

ImageIO-EXT GDAL

Module:	gt-imageio-ext-gdal
Module Maintainer:	Daniele Romagnoli
Email Help:	Geotools-gt2-users@lists.sourceforge.net
Volunteer:	geotools-devel@lists.sourceforge.net
Status:	★★★★★
IP Review:	review.ap

Gold Star Quality Assurance Check:

- ★ IP Check: review.ap added, all headers are in place.
- ★ Releasable: no blocking issues, but feedback from potential users is appreciated
- ★ Used in anger: Used by GeoServer and uDig
- ★ Optimized.
- ★ Supported: [Javadoc](#) available, module maintainer does watches user list, answers email.

Background

In the GeoSpatial context, a wide range of raster data format exists, however, GeoTools successfully handles only a small subset of them (GeoTiff, ArcGrid, WorldImage and Gtopo30). Developing a coverage reader to support a new format is not a trivial task, hence it makes sense to try and leverage other libraries that already have an extensive set of supported formats.

Leveraging GDAL

[GDAL](#) is a well known open source library providing access access [tens of different coverage formats](#) with a common API.

Programming language wise, GDAL is a C++ api, but it provides bindings for integration with [various other languages \(python, VB6, java...\)](#) based on the [SWIG code generator](#).

License wise, GDAL is based on the [X/MIT license](#).

The Image I/O-Ext project

The [Image I/O-Ext project](#) leverages existing native APIs to extend the set of formats that the [JAI ImageIO](#) API can read. At the moment, the project is building on top of GDAL java bindings and ImageMagick java bindings, and hopefully the list will grow in the future. The project is compatible with GeoTools both from the license point of view (LGPL) and the API point of view, since most of the current coverage readers are leveraging ImageIO already. At the moment Image I/O-Ext does not support all the formats GDAL provides, but adding a new format is usually a matter of few hours of work (for a complete list see of the currently supported formats see [Image I/O-Ext's GDAL Supported formats](#)).

Actual stable GeoTools version leverages on Image I/O-Ext is 1.0.8. GeoTools 8.x will be updated to leverage on the latest Image I/O-Ext 1.1.0.

Libraries and native bindings

Of course, in order to leverage the native readers provided by GDAL, the GDAL libraries and java bindings have to be built for the target platform.

Image I/O Ext provides some options in this regard:

- A set of ready-to-use distribution files containing requested DLLs/SOs are available in the [ImageI/O-Ext Download page](#). (Read available notes and download the proper distribution).
- Step-by-step instructions provided in the ImageI/O-Ext setup guide, available either as an [OpenOffice document](#) or as a [PDF document](#). This option suits people that have the build tools ready, are proficient with them, and need custom builds for some reasons (for example, license ones, the Kakadu JPEG 2000 readers are not open source so they cannot be redistributed along with Image I/O Ext).

The gdal module

The purpose of the proposed module is to extend the raster data access capabilities of the GeoTools library and, consequently, the GeoServer capabilities too, by building on top of Image I/O-Ext, and adding the necessary GIS features to the otherwise plain ImageIO API offered by Image I/O-Ext.

Given that most of the hard work is already done by the lower levels, the GeoTools readers only have to deal with format specific metadata that aren't already handled in a generic way by GDAL itself (informations such as the CRS, the bounds and the bands structure are already provided thru a generic API).

The only residual difficulty is the distribution one, which can be handled by providing users with install packages based on the pre-built binaries of the GDAL libraries, much like JAI ImageIO is already doing for its native counterparts.

Supported Formats

Actually supported formats on the gt-imageio-ext-gdal module are:

- MrSID
- Arc/Info Binary GRID (AIG)
- DTED
- Erdas Imagine
- ECW
- JPEG2000 (based either Kakadu or MrSid libraries)
- NITF
- ESRI Hdr
- ENVI Hdr

Extending the raster data access capabilities with add for additional formats is a simple task due to the GDAL power capabilities integration in the ImageI/O-Ext.

Usually, writing a new ImageI/O-Ext plugin for a "not too much complex" format requires 10-15 minutes of work (you only need to setup a XXXImageReader as well as a XXXImageReaderSpi). The same (or minor time) is requested to setup the related GeoTools plugin in the gt-imageio-ext-gdal module (You only need to setup a XXXFormat, a XXXFormatFactory and a XXXReader), where XXX is usually the name of the new format to be supported. With further developments it would be nice to extend GeoTools raster data access capabilities to support any format supported by GDAL (See: http://www.gdal.org/formats_list.html).

