

Writing ImageJ Plugins—A Tutorial

Werner Bailer*
ijtutorial@fh-hagenberg.at[†]

Upper Austria University of Applied Sciences
Dept. of Media Technology and Design
Hagenberg, Austria[‡]

Version 1.6
Based on ImageJ Release 1.28
June 26, 2003

Contents

1	Getting Started	3
1.1	About ImageJ	3
1.2	About this Tutorial	3
1.3	Setting up your Environment	4
1.4	Updating ImageJ	5
2	ImageJ Class Structure	6
3	The Plugin Concept of ImageJ	8
3.1	Types of Plugins	8
3.2	Interfaces	8
3.3	Plugins Folder—Installing Plugins	9
3.4	A Sample Plugin (Example)	10
3.5	Recording Plugins	11
3.6	Compiling and Running Plugins	11
3.7	Integrating Plugins into the ImageJ GUI	13
4	Image Representation in ImageJ	13
4.1	Types of Images	13
4.2	Images	13
4.3	Processors	15
4.4	Accessing Pixel Values	15
4.5	Regions of Interest	17

*The author is now with the *Institute of Information Systems and Information Management at Joanneum Research GmbH*, in Graz, Austria.

[†]This document can be downloaded from www.fh-hagenberg.at/mtd/depot/imaging/imagej/

[‡]Fachhochschule Hagenberg, Medientechnik und -design

4.6	Creating New Images	18
4.7	Displaying Images	19
4.8	ColorInverter PlugIn (Example)	20
4.9	Stacks	22
4.10	StackAverage PlugIn (Example)	23
4.11	Additional Reference	25
5	ImageJ's Utility Methods and Image Conversion	30
5.1	(Error) Messages	30
5.2	ImageJ Window, Status Bar and Progress Bar	30
5.3	User input	32
5.4	Calling Menu Commands	32
5.5	Calling Other PlugIns	32
5.6	MessageTest PlugIn (Example)	32
5.7	More Utilities	33
5.8	Image Type Conversion	34
6	Windows	35
6.1	PlugInFrame	36
6.2	GenericDialog	36
6.3	FrameDemo PlugIn (Example)	38
6.4	ImageWindow	39
6.5	ImageCanvas	40
6.6	Subclasses of ImageWindow	40
6.7	Event Handling (Example)	41
7	Advanced Topics	43
7.1	Data Import/Export	43
7.2	Using the ImageJ Library outside ImageJ	44
8	Troubleshooting	47
8.1	ImageJ runs out of memory.	47
8.2	A plugin is not displayed in ImageJ's plugins menu.	48
8.3	When you call the "Plugins/Compile and Run ..." menu	48
8.4	ImageJ throws an exception in a web application (Unix/Linux)	48
9	Frequently Asked Questions	48
9.1	How to change the URL for the sample images	48
9.2	How to include user plugins when running ImageJ as applet?	49
10	Further Resources	49
10.1	API Documentation, Source Code	49
10.2	Plugins Page	49
10.3	ImageJ Mailing List	49
10.4	Java Resources	49

1 Getting Started

1.1 About ImageJ¹

ImageJ is a public domain Java image processing program inspired by NIH Image for the Macintosh. It runs, either as an online applet or as a downloadable application, on any computer with a Java 1.1 or later virtual machine.

It can display, edit, analyze, process, save and print 8-bit, 16-bit and 32-bit images. It can read many image formats including TIFF, GIF, JPEG, BMP, DICOM, FITS and “raw”. It supports “stacks”, a series of images that share a single window. It is multithreaded, so time-consuming operations such as image file reading can be performed in parallel with other operations.

It can calculate area and pixel value statistics of user-defined selections. It can measure distances and angles. It can create density histograms and line profile plots. It supports standard image processing functions such as contrast manipulation, sharpening, smoothing, edge detection and median filtering.

It does geometric transformations such as scaling, rotation and flips. Images can be zoomed up to 32:1 and down to 1:32. All analysis and processing functions are available at any magnification factor. The program supports any number of windows (images) simultaneously, limited only by available memory.

Spatial calibration is available to provide real world dimensional measurements in units such as millimeters. Density or gray scale calibration is also available.

ImageJ was designed with an open architecture that provides extensibility via Java plugins. Custom acquisition, analysis and processing plugins can be developed using ImageJ’s built in editor and Java compiler. User-written plugins make it possible to solve almost any image processing or analysis problem.

ImageJ is being developed on Mac OS X using its built in editor and Java compiler, plus the BBEdit editor and the Ant build tool. The source code is freely available. The author, Wayne Rasband (wayne@codon.nih.gov), is at the Research Services Branch, National Institute of Mental Health, Bethesda, Maryland, USA.

1.2 About this Tutorial

This tutorial is an introduction to writing plugins for ImageJ. It explains the concept of plugins in ImageJ, starting with the sample plugins that are part of the ImageJ distribution, and covers those parts of the ImageJ API, which are essential for writing plugins. A reference of the most important classes, methods and constants is provided and some advanced topics are discussed.

A basic knowledge of the Java programming language is required. (Resources for Java beginners can be found in section 10.4). You should also try to get familiar with the use of ImageJ before you start writing plugins. The ImageJ documentation can be found at <http://rsb.info.nih.gov/ij/docs/index.html>, including two handouts by Larry Reinking.

Plugins which just consist of a sequence of ImageJ commands can be recorded using the plugin recorder which generates Java code (cf. Section 3.5). Another alternative is the JavaScript editor by Kas Thomas (<http://rsb.info.nih.gov/ij/plugins/javascript.html>), which allows controlling ImageJ using JavaScript.

¹Description taken from <http://rsb.info.nih.gov/ij/docs/intro.html>

For the development of plugins you need ImageJ and a Java compiler. You can write your plugins using ImageJ's built-in editor or any other text editor or Java IDE (Integrated Development Environment). You can compile them using a Java compiler of your choice or inside ImageJ.

1.3 Setting up your Environment

For running ImageJ you need a the ImageJ class and configuration files, a Java Runtime Environment (JRE) and for compiling your own plugins a Java compiler with the required libraries, as for example included in the Java 2 SDK Standard Edition (J2SE) from Sun Microsystems. Depending on the ImageJ distribution you are using, some or all of this may be included.

1.3.1 Installing ImageJ

The latest distribution of ImageJ can be downloaded from <http://rsb.info.nih.gov/ij/download.html>

In the following the installation of ImageJ will be described briefly for different operating systems. More detailed and up-to-date installation instructions can be found at <http://rsb.info.nih.gov/ij/docs/install>

If you already have a JRE (and a Java compiler) installed on your computer and you are familiar with Java, you just need to download the ImageJ class and configuration files which are available as a ZIP archive. To run ImageJ, add `ij.jar` to your classpath and execute class `ij.ImageJ`. This works for all operating systems for which there is no specific distribution but for which a Java runtime environment is available.

Windows

The Windows version is available as a self-extracting ZIP archive which includes a Java Runtime Environment (JRE) and a Java compiler. To install it, run the self-extracting file you have downloaded. In the destination directory you will find a shortcut called "ImageJ". Double-click it to run ImageJ.

Note: The "ImageJ" shortcut assumes that the ImageJ folder is `C:\ImageJ`. If this is not the case, right-click on the shortcut and edit the "Target:" and "Start in" fields of the "Shortcut" properties to reflect the new location. For example, if the ImageJ folder is located in the `C:\Program Files` folder, change these fields to

```
"C:\Program Files\ImageJ\jre\bin\jre.exe" -mx80m cp ij.jar ij.ImageJ  
and  
C:\Program Files\ImageJ
```

This shortcut opens ImageJ without a console window in the background. If you want to use `System.out` and `System.err` to write debug information to the console, start ImageJ using the shortcut "ImageJ (old shortcut)".

Mac OS

To run ImageJ on Mac OS you need the Mac OS Runtime for Java (MRJ). It can be downloaded from <http://www.apple.com/java>. Installation instructions can be found on the download page.

