proposed change

Implementations of `DirectPosition` should not be forced to allow cloning. The `Cloneable` status of a `DirectPosition` should be left to implementors.

5.14.3 Reasons for change

This is a user request. Some `DirectPosition` implementations are not amenable to a cloneable status. For example `DirectPosition`s obtained from an `ArrayList` are often thin wrapper backed by the array list. Cloning such `DirectPosition` will not create a copy as most users would expect, since the clone would still be backed by the same array list. It would be possible to define the `clone()` method in such a way that it returns a different implementation not backed by the array list, but some users found such approach counter-intuitive.

So the proposal is to let the cloneable status to implementor choice. The functionality would be replaced by the use of `PositionFactory`, a more versatile alternative allowing transformation into alternate representations.

5.14.4 Specific suggested changes

The suggested changes to `DirectPosition` are:

- deprecate the method `clone()`

5.14.5 Consequences of the change

Old code that uses the clone() method will still work, but the developers will become aware of its deprecated status so they can remove the dependency and implement it how they see fit.

5.14.6 Consequences if not approved

Some `DirectPosition` implementations will implement the `clone()` method in a way that some users find counter-intuitive.

5.15 Modify return types of `SurfaceBoundary.getInteriors()`

The return type of the method `getInteriors()` in the interface `geometry.primitive.SurfaceBoundary` should be `List<Ring>` rather than `Ring[]`
JIRA task: http://jira.codehaus.org/browse/GEO-65

The method `getInteriors()` should return a paramatrized List of Ring objects rather than an array of Ring objects.

5.15.1 Affected section(s), table(s), and figure(s)

None.

5.15.2 Purposes of the proposed change

The purpose of this change is to conform with the rest of the GeoAPI which returns collections, this method should also return a type of collection.

5.15.3 Reasons for change

This is a user request in order to conform with the rest of the GeoAPI methods.

5.15.4 Specific suggested changes

The proposed changes for `SurfaceBoundary` are:

- update the `getInteriors()` return type

5.15.5 Consequences of the change

This change will break old code until they update the return type to expect a list instead of an array.

5.15.6 Consequences if not approved

No code will be broken. `SurfaceBoundary.getInteriors()` will still slightly inconsistent compared to the rest of GeoAPI.

6. Referencing

6.1 Modify `ProjectedCRS` factory

Provides a better way to instantiate `ProjectedCRS` using factories
JIRA task: http://jira.codehaus.org/browse/GEO-61

The current factory methods for creating `ProjectedCRS` and `DerivedCRS` instances are uneasy and not in phase with the ISO 19111 spirit. They were an unsuccessful attempt to avoid a "chicken and egg problem": `SC_DerivedCRS` need a `CC_Conversion`, and it would have been nice for user convenience (although not required by ISO 19111, as discussed in C.4.2. "*Coordinate conversions*") that the `CC_Conversion.targetCRS` association point toward the `SC_DerivedCRS` instance.

We propose to replace (deprecate now, remove later) the current `createProjectedCRS(…)` and `createDerivedCRS(…)` methods by 3 new ones modeling ISO 19111 in a more straightforward way:

- One method for creating a defining conversion.
- One method for creating a projected CRS from a defining conversion
- One method for creating a derived CRS from a defining conversion.

6.1.1 Affected section(s), table(s), and figure(s)

In [OGC 03-064r10]: figure 23, 24.

6.1.2 Purposes of the proposed change

- To make the factory API (for creating instances of various ISO 19111 objects) more straightforward to users familiar with ISO 19111.
- More freedom for both users and implementers, since the creation steps (*defining conversion* first, *projected CRS* next) would be more explicit.

6.1.3 Reasons for change

The current factory API is reasonably straightforward for all ISO 19111 objects except `DerivedCRS` and `ProjectedCRS` (because of their dependence toward a `Conversion` object). Numerous emails on the Geotools mailing list show that users have a hard time creating `ProjectedCRS` instances.

This proposal is the result of suggestions sent by Geotools users, and has been tested in the Geotools implementation.

6.1.4 Specific suggested changes

Deprecate all current `createDerivedCRS(…)` and `createProjectedCRS(…)` methods in `CRSFactory`. Add the following method in `CoordinateOperationFactory`:

- ```
  createDefiningConversion(Map                 properties,
                           OperationMethod     method,
                           ParameterValueGroup parameterValues);
  ```

Add the following methods in `CRSFactory`:

- ```
  createProjectedCRS(Map                       properties,
                     GeographicCRS             baseCRS,
                     Conversion                conversionFromBase,
                     CartesianCS               cs);
  ```

- ```
  createDerivedCRS  (Map                       properties,
                     CoordinateReferenceSystem baseCRS,
                     Conversion                conversionFromBase,
                     CoordinateSystem          cs);
  ```

6.1.5 Consequences of the change

Deprecation of 2 existing methods in `CRSFactory` and addition of 3 new methods. Users who were using the deprecated methods will need to switch to the new ones. However (based on feedback on mailing lists), we expect that some users already avoided direct usage of the 2 existing methods anyway.

