

# Naming policy for GeoAPI implementation classes

|                    |   |
|--------------------|---|
| <b>Motivation:</b> | Make GeoAPI implementation class names consistent |
| <b>Contact:</b>    | <a href="#">Martin Desruisseaux</a>               |
| <b>Tracker:</b>    |   |
| <b>Tagline:</b>    |   |

Implementation of GeoAPI classes have different naming pattern in different modules. In the referencing module, they are prefixed with `Default` or `Abstract`. In the metadata module, they are suffixed with `Impl`. This proposal is about a more consistent pattern to be applied at least accross the metadata, referencing and coverage modules. More specifically we suggest to adopt the `Default/Abstract` prefix scheme used in referencing, and rename all metadata implementation classes accordingly.

## Why Default/Abstract prefix instead of Impl suffix

- Prefix allow us to see at a first glance which classes are GeoAPI implementations and which ones are GeoTools specific. The argument for `Impl` suffix was to preserve alphabetical order. However experience suggests that it is not so convenient, since it is often more useful to distinguish between GeoAPI implementations and GeoTools specific classes. The GeoAPI implementation could be hidden from the "beginner" javadoc (make is visible only to "advanced" users). Most users could be encouraged to read the GeoAPI javadoc instead.
- Distinguish between abstract and concrete ISO classes. Classes that are abstract according ISO are not necessarily abstract in GeoTools implementation. For example `AbstractCS` (the base class for `DefaultCartesianCS`, `DefaultVerticalCS`, *etc.*) is abstract according ISO, but not in the Java sense since it can be instantiated. Actually `AbstractCS` **is** really instantiated in a few cases by the WKT parser, meaning "I really don't know what this coordinate system is; I just know its axes" (this case occurs because of limitations in WKT syntax).
- Make abstract classes to appear before concrete ones in alphabetical order.
- Consistency with usage in the JDK collection framework.

**Example:** The [CRS package](#) has the following classes. We can see immediately that `CRS`, `SingleCRS` and `DerivedCRS` are abstract (in ISO sense), and that `UnprefixedMap` has nothing to do with GeoAPI classes. We would like the same advantages for metadata.

- `AbstractCRS.java`
- `AbstractDerivedCRS.java`
- `AbstractSingleCRS.java`
- `DefaultCompoundCRS.java`
- `DefaultDerivedCRS.java`
- `DefaultEngineeringCRS.java`
- `DefaultGeocentricCRS.java`
- `DefaultGeographicCRS.java`
- `DefaultImageCRS.java`
- `DefaultProjectedCRS.java`
- `DefaultTemporalCRS.java`
- `DefaultVerticalCRS.java`

- `UnprefixedMap.java`

## Proposed action

- Rename all metadata implementation classes, replacing the `Impl` suffix by `Default` or `Abstract` depending on whatever the class is abstract or not according ISO.
- Refactor the old `Impl` classes as extending the new renamed classes, and deprecate them for GeoTools 2.5 release.
- Delete the deprecated `Impl` classes on trunk after GeoTools 2.5 release.

No other classes than metadata would be renamed at this stage; current class names in referencing and coverage are considered okay.

Note that a few referencing and coverage classes use other prefix than `Abstract` or `Default`. For example "General" in `GeneralEnvelope` to stress-out its  $n$ -dimensional aspect compared to `Envelope2D`, or "Grid" in `GridSampleDimension` to stress-out that they apply to `GridCoverage` rather than arbitrary `Coverage`. This is okay and don't need to be forced into a `Default/Abstract` naming scheme. The above is just a proposed rule of thumbs, not a strict rule.

Voting is open:

- [Andrea Aime](#)
- [Ian Turton](#)
- [Justin Deoliveira](#)
- [Jody Garnett](#)
- [Martin Desruisseaux](#) +1
- [Simone Giannecchini](#)

Community support:

- [Cédric Briçon](#) +1
- [Johann Sorel](#) +1