Version 1.6

The ImageJ distribution is a self-extracting archive (If it does not expand automatically after downloading, use e.g. StuffIt Expander). Double-click the “ImageJ” icon in the newly created folder to run it.

Mac OS X

Download the Mac OS X `tar.gz` file and double-click it to expand. Double-click the “ImageJ” icon to run ImageJ. Alternatively, you can also run ImageJ from the command line using

```
java -jar -mx128m ImageJ.app/Contents/Resources/Java/ij.jar
```

Running ImageJ from the command line has the advantage, that you can use `System.out` and `System.err` to write debug information to the console.

Linux x86

Download the Linux x86 `tar.gz` file, which contains Sun's Java Runtime Environment 1.3.1, and extract it using e.g.

```
tar xvzf ij123-x86.tar.gz
```

and execute the `run` script in the ImageJ directory.

1.3.2 Installing the Java Compiler

Installing a Java compiler is only necessary if it is not included in the ImageJ distribution or provided by the operating system. In any case, also if you are using an operating system which is not mentioned here but for which a Java compiler is available, you can use any Java compiler of your choice to compile your plugins (e.g. J2SE SDK from Sun Microsystems, which you can download from <http://www.javasoft.com>).

Details on compiling plugins can be found in Section 3.6.

Windows

The ImageJ distribution for Windows includes a Java compiler which allows you to compile plugins from inside ImageJ.

Mac OS

In addition to the MRJ you need the MRJ SDK. It can be downloaded from <http://developer.apple.com/java>. Run the installer you have downloaded. After the installation it is possible to compile plugins inside ImageJ.

Mac OS X

A Java compiler is included in Mac OS X Java, so you can compile plugins from inside ImageJ.

Linux

The ImageJ distribution for Linux includes a Java compiler which allows you to compile plugins from inside ImageJ.

1.4 Updating ImageJ

You can update ImageJ by replacing the ImageJ JAR file (`ij.jar`). The latest version is available at <http://rsb.info.nih.gov/ij/upgrade/index.html>. Just replace your existing `ij.jar` file with the one you downloaded. The `ij.jar` file can be found directly in your ImageJ folder. On Mac OS X, it can be found in `ImageJ.app/Contents/Resources/Java` where `ImageJ.app` is the directory that Finder displays as the ImageJ application.

Note: The ImageJ JAR file also contains the configuration file `IJProps.txt`. If you want to save your settings, extract the file from your old `ij.jar` and replace it in the new one. You can edit JAR files with most ZIP utilities (e. g. WinZip).

2 ImageJ Class Structure

This is an overview of the class structure of ImageJ. It is by far not complete, just the most important classes for plugin programming are listed and briefly described. More detailed descriptions of classes and methods can be found in Sections 3 through 6. A UML class diagram is available at <http://rsb.info.nih.gov/ij/developer/diagram.html>.

`ij`

`ImageJApplet`

ImageJ can be run as applet or as application. This is the applet class of ImageJ. The advantage of running ImageJ as applet is that it can be run (remotely) inside a browser, the biggest disadvantage is the limited access to files on disk because of the Java applet security concept, if the applet is not signed.

`ImageJ`

The main class of the ImageJ application. This class contains the `run` method, which is the program's main entry point, and the ImageJ main window.

`Executer`

A class for executing menu commands in separate threads (without blocking the rest of the program).

`IJ`

A class containing many utility methods (discussed in Section 5).

`ImagePlus`

The representation of an image in ImageJ, which is based on an `ImageProcessor` (see Section 4).

`ImageStack`

An `ImageStack` is an expandable array of images (see Section 4).

`WindowManager`

This class manages the list of open windows.

`ij.gui`

`ProgressBar`

A bar in the ImageJ main window that informs graphically about the progress of a running operation.

`GenericDialog`

A modal dialog that can be customized and called on the fly, e.g. for getting user input before running a plugin (see Section 6).

`NewImage`

A class for creating a new image of a certain type from scratch.

`Roi`

A class representing a region of interest of an image. If supported by a plugin, it can process just the ROI and not the whole image.

ImageCanvas

A canvas derived from `java.awt.Canvas` on which an image is painted (see Section 6).

ImageWindow

A frame derived from `java.awt.Frame` that displays an image (see Section 6).

StackWindow

An `ImageWindow` designed for displaying stacks (see Section 6).

HistogramWindow

An `ImageWindow` designed for displaying histograms (see Section 6).

PlotWindow

An `ImageWindow` designed for displaying plots (see Section 6).

ij.io

This package contains classes for reading/decoding and writing/encoding image files.

ij.measure

Contains classes for measurements.

ij.plugin

Most ImageJ menu commands are implemented as plugins and can therefore be found in the classes of `ij.plugin` and its subpackages.

PlugIn

This interface has to be implemented by plugins, that do not require an image as input (see Section 3).

Converter

Implements a method for conveniently converting an `ImagePlus` from one type to another (see Section 5.8)

ij.plugin.filter

PlugInFilter

This interface has to be implemented by plugins, that require an image as input (see Section 3).

ij.plugin.frame

PlugInFrame

A window class that can be subclassed by a plugin (see Section 3).

ij.process

ImageConverter

A class that contains methods for converting images from one image type to another.

ImageProcessor

An abstract superclass of image processors for certain image types. An image processor provides methods for actually working on the image data (see Section 4).

StackConverter

A class for converting stacks from one image type to another.

StackProcessor

A class for processing image stacks.

ij.text

This package contains classes for displaying and editing text.

3 The Plugin Concept of ImageJ

The functions provided by ImageJ's menu commands (most of them are in fact plugins themselves) can be extended by user plugins. These plugins are Java classes implementing the necessary interfaces that are placed in a certain folder. Plugins can be written with ImageJ's built-in plugin editor (accessible via the menus "Plugins/New..." and "Plugins/Edit..."), with a text editor of your choice or they can be generated using ImageJ's plugin recorder. In any case plugins can be compiled and run inside ImageJ. Plugins found by ImageJ are placed in the Plugins menu or (since version 1.20) in submenus of it.

3.1 Types of Plugins

There are basically two types of plugins: those that do not require an image as input (implementing the interface `PlugIn`) and plugin filters, that require an image as input (implementing the interface `PlugInFilter`).

3.2 Interfaces

3.2.1 PlugIn

This interface has just one method:

```
void run(java.lang.String arg)
```

This method runs the plugin, what you implement here is what the plugin actually does. `arg` is a string passed as an argument to the plugin, and it can also be an empty string. You can install plugins more than once, so that each of the commands can call the same plugin class with a different argument.

3.2.2 PlugInFilter

This interface also has a method

```
void run(ImageProcessor ip)
```

This method runs the plugin, what you implement here is what the plugin actually does. It takes the image processor it works on as an argument. The processor can be modified directly or a new processor and a new image can be based on its data, so that the original image is left unchanged. The original image is locked while the plugin is running. In contrast to the `PlugIn` interface the `run` method does not take a string argument. The argument can be passed using

```
int setup(java.lang.String arg, ImagePlus imp)
```

This method sets up the plugin filter for use. The `arg` string has the same function as in the `run` method of the `PlugIn` interface. You do not have to care for the argument `imp`—this is handled by ImageJ and the currently active image is passed. The `setup` method returns a flag word that represents the filters capabilities (i.e. which types of images it can handle).