Note that, actually, only reading capabilities are provided.

Licensing Issues

Not all the GDAL supported formats are freely available and freely redistributable. Notable exceptions are:

- ECW: the SDK license agreement prohibits the SDK use on a server (So, for the moment, we can't distribute

ECW support with GeoServer).

- JPEG2000 Kakadu reader: this is a high performance, commercial (and quite expensive) library that needs to be bought in order to be used at all (unlikely the ECW license, you have to buy a license even for desktop usage).

Notes on old unsupported modules

Actually, in the unsupported section, there are an ECW and a MrSID modules. Since some GeoTools/Udig/GeoServer people use these modules, they wouldn't be removed until the GDAL based Geotools plugin will ready and fully tested.

Unofficial GeoServer release leveraging on ImageIO-EXT GDAL

For testing purpose, while waiting for a 1.7 geoserver stable release, an unofficial GeoServer release has been developed which leverages on this module.

The zipped geoserver WAR is available [here](#). A brief guide containing instructions on how to setup this release is available [here](#). (It also contains links to needed Jars and Dynamic Libraries)

Note that, due to the licensing issues introduced before, the GDAL Libraries available in the distribution section doesn't contain JP2KAK library. See [ImageIO-EXT Setup Guide](#), sections **GDAL Requirements** and **GDAL Configurations** for further information on enabling Kakadu support on GDAL.

Some notes on the GeoServer usage

- **LOW PERFORMANCES?:** Some available images could be striped (where tiles are in the form Nx1 pixels). In such a case, set the **SUGGESTED_TILE_SIZE** read parameter to speed up the rendering when using JAI ImageRead (otherwise a lot of "single line" JAI image reads will be computed). We will add some auto-tilesize-tuning capabilities in the future.



The screenshot shows a configuration form with the following fields and values:

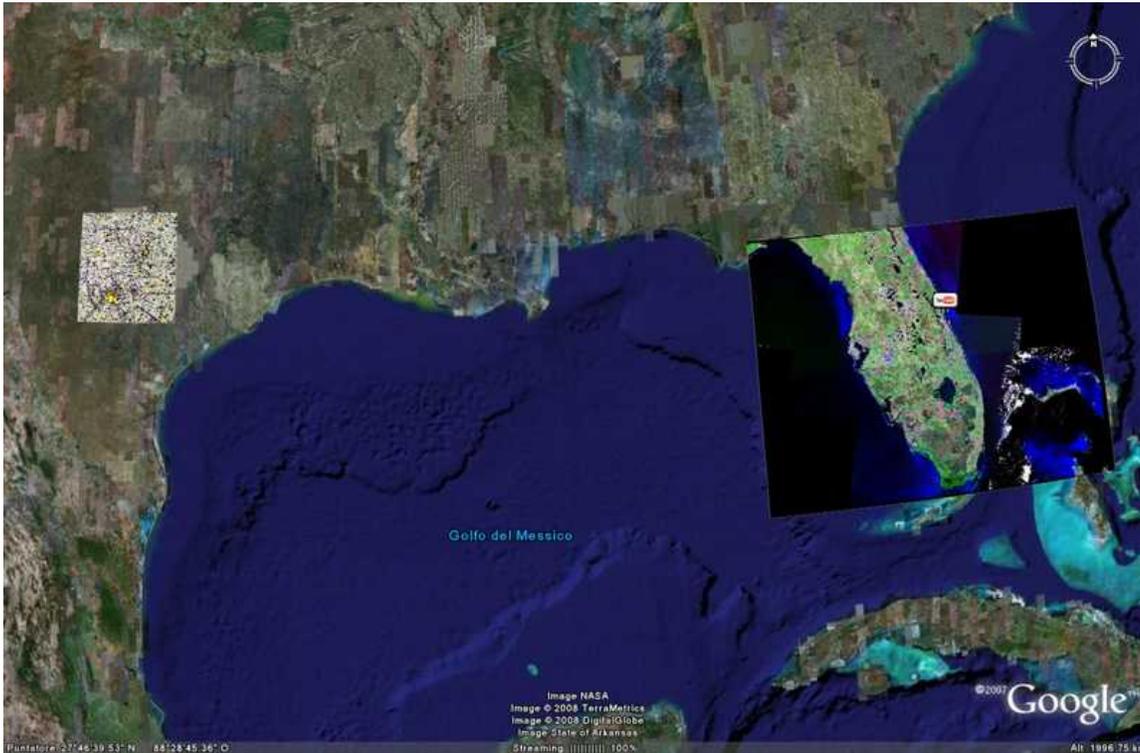
- Default Int. Method: nearest neighbor
- Interpolation Methods: nearest neighbor, bilinear, bicubic
- USE_MULTITHREADING: false
- SUGGESTED_TILE_SIZE: (empty field)
- USE_JAI_IMAGEREAD: true

