Image I/O-EXT – User Guide
V. 1.0

Eng. Daniele Romagnoli

Eng. Simone Giannecchini

1 – Introduction

This simple guide will provide you some instructions about how to use the Image I/O-Ext capabilities as well as how to extend it by adding new plugins.

2 – Pre-Requirements

Before using the Image I/O-Ext, you should have already setup all the required elements. In case you need help on achieving this, you may look at the Image I/O-Ext Setup Guide, available here:

https://imageio-ext.dev.java.net/svn/imageio-ext/trunk/documentation/ImageioExt-SetupGuide.odt

After completing all the required steps suggested in that guide you should be ready to use the Image I/O-Ext Project. During the following instructions we assume you are using Eclipse as IDE.

First of all, you may download Eclipse from this site: http://www.eclipse.org/downloads/

From the "*Eclipse IDE for Java Developers*" section, click on the link located in the right side, referring the proper OS version you need (Windows/Linux).

3 – Usage

Image I/O-Ext extends the Java SUN's Image I/O which is a pluggable architecture for working with images stored in files and accessed across the network by means of a wide set of packages which allow to perform data access (read/write operations) and data manipulation, as well as a set of classes to define new image readers, image writers. Basically you would get access to a data source using a specific plugin which is able to manage that specific data format. Let us now introduce some tips on how to leverage on the Image I/O-Ext capabilities.

3.1 – Setup Customizations

When creating a new project which requires to use the Image I/O-Ext, you need to add several required libraries. Supposing you have already built and installed the Image I/O-Ext project as explained in the Setup Guide, you will find all what you need in your Maven2 Repository. Basically the Image I/O-Ext core is built on top of 3 main libraries, available with the following JARs:

- **imageio-ext-gdal**
- **imageio-ext-gdalframework**
- **imageio-ext-customstreams**

Finally, depending on the specific format on which you need to get access, you must also add the proper library which provides access to that. As an instance, if you need to work on ECW files, you also need to add the **imageio-ext-gdalecw** jar.

Before starting introducing some examples on how to perform data access and manipulation let us point out that any module composing the Image I/O-Ext project contains a set of Junit test case classes which are used by maven to test the project functionalities. To acquire confidence with the basic Image I/O (and Image I/O-Ext) data access/data manipulation ways, you can take a look on them with Eclipse, as explained in the following subsections.

3.1.1 – Build and import Eclipse projects

Maven allows to build ready-to-use Eclipse projects by automatically setting the required dependencies of each project. In case you need to setup eclipse projects for the main framework go in your `imageio-ext\trunk\library` folder and run:

`mvn eclipse:eclipse` (this will build ready-to-use projects containing the main framework).

Finally, if you are interested in building eclipse projects for all the available Image I/O-Ext plugins, you should enter in your `imageio-ext\trunk\plugin` folder and run again

`mvn eclipse:eclipse`

Alternatively, if you are interested in a single plugin, you may enter in the proper subfolder, as an instance, `imageio-ext\trunk\plugin\gdalecw` and run the same command.

At this point, you should be ready to run Eclipse and import the just produced projects as follow. From the Eclipse *File* menu: *File->"Import"->"General"->"Existings Projects into Workspace"->* and select the root directory where you previously downloaded the whole Image I/O-Ext project. When ready, the "*Projects:"* window should contain all the projects previously built with "`mvn eclipse:eclipse`". Select the ones you are interested in and go on.

