Image I/O-EXT – Setup Guide
V. 1.0

Eng. Daniele Romagnoli

Eng. Simone Giannecchini

1 – Introduction

This simple guide will provide you some instructions about how to setup all the libraries and tools needed to proficiently use the Image I/O-Ext Project.

2 – Requirements

The Image I/O-Ext project requires the same tools for both Windows and Linux OSs. However, the set of operations needed to properly configure all of them or specific versions of a tool may be different for one OS with respect to the other one. For this reason, these instructions have been split in two main chapters to differentiate the two typologies of setup procedures in order to provide all the OS specific instructions into a stand-alone section. By this way, Windows Users may read the proper chapter while Linux Users may skip it and go on to the linux dedicated chapter and viceversa although some instructions may result duplicated.

3 – Windows instructions

In some of the following sections, we are assuming you has Visual Studio 2003 properly installed.

3.1 – JAVA

First of all, you need your machine has JAVA 1.5.0_14 installed. You may download it from this site: http://java.sun.com/javase/downloads/index_jdk5.jsp

Select the JDK 5.0 Update 14 and download the Windows Offline Installation. Download it on your hard-disk and run the installer.

Finally, be sure to properly set a `JAVA_HOME` environment variable[1] referring to the location of the JDK (As an instance, `C:\ProgramFiles\Java\jdk.1.5.0_14`)

On Windows Vista you will need to choose an install location other than the default (the program files folder has access restrictions on it which will prevent maven from installing additional DLL files as part of our build process). As an example `C:\java\jdk.1.5.0_14` will work just fine.

3.2 – ANT

Apache ANT is another needed tool. You can download it from: http://ant.apache.org/

When you downloaded it (as an instance, on `C:\ProgramFiles\Apache-ant-1.7.0`), be sure to properly set an `ANT_HOME` environment variable referring to that location.

[1]To set an environment variable on Windows XP, open the Control Panel -> System. Then, in the "Advanced" tab, click "Environment Variables". Lastly, click the "NEW" button from the System variables box to add a new Environment Variable. Define the name of the Environment Variable (as an instance: JAVA_HOME) and provide a value for this variable (as an instance, the path of your JDK -> `C:\ProgramFiles\Java\jdk.1.5.0_14`). Note that if you open a windows command line or the Visual Studio Command Prompt prior to change or set new environment variables via the Control Panel, these changes will not be updated on your command line window. Thus you need to close your command line and open a new one.

Then, edit the `PATH` environment variable by adding the Ant's `bin` directory to (As an instance, `C:\ProgramFiles\Apache-ant-1.7.0\bin`).

3.3 – MAVEN 2

Maven 2 (in the following instructions it will be simply called "Maven") is another important tool needed by the Image I/O-Ext project. You can download it from http://maven.apache.org/download.html

Download the last `maven-xxx-bin.zip` version and unzip it somewhere on your hard-disk, as an instance on `C:\ProgramFiles\Apache-maven-2.0.7`.

Then, edit the `PATH` environment variable by adding the Maven's `bin` directory to (As an instance, `C:\ProgramFiles\Apache-maven-2.0.7\bin`).

3.4 – JAI

Go to https://jai.dev.java.net/binary-builds.html and select the daily builds link. Then, download the proper windows version. After you downloaded it, extract the content of the lib folder on your `JAVA_HOME\lib` folder as well as on your `JAVA_HOME\JRE\lib`. (where `JAVA_HOME` defines your JDK, as an instance, `c:\programFiles\java\jdk1.5.0_14`).

3.5 – JAI-ImageIO Toolkit

Go to https://jai-imageio.dev.java.net/binary-builds.html and select the daily builds link. Then, download the proper Linux version. After you downloaded it, extract the content of the lib folder on your `JAVA_HOME\lib` folder and on your `JAVA_HOME\JRE\lib`. (where `JAVA_HOME` defines your JDK, as an instance, `c:\programFiles\java\jdk1.5.0_14`).

3.6 – SWIG

Be sure you have properly downloaded SWIG, the Simplified Wrapper and Interface Generator which allows to produce JAVA bindings for C/C++ code. You can obtain it at this site:

http://www.swig.org/download.html

You should download the last swigwin version which includes a prebuilt executable. (When this guide has been released, the last available swigwin version was 1.3.31 available at: http://prdownloads.sourceforge.net/swig/swigwin-1.3.31.zip)

After you downloaded it, extract the zipped file on your hard-disk (as an instance on `C:\ProgramFiles\Swig`)

3.7 – GDAL

This is one of the most important sections. GDAL is the Geospatial Data Abstraction Library which provides data access to several data formats. Image I/O-Ext deeply leverages on this complex library which need to be properly configured.

