

# Replacing the OLD GCE interfaces

<b>Motivation:</b>	The Grid Coverage Exchange specification is dead; we need a replacement along the lines of DataStore
<b>Contact:</b>	<a href="#">Simone Giannecchini</a> , <a href="#">Jody Garnett</a>
<b>Tracker:</b>	<a href="http://jira.codehaus.org/browse/GEOT-XXXX">http://jira.codehaus.org/browse/GEOT-XXXX</a>
<b>Tagline:</b>	GCE is dead, lets move on

This page describes the work that is being carried out in order to replace the actual GCE API with a new one. For the moment it just tries to capture what people are doing as well as the various opinions and feedback.

## Description

This proposal:

- introduces new interface in a manner consistent with [Data Access Design Goals](#)
- where possible abstractions and method names are taken from various OGC-ISO specifications, namely:
  - WCS 1.1.1
  - ISO 19123
  - ISO 19111
  - ISO 19108
  - ISO 19103

## Status

This proposal is under construction; please help put it together. You may find the previous proposal [Dry Run at DataAccess Story](#) informative, as well as this [email thread](#). Anyway, the API as well as some helper can be found under the spike directory of GeoTools trunk.

Voting has not started yet:

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

## Tasks

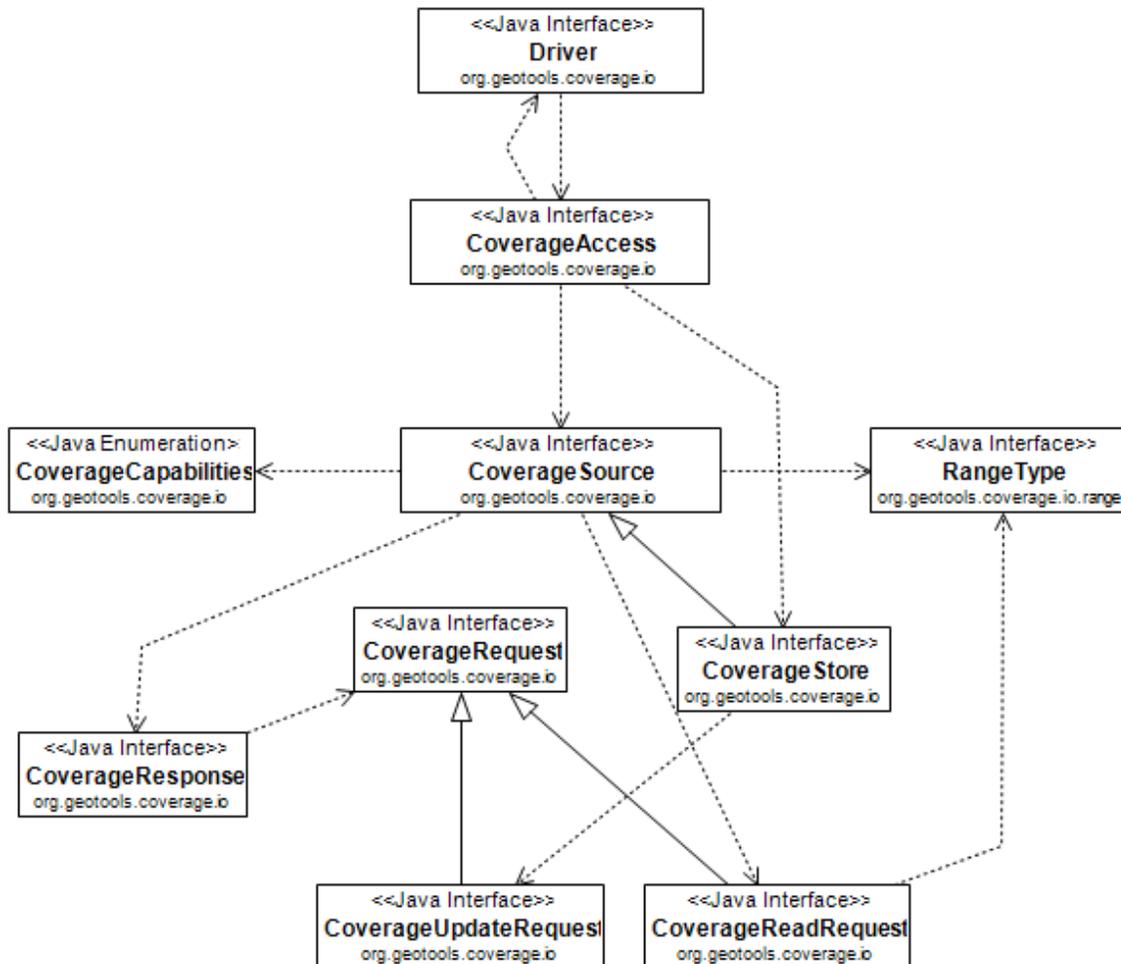
*This section is used to make sure your proposal is complete (did you remember documentation?) and has enough paid or volunteer time lined up to be a success*