The following capability flags are defined in `PlugInFilter`:

```
static int DOES_16 The plugin filter handles 16 bit grayscale images.
```

```
static int DOES_32 The plugin filter handles 32 bit floating point grayscale images.
```

```
static int DOES_8C The plugin filter handles 8 bit color images.
```

```
static int DOES_8G The plugin filter handles 8 bit grayscale images.
```

```
static int DOES_ALL The plugin filter handles all types of images.
```

```
static int DOES_RGB The plugin filter handles RGB images.
```

```
static int DOES_STACKS The plugin filter supports stacks, ImageJ will call it for each slice in a stack.
```

```
static int DONE If the setup method returns DONE the run method will not be called.
```

```
static int NO_CHANGES The plugin filter does not change the pixel data.
```

```
static int NO_IMAGE_REQUIRED The plugin filter does not require an image to be open.
```

```
static int NO_UNDO The plugin filter does not require undo.
```

```
static int ROI_REQUIRED The plugin filter requires a region of interest (ROI).
```

```
Static int STACK_REQUIRED The plugin filter requires a stack.
```

```
static int SUPPORTS_MASKING Plugin filters always work on the bounding rectangle of the ROI. If this flag is set and there is a non-rectangular ROI, ImageJ will restore the pixels that are inside the bounding rectangle but outside the ROI.
```

3.3 Plugins Folder—Installing Plugins

ImageJ user plugins have to be located in a folder called `plugins`, which is a subfolder of the ImageJ folder. But only class files in the `plugins` folder with at least one underscore in their name appear automatically in the “Plugins” menu. Since version 1.20 it is also possible to create subfolders of the `plugins` folder and place plugin files there. The subfolders are displayed as submenus of ImageJ’s “Plugins” menu.

To install a plugin (e.g. one you have downloaded from the ImageJ plugins page) copy the `.class` file into the `plugins` folder or one of its subfolders. The plugin will appear in the plugin menu (or one of its submenus) the next time you start ImageJ. You can add it to a menu and assign a shortcut to it using the “Plugins/ Shortcut/ Install plugin...” menu. In this case, the plugin will appear in the menu without restarting ImageJ.

Alternatively, if you have the source code of a plugin, you can compile and run it from within ImageJ. More about compiling and running plugins can be found in Section 3.6.

Since ImageJ 1.24 you can specify the plugins directory using the `plugins.dir` property. On Windows, change the “Target” field of the shortcut to

```
"C:\Program Files\ImageJ\jre\bin\jre.exe" -mx80m cp ij.jar
-Dplugins.dir=c:\data ij.ImageJ
```

Under Linux, start ImageJ with

```
./jre/bin/jre -mx80m -cp ij.jar:tools.jar -Dplugins.dir=/home/wayne
ij.ImageJ
```

3.4 A Sample Plugin (Example)

If you look into the plugins folder right after installing ImageJ you will find the sample plugins that come with ImageJ. In this section we will take a closer look at one of them. `Inverter_` is a plugin that inverts 8 bit grayscale images.

Here we import the necessary packages, `ij` for the basic ImageJ classes, `ij.process` for image processors and the interface `ij.plugin.filter.PlugInFilter` is the interface we have to implement for a plugin filter.

```
import ij.*;
import ij.plugin.filter.PlugInFilter;
import ij.process.*;
import java.awt.*;
```

Note: Do not use a `package` statement inside plugin classes—they have to be in the default package!

Our plugin has the necessary underscore appended. It needs an image as input, so it has to implement `PlugInFilter`:

```
public class Inverter_ implements PlugInFilter {
```

What comes next is the method for setting up the plugin. For the case that we get “about” as argument, we call the method `showAbout` that displays an about dialog. In that case we return `DONE` because we do not want the `run` method to be called. In any other case we return the capability flags for this plugin: It works on 8 bit grayscale images, also on stacks and in the case that there is a ROI (region of interest) defined the plugin will just work on the masked region (ROI).

```
    public int setup(String arg, ImagePlus imp) {
        if (arg.equals("about")) {
            showAbout();
            return DONE;
        }
        return DOES_8G+DOES_STACKS+SUPPORTS_MASKING;
    }
}
```

The `run` method implements the actual function of the plugin. We get the processor of the original image. Then we get the image as an array of pixels from the processor—as it is a 8 bit grayscale image (= 256 possible values) we can use a `byte` array. Note that the pixel array is one-dimensional, containing one scan line after the other. Then we read the width of the image (because we need to know the length of a scan line) and the bounding rectangle of the ROI.

```

public void run(ImageProcessor ip) {
    byte[] pixels = (byte[])ip.getPixels();
    int width = ip.getWidth();
    Rectangle r = ip.getRoi();

```

We now declare two variables to avoid calculating the position in the one dimensional image array every time. In the outer loop we go from the first line of the ROI to its last line. We calculate the offset (= position of the first pixel of the current scan line) and go in the inner loop from the left most pixel of the ROI to its right most pixel. We assign the current position to `i` and invert the pixel value by subtracting its value from 255.

```

    int offset, i;
    for (int y=r.y; y<(r.y+r.height); y++) {
        offset = y*width;
        for (int x=r.x; x<(r.x+r.width); x++) {
            i = offset + x;
            pixels[i] = (byte)(255-pixels[i]);
        }
    }
}

```

`showAbout` uses the static method `showMessage` from class `IJ` to display a text in a message box. The first parameter specifies its title, the second the message text.

```

void showAbout() {
    IJ.showMessage("About Inverter_...",
        "This sample plugin filter inverts 8-bit images. Look\n" +
        "at the 'Inverter_.java' source file to see how easy it is\n" +
        "in ImageJ to process non-rectangular ROIs, to process\n" +
        "all the slices in a stack, and to display an About box."
    );
}
}

```

3.5 Recording Plugins

If your plugin shall just execute a sequence of ImageJ menu commands, you do not have to write your plugin, you can simply record it. “Plugins/Record...” opens a window and records your actions (menu commands, regions of interest, etc.) as long as the “Record” checkbox is checked. The pseudocode of this operations will be displayed in the window. The “Create Plugin” button generates a Java class from this pseudocode using the utility methods from class `IJ` (discussed in section 5). The plugin class will be opened in the built in editor from where you can also compile and run it. You may want to take such a generated plugin code as a basis for writing more complex plugins.

3.6 Compiling and Running Plugins

Now that we have looked at one of the sample plugins we want to compile and run it.

If the Java runtime environment you are using includes a Java compiler (e. g. as in the ImageJ distribution for Windows) or your operating system provides the Java compiler you can compile and run plugins inside ImageJ. There are basically two ways:

- Using the menu “Plugins/Compile and run...”, which opens a file dialog which lets you select a `.java` file which will be compiled into a class file and executed as plugin.
- Using “File/Compile and run...” in the built-in plugin editor which will compile and run the code in the editor window.

If your plugin requires other libraries than ImageJ and the standard Java libraries, you have to modify the classpath of your Java environment in order to have these libraries available when compiling and running plugins. Here is how to modify the classpath:

Windows

Edit the classpath option in the “Target:” field of the of the “Shortcut” properties of the ImageJ shortcut. The entries are separated by semicolons. For example, to use `mylib.jar`, change the “Target” line to

```
C:\ImageJ\jre\bin\jre.exe mx80m -cp ij.jar;tools11.jar;mylib.jar ij.ImageJ
```

Mac OS

Make sure that all libraries you want to use are located in

`...:System Folder:Extensions:MRJLibraries:MRJClasses` or create an alias for the library there.

Alternatively, the classpath for the application can be set in its ‘STR ’ (0) resource.

Mac OS X

To change the classpath used by ImageJ, edit the `com.apple.mrj.application.classpath` property in the `ImageJ.app/Contents/Resources/MRJApp.properties` text file, where `ImageJ.app` is the directory (bundle) that the Finder displays as the ImageJ application. You can view the files and folders in the ImageJ bundle by control-clicking on the ImageJ application and selecting “Show Package Contents”.

Linux, Unix, other OS

You can modify the classpath by using the `cp` switch followed by the libraries to be used. For example, to use `mylib.jar`, call the JVM using:

```
./jre/bin/jre mx80m -cp ij.jar:tools.jar:mylib.jar ij.ImageJ
```

If you want to use this setting all the time, make the modification in the run script.