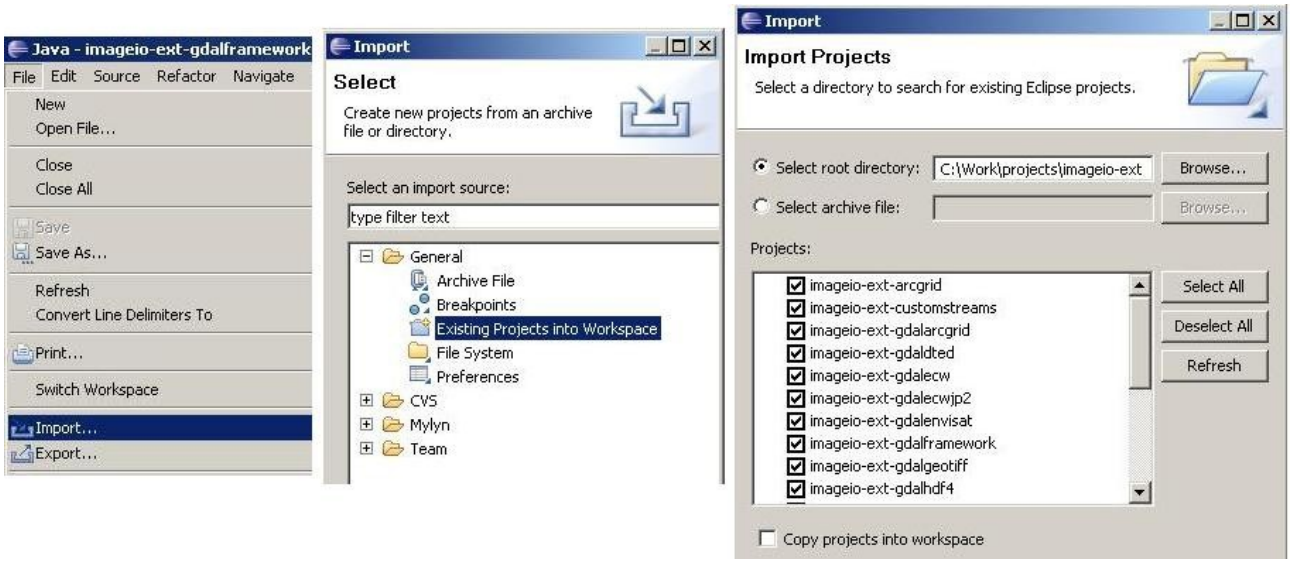

Figure 1: Importing Projects from Eclipse

3.1.2 – Setup dependencies on Eclipse

As stated in 3.1 some of these projects need several dependencies (as an instance, gdal based plugins require `imageio-ext-gdalframework.jar`, `imageio-ext-gdal.jar`, `imageio-ext-customstream.jar` and some others) which are contained in the maven2 repository.

Be sure you have properly set the *M2_REPO* classpath variable as explained here below.
Open the properties of one of your just imported projects and select "*Java Build Path*" entry in the left column. Then go to the "*Libraries*" Tab and check if your M2_REPO variable has been defined. To define it, click on "*Add Variable"->"Configure Variables..."->"New..."* Set "M2_REPO" as Name

and a proper location as Path. Usually, the maven2 repository is located on the user folder of your hard disk, as an instance:

- "`C:\Documents and settings\YourUser\.m2\repository`" on Windows XP
- "`C:\Users\YourUser\.m2\repository`" on Windows Vista
- "`/home/youruser/.m2/repository`" on Linux

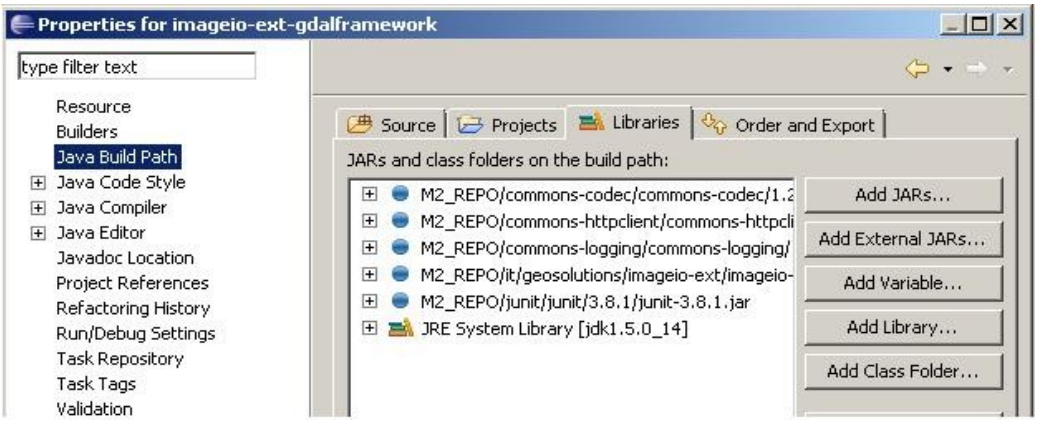

Figure 2: Configuring M2_REPO dependencies

At this point, everything should be ready to run your first test.