	no progress		done		impeded		lack mandate /funds/time		volunteer needed
--	-------------	-------------------------------------------------------------------------------------	------	-------------------------------------------------------------------------------------	---------	-------------------------------------------------------------------------------------	--------------------------	---------------------------------------------------------------------------------------	------------------

1. Define interface based from the WCS specification
2. Produce initial prototype for world + image
3. Roll out the above prototype for a round of community feedback and review
4. Port the existing coverage formats one by one, either:
  - as a wrapper around existing GridCoverageReaders / GridCoverageWriter implementations
  - or direct implementation of CoverageStore / CoverageSource and CoverageStore

## API

A UML diagram capturing the actual version of the API can be found here below:



Relevant classes are:

1. Class/Interface	Description
--------------------	-------------

<a href="#">Driver</a>	<p>A driver adding the ability to work with a coverage format or service.</p> <p>Classes implementing this interface basically act as factory for creating connections to coverage sources like files, WCS services, WMS services, databases, etc...</p> <p>This class also offers basic create / delete functionality (which can be useful for file based coverage formats).</p> <p>Purpose of this class is to provide basic information about a certain coverage service/format as well as about the parameters needed in order to connect to a source which such a service/format is able to work against.</p> <p>Notice that as part as the roll of a "factory" interface this class makes available an <code>isAvailable()</code> method which should check if all the needed dependencies which can be jars as well as native libs or configuration files.</p>
<a href="#">CoverageAccess</a>	<p>Represents a Physical storage of coverage data (that we have a connection to).</p> <p>Please note that this service may be remote (or otherwise slow). You are doing IO here and should treat this class with respect - please do not access these methods from a display thread.</p>
<a href="#">CoverageSource</a>	<p>Allows read-only access to a Coverage</p>
<a href="#">CoverageStore</a>	<p>Provides read-write access to a coverage data product</p>
<a href="#">CoverageCapabilities</a>	<p>Describes the capabilities of this <a href="#">CoverageSource</a></p>
<a href="#">CoverageRequest</a>	<p>Captures the base elements of a request for a coverage source subclass.</p> <p>The request can be a request for reading data, a request for writing data, etc.</p> <p>Notice that using additional parameters and Hints a user can customize the request behavior and meaning.</p>
<a href="#">CoverageReadRequest</a>	<p>Captures element of a request to read data from an underlying coverage.</p>
<a href="#">CoverageResponse</a>	<p>Captures the status of the response to a coverage request.</p>

<a href="#">RangeType</a>	Defines the fields (categories, measures, or values) in the range records available for each location in the coverage domain.

We still need to cope with some topics:

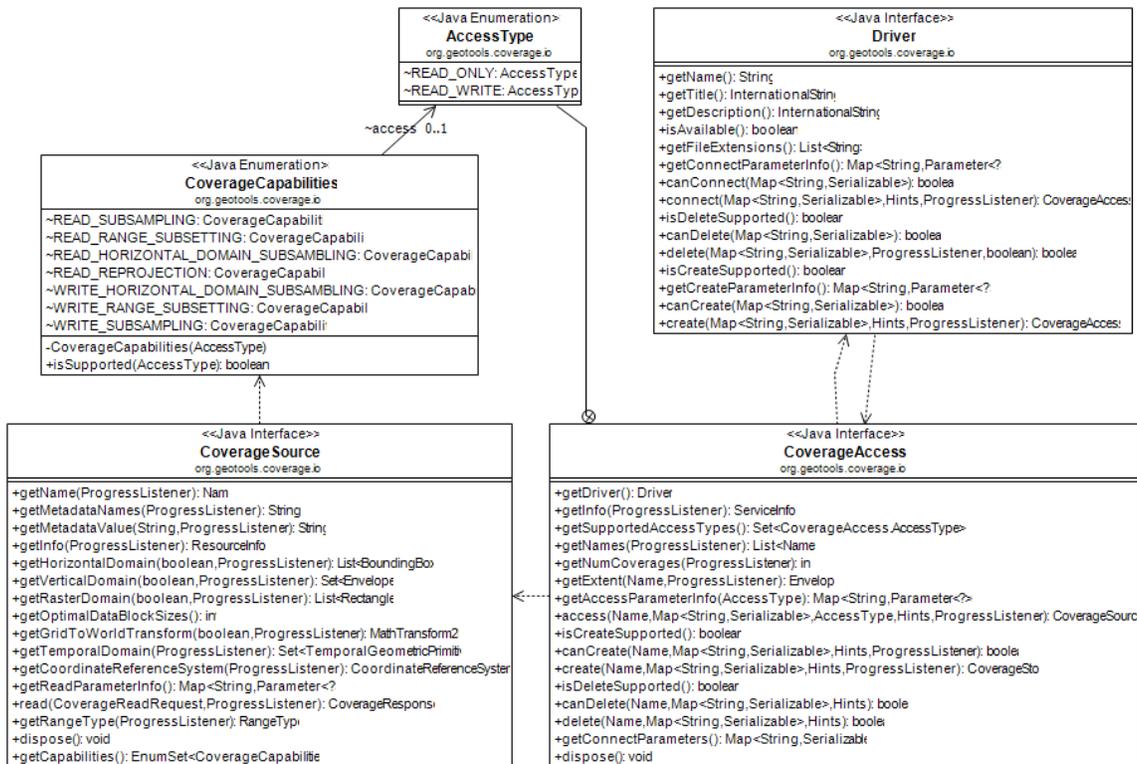
- How to/Where to handle Ground Control Points and management of unrectified coverage
- Support for a tree of coverage with real and fake nodes to better suit WCS 1.1.2
- Harmonize with range with Record/RecordType from ISO 19103 and also look at the SimpleFeatureType work in geotools
- Statistics information management
- Overviews management
- Metadata management should be improved. Actually it has been copied from old GCE API.