3.7.1 – GDAL requirements

Depending on the format you wish to support, you need to properly download and setup several libraries prior to build GDAL. The following instructions describe how to achieve this for these formats:

- Kakadu (v. 5.2.6)
- MrSID (v 6.0.7)
- ECW (v 3.3)
- HDF4 (v. 4.2r1)

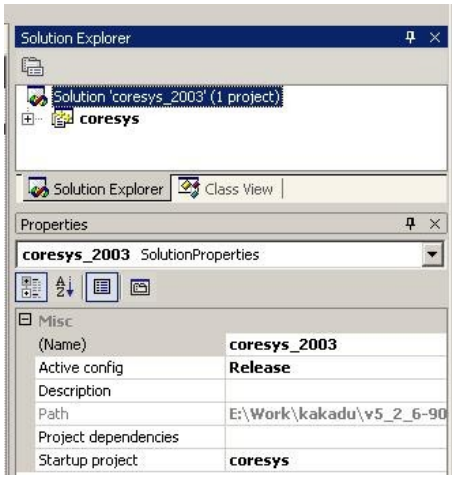
NOTE: If you have no time to follow all the instructions contained in the following sections or you encountered several problems which you cannot solve, you may leverage on the ready-to-use DLLs available with the Image I/O-EXT project using the deploy module. This module will deploy all the DLLs in the proper location (In such a case, the available GDAL DLL is built to support MrSID, ECW and HDF4). **However** it is worth to point out that this approach is not recommended and it should be used only as last resort. Anyway, the instructions to auto deploy DLL are contained in section 3.9.1

3.7.1.1 – Kakadu

Supposing you have your own Kakadu licensed source code, you need to build the Kakadu DLL.

The visual studio solution for kakadu building allows to build a shared Debug DLL. We need to change some settings to build a shared Release DLL. Otherwise, sometimes, memory allocations errors could occur especially when you build GDAL with support for several external formats (which need additional DLLs) since some libs may use MSVCRT71D and some others MSVCR71.

First step is opening the proper ready-to-use Visual Studio Solution[2] of kakadu coresys (located in `kakadu\VERSION\coresys`) and change the solution properties (right click on the solution -> Properties). Select *"Configuration Properties"* and switch the *Configuration* value from "*Debug*" to "*Release*".



Then, you are ready to build your solution. After you done this, open the Kakadu apps solution (located in `kakadu\VERSION\apps`). Change the Configuration properties to Release in the same

[2]As an instance, if you are using Visual Studio .Net 2003, open the `coresys_2003.sln` solution file.

way you just do it for Coresys solution and build this solution. If some errors occur for a specific project, rebuild that one. Once your build has done, copy the produced `.obj` files in the `kakadu\VERSION\apps\make` folder. The required `.obj` and the location from where to get them are listed in the following table.

File	Originating location
`args.obj`	`\v5_generated\v_compress\release`
`jp2.obj`	`\v5_generated\v_compress\release`
`mj2.obj`	`\v5_generated\v_compress\release`
`image_in.obj`	`\v5_generated\compress\release`
`palette.obj`	`\v5_generated\compress\release`
`roi_sources.obj`	`\v5_generated\compress\release`
`kdu_tiff.obj`	`\v5_generated\compress\release`
`jpx.obj`	`\v5_generated\compress\release`
`kdu_stripe_decompressor.obj`	`\v5_generated\buffer_expand\release`
`image_out.obj`	`\v5_generated\expand\release`

3.7.1.2 – MrSID

As a first requirement, you need the LizardTech Decoding Software Development Kit (DSDK). You can download it free of charge from this site: http://developer.lizardtech.com (You need to be registered in order to download it). After logged in, select *"Download" -> "Software Development Kits" -> "Download SDK's"*. Select the proper version of SDK to be downloaded (select the **GeoExpress SDK for Windows - VC7.1**).

When your download is completed, unzip the DSDK somewhere on your hard disk, as an instance, on `C:\ExternalLibraries\MrSid`

3.7.1.3 – ECW

Download the Image Compression SDK (source code) from ER Mapper site at this address:

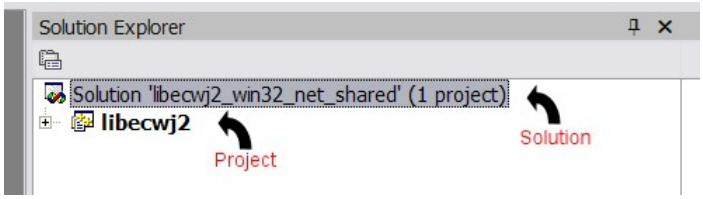
http://www.ermapper.com (You need to be registered in order to download it).

From the main site, select the menu *"products & downloads" -> "Image Compression SDK"* from the *Desktop* section. Download the *Image Compression SDK Source Code 3.3* file and extract this somewhere on your hard disk, as an instance on `C:\ExternalLibraries\libecwj2-3.3`.

Be sure you have *Microsoft Windows® Server 2003 R2 Platform SDK installed*. If not yet installed, download it from this location: http://www.microsoft.com/downloads/details.aspx?FamilyID=484269e2-3b89-47e3-8eb7-1f2be6d7123a&DisplayLang=en

Then, open the ready-to-use `libecwj2_win32_net_shared.vcproj` Visual Studio Project available in `C:\ExternalLibraries\libecwj2-3.3\Source\NCSBuildQmake`.

You will be asked to create a new solution. Select a location where to create the solution and, when done, change the solution properties (right click on the solution -> Properties).