At the bottom of the form are two buttons: "Submit" and "Reset".

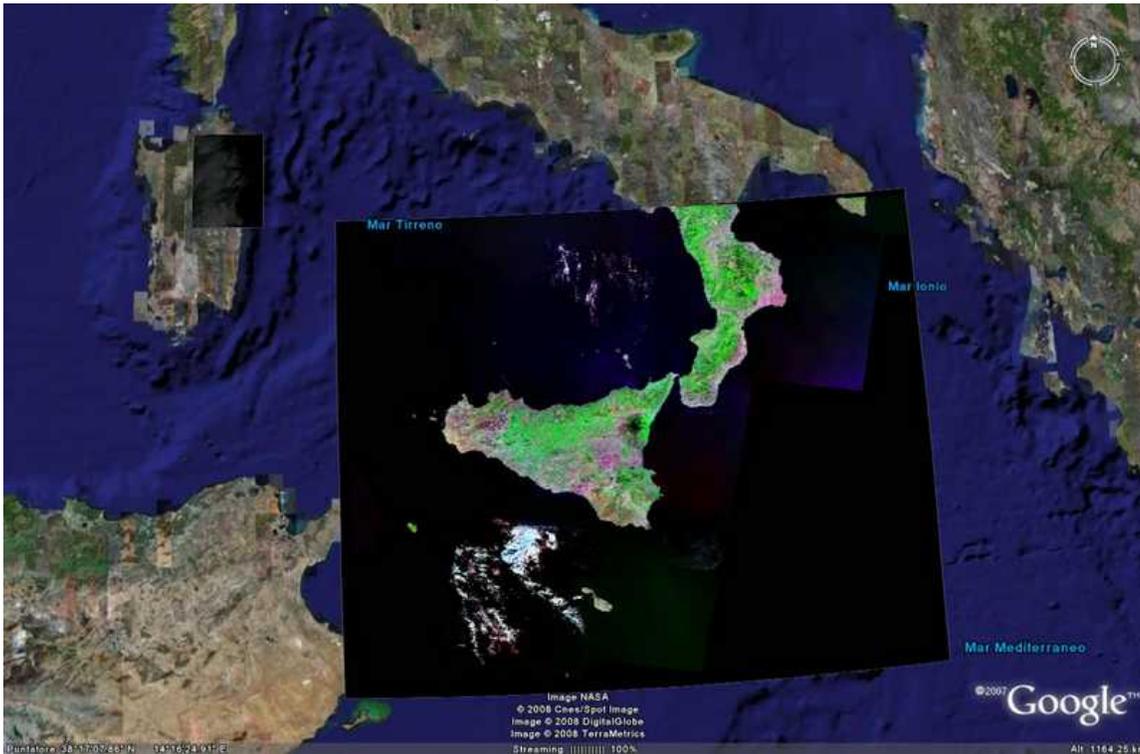
- Due to the underlying GDAL mechanisms, if you have built GDAL with support for several JP2K Drivers (JP2KAK, JP2MrSID, JP2ECW), you need to set a GDAL_SKIP environment variable to specify the JP2K Driver you wish to use. As an instance, if you want to use the JP2MrSID driver and GDAL has been built against Kakadu too, set GDAL_SKIP=JP2KAK. The cause of this is the driver registration mechanisms of GDAL. As a workaround, we will add more checks on the formatFactories.