6.1.6 Consequences if not approved

Some users will continue to avoid direct use of `CRSFactory.createProjectedCRS(…)` method, and will continue to use implementation-specific workarounds.

6.2 Add convenience methods to `MathTransformFactory`

`MathTransformFactory.createBaseToDerived(...)`

`MathTransformFactory` provides a `createParameterizedTransform(...)` method for the creation of an arbitrary transform, including projections. Users are responsible for handling units conversions and axis order. In the projection case, the handling of (*latitude*, *longitude*) axis order caused a considerable amount of confusion. See:

http://docs.codehaus.org/display/GEOTOOLS/The+axis+order+issue

Experience in the GeoTools community suggests that improper axis order in user's CRS still a frequent error. We propose the addition of a convenience method handling performing the `createParameterizedTransform` work with the addition of unit conversions and axis order handling. In addition of the `parameters` to be given to the parameterized transform, this convenience method expect a `baseCRS` and a `derivedCS` that can be used for the creation of affine transforms to be concatenated with the parameterized transform.
  ```

6.2.1 Affected section(s), table(s), and figure(s)

In [OGC 03-064r10]: section 6.4.2.7.

6.2.2 Purposes of the proposed change

Help users to avoid the most frequent error seen on mailing lists.

6.2.3 Reasons for change

The axis order issue has been a source of considerable confusion, and a large fraction of users don't handle axis order correctly. This convenience method may help to reduce the occurrence of wrong axis order in projections.

6.2.4 Specific suggested changes

Add a method with the following signature:

```
createBaseToDerived(CoordinateReferenceSystem baseCRS,
                    ParameterValueGroup       parameters,
                    CoordinateSystem          derivedCS);
```

Also add the following method, applicable to both `createParameterizedTransform` and `createBaseToDerived`. It make creation of `ProjectedCRS` and `DerivedCRS` easier:

```
OperationMethod getLastMethodUsed();
```

6.2.5 Consequences of the change

GeoAPI implementors would be required to add the above-cited method in their implementations.

6.2.6 Consequences if not approved

Axis order is likely to stay a main source of errors.

7. GO

7.1 Refactor GO enumerations

Bring GO package enumerations into conformance with the rest of the GeoAPI CodeList classes by (1) making the enumeration classes final, (2) parametrizing the enumerations when they are used in the style classes, and (3) altering a few methods as appropriate for the enumeration system.

JIRA task: none

The GeoAPI enumeration system follows a consistent pattern but the GO package in GeoAPI 2.0 did not conform to this system. We propose to alter the GO package to conform to this system.

7.1.1 Affected section(s), table(s), and figure(s)

This change does not affect any specification documents.

7.1.2 Purposes of the proposed change

Implementors and users will benefit from a more consistent system for the use of enumerations in the GeoAPI interface library.

7.1.3 Reasons for change

The GO package was confusingly different from the rest of the GeoAPI library.

7.1.4 Specific suggested changes

We propose to modify the enumerations in the `go.spatial` packages:

```
go.spatial.GlobalPathType
go.spatial.UnprojectedPathType
go.spatial.VectorPathType
```

to be final.

We propose to modify the constructors of the classes in the `go.spatial` package:

```
go.spatial.GlobalPathType
go.spatial.UnprojectedPathType
go.spatial.VectorPathType
```

and in the `go.display.style` package

```
go.display.style.DashArray
go.display.style.LinePattern
```

to match the enumeration system.

We propose to modify the classes in the `go.display.style` package which use `util.SimpleEnumerationType` to parameterize the type of the enumeration.

We propose to add the methods `values()` and `family()` to several classes in order to conform with the GeoAPI enumeration system. We wish to add:

```
go.spatial.GlobalPathType.values()
go.spatial.GlobalPathType.family()
go.spatial.PathType.values()
go.spatial.PathType.family()
go.spatial.UnprojectedPathType.values()
go.spatial.UnprojectedPathType.family()
go.spatial.VectorPathType.values()
go.spatial.VectorPathType.family()
```

We propose to deprecate, then drop in the subsequent revision of GeoAPI, the method `go.display.style.YAnchor.getNumberOfStyles()` since it duplicates the information available from the already existing `values().length`.

7.1.5 Consequences of the change

This is a compatible change. The changes will make the go package conform better to the GeoAPI `CodeList` enumeration system.

7.1.6 Consequences if not approved

The GO package would retain distinct semantics in the use of enumerations when compared to the rest of GeoAPI.

8. Filter

8.1 Extend the `FilterFactory` interface

Create the `FilterFactory2` interface which extends the `FilterFactory` interface to support SFSQL type queries and to create an easier `BoundingBox` system than that adopted by the specification.
JIRA task: http://jira.codehaus.org/browse/GEO-121

We propose a new interface, the `FilterFactory2` interface to extend the Filter system beyond the support for the Common Catalog Query Language (CQL) required by the specification to also support queries in the Simple Features for SQL (SFSQL) format. As proposed, `FilterFactory2` also provides smoother support for `BoundingBoxes` than that proposed by the Filter Encoding specification.

The new `FilterFactory2` interface also restores the symmetrical treatment of operations which was present in GeoAPI 2.0. The format of operations in the 1.1 specification implies the operation follows the form