3.1.3 – Test run customizations on Eclipse

Finally, select a test class from src/test/java folder of an imported plugin and select *Run As->Java Application*. Note that all tests do not display image loaded. If you want to view the images in the available tests, just specify a proper JVM argument for the test run. Select a test class and select "*Run As" -> "Run..."*[1] . Then go in the *Arguments* TAB and add the following line in the "*VM arguments:*" box: `-Dorg.geotools.test.interactive=true`

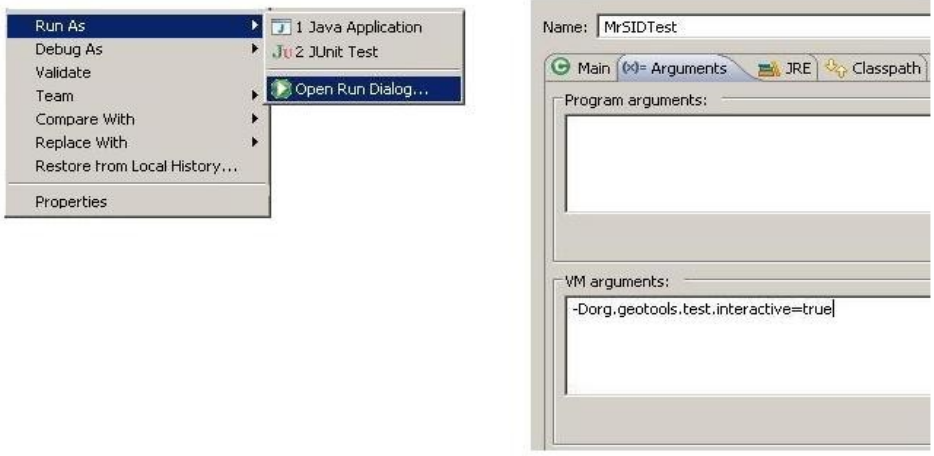

Figure 3: Customizing Run

[1]Depending on your Eclipse IDE version, the available menu command may be "*Open Run Dialog...*" instead of "*Run...*"

On Linux, in case the required libraries (.so) are not found during tests, you could check the LD_LIBRARY_PATH environment variable[2]. Select the test class from src/test/java of a plugin project and select *Run As->Open Run Dialog*. Then select the "*Environment*" tab and check whether the LD_LIBRARY_PATH has been properly specified. If such a variable is not yet set, just add it using the "Select..." button. Be sure it also contains a path where the required libraries have be placed, as an instance, the `/usr/local/lib/` path. If missing, simply add it using the "Edit..." button, and append the `:/usr/local/lib/` string to the value of such a variable.

Let us now provide some example on how to use the Image I/O-Ext project.

3.2 – Read Access

Let suppose you need to get an image from a file stored on your disk (as an instance on `C:\data\sample.ecw`). Depending on your needs, the read access could be performed in several different ways. Let us now start with the simplest type of read operation.

3.2.1 – Simplest Read Access

Use the following code in a method of your class to get an image by reading all the data available from the underlying file:

```
final File file = new File("C:/data/sample.ecw");
final ImageReader reader = new ECWImageReaderSpi().createReaderInstance();
reader.setInput(file);
final RenderedImage image = reader.read(0);
```

The just introduced example may be useful to read a whole dataset. However, customizing a read operation is a more frequent task since, as an instance, you could need a reduced part of the dataset or you could need to minimize the memory request. To achieve this objective, you may properly set an image read parameter to be passed as argument of the read operation. Let us illustrate this approach in the following example.

3.2.2 – Source settings parametrization read

Let us now suppose your ECW file is very big (a 10000x10000 pixels dataset representing sea and shores) and, as an instance, you need to load a rescaled view (4 times smaller) of a reduced part of the image, let's say, of a region composed of 5000x3600 pixels starting at the x,y pixel coordinates (5000, 6400), as illustrated in Figure 4.