Some screenshots:

In the left side of the image, the rendering of a NITF-CADRG data sample;
In the right side, a Landsat7 MrSID data sample



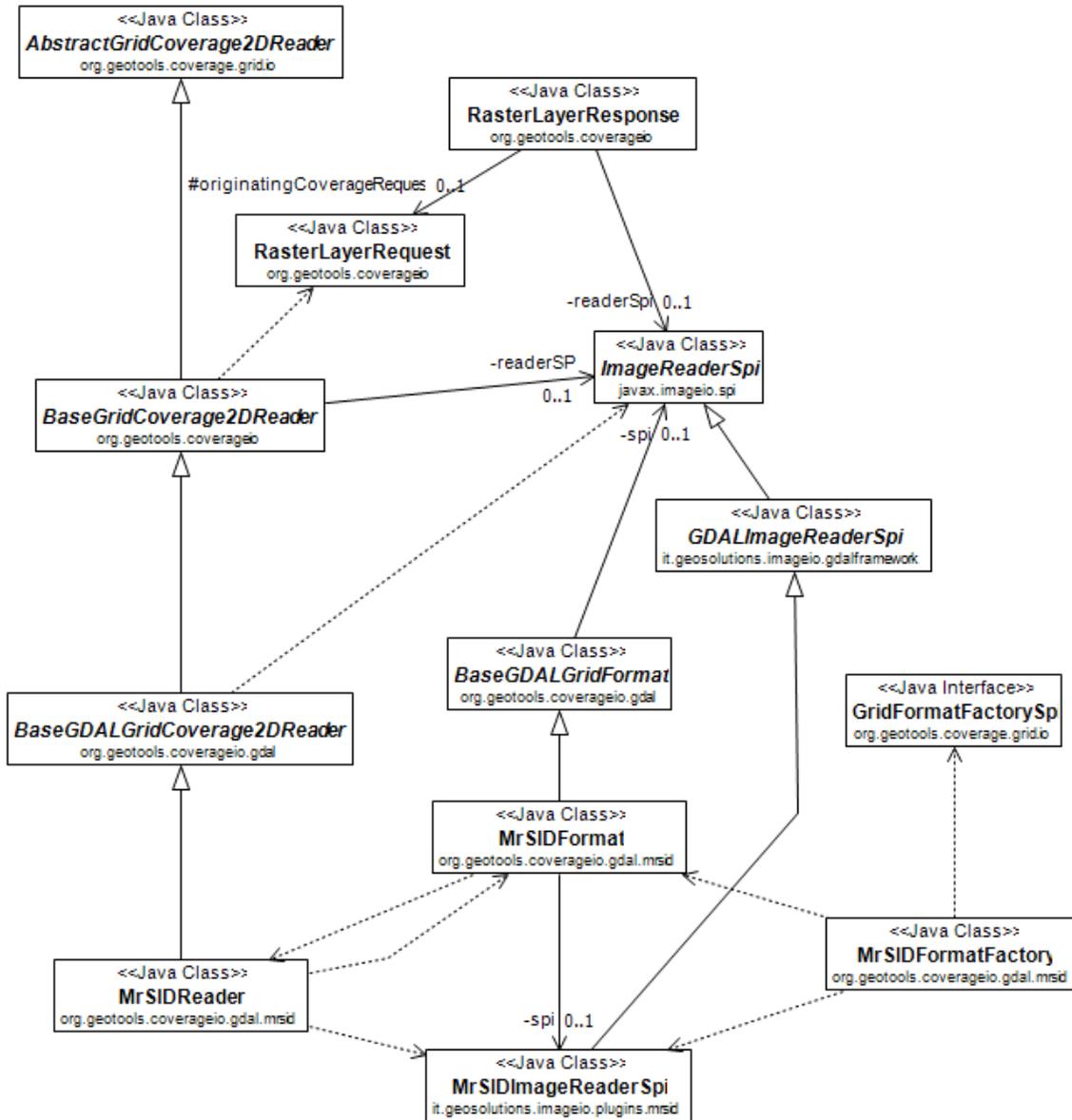
In the left side of the image (Sardegna), the rendering of a DTED data sample;
 In the center, a Landsat7 MrSID data sample