If your Java environment does not contain a Java compiler you can compile plugins on the command line (or using a batch file or shell script respectively) under Windows, Linux and Mac OS X. The syntax for calling the Java compiler is

```
javac cp ij.jar;tools.jar .\plugins\MyPlugin_.java under Windows and  
javac cp ij.jar:tools.jar ./plugins/MyPlugin_.java under Linux.
```

3.7 Integrating Plugins into the ImageJ GUI

Like commands, plugins can be accessed via hot-keys. You can create a new hot-key by selecting “Create Shortcut” from the menu “Plugins / Shortcuts”.

When we discussed the plugin interfaces we talked about arguments that can be passed to plugins. Installing a plugin using the menu command “Plugins / Shortcuts / Install Plugin ...” places the plugin into a selected menu, assigns a hot-key and passes an argument. “Plugins / Shortcuts / Remove ...” removes a plugin from the menu.

4 Image Representation in ImageJ

When we looked at the sample plugin in the previous section we saw that images are represented by `ImagePlus` and `ImageProcessor` objects in ImageJ. In this section we take a closer look at the way images are handled by ImageJ. Methods that are not discussed in the text but are of some importance for writing plugins can be found in the reference in Section 4.11.

4.1 Types of Images

Images are large arrays of pixel values. But it is important to know how these pixel values should be interpreted. This is specified by the type of the image. ImageJ knows five image types:

8 bit grayscale image can display 256 grayscales, a pixel is represented by a `byte` variable.

8 bit color image can display 256 colors that are specified in a lookup table (LUT), a pixel is represented by a `byte` variable.

16 bit grayscale image can display 65,536 grayscales, a pixel is represented by a `short` variable.

RGB color image can display 256 values per channel, a pixel is represented by an `int` variable.

32 bit image floating point grayscale image, a pixel is represented by a `float` variable.

For information about conversion between different image types, see Section 5.8. D. Sage and M. Unser (Biomedical Imaging Group, Swiss Federal Institute of Technology Lausanne) contributed a package called `ImageAccess`, which unifies access to images regardless of data types.²

4.2 Images

An `ImagePlus` is an object that represents an image. It is based on an `ImageProcessor`, a class that holds the pixel array and does the actual work on the image. The type of the `ImageProcessor` used depends on the type of the image. The image types are represented by constants declared in `ImagePlus`:

`COLOR_256` A 8 bit color image with a look-up table.

²More information can be found at <http://bigwww.epfl.ch/teaching/iplabsite/>.

`COLOR_RGB` A RGB color image.

`GRAY16` A 16 bit grayscale image.

`GRAY32` A 32 bit floating point grayscale image.

`GRAY8` A 8 bit grayscale image.

ImageJ displays images using a class called `ImageWindow`. It handles repainting, zooming, changing masks etc.

To construct an `ImagePlus` use one of the following constructors:

`ImagePlus()`

Default constructor, creates a new empty `ImagePlus` and does no initialization.

`ImagePlus(java.lang.String urlString)`

Constructs a new `ImagePlus`, loading the Image from the URL specified.

`ImagePlus(java.lang.String title, java.awt.Image img)`

Constructs a new `ImagePlus` based on a Java AWT image. The first argument is the title of the `ImageWindow` that displays the image.

`ImagePlus(java.lang.String title, ImageProcessor ip)`

Constructs a new `ImagePlus` that uses the specified `ImageProcessor`. The first argument is the title of the `ImageWindow` that displays the image.

`ImagePlus(java.lang.String title, ImageStack stack)`

Constructs a new `ImagePlus` from an `ImageStack`. The first argument is the title of the `ImageWindow` that displays the image.

The type of an `ImagePlus` can be retrieved using

`int getType()`

Similar methods exist for getting the image dimension, the title (= name of the `ImageWindow` that displays this image), the AWT image that represents the `ImagePlus` and the file information:

`int getHeight()`

`int getWidth()`

`java.lang.String getTitle()`

`java.awt.Image getImage()`

`ij.io.FileInfo getFileInfo()`

The AWT image on which the `ImagePlus` is based and the image's title can be set using

`void setImage(java.awt.Image img)`

`void setTitle(java.lang.String title)`

An `ImagePlus` can have a list of additional properties that can be defined by the user. They are indexed using a string and can be any type of object. These properties can be read and set using the methods:

`java.util.Properties getProperties()`

Returns this image's Properties.

`java.lang.Object getProperty(java.lang.String key)`

Returns the property associated with key.

```
void setProperty(java.lang.String key, java.lang.Object value)
```

Adds a key-value pair to this image's properties.

4.3 Processors

Each image is based on an image processor. The type of the processor depends on the type of the image. You can get and set the image processor using these two methods of an `ImagePlus`:

```
ImageProcessor getProcessor()
```

Returns a reference to the image's `ImageProcessor`.

```
void setProcessor(java.lang.String title, ImageProcessor ip)
```

Sets the image processor to the one specified.

When working with plugin filters you do not have to care about retrieving the processor from the `ImagePlus`, it is passed as argument to the `run` method.

`ImageProcessor` is an abstract class. Depending on the type of the image we use a subclass of `ImageProcessor`. There are five of them:

ByteProcessor

Used for 8 bit grayscale and color images. It has a subclass called `BinaryProcessor` for grayscale images that only contain the pixel values 0 and 255.

ShortProcessor

Used for 16 bit grayscale images.

ColorProcessor

Used for 32 bit integer images (RGB with 8 bit per channel).

FloatProcessor

Used for 32 bit floating point images.

4.4 Accessing Pixel Values

To work with the image we need access to its pixels. We know how to get the image's `ImageProcessor`. Retrieving the pixel values can be done by using an `ImageProcessor`'s

```
java.lang.Object getPixels()
```

method. It returns a reference to this image's pixel array. As the type of this array depends on the image type we need to cast this array to the appropriate type when we get it:

```
int[] pixels = (int[]) myProcessor.getPixels()
```

This example would work for an RGB image. As you have noticed we get back a one-dimensional array. It contains the image scanline by scanline. To convert a position in this array to a (x,y) coordinate in an image, we need at least the width of a scanline.

The width and height of an `ImageProcessor` can be retrieved using these methods:

```
int getHeight()
```

```
int getWidth()
```

Now we have everything to iterate through the pixel array. As you have seen in the sample plugin this can be done using two nested loops.

Some cases need a bit more explanation: Reading pixels from `ByteProcessor`, `ShortProcessor` and from `ColorProcessor`.

Java's `byte` data type is signed and has values ranging from 128 to 127, while we would expect a 8 bit grayscale image to have values from 0 to 255. If we cast a `byte` variable to another type we have to make sure, that the sign bit is eliminated. This can be done using a binary AND operation (`&`):

```
int pix = 0xff & pixels[i];
...
pixels[i] = (byte) pix;
```

It's the same with Java's `short` data type, which is also signed and has values ranging from -32768 to 32767 , while we would expect a 16 bit grayscale image to have values from 0 to 65535. If we cast a `short` variable to another type we have to make sure that the sign bit is eliminated. This can again be done using a binary AND operation:

```
int pix = pixels[i] & 0xffff;
...
pixels[i] = (short) pix;
```

`ColorProcessors` return the pixel array as an `int[]`. The values of the three color components are packed into one `int` variable. They can be accessed as follows:

```
int red   = (int)(pixels[i] & 0xff0000)>>16;
int green = (int)(pixels[i] & 0x00ff00)>>8;
int blue  = (int)(pixels[i] & 0x0000ff);
...
pixels[i] = ((red & 0xff)<<16)+((green & 0xff)<<8) + (blue & 0xff);
```

The pixel array you work on is a reference to the `ImageProcessor`'s pixel array. So any modifications effect the `ImageProcessor` immediately. However, if you want the `ImageProcessor` to use another (perhaps newly created) array, you can do this using

```
void setPixels(java.lang.Object pixels)
```

You do not always have to retrieve or set the whole pixel array. `ImageProcessor` offers some other methods for retrieving or setting pixel values:

```
int getPixel(int x, int y)
```

Returns the value of the specified pixel.

```
void putPixel(int x, int y, int value)
```

Sets the pixel at (x, y) to the specified value.

```
float getPixelValue(int x, int y)
```

Returns the value of the specified pixel.

```
void getColumn(int x, int y, int[] data, int length)
```

Returns the pixels down the column starting at (x, y) in `data`.

```
void putColumn(int x, int y, int[] data, int length)
```

Inserts the pixels contained in `data` into a column starting at (x, y).

```
void getRow(int x, int y, int[] data, int length)
```

Returns the pixels along the horizontal line starting at (x,y) in `data`.

```
void putRow(int x, int y, int[] data, int length)
```

Inserts the pixels contained in `data` into a horizontal line starting at (x,y).

```
double[] getLine(int x1, int y1, int x2, int y2)
```

Returns the pixels along the line with start point (x1,y1) and end point (x2,y2).

