Image I/O-EXT – User Guide
V. 1.0

Dott. Ing. Daniele Romagnoli

Dott. Ing. 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. During the following instructions we assume you are using Eclipse as IDE.

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.

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 the Image I/O-Ext use, you need to customize your Java Build Path by adding a set of 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 that you need in your Maven2 Repository. Basically the core of Image I/O-Ext is built on top of 3 main libraries, available with the following JARs, which need to be added in your build path:

- **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.

Let us now introduce some examples on how to perform a read access. Anyway, 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, which are located in the test folders.

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 access.

3.2.1 – Simplest Read Access

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

```
File file = new File("C:/data/sample.ecw");
ImageReader reader = new ECWImageReaderSpi().createReaderInstance();
reader.setInput(file);
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 containing 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 1.

Figure 1: source region / subsampling settings

The result will be an image of 1250x900.

The following lines of code allows to obtain this result:

```
File file = new File("C:/data/bigsample.ecw");
ImageReader reader = new ECWImageReaderSpi().createReaderInstance();
ImageReadParam param = reader.getDefaultReadParam();
param.setSourceSubsampling(4, 4, 0, 0);
param.setSourceRegion(new Rectangle(5000, 6400, 5000, 3600));
reader.setInput(file);
RenderedImage image = reader.read(0, param);
```

3.2.3 – JAI ImageRead

In the following examples we have performed data access by 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 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 used to perform JAI ImageRead and JAI ImageWrite operations. Let us now provide an example of how to perform a JAI ImageRead operation.

Note that when using the JAI ImageRead, a call to the `ImageReader`'s `read` method will be performed for any tile composing the image, which needs to be accessed for the data loading. This is different from the type of data loading performed by a manual call to the read method which simply load all you need at-once. For this reason, sometime 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 supported yet 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[1].

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