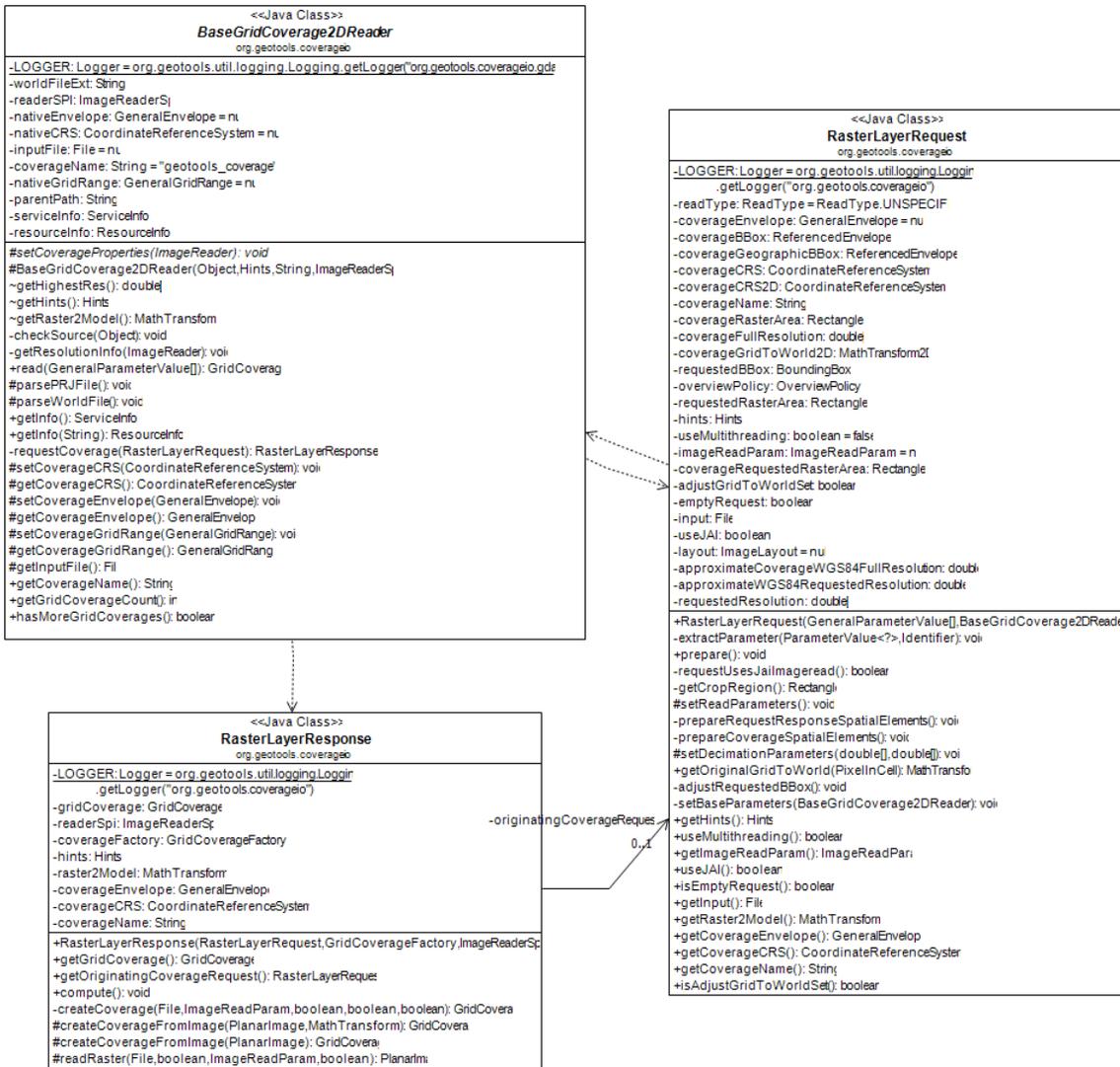


Technical details

The proposed module is composed of a main abstract BaseGridCoverage2DReader (extending AbstractGridCoverage2DReader) which allows to read coverages by leveraging on proper coverage requestes (represented by instances of RasterLayerRequest class) computed returning coverage responses (represented by

instances of RasterLayerResponse class). [see the related javadoc for further details: [GT ImageI/O-Ext-GDAL Javadoc](#)]. BaseGridCoverage2DReader is extended by BaseGDALGridCoverage2DReader which leverages on the core of the ImageI/O-Ext project: the **gdalframework** module. Since, the hard work of data access and management is performed by the GdalFramework, the Geotools reader simply needs to query it to obtain all the needed information (such as CRS, Envelope, raster properties) and delegate the read to the ImageI/O-Ext.