Select *"Configuration Properties"* and switch the *Configuration* value from "*Debug*" to "*Release*".

Finally, select the libecwj2 project in your solution explorer and change its properties as follow:

Configuration Properties -> C/C++ -> General -> Additional Include Directories:

add an entry referring the `include` subfolder of the Microsoft Windows® Server 2003 R2 Platform SDK (As an instance: `"C:\ProgramFiles\Microsoft Platform SDK for Windows Server 2003 R2\Include"`). Then, you are ready to build your solution.

3.7.1.4 – HDF4

As a first requirement, you need to download the binary distribution of HDF4 release from this site:

http://hdf.ncsa.uiuc.edu/release4/obtain.html

Scroll this page until you find the link to the binary distribution file for Windows and download it. When your download is completed, unzip the binary somewhere on your hard disk, as an instance, on `C:\ExternalLibraries\HDF4`.

Then, enter in the `release` subfolder and create a `libpath` folder where you need to copy 4 `*.lib` files contained in `release\lib` and `release\dll` subfolders. They are: `hd421.lib, hd421m.lib, hm421.lib, hm421m.lib`

This could seem a strange workaround but it avoids errors when building GDAL against HDF4.

It is finally worth to point out that HDF4 leverages on some external libs: JPEG, ZLIB, SZIP. Be sure you have them. http://hdf.ncsa.uiuc.edu/release4/obtain.html also contains 3 links to download the required libraries: From the *External Software* section -> *External Libraries used by HDF, f*or each library, you need to select the "*Pre-Compiled Binaries*" link and select the Windows version.

3.7.2 – GDAL Configuration

Firstly, you need to download GDAL 1.4.4 from OSGeo SVN.

You may use Tortoise SVN (available at: http://tortoisesvn.net/downloads) to download it.

Create a `GDAL` folder on your Hard-disk and open the contextual menu on it (It's the menu which appears when you click on some element with the right's button of your mouse) and select "*SVN Checkout...*". Finally, specify https://svn.osgeo.org/gdal/tags/1.4.4/gdal as "URL of repository".

Although TortoiseSVN is a very helpful/easy-to-use program, it can reduce the performances of your machine. If you need a very light SVN client[3] you can download the Collabnet Subversion Command Line client, available at: http://downloads.open.collab.net/collabnet-subversion.html

When installed SVN, to checkout GDAL, you simply need to run the following command:

```
svn co http://svn.osgeo.org/gdal/tags/1.4.4/gdal gdal-1.4.4
```

After this, you need to apply the patch available at this location:

https://imageio-ext.dev.java.net/svn/imageio-ext/trunk/patches/gdal1.4.4.patch

This patch contains several changes for:

- Kakadu support: More Kakadu create options and error management supported. Makefile modified
- Java bindings: improved data access (read/write Dataset at once instead of read/write RasterBands at once; BandMap, PixelSpace, LineSpace and BandSpace parameters now are allowed), added setMetadataItem capabilities

To apply the patch, supposing you are using Tortoise SVN, you need to select the `GDAL` folder on your hard-disk, then, from the contextual menu, select *TortoiseSVN ->Apply patch...* At this point, you need to specify the previously downloaded `gdal1.4.4.patch` file and apply.

Alternatively, if you dont have TortoiseSVN, you may use the *patch* command distributed with MSYS[4]. In such a case, to apply the patch, enter in the GDAL main folder and run:

```
patch -p0 -f -i PATH_TO_DOWNLOADED_PATCH/gdal1.4.4.patch
```

(where `PATH_TO_DOWNLOADED_PATCH` represents the location where you have previously downloaded the patch)

Finally, you need to modify your `GDAL\NMAKE.opt` as explained in the following sections.

3.7.2.1 – Kakadu configuration option

Find the KAKADU Setting properties in `GDAL\NMAKE.opt` by looking for the following line:

```
# Uncommment if you have Kakadu 4.0 or newer
```

Supposing your KAKADU library has been placed in `C:\ExternalLibraries\kakadu`, edit the next line, like this: `KAKDIR = C:\ExternalLibraries\kakadu\v5_2_6-90032L`

Be sure the proper version subfolder is set. (In this case: v5_2_6-90032L).

To enable kakadu support we need to change only another property. Go ahead in the `NMAKE.OPT` and look for the following line:

[3]Alternatively, you could also download SmartSVN at: http://www.syntevo.com/smartsvn/download.html

[4]You can download MSYS from here: http://prdownloads.sf.net/mingw/MSYS-1.0.10.exe?download

After the download, run the MSYS-1.0.10 executable which will install MSYS on your hard disk, as an instance on `C:\MSYS\1.0`. Finally, edit the PATH environment variable (as explained in note 1) by adding the `bin` subfolder of your MSYS installation (as an instance, `C:\MSYS\1.0\bin`)

```
# Any extra libraries needed on this platform?
```

Then, edit the `ADD_LIBS` variable by adding the kakadu lib, like this:

```
ADD_LIBS = C:\ExternalLibraries\kakadu\lib\kdu_v52R.lib
```