The method

```
int[] getPixel(int x, int y)
```

of `ImagePlus` returns the pixel value at (x,y) as a four element array.

All these methods should only be used if you intend to modify just a few pixels. If you want to modify large parts of the image it is faster to work with the pixel array.

4.5 Regions of Interest

A plugin filter does not always have to work on the whole image. ImageJ supports regions of interest (ROI) which can be rectangular, oval, polygonal, freeform or text selections of regions of the image.

The bounding rectangle of the current ROI can be retrieved from the `ImageProcessor` using

```
java.awt.Rectangle getRoi()
```

This makes it possible to just handle the pixels that are inside this rectangle. It is also possible to set a processor's ROI with

```
void setRoi(int x, int y, int rwidth, int rheight)
```

This sets the ROI to the rectangle starting at (x,y) with specified width and height.

More methods for working with ROIs can be found in `ImagePlus`. Remember that a plugin filter's `run` method receives an `ImageProcessor` as argument, but you can access the `ImagePlus` in the `setup` method.

```
void setRoi(int x, int y, int width, int height)
```

Creates a rectangular selection starting at (x,y) with specified width and height.

```
void setRoi(java.awt.Rectangle r)
```

Creates a rectangular selection.

```
void setRoi(Roi roi)
```

Creates a selection based on the specified ROI object.

```
Roi getRoi()
```

Returns a ROI object representing the current selection.

The classes representing the different types of ROIs can be found in `ij.gui`. These classes are:

- `FreehandROI`
- `OvalROI`
- `PolygonROI`
- `ROI`
- `TextROI`

4.6 Creating New Images

In many cases it will make sense that a plugin does not modify the original image, but creates a new image that contains the modifications.

`ImagePlus`' method

```
ImagePlus createImagePlus()
```

returns a new `ImagePlus` with this `ImagePlus`' attributes, but no image. A similar function is provided by `ImageProcessor`'s

```
ImageProcessor createProcessor(int width, int height)
```

which returns a new, blank processor with specified width and height which can be used to create a new `ImagePlus` using the constructor

```
ImagePlus(java.lang.String title, ImageProcessor ip)
```

The class `NewImage` offers some useful static methods for creating a new `ImagePlus` of a certain type.

```
static ImagePlus createByteImage(java.lang.String title, int width, int height, int slices, int fill)
```

Creates a new 8 bit grayscale or color image with the specified title, width and height and number of slices. `fill` is one of the constants listed below that determine how the image is initially filled.

```
static ImagePlus createFloatImage(java.lang.String title, int width, int height, int slices, int fill)
```

Creates a new 32 bit floating point image with the specified title, width and height and number of slices. `fill` is one of the constants listed below that determine how the image is initially filled.

```
static ImagePlus createRGBImage(java.lang.String title, int width, int height, int slices, int fill)
```

Creates a new RGB image with the specified title, width and height and number of slices. `fill` is one of the constants listed below that determine how the image is initially filled.

```
static ImagePlus createShortImage(java.lang.String title, int width, int height, int slices, int fill)
```

Creates a new 16 bit grayscale image with the specified title, width and height and number of slices. `fill` is one of the constants listed below that determine how the image is initially filled.

These are the possible values for the `fill` argument defined in class `NewImage`:

`FILL_BLACK` Fills the image with black color.

`FILL_WHITE` Fills the image with white color

`FILL_RAMP` Fills the image with a horizontal grayscale ramp.

There are two methods to copy pixel values between different `ImageProcessors`:

`void copyBits(ImageProcessor ip, int xloc, int yloc, int mode)`
 Copies the image represented by `ip` to `xloc`, `yloc` using the specified blitting mode.
 This is one of the following constants defined in the interface `Blitter`:

`ADD` destination = destination+source

`AND` destination = destination AND source

`AVERAGE` destination = (destination+source)/2

`COPY` destination = source

`COPY_INVERTED` destination = 255-source

`COPY_TRANSPARENT` White pixels are assumed as transparent.

`DIFFERENCE` destination = |destination-source|

`DIVIDE` destination = destination/source

`MAX` destination = maximum(destination,source)

`MIN` destination = minimum(destination,source)

`MULTIPLY` destination = destination * source

`OR` destination = destination OR source

`SUBTRACT` destination = destination-source

`XOR` destination = destination XOR source

`void insert(ImageProcessor ip, int xloc, int yloc)`

Inserts the image contained in `ip` at (`xloc`, `yloc`).

If you need a Java AWT image, you can retrieve it from the `ImageProcessor` using

```
java.awt.Image createImage()
```

The same function is provided by `ImagePlus`' method

```
java.awt.Image getImage()
```

4.7 Displaying Images

Now that we can modify images we need to know how the changes can be made visible. `ImageJ` uses a class called `ImageWindow` to display `ImagePlus` images. `ImagePlus` contains everything that is necessary for updating or showing newly created images.

```
void draw()
```

Displays the image.

```
void draw(int x, int y, int width, int height)
```

Displays the image and draws the ROI outline using a clipping rectangle.

```

void updateAndDraw()
    Updates the image from the pixel data in its associated ImageProcessor and displays it.

void updateAndRepaintWindow()
    Calls updateAndDraw to update from the pixel data and draw the image. The method also repaints the image window to force the information displayed above the image (dimension, type, size) to be updated.

void show()
    Opens a window to display the image and clears the status bar.

void show(java.lang.String statusMessage)
    Opens a window to display the image and displays statusMessage in the status bar.

void hide()
    Closes the window, if any, that is displaying the image.

```

4.8 ColorInverter PlugIn (Example)

With the knowledge of the previous sections we can write our first own plugin. We will modify the `Inverter_` plugin so that it handles RGB images. It will invert the colors of the pixels of the original image's ROI and display the result in a new window.

As mentioned before, we start from the existing plugin `Inverter_`. First of all we modify the class name.

```

import ij.*;
import ij.gui.*;
import ij.process.*;
import ij.plugin.filter.PlugInFilter;
import java.awt.*;

public class ColorInverter_ implements PlugInFilter {
    ...

```

Don't forget to rename the file to `ColorInverter_.java`, otherwise you won't be able to compile it!