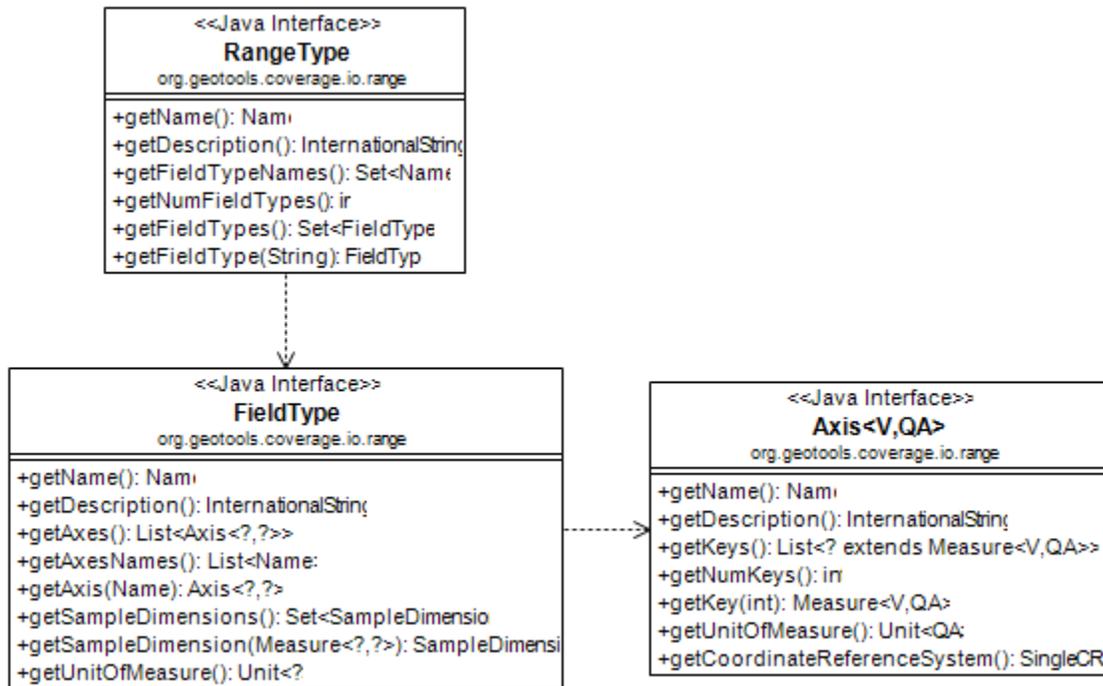
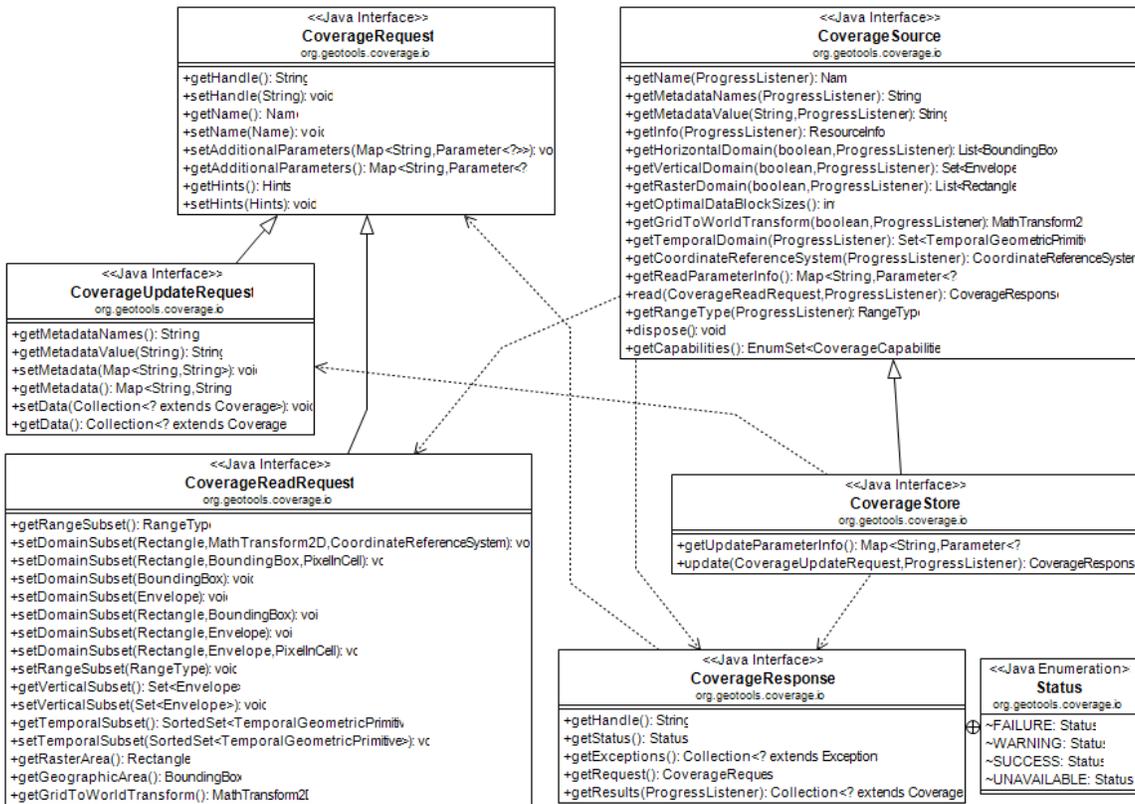
See TBD comments on [CoverageSource javadoc](#).

Javadocs are available at <ftp://ftp.geo-solutions.it> as anonymous user, under /incoming/javadoc/coverage-api-docs.zip

### UML Diagrams

Here below, more detailed UML class diagrams with available methods:





## Out of scope

The following extensions are possible:

- GridServiceInfo - we will wait for experience to dictate what extra information is useful

- GridResourceInfo - we will wait for experience to dictate what extra information is useful

## Documentation Changes

*list the pages effected by this proposal*

- [Data Module](#) from the [Module matrix](#) page