3.7.2.2 – MrSID configuration option

Find the MrSID Setting properties in `GDAL\NMAKE.opt` by looking for the following line:

```
#Uncomment the following for MrSID support
```

Supposing your MrSID library has been placed in `C:\ExternalLibraries\MrSid`, edit the next lines, like this

```
MRSID_DIR = C:\ExternalLibraries\MrSid
MRSID_INCLUDE = -I$(MRSID_DIR)\include\base -I$(MRSID_DIR)\include\support \
      -I$(MRSID_DIR)\include\metadata \
      -I$(MRSID_DIR)\include\mrsid_readers \
      -I$(MRSID_DIR)\include\j2k_readers
MRSID_LIB = $(MRSID_DIR)\lib\Release_md\lti_dsdk_dll.lib advapi32.lib
user32.lib
```

Lastly, if you also need to enable JPEG2000 support by means of MrSID library, you need to add the following line:

```
MRSID_FLAGS = -DMRSID_J2K
```

3.7.2.3 – ECW configuration option

Find the ECW Setting properties in `GDAL\NMAKE.opt` by looking for the following line:

```
# Uncomment the following and update to enable ECW support.
```

Supposing your ECW library has been placed in `C:\ExternalLibraries\libecwj2-3.3`, edit the next 2 lines, like this:

```
ECWDIR  =  C:\ExternalLibraries\libecwj2-3.3
ECWLIB  =  $(ECWDIR)\lib\libecwj2.lib
```

3.7.2.4 – HDF4 configuration option

Find the HDF4 Setting properties in `GDAL\NMAKE.opt` by looking for the following line:

```
# Uncomment the following and update to enable NCSA HDF Release 4 support.
```

Supposing your HDF4 library has been placed in `C:\ExternalLibraries\HDF4`, edit the next 2 lines, like this:

```
HDF4_DIR = c:\ExternalLibraries\HDF4\release
HDF4_LIB = /LIBPATH:$(HDF4_DIR)\libpath
```

3.7.3 – GDAL Building

Now, you are ready to build GDAL. Open Visual Studio Command Prompt, and enter in your `GDAL` home folder. At this point, you are ready to start the build process by running the following command:

```
nmake /f makefile.vc
```

When the build is terminated, you need to generate JAVA bindings.

3.7.4 – Generating JAVA Bindings

3.7.4.1 - Variable settings

Be sure the `SWIG` variable in your `GDAL\NMAKE.opt` is properly set. Check this by finding the following lines:

```
# Set the location of your SWIG installation
!IFNDEF SWIG
SWIG = C:\ProgramFiles\swigwin-1.3.31\swig.exe
!ENDIF
```

Be sure `SWIG` variable refers to the proper `swig.exe` path.

Then, check your `GDAL\SWIG\JAVA\java.opt` is properly configured. (Basically, you need to check the `JAVA_HOME` and `ANT_HOME` variables are properly set)

3.7.4.2 - Running SWIG

Now, you are ready to generate java bindings. From the command line, enter in your `GDAL\SWIG` folder and run `nmake /f makefile.vc java`

This command will automatically generate wrappers and bindings.

3.7.5 – Final Settings

At this point, you should have:

- some external DLLs (for Kakadu, ECW, MrSID, HDF4)
- a GDAL DLL (`gdal14.dll`)
- 4 JNI DLL (`gdalconstjni.dll, gdaljni.dll, ogrjni.dll, osrjni.dll`)
- a jar file (`gdal.jar`)

You need to place all the DLLs in the folder where your application will look for libraries. Your `JAVA_HOME/BIN` folder could be a typical location where to place them.

Finally, be sure you properly set the `GDAL_DATA` environment variable. This needs to be set with your `GDAL\DATA` location in order to properly evaluate EPSG codes.

3.8 – ImageMagick & Jmagick

The Image I/O-Ext project provides an additional plugin to handle JPEG format, leveraging on Jmagick. In order to build and use this plugin, you need to download and build the ImageMagick library from this site:

ftp://ftp.imagemagick.org/pub/ImageMagick/windows/ImageMagick-windows.zip

Prior to build ImageMagick you need to configure it:

Run the Visual Studio IDE and from the "Open->Project" menu, select the configure workspace available at the `ImageMagick-6.X.X\VisualMagick\configure` folder and press Open. Choose "Build->Build Solution" to compile the configuration tool and, when finished, you will find a configure.exe tool on the `configure` folder. Now, you are ready to configure the build of your ImageMagick libraries:

Run `configure.exe` and the wizard will be opened. Press Next and click on the multi-threaded DLL. Now press, on Next twice and finally Finish. The configuration utility just created a workspace required to build ImageMagick from source. Open the `VisualDynamicMT.sln` Visual Studio Solution from the `ImageMagick-6.X.X\VisualMagick` folder. Change the solution properties (right click on the solution -> Properties). Select *"Configuration Properties"* and switch the *Configuration* value from "*Debug*" to "*Release*".

Finally, choose Build->Build Solution to compile and build the ImageMagick distribution. (Advanced Users may manually disable unrequired modules. Actually, the Image I/O-Ext plugin module leveraging on Jmagick, only provides support for JPEG files.