The GDAL Data model allows to represent any datasource (of any format) with a basic [Dataset](#) object. ImageIO-Ext wraps all the properties of this object within a GDALCommonIOImageMetadata object extending the ImageIO IOMetadata class, which can be leveraged at two levels:

- The implementation of the BaseGDALGridCoverage2DReader will query an instance of this object obtained by the framework to setup Envelope, CRS, GeoTransformation and GridRange.
- Some formats allow to specify/handle an additional set of metadata. In such a context, an extended implementation of the geotools GDAL reader will leverage on a proper GDAL metadata subclass. (This will be achieved by overloading the setCoverageProperties method).

The GDAL based plugin allows to customize the data access by means of a set of read parameters. First of all, it leverages on the AbstractGridFormat USE_JAI_IMAGEREAD parameter/Hint which allows to specify whether the underlying ImageIO plugin should use a JAI ImageRead operation (and thus deferred execution model, tile caching/scheduling, ...) or a more simple direct call to the read method of the ImageReader. Note that when performing a JAI ImageRead operation, the internal computations get access to any involved tile using an ImageReader read operation. In case some raster data are striped (where tiles are in the form Nx1 pixels) the tiles read process may be time consuming. To improve this, the parameter SUGGESTED_TILE_SIZE allows to specify a different layout to be used by JAI ImageRead operation when consuming tiles. Values to be specified are in the form "W,H" where W represents the suggested tile width and H represents the suggested tile height. Moreover, ImageIO-Ext introduces a very useful JAI operation: ImageReadMT. This allows to perform multithreaded

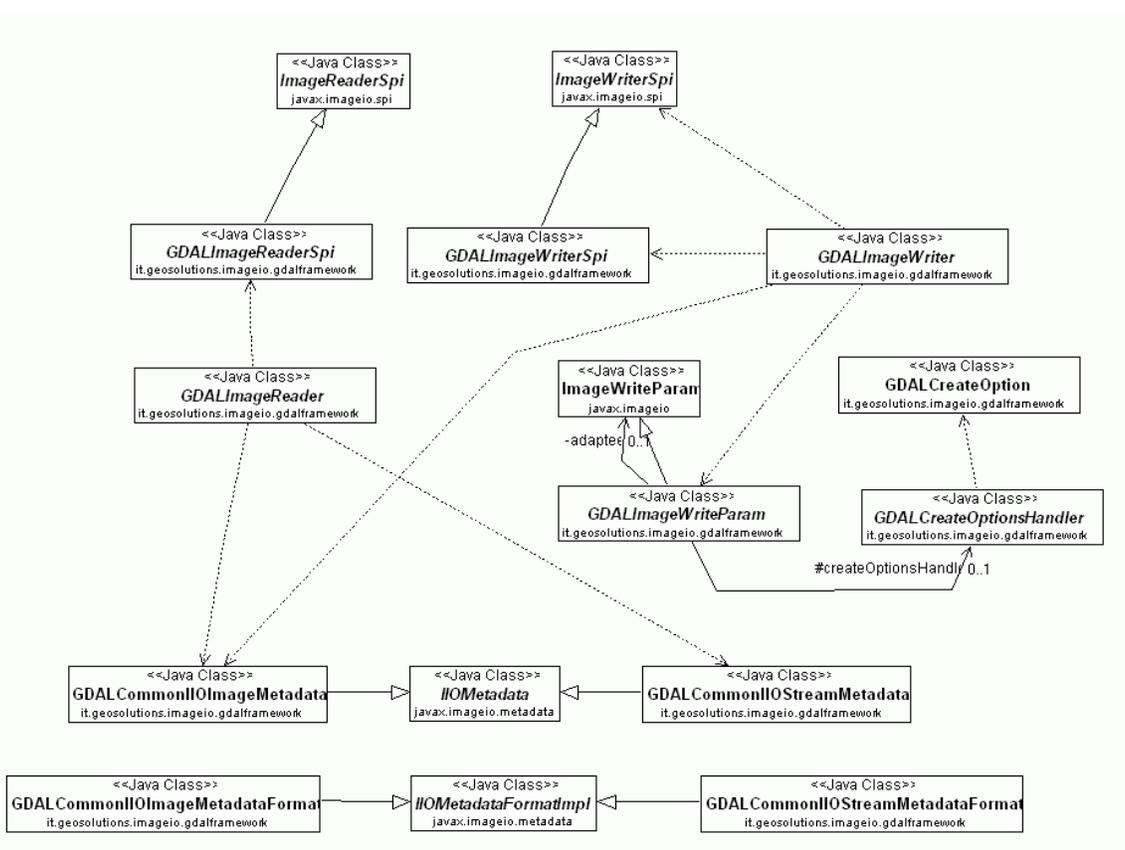
JAI ImageRead operations.