We want to handle RGB files, we do not want to apply it to stacks, we want to support non-rectangular ROIs and because we display the results in a new image we do not modify the original, so we change the capabilities returned by the `setup` method to `DOES_RGB + SUPPORTS_MASKING + NO_CHANGES`.

```

    public int setup(String arg, ImagePlus imp) {
        if (arg.equals("about")) {
            showAbout();
            return DONE;
        }
        return DOES_RGB+SUPPORTS_MASKING+NO_CHANGES;
    }

```

The `run` method will do the actual work.

```
public void run(ImageProcessor ip) {
```

First we save the dimension and the ROI of the original image to local variables.

```
int w = ip.getWidth();
int h = ip.getHeight();
Rectangle roi = ip.getRoi();
```

We want to have the result written to a new image, so we create a new RGB image of the same size, with one slice and initially black and get the new image's processor.

```
ImagePlus inverted = NewImage.createRGBImage("Inverted image", w, h,
                                             1, NewImage.FILL_BLACK);
ImageProcessor inv_ip = inverted.getProcessor();
```

Then we copy the image from the original `ImageProcessor` to (0,0) in the new image, using `COPY` blitting mode (this mode just overwrites the pixels in the destination processor). We then get the pixel array of the new image (which is of course identical to the old one). It's a RGB image, so we get an `int` array.

```
inv_ip.copyBits(ip,0,0,Blitter.COPY);
int[] pixels = (int[]) inv_ip.getPixels();
```

We now go through the bounding rectangle of the ROI with two nested loops. The outer one runs through the lines in the ROI, the inner one through the columns in each line. The offset in the one-dimensional array is the start of the current line (= width of the image × number of scanlines).

```
for (int i=roi.y; i<roi.y+roi.height; i++) {
    int offset =i*w;
    for (int j=roi.x; j<roi.x+roi.width; j++) {
```

In the inner loop we calculate the position of the current pixel in the one-dimensional array (we save it in a variable because we need it twice). We then get the value of the current pixel. Note that we can access the pixel array of the new image, as it contains a copy of the old one.

```
int pos = offset+j;
int c = pixels[pos];
```

We extract the three color components as described above.

```
int r = (c&0xff0000)>>16;
int g = (c&0x00ff00)>>8;
int b = (c&0x0000ff);
```

We invert each component by subtracting it's value from 255. Then we pack the modified color components into an integer again.

```

        r=255-r;
        g=255-g;
        b=255-b;
        pixels[pos] = ((r & 0xff) << 16) +
                      ((g & 0xff) << 8) +
                      (b & 0xff);
    }
}

```

We have now done all necessary modifications to the pixel array. Our image is still not visible, so we call `show` to open an `ImageWindow` that displays it. Then we call `updateAndDraw` to force the pixel array to be read and the image to be updated.

```

        inverted.show();
        inverted.updateAndDraw();
    }
}

```

4.9 Stacks

ImageJ supports expandable arrays of images called image stacks, that consist of images (slices) of the same size. In a plugin filter you can access the currently open stack by retrieving it from the current `ImagePlus` using

```
ImageStack getStack()
```

`ImagePlus` also offers a method for creating a new stack:

```
ImageStack createEmptyStack()
```

Returns an empty image stack that has the same width, height and color table as this image.

Alternatively you can create an `ImageStack` using one of these constructors:

```
ImageStack(int width, int height)
```

Creates a new, empty image stack with specified height and width.

```
ImageStack(int width, int height, java.awt.image.ColorModel cm)
```

Creates a new, empty image stack with specified height, width and color model.

To set the newly created stack as the stack of an image use

```
void setStack(java.lang.String title, ImageStack stack)
```

The number of slices of a stack can be retrieved using the methods

```
int getSize()
```

of class `ImageStack` or

```
int getStackSize()
```

of class `ImagePlus`.

The currently displayed slice of an `ImagePlus` can be retrieved and set using

```
int getCurrentSlice()
```

```
void setSlice(int index)
```

A stack offers several methods for retrieving and setting its properties:

`int getHeight()`

Returns the height of the stack.

`int getWidth()`

Returns the width of the stack.

`java.lang.Object getPixels(int n)`

Returns the pixel array of the specified slice, where `n` is a number from 1 to the number of slices. See also Section 4.4.

`void setPixels(java.lang.Object pixels, int n)`

Assigns a pixel array to the specified slice, where `n` is a number from 1 to the number of slices. Section 4.4.

`ImageProcessor getProcessor(int n)`

Returns an `ImageProcessor` for the specified slice, where `n` is a number from 1 to the number of slices. See also Section 4.3.

`java.lang.String getSliceLabel(int n)`

Returns the label of the specified slice, where `n` is a number from 1 to the number of slices.

`void setSliceLabel(java.lang.String label, int n)`

Sets the label of the specified slice, where `n` is a number from 1 to the number of slices.

`java.awt.Rectangle getRoi()`

Returns the bounding rectangle of the stack's ROI. For more information on ROIs, see Section 4.5.

`void setRoi(java.awt.Rectangle roi)`

Sets the stacks ROI to the specified rectangle. For more information on ROIs, see Section 4.5.

Slices can be added to and removed from the `ImageStack` using these methods:

`void addSlice(java.lang.String sliceLabel, ImageProcessor ip)`

Adds the image represented by `ip` to the end of the stack.

`void addSlice(java.lang.String sliceLabel, ImageProcessor ip, int n)`

Adds the image represented by `ip` to the stack following slice `n`.

`void addSlice(java.lang.String sliceLabel, java.lang.Object pixels)`

Adds an image represented by its pixel array to the end of the stack.

`void deleteLastSlice()`

Deletes the last slice in the stack.

`void deleteSlice(int n)`

Deletes the specified slice, where `n` is in the range 1...number of slices.

4.10 StackAverage PlugIn (Example)

This example shows how to handle stacks. It calculates the average values of pixels located at the same position in each slice of the stack and adds a slice showing the average values to the end of the stack.

First of all, we import the necessary packages. We want to work on the current stack so we need to implement `PlugInFilter`.

```
import ij.*;
import ij.plugin.filter.PlugInFilter;
import ij.process.*;

public class StackAverage_ implements PlugInFilter {
```

We define the stack as instance variable because we will retrieve it in `setup` and use it in `run`.

```
    protected ImageStack stack;
```

In this method we get the stack from the current image and return the plugin's capabilities—in this case we indicate that it handles 8 bit grayscale images and requires a stack as input.

```
    public int setup(String arg, ImagePlus imp) {
        stack = imp.getStack();
        return DOES_8G + STACK_REQUIRED;
    }
```

In the `run` method we declare a `byte` array that will hold the pixels of the current slice. Then we get width and height of the stack and calculate the length of the pixel array of each slice as the product of width and height. `sum` is the array to hold the summed pixel values.

```
    public void run(ImageProcessor ip) {
        byte[] pixels;
        int dimension = stack.getWidth()*stack.getHeight();
        int[] sum = new int[dimension];
```

In the outer loop we iterate through the slices of the stack and get the pixel array from each slice. In the inner loop we go through the pixel array of the current slice and add the pixel value to the corresponding pixel in the `sum` array.

```
        for (int i=1;i<=stack.getSize();i++) {
            pixels = (byte[]) stack.getPixels(i);
            for (int j=0;j<dimension;j++) {
                sum[j]+=0xff & pixels[j];
            }
        }
```

We have now gone through the whole stack. The image containing the averages will be a 8 bit grayscale image again, so we create a `byte` array for it. Then we iterate through the pixels in the `sum` array and divide each of them through the number of slices to get pixel values in the range 0...255.

```
        byte[] average = new byte[dimension];
        for (int j=0;j<dimension;j++) {
            average[j] = (byte) ((sum[j]/stack.getSize()) & 0xff);
        }
```

Finally we add a new slice to the stack. It is called “Average” and represented by the pixel array that contains the average values.

```
        stack.addSlice("Average", average);
    }
```

4.11 Additional Reference

This reference is thought as a supplement to the concepts presented in this section. It is not complete—it just covers what you will normally need for writing plugins. For a complete reference see the API documentation and the source code.

4.11.1 ImagePlus

Windows

```
void setWindow(ImageWindow win)
```

Sets the window that displays the image.

```
ImageWindow getWindow()
```

Gets the window that is used to display the image.

```
void mouseMoved(int x, int y)
```

Displays the cursor coordinates and pixel value in the status bar.

Multithreading

```
boolean lock()
```

Locks the image so that it cannot be accessed by another thread.

```
boolean lockSilently()
```

Similar to lock, but doesn't beep and display an error message if the attempt to lock the image fails.

```
void unlock()
```

Unlocks the image.

Lookup Tables

```
LookupTable createLut()
```

Creates a LookupTable based on the image.

Statistics

```
ij.process.ImageStatistics getStatistics()
```

Returns an ImageStatistics object generated using the standard measurement options (area, mean, mode, min and max).

```
ij.process.ImageStatistics getStatistics(int mOptions)
```

Returns an ImageStatistics object generated using the specified measurement options.

```
ij.process.ImageStatistics getStatistics(int mOptions, int nBins)
```

Returns an ImageStatistics object generated using the specified measurement options and number of histogram bins.