When finished, you need to place all the DLLs from the `ImageMagick-6.X.X\VisualMagick\bin` folder in the folder where your application will look for libraries. Your `JAVA_HOME/BIN` folder could be a typical location where to place them.

Future versions of this document will provide better instructions about how to customize the ImageMagick build.

3.9 – Image I/O-EXT Project

You need to download the **imageio-ext** project from Java.net SVN. To do this, you need to create an `imageio-ext` folder and use Tortoise SVN or another SVN client as explained in section 3.7.1.

The URL of repository for the *SVN Checkout* command is:

https://imageio-ext.dev.java.net/svn/imageio-ext/trunk

3.9.1 – DLL deployment instructions (Optional) - [deploylibs] property

In case you have skipped the manual building process introduced in the previous sections, you should leverage on the DLL available via the deploy module although this approach is not recommended (Testing some modules may fail since you need several licensed DLLs, which are not included in the deploy module).

To deploy available DLLs and additional required elements, you need to set the *deploylibs* properties as true with the `-Ddeploylibs` option when building the project (as explained in 3.9.3).

> **IMPORTANT NOTE:**
>
> you also need to setup a GDAL_DATA environment variable with the path of the location where you want the deploy module will store required files for EPSG codes parsing.

3.9.2 – JMagick libraries deployment instructions (Optional)- [jmagick] profile

In case you want to deploy and add Jmagick library support to the Image I/O-Ext project, specify the *jmagick* profile when building the project (as explained in 3.9.3). Multiple profiles may be specified; simply use the comma sign to specify more profiles (as an instance, `-Pbase,jmagick`)

3.9.3 – Image I/O-Ext Project building

Actually, depending on the set of available DLLs or the formats you need, it is possible to build the project in 3 configuration.

1. **base**: only the plugins which don't depend on external libraries are built
2. **full**: All the available plugins are built except kakadu. (that is: MrSID, ECW, HDF4)
3. **fullkakadu**: All the available plugins are built, kakadu included.

When executing tests, any configuration requires the proper set of DLLs. In case you select the **base** or the **full** configuration you could also leverage on the DLLs available via the deploy module, as explained in chapter 3.9.1 using `-Ddeploylibs`[5]

To build the Image I/O-Ext project, enter in `imageio-ext\trunk\` and, select the configuration you need, using the proper configProfile running the command:

`mvn install -PconfigProfile (`where configProfile is one of **base**, **full**, **fullkakadu**)

(Remind to add the `-Ddeploylibs=true` in case you need the DLLs from the deploy module. In case you also need the jmagick build, add the `jmagick` profile as explained in 3.9.2).

This command will build and test all required modules and plugins and store the produced JARS in the local maven repository.

In case you need to perform a fast build of the Image I/O-Ext project, without tests, just add the `-Dmaven.test.skip` option to the previous `mvn` command.

3.9.4 – Testing Image I/O-Ext modules with Maven.

In case you simply need testing some Image I/O-Ext modules, as an instance in order to check if everything is working fine, you can enter in the module you are interested in and run the maven test. As an instance:

```
C:\Projects\imageio-ext\trunk>cd plugin\gdalarcgrid

C:\Projects\imageio-ext\trunk\plugin\gdalarcgrid>mvn test
```

Lastly, if you want to perform interactive tests (which usually display data read on a windows), you should use the `interactive.tests` profile like this:

```
C:\Projects\imageio-ext\trunk\plugin\gdalarcgrid>mvn test
-Pinteractive.tests
```

Anyway, displaying the image is a non blocking/waiting operation so you will see the image just for a brief instant (when displayed, it will be automatically closed. Future versions may include a property to customize “waiting time” before close).

[5]Depending on the selected configuration, the proper set of DLL will be deployed by the deploy module. In case you select **fullkakadu**, no DLLs are available and you should have built them as explained in the previous chapters.

3.9.4.1 – Testing JPEG2000 Kakadu (GDAL) writer capabilities

The JPEG2000 Kakadu (GDAL based) plugin contains a wide suite for testing write operations with different write parameters leveraging on the available create options / Kakadu customizations. As default, the test performs only a simple write operation without testing any supported kakadu create option to reduce build time. In case you need to test all these operations simply use the `extensive.tests` profile like this:

```
C:\Projects\imageio-ext\trunk\plugin\gdalkakadujp2>mvn test
-Pextensive.tests
```

Moreover, as default, when the test is terminated, all the written files are automatically deleted. In case you would like to maintain the produced files, avoiding delete, you should add the profile `tests.holdwrittenfiles` to the previous one, using the following command:

```
C:\Projects\imageio-ext\trunk\plugin\gdalkakadujp2>mvn test
-Pextensive.tests,tests.holdwrittenfiles
```

4 – LINUX instructions

The following instructions have been tested on Linux Fedora Core 5 distribution.

Note: On some Linux distributions (as an instance, Ubuntu), you can install Java, Swig and Ant using the Package Manager.