```
PropertyName OPERATION Expression
```

but this does not always work out for users . Sometimes, what users want is something like

```
BUFFER( propertyName ) OPERATION Expression
```

which was possible in GeoAPI 2.0 and is restored by `FilterFactory2`. The new interface would restore these spatial operators.

8.1.1 Affected section(s), table(s), and figure(s)

These changes do not effect any specification document but, instead, extend the Filter specification adding support for similar constructs in different formats.

8.1.2 Purposes of the proposed change

The changes provide greater flexibility to users of the GeoAPI interfaces by extending the Filter system to cover the older but popular SFSQL language. This means users can mix SFSQL and CQL statements.

8.1.3 Reasons for change

SFSQL is still widely used.

8.1.4 Specific suggested changes

Provide `FilterFactory2` interface with the methods illustrated below:

8.1.5 Consequences of the change

This is a compatible change.

8.1.6 Consequences if not approved

Some applications may ignore the interfaces provided, and continue to work with custom extensions.

8.2 `BoundedSpatialOperator`

Create the `BoundedSpatialOperator` marker interface to allow for simpler implementation code.
JIRA task: http://jira.codehaus.org/browse/GEO-122

We propose creating the `BoundedSpatialOperator` interface with no methods simply to mark those operations which are a subset of the Bounds operator. This change greatly simplifies implementation code from as series of if/then/else constructs to a single instance of check.

8.2.1 Affected section(s), table(s), and figure(s)

This change does not affect any specification documents.

8.2.2 Purposes of the proposed change

The change greatly simplifies implementations of spatial operators in which the Bounding Box is involved.

8.2.3 Reasons for change

The change is proposed to simplify implementation code.

8.2.4 Specific suggested changes

Create the interface `BoundedSpatialOperator.java`.

Make the following interfaces extend this marker interface:

- `filter.spatial.Contains`
- `filter.spatial.Crosses`
- `filter.spatial.Equals`
- `filter.spatial.Intersects`
- `filter.spatial.Overlaps`
- `filter.spatial.Touches`
- `filter.spatial.Within`

8.2.5 Consequences of the change

The changes have no effect on user code and require only trivial changes to implementation code. However the changes permit a great simplification of implementation code if the implementors decide to leverage the marker interface.

8.2.6 Consequences if not approved

Implementors and client code would continue to perform a series of if than else checks, with the possibility of error.

The GeoTools implementation of the GeoAPI interfaces revert to a utility method to perform this common check.

8.3 `Function.getParameters()`

Change the return type of `filter.expression.Function.getParameters()` from an array to a parametrized list to conform with the use of collections and generics elsewhere in the API
JIRA task: http://jira.codehaus.org/browse/GEO-123

The return type of the `filter.expression.Function.getParameters()` was changed from an array of `Expression` to a `List` parameterized by Expression: `List<Expression>`.

8.3.1 Affected section(s), table(s), and figure(s)

In [OGC 03-064r10]: Figures 36.

8.3.2 Purposes of the proposed change

This change makes the representation of collection in the Function interface consistent with other GeoAPI interfaces.

8.3.3 Reasons for change

Handling collections in a manner consistent with the rest of GeoAPI makes the Function interface easier to learn and implement.

8.3.4 Specific suggested changes

- Change the return type of `getParameters()` from `Expression[]` to `List<Expression>`.

8.3.5 Consequences of the change

This is an incompatible change to the API, client code would need to be modified to use a collection of expressions.

8.3.6 Consequences if not approved

The representation of collections would vary across GeoAPI interfaces.

9. All packages

9.1 Replace the dependency to JSR-108 to a dependency to JSR-275

GeoAPI 2.0 relies on JSR-108 for units management. But this JSR has been withdrawn by the Java Community Process (JCP) before completion and is now replaced by JSR-275, with identical goals.

GeoAPI do not defines and interface for units of measurement. Instead we relies on the package under development in the Java Community Process (JCP). We propose to replace the dependency to `javax.units.Unit` from JSR-108 by a dependency to `javax.measure.units.Unit` from JSR-275. Note that JSR-275 may be included in Java 7 (not yet confirmed).

This change is targeted for GeoAPI 3.0 at best, or at a slightly later release depending when the community will feel ready to abandon Java 1.4 support, since JSR-275 is a Java 5 library.

9.1.1 Affected section(s), table(s), and figure(s)

None.

9.1.2 Purposes of the proposed change

Update GeoAPI dependencies to the current units of measurement framework.

9.1.3 Reasons for change

JSR-108 has been withdrawn by the Java Community Process (JCP) and replaced by JSR-275.

9.1.4 Specific suggested changes

Replace dependencies to `javax.units.Unit` from JSR-108 by dependencies to `javax.measure.units.Unit` from JSR-275.

9.1.5 Consequences of the change

This is an incompatible change. Client code will need to update their import statements. The way to performs units conversions in JSR-275 is also slightly different than it was in JSR-108.

9.1.6 Consequences if not approved

GeoAPI would continue to depends on an deprecated API.