Calibration

```
void setCalibration(ij.measure.Calibration cal)
    Sets this image's calibration.

void setGlobalCalibration(ij.measure.Calibration global)
    Sets the system-wide calibration.

ij.measure.Calibration getCalibration()
    Returns this image's calibration.
```

4.11.2 ImageProcessor

Geometric Transforms

```
void flipHorizontal()
    Flips the image horizontally.

void flipVertical()
    Flips the image vertically.

void rotate(double angle)
    Rotates the image angle degrees clockwise.

void scale(double xScale, double yScale)
    Scales the image by the specified factors.

ImageProcessor crop()
    Crops the image to the bounding rectangle of the current ROI. Returns a new image
    processor that represents the cropped image.

ImageProcessor resize(int dstWidth, int dstHeight)
    Resizes the image to the specified destination size. Returns a new image processor
    that represents the resized image.

ImageProcessor rotateLeft()
    Rotates the image 90 degrees counter-clockwise. Returns a new image processor
    that represents the rotated image.

ImageProcessor rotateRight()
    Rotates the image 90 degrees clockwise. Returns a new image processor that repre-
    sents the rotated image.

void setInterpolate(boolean interpolate)
    Setting interpolate to true causes scale(), resize() and rotate() to do bilinear
    interpolation, otherwise nearest-neighbor interpolation is used.
```

Filters

```
void convolve3x3(int[] kernel)
    Convolves the image with the specified 3×3 convolution matrix.

void sharpen()
    Sharpens the image using a 3×3 convolution kernel.
```

`void smooth()`
Replaces each pixel with the 3×3 neighborhood mean.

`void filter(int type)`
A 3×3 filter operation, the argument defines the filter type.

`void dilate()`
Dilates the image using a 3×3 minimum filter.

`void erode()`
Erodes the image using a 3×3 maximum filter.

`void findEdges()`
Finds edges using a Sobel operator.

`void medianFilter()`
A 3×3 median filter.

`void gamma(double value)`
A gamma correction.

`void invert()`
Inverts an image.

`void add(int value)`
Adds the argument to each pixel value.

`void multiply(double value)`
Multiplies each pixel value with the argument.

`void and(int value)`
Binary AND of each pixel value with the argument.

`void or(int value)`
Binary OR of each pixel value with the argument.

`void xor(int value)`
Binary exclusive OR of each pixel value with the argument.

`void log()`
Calculates pixel values on a logarithmic scale.

`void noise(double range)`
Adds random noise (random numbers within range) to the image.

Drawing

`void setColor(java.awt.Color color)`
Sets the foreground color. This will set the default fill/draw value to the pixel value that represents this color.

`void setValue(double value)`
Sets the default fill/draw value.

```
void setLineWidth(int width)
    Sets the line width.
void moveTo(int x, int y)
    Sets the current drawing location to (x,y).
void lineTo(int x2, int y2)
    Draws a line from the current drawing location to (x2,y2).
void drawPixel(int x, int y)
    Sets the pixel at (x,y) to the current drawing color.
void drawDot(int xcenter, int ycenter)
    Draws a dot using the current line width and color.
void drawDot2(int x, int y)
    Draws a 2×2 dot in the current color.
void fill()
    Fills the current rectangular ROI with the current drawing color.
void fill(int[] mask)
    Fills pixels that are within the current ROI and part of the mask (i.e. pixels that
    have value 0 (= black) in the mask array).
void drawString(java.lang.String s)
    Draws the string s at the current location with the current color.
int getStringWidth(java.lang.String s)
    Returns the width of the specified string in pixels.
```

Colors

```
int getBestIndex(java.awt.Color c)
    Returns the LUT index that matches the specified color best.
java.awt.image.ColorModel getColorModel()
    Returns this processor's color model.
void invertLut()
    Inverts the values in the lookup table.
```

Minimum, Maximum and Threshold

```
double getMin()
    Returns the smallest displayed pixel value.
double getMax()
    Returns the largest displayed pixel value.
void setMinAndMax(double min, double max)
    Maps the pixels in this image from min...max to the value range of this type of
    image.
```

`void resetMinAndMax()`
For short and float images, recalculates the minimum and maximum image values needed to correctly display the image (i.e. maps the color values to the 255 displayable grayscales).

`void autoThreshold()`
Calculates the auto threshold of an image and applies it.

`double getMinThreshold()`
Returns the minimum threshold.

`double getMaxThreshold()`
Returns the maximum threshold.

`void setThreshold(double minThreshold, double maxThreshold, int lutUpdate)`
Sets the minimum and maximum threshold levels, the third parameters specifies if the lookup table will be recalculated.

Histograms

`int[] getHistogram()`
Returns the histogram of the image. This method will return a luminosity histogram for RGB images and `null` for floating point images.

`int getHistogramSize()`
The size of the histogram is 256 for 8 bit and RGB images and `maxmin+1` for 16 bit integer images.

Snapshots (Undo)

`void snapshot()`
Saves the current state of the processor as snapshot.

`java.lang.Object getPixelsCopy()`
Returns a reference to this image's snapshot (undo) array, i.e. the pixel array before the last modification.

`void reset()`
Resets the processor to the state saved in the snapshot.

`void reset(int[] mask)`
Resets the processor to the state saved in the snapshot, excluding pixels that are part of mask.

4.11.3 Stacks

Accessing Images

`java.lang.Object[] getImageArray()`
Returns the stack as an array of `ImagePlus` objects.

Color

`boolean isHSB()`
Returns `true`, if this is a 3-slice HSB stack.

```
boolean isRGB()  
    Returns true, if this is a 3-slice RGB stack.  
java.awt.image.ColorModel getColorModel()  
    Returns the stack's color model.  
void setColorModel(java.awt.image.ColorModel cm)  
    Assigns a new color model to the stack.
```

5 ImageJ's Utility Methods and Image Conversion

The ImageJ API contains a class called IJ that contains some very useful static methods. These utility methods are also used in plugins generated using “Plugins/Record”. They are discussed in this section. Section 5.8 deals with conversion from one image type to another.

5.1 (Error) Messages

It is often necessary that a plugin displays a message—be it an error message or any other information. In the first case you will use

```
static void error(java.lang.String msg)  
    which displays a message in a dialog box entitled “Error”, in the second case  
static void showMessage(java.lang.String msg)  
    which displays a message in a dialog box entitled “Message”.
```

You can also specify the title of the message box using

```
static void showMessage(java.lang.String title, java.lang.String msg)
```

All these methods display messages that the user has to accept. If you want to let the user choose whether to cancel the plugin or to let it continue, use

```
static boolean showMessageWithCancel(java.lang.String title,  
java.lang.String msg)
```

This method returns `false` if the user clicks cancel and `true` otherwise.

There are also some predefined messages:

```
static void noImage()  
    Displays a “no images are open” dialog box.  
static void outOfMemory(java.lang.String name)  
    Displays an “out of memory” message in the ImageJ window.  
static boolean versionLessThan(java.lang.String version)  
    Displays an error message and returns false if the ImageJ version is less than the  
    one specified.
```

5.2 ImageJ Window, Status Bar and Progress Bar

The ImageJ main window and its components are shown in Figure 1. Until version 1.25, the results window (Figure 2) was part of the main window in form of a text panel.

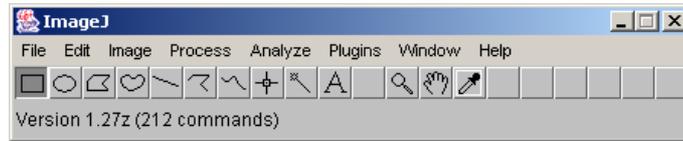


Figure 1: The ImageJ main window and its components: Menu bar, tool bar and status bar.