4.1 – JAVA

First of all, you need your machine has JAVA 1.5.0_14 installed. (This version was available when writing this guide). You may download it from this site:
http://java.sun.com/javase/downloads/index_jdk5.jsp

Select the JDK 5.0 Update 14 and download the Linux Self-extracting file (A `jdk-1_5_0_14-linux-i586.bin` file). After you have downloaded it (as an instance on `/usr/local/java`), be sure that execute permissions are set on the downloaded file, by running this command:

```
chmod +x jdk-1_5_0_14-linux-i586.bin
```

Then, go in `/usr/local/java` and run: `./jdk-1_5_0_14-linux-i586.bin`

Usually, Fedora Core 5 distribution comes with a OLD java version (as an instance, 1.4.2). Now, you could add symbolic links on your `alternatives`.
Just run the following commands:

```
alternatives --install /usr/bin/java java /usr/.../jdk1.5.0_14/bin/java 2
```

next you can configure alternatives for java by using the following command:

```
alternatives --config java
```

```
alternatives --display java
```

Some Linux distributions come with Java 6 version. When building ImageIO-Ext with Java 6, an error related to `customstreams` module appears.

4.2 – ANT

Apache ANT is another needed tool. You can download it from: http://ant.apache.org/

When you downloaded it (as an instance, on `/usr/local/apache-ant-1.7.0`), you may create a symbolic link as follow: `ln -s /usr/local/apache-ant-1.7.0/bin/ant /usr/bin/ant`

4.3 – MAVEN

Maven is another important tool needed by the Image I/O-Ext project. You can download it from http://maven.apache.org/download.html.

When you downloaded it, extract the archive to the directory where you wish to install it, as an instance on `/usr/local/maven-2.0.7`

At this point, on your `/etc` folder, edit the "`profile`" file by adding the following lines just before the "`export`" section (`export PATH USER` ...):

```
PATH=/usr/local/maven-2.0.7/bin:$PATH
export JAVA_HOME=/usr/local/java/jdk1.5.0_14/
export JRE_HOME=/usr/local/java/jdk1.5.0_14/jre/
export MAVEN_HOME=/usr/local/maven-2.0.7/
export ANT_HOME=/usr/local/apache-ant-1.7.0/
export LD_LIBRARY_PATH=/usr/local/lib
```

4.4 – JAI

Go to https://jai.dev.java.net/binary-builds.html and select the daily builds link. Then, download the proper Linux version. After you downloaded it, extract the content of the lib folder on your `JAVA_HOME/lib` folder as well as on your `JAVA_HOME/JRE/lib`. (where `JAVA_HOME` defines your JDK, as an instance, `/usr/java/jdk1.5.0_14`).

4.5 – JAI-ImageIO Toolkit

Go to https://jai-imageio.dev.java.net/binary-builds.html and select the daily builds link. Then, download the proper Linux version. After you downloaded it, extract the content of the lib folder on your `JAVA_HOME/lib` folder and on your `JAVA_HOME/JRE/lib`. (where `JAVA_HOME` defines your JDK, as an instance, `/usr/java/jdk1.5.0_14`).

4.6 – SWIG

Be sure you have properly downloaded SWIG, the Simplified Wrapper and Interface Generator which allow to produce JAVA bindings for C/C++ code. You can obtain it by simply running:

```
yum update swig
```

4.6.1 – Manual SWIG installation

In case `yum` is not supported by your distribution, just download swig from: http://mesh.dl.sourceforge.net/sourceforge/swig/swig-1.3.32.tar.gz.

Unzip this somewhere on your hard disk and then run:

```
./configure
make
```

`sudo make install` (As you may notice, this command requires superuser privileges)

4.7 – GDAL

This is one of the most important sections. GDAL is the Geospatial Data Abstraction Library which provide data access to several data formats. Image I/O-Ext deeply leverages on this complex library which need to be properly configured.

4.7.1 – GDAL requirements

Depending on the format you wish to support, you need to properly download and setup several libraries prior to build GDAL. The following instructions describe how to achieve this for these formats:

- Kakadu (v. 5.2.6)
- MrSID (v 6.0.7)
- ECW (v 3.3)

4.7.1.1 – Kakadu

Supposing you have your own Kakadu licensed source code, go in the main kakadu folder (as an instance on `usr/local/kakadu`).

Enter in `coresys/make` and modify the `Makefile-Linux-x86-gcc` file as follow:

Enable the static build by setting `KDU_GLIBS = -static -static-libgcc`

Then run make: `make -f Makefile-Linux-x86-gcc`

This will generate libs in `kakadu/lib/Linux-x86-gcc`.

From the `kakadu` folder, run: `sudo cp lib/Linux-x86-gcc/* /usr/local/lib`

After this, enter in `apps/make` and modify the `Makefile-Linux-x86-gcc` file as follow:

Enable the static build by setting `KDU_GLIBS = -static -static-libgcc`

Set `LIB_SRC` as follow: `LIB_SRC=$(LIB_DIR)/libkdu.a`

Run make: `make -f Makefile-Linux-x86-gcc`

Then, run `sudo ldconfig`

The following additional step is not required by GDAL but it is needed by an Image I/O-Ext plugin which directly leverages on the Kakadu Library.