With the following settings the read operation will get an image of 1250x900 pixels.

The following lines of code allow to obtain the requested parametrized read operation:

Figure 4: source settings

```
final File file = new File("C:/data/bigsample.ecw");
final ImageReader reader = new ECWImageReaderSpi().createReaderInstance();
final ImageReadParam param = reader.getDefaultReadParam();
param.setSourceSubsampling(4, 4, 0, 0);
```

[2]In case you have used the deploy module, all the required libraries should have be placed by Maven in your JRE and everything should already work.

```
param.setSourceRegion(new Rectangle(5000, 6400, 5000, 3600));
reader.setInput(file);
final RenderedImage image = reader.read(0, param);
```

3.2.3 – JAI ImageRead

In the previous examples we have performed data access directly using the `read` methods of an `ImageReader` instance. However, there is another way to perform data access and data manipulation with better performances: using the JAI-Image I/O Toolkit.

3.2.3.1 – Little Introduction on JAI and JAI-Image I/O Toolkit

JAI-Image I/O Toolkit provides Image I/O-based read and write operations for Java Advanced Imaging (JAI ImageRead and ImageWrite operations). The JAI is a set of APIs which allows sophisticated, high performance image processing functionalities, such as rescales, rotations, crops, convolutions, bands compositions, shears, sub-samplings and much more. Moreover, JAI provides built-in support for a wide set of mechanisms such as tiling, tile-caching, deferred execution and operations chaining. Let us provide a minimal introduction on these topics in order to know how data may be accessed/manipulated. Basically:

- Tiling refers to the technic of building a tessellation of a big image in smaller squares, allowing to load and process only a reduced subset of this with the advantage of a reduced memory consumption and a minor loading time.
- Tile-Caching refers to the capability of caching tiles which need to be frequently used or involved in some type of processing.
- Deferred execution refers to a mechanism which allows loading data only when they are really need.
- Operations chaining refers to a technic which allows the user to sets a chain of operations by concatenating them one after the other as needed, building directed acyclic graph. The graph starts from a source (as an instance an originating image) and ends with a sink (as an instance, the rendering on the monitor). In such a context, the meaning of the term deferred execution is that no data pixels are loaded in memory until a sink is reached.

Being the Image I/O-Ext, as its name suggests, an extension of the standard Image I/O architecture, it may be easily involved in JAI ImageRead and JAI ImageWrite operations.

Note that when using the JAI ImageRead, a call to the adopted `ImageReader`'s `read` method will be performed for any tile which needs to be accessed for the data loading. This approach is different with respect to the type of data loading performed by a manual call to the `read` method which simply loads all you need at-once. For this reason, in several circumstances you may notice that loading an image using a not properly set JAI ImageRead operation may require a lot of time. This mainly happens when the underlying dataset is striped, having each tile composed of a data row, requiring a JNI access to the underlying GDAL DLLs for each row/tile to be managed. However, this issue may be easily solved by specifying an ImageLayout when creating the JAI operation.

3.3 – Write Access

3.4 – Metadata

4 – Capabilities Extensions

Although Image I/O-Ext provides several unrelated capabilities, its main feature is allowing to access and manipulate a set of raster data formats. The home page of the project (https://imageio-ext.dev.java.net/) reports a list of all the actually supported formats as well as the type of supported access (read/write). Basically, the data access leverages on a set of JNI Bindings to GDAL which is a raster Geospatial Data Abstraction Library capable of managing a very large set of raster formats. For this reason a good objective of this Project could be to expose a plugin for almost any raster format supported by GDAL. You may find the list of all GDAL's supported formats at: http://www.gdal.org/formats_list.html

In case a format is not yet supported by Image I/O-Ext, it is possible to define a new plugin for it.

Basically all you need to do is writing a specific ImageReader/ImageReaderSpi as well as a specific ImageWriter/ImageWriterSpi in case you need to support write operations too[3].

[3]Be sure the underlying GDAL Driver supports creation for that format.