To tell the geotools reader to use multithreading, you simply need to specify the USE_MULTITHREADING parameter (Set as TRUE).

These parameters are declared within a BaseGDALGridFormat extending the AbstractGridFormat.

ImageI/O-Ext additional information

Here are a couple of class diagrams showing how ImageI/O-Ext GDAL framework is set up (see the related javadoc for further details: [ImageI/O-Ext Javadoc](#))



... and how the MrSid plugin is using it.

```

<<Java Class>>
GDALCommonIOImageMetadata
it.geosolutions.imageio.gdalframework
+GDALCommonIOImageMetadata
+GDALCommonIOImageMetadata
+GDALCommonIOImageMetadata
+GDALCommonIOImageMetadata
+GDALCommonIOImageMetadata
-GDALCommonIOImageMetadata
-setGeoreferencingInfo
-setMembers()
-createCommonNativeTree()
+getAsTree()
+isReadOnly()
+mergeTree()
+reset()
+getDatasetName()
+getDescription()
+getDriverName()
+getDriverDescription()
+getNumBands()
+getWidth()
+getHeight()
+getTileHeight()
+getTileWidth()
+getColorModel()
+getSampleModel()
+getProjection()
+getGeoTransformation()
+getGcpNumber()
+getGcpProjection()
+getGcps()
+getNumOverviews()
+getColorInterpretations()
+getMaximum()
+getMinimum()
+getScale()
+getOffset()
+getNoDataValue()
-checkBandIndex()
-performTileSizeTuning
#getGdalMetadataDomain
#getGdalMetadataDomainsList
+asWritable()

```

```

<<Java Class>>
GDALImageReaderSpi
it.geosolutions.imageio.gdalframework
+getSupportedFormats()
+GDALImageReaderSpi
+canDecodeInput()
#isDecodable()
+isAvailable()

```

```

<<Java Class>>
GDALImageReader
it.geosolutions.imageio.gdalframework
+setInput()
+getDatasetMetadata()
+GDALImageReader
+GDALImageReader
#checkImageIndex()
-initialize()
#createDatasetMetadata()
#createDatasetMetadata()
-readDatasetRaster()
-readDatasetRaster()
#getDatasetSource()
+setInput()
+dispose()
+reset()
+getImageTypes()
+read()
+readRaster()
+read()
+getNumImages()
+getWidth()
+getHeight()
+getTileHeight()
+getTileWidth()
+getProjection()
+getGeoTransform()
+getGcps()
+getGCPProjection()
+getGCPCount()
+getNoDataValue()
+getOffset()
+getScale()
+getMinimum()
+getMaximum()
+getStreamMetadata()
+getImageMetadata()

```

```

<<Java Class>>
MrSIDImageReaderSpi
it.geosolutions.imageio.plugins.msids

```

```

<<Java Class>>
MrSIDImageReader
it.geosolutions.imageio.plugins.msids

```

```

<<Java Class>>
MrSIDIOImageMetadata
it.geosolutions.imageio.plugins.msids

```

```

<<Java Class>>
IIOMetadataFormatMpi
javax.imageio.metadata

```

```

<<Java Class>>
MrSIDIOImageMetadataFormat
it.geosolutions.imageio.plugins.msids
#mrsidMetadataInstance
0..1

```