Enter in `managed/make` and modify the `Makefile-Linux-x86-gcc` file. You will notice the presence of a `INCLUDES += -I../all_includes` row. In top of this, add the following additional setting:

```
INCLUDES += -I$(JAVA_HOME)/include -I$(JAVA_HOME)/include/linux
```

Run make: `make -f Makefile-Linux-x86-gcc`

Then, from the `kakadu` folder, run: `sudo cp /lib/Linux-x86-gcc/libkdu_jni.so /usr/local/lib`

Finally, run `sudo ldconfig`

4.7.1.2 – MrSID

As a first requirement, you need the LizardTech Decoding Software Development Kit (DSDK). You can download it free of charge from this site: http://developer.lizardtech.com (You need to be registered in order to download it). After logged in, select *"Download" -> "Software*

Development Kits" -> "Download SDK's". Select the proper version of SDK to be downloaded (select the **GeoExpress SDK for Linux (x86) - gcc 3.4**)

4.7.1.3 – ECW

Download the Image Compression SDK (source code) from ER Mapper site at this address:

http://www.ermapper.com (You need to be registered in order to download it).

From the main site, select the menu *"products & downloads" -> "Image Compression SDK"* from the *Desktop* section. Download the *Image Compression SDK Source Code 3.3* file and extract this somewhere on your hard disk, as an instance on `/home/youruser/libs/libecwj2`

From the command line, just enter this folder and simply run:

```
./configure
```

```
make
```

`sudo make install` (As you may notice, this command requires superuser privileges)

The last command will copy all the libs in the `usr/local/lib` folder. By this way, when configuring GDAL (as explained in the next chapter) it will automatically setup the build to support the ECW format.

4.7.2 – GDAL Configuration

Firstly, you need to download GDAL 1.4.4 from OSGeo SVN. Enter the folder where you want to download GDAL and run:

```
svn co http://svn.osgeo.org/gdal/tags/1.4.4/gdal gdal1.4.4
```

Then, you need to apply the patch available at this location:

https://imageio-ext.dev.java.net/svn/imageio-ext/trunk/patches/gdal1.4.4.patch

This patch contains several changes for:

- Kakadu support: More Kakadu create options and error management supported. Makefile modified
- Java bindings: improved data access (read/write Dataset at once instead of read/write RasterBands at once; BandMap, PixelSpace, LineSpace and BandSpace parameters now are allowed), added setMetadataItem capabilities

To apply the patch, enter in the GDAL main folder and run:

```
patch -p0 -f -i PATH_TO_DOWNLOADED_PATCH/gdal1.4.4.patch
```

(where `PATH_TO_DOWNLOADED_PATCH` represents the location where you downloaded the patch, as an instance, `/home/youruser/Desktop/`)

Be sure you properly set the `GDAL_DATA` environment variable. This need to be set with your `GDAL/DATA` location in order to properly evaluate EPSG codes. As an instance, supposing you installed GDAL on `/home/youruser/gdal1.4.4`, you can use the following command:

```
export GDAL_DATA=/home/youruser/gdal1.4.4/data/
```

Next step is configuring GDAL by means of the `./configure` command. Such a command allows to specify several options to enable formats, change build properties, customize libraries and much more. Depending on the required formats you wish to enable on GDAL, you need to add some options to this command as explained in the following sections.

4.7.2.1 – Kakadu configuration option

Add `--with-kakadu=KAKADU_FOLDER` option to the `./configure` command, where `KAKADU_FOLDER` represents the path where your Kakadu library is located.

4.7.2.2 – MrSID configuration option

Add `--with-mrsid=MRSID_FOLDER` option to the `./configure` command, where `MRSID_FOLDER` represents the path where you previously downloaded GeoDSDK.

Note: During the future build process (4.7.3) a similar error could occur:

```
/......./include/base/lti_sceneBuffer.h:356:
error: extra qualification 'LizardTech::LTISceneBuffer::' on member
```

You need to fix the issue in the header `MRSID_FOLDER/include/base/lti_sceneBuffer.h` by simply removing the extra qualification from the `inWindow` declaration. Line 356 should look like this: `bool inWindow(lt_uint32 x, lt_uint32 y) const;` Then repeat build process as suggested in 4.7.3.

4.7.3 – GDAL Building

Finally, you are ready to build GDAL. Supposing you properly configured it as explained in section 4.7.2, run the following commands:

```
make clean
```

```
make
```

`sudo make install` (As you may notice, this command requires superuser privileges)

When the build is terminated, run `sudo ldconfig`.

Next step is generating JAVA bindings.

4.7.4 – Generating JAVA Bindings

SWIG will generate java bindings for you.

As a first step, check your `GDAL/SWIG/JAVA/java.opt` is properly configured.

Basically, you need to check the `JAVA_HOME, JAVA_INCLUDE` and `ANT_HOME` variables are properly set. Be sure the following line exists: `JAVA_INCLUDE=-I$(JAVA_HOME)/include -I$(JAVA_HOME)/include/linux`)

A second step is required in order to customize the compiler options (This is needed to change default optimizations settings, which may cause JVM crashes). Just redefine the `CXX_OPTFLAGS` and `C_OPTFLAGS` in the `GDALmake.opt` file on your GDAL main folder. You need to set these 2

flags with the `-O1` value (Note that the minus sign ("-") is followed by the "O" letter instead of the "zero" digit). Search the following line (`#Flags to build optimized relese version`) in `GDALmake.opt` and change the flags like this:

```
#Flags to build optimized relese version
CXX_OPTFLAGS =   -O1
C_OPTFLAGS =     -O1
```

Then, enter in your main `GDAL` folder and run:

```
cd swig
cd java
make clean
make generate
make build
```

This command will automatically generate wrappers and bindings. Then, copy the generated libs in `/usr/local/lib` using the command:

`sudo cp *.so /usr/local/lib` (As you may notice, this command requires superuser privileges)

Finally, run `sudo ldconfig`.

4.8 – Image I/O-EXT Project

You need to download the **imageio-ext** project from Java.net SVN. Enter the folder where you want to download Image I/O-EXT and run:

```
svn co https://imageio-ext.dev.java.net/svn/imageio-ext/trunk imageio-ext --username MYUSERNAME
```

(Note: MYUSERNAME need to be replaced with your own username)

4.8.1 – Libraries deployment instructions (Optional) - [deploylibs] property

In case you have skipped the GDAL manual building process introduced in the previous sections, you should leverage on the libraries contained in the deploy module although this approach is not recommended.

In that case, you need to deploy available libraries before building the Image I/O-Ext project. You also need to setup a GDAL_DATA environment variable with the path of the location where you want the deploy module will store required files for EPSG codes parsing, as explained in 4.7.2.

Moreover, depending on the set of required libraries, you need to specify a configuration profile from the following set:

1. **base**: in case you need the libraries built to support only the plugins which don't depend on external libraries
2. **full**: in case you need the libraries built to support MrSID and ECW plugins (HDF4 and Kakadu are not available as deployed libraries).

When you have made your choices, you need to enter in `imageio-ext\trunk\` and run the following command:

`sudo mvn generate-resources -PconfigProfile -Ddeploylibs` (where configProfile is one of **base** or **full**). Note that you need superuser privileges since this command will deploy all the required libraries (*.so) on the `JRE/lib/i386 folder`

4.8.2 – Image I/O-Ext Project building

Actually, it is possible to build the project in 3 configuration.

1. **base**: only the plugins which don't depend on external libraries are built
2. **full**: All the available plugins are built except kakadu
3. **fullkakadu**: All the available plugins are built, kakadu included.

When executing tests, any configuration requires the proper set of libraries. In case you select the **base** or the **full** configuration you could also leverage on the libraries available via the deploy module, as explained in chapter 4.8.1.

To build the Image I/O-Ext project, enter in `imageio-ext\trunk\` and, select the configuration you need, using the proper configProfile running the command: `mvn install -PconfigProfile` (where configProfile is one of **base**, **full** and **fullkakadu**)

This command will build and test all required modules and plugins and store the produced JARS in the local maven repository. (Note that actually HDF4 plugin is not tested on Linux).

In case you need to do a fast build of the Image I/O-Ext project, without tests, just add the `-Dmaven.test.skip` option to the previous `mvn` command.

NOTE: Maven should automatically-recognize the running operative system in order to use the proper deploy module. In case some errors occur due to an unrecognized OS name (which should be “Linux”) you can explicitly specify it with the additional `linux` profile. As an instance, if you are building the full configuration, you can use the `-Pfull,linux` profiles.

4.8.3 – Testing Image I/O-Ext modules with Maven.

In case you simply need testing some Image I/O-Ext modules, as an instance in order to check if everything is working fine, you can enter in the module you are interested in and run the maven test. As an instance:

```
imageio-ext/trunk/plugin>cd plugin/gdalarcgrid

imageio-ext/trunk/plugin/gdalarcgrid>mvn test
```

Lastly, if you want to perform interactive tests (which usually display data read on a windows), you should use the `interactive.tests` profile like this:

```
imageio-ext/trunk/plugin/gdalarcgrid>mvn test -Pinteractive.tests
```

Anyway, displaying the image is a non blocking/waiting operation so you will see the image just for a brief instant (when displayed, it will be automatically closed. Future versions may include a property to customize “waiting time” before close).

4.8.3.1 – Testing JPEG2000 Kakadu (GDAL) writer capabilities

The JPEG2000 Kakadu (GDAL based) plugin contains a wide suite for testing write operations with different write parameters leveraging on the available create options / Kakadu customizations. As default, the test performs only a simple write operation without testing any supported kakadu create option to reduce build time. In case you need to test all these operations simply use the `extensive.tests` profile like this:

```
imageio-ext/trunk/plugin/gdalkakadujp2>mvn test -Pextensive.tests
```

Moreover, as default, when the test is terminated, all the written files are automatically deleted. In case you would like to maintain the produced files, avoiding delete, you should add the profile `tests.holdwrittenfiles` to the previous one, using the following command:

```
imageio-ext/trunk/plugin/gdalkakadujp2>mvn test
-Pextensive.tests,tests.holdwrittenfiles
```