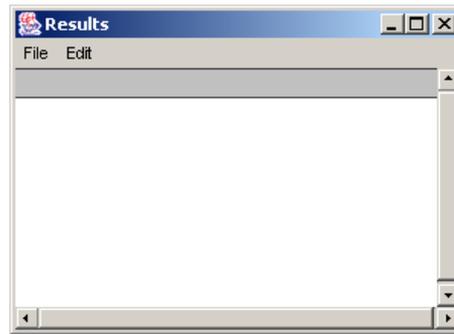


Figure 2: The ImageJ results window displays messages, measurements, etc. It opens automatically, when text is written to it.

5.2.1 Displaying Text

To display a line of text in the results window (Figure 2) use

```
static void write(java.lang.String s)
```

It is possible to use the results window's text panel as a table (e.g. for displaying statistics, measurements, etc.). In that case ImageJ lets you set the headings of the columns using

```
static void setColumnHeadings(java.lang.String headings)
```

Note that this method clears the entire text panel.

You will often want to displays numbers, which you can format for output using

```
static java.lang.String d2s(double n)
```

Converts a number to a formatted string using two digits to the right of the decimal point.

```
static java.lang.String d2s(double n, int precision)
```

Converts a number to a rounded formatted string.

5.2.2 Status Bar

Text can also be displayed in the status bar at the bottom of the main window using the method

```
static void showStatus(java.lang.String s)
```

It can be useful to display the time that was needed for an operation:

```
static void showTime(ImagePlus imp, long start, java.lang.String str)
```

will display the string argument you specify, followed by the time elapsed since the specified start value and the rate of processed pixels per second.

5.2.3 Progress Bar

The progress of the current operation can be visualized using ImageJ's progress bar.

```
static void showProgress(double progress)
    updates the position of the progress bar to the specified value (in the range from 0.0
    to 1.0).
```

5.3 User input

Often user input (e. g. a parameter) is required in a plugin. ImageJ offers two simple methods for that purpose.

```
static double getNumber(java.lang.String prompt, double defaultNumber)
    Allows the user to enter a number in a dialog box.
```

```
static java.lang.String getString(java.lang.String prompt,
java.lang.String defaultString)
    Allows the user to enter a string in a dialog box.
```

A way to build more sophisticated dialogs is presented in Section 6.2, accessing mouse and keyboard events is discussed in Section 6.7.

5.4 Calling Menu Commands

You can access all menu commands from a plugin. There are two different methods:

```
static void doCommand(java.lang.String command)
    Starts executing a menu command in a separate thread and returns immediately.
    Executing the command in a separate thread means that the program will not wait
    until the command is executed, it will immediately proceed. This has the advantage
    that the program is not blocked while the command is running.
```

```
static void run(java.lang.String command)
    Runs a menu command in the current thread, the program is will continue after the
    command has finished.
```

5.5 Calling Other PlugIns

Like menu commands you can also run other plugins.

```
static java.lang.Object runPlugIn(java.lang.String className,
java.lang.String arg)
    Runs the plugin specified by its class name and initializes it with the specified
    argument.
```

5.6 MessageTest PlugIn (Example)

We will now look at a plugin that uses some of the utility methods presented in this chapter. This time, we do not need an image, so we implement the interface `PlugIn`. We also have to import the package `ij` as we need the class `IJ` from there.

```
import ij.*;
import ij.plugin.PlugIn;
```

```
public class Message_Test implements PlugIn {
```

All we have to implement is the `run` method. We do not need the argument, so we ignore it. First of all we display a string in the status bar that informs the user that the plugin was started. Then we set the progress bar to 0% and show an error message.

```
    public void run(String arg) {
        IJ.showStatus("Plugin Message Test started.");
        IJ.showProgress(0.0);
        IJ.error("I need user input!");
```

We want the user to input a string and set the progress bar to 50% after that. Then we write a message into the main window saying that we were going to start the sample plugin `RedAndBlue` (this is one of the plugins that come with ImageJ and displays a new image with a red/blue gradient) and run the plugin. Finally we set the progress bar to 100% and show a custom message box.

```
        String name = IJ.getString("Please enter your name: ", "I.J. User");
        IJ.showProgress(0.5);
        IJ.write("Starting sample plugin RedAndBlue ... ");
        IJ.runPlugIn("RedAndBlue_", "");
        IJ.showProgress(1.0);
        IJ.showMessage("Finished.", name + ", thank you for running this plugin");
    }
}
```

5.7 More Utilities

Keyboard & Sound

```
static void beep()
    Emits an audio beep.

static boolean altKeyDown()
    Returns true if the Alt key is down.

static boolean spaceBarDown()
    Returns true if the space bar is down.
```

Accessing GUI Elements

```
static ImageJ getInstance()
    Returns a reference to the "ImageJ" frame.

static java.applet.Applet getApplet()
    Returns the applet that created this ImageJ or null if running as an application.

static TextPanel getTextPanel()
    Returns a reference to the text panel in ImageJ's results window.
```

Misc

```
static boolean isMacintosh()
    Returns true if the machine on which ImageJ is running is a Macintosh.

static void wait(int msec)
    Delays msec milliseconds.

static java.lang.String freeMemory()
    Returns the amount of free memory in kByte as string.
```

5.8 Image Type Conversion

The simplest way to convert an image from one type to another is to use the conversion methods of the image's `ImageProcessor`.

```
ImageProcessor convertToByte(boolean doScaling)
    Converts the processor to a ByteProcessor (8 bit grayscale). If doScaling is set, the pixel values are scaled to the range 0 – 255, otherwise the values are clipped.
```

```
ImageProcessor convertToFloat()
    Converts the processor to a FloatProcessor (32 bit grayscale). If a calibration table has been set, the calibration function is used.
```

```
ImageProcessor convertToRGB()
    Converts the processor to a ColorProcessor (RGB image).
```

```
ImageProcessor convertToShort(boolean doScaling)
    Converts the processor to a ShortProcessor (16 bit grayscale). If doScaling is set, the pixel values are scaled to the range 0 – 65536, otherwise the values are clipped.
```

The class `ImageConverter` in `ij.process` provides a number of methods for image type conversion, also methods for converting RGB and HSB to stacks and vice versa. They can be accessed either directly or by using the plugin class `ij.plugin.Converter` as a convenient interface.

An instance of the converter can be constructed using

```
Converter()
```

and works on the current image.

The only method of this class is

```
public void convert(java.lang.String item)
    where item is a string specifying the destination type. It can have one of the values "8-bit", "16-bit", "32-bit", "8-bit Color", "RGB Color", "RGB Stack" and "HSB Stack".
```

Similarly, an `ImageConverter` instance can be created using

```
ImageConverter(ImagePlus imp)
```

The methods for conversion are:

```
public void convertToGray8()
    Converts the ImagePlus to 8 bit grayscale.
```

```
public void convertToGray16()
    Converts the ImagePlus to 16 bit grayscale.
```

```
public void convertToGray32()
```

Converts the `ImagePlus` to 32 bit grayscale.

```
public void convertToRGB()
```

Converts the `ImagePlus` to RGB.

```
public void convertToRGBStack()
```

Converts an RGB image to an RGB stack (i. e. a stack with 3 slices representing red, green and blue channel).

```
public void convertToHSB()
```

Converts an RGB image to a HSB stack (i. e. a stack with 3 slices representing hue, saturation and brightness channel).

```
public void convertRGBStackToRGB()
```

Converts a 2 or 3 slice 8-bit stack to RGB.

```
public void convertHSBToRGB()
```

Converts a 3-slice (hue, saturation, brightness) 8-bit stack to RGB.

```
public void convertRGBtoIndexedColor(int nColors)
```

Converts an RGB image to 8-bits indexed color. `nColors` must be greater than 1 and less than or equal to 256.

To scale to 0–255 when converting `short` or `float` images to `byte` images and to 0–65535 when converting `float` to `short` images set scaling `true` using

```
public static void setDoScaling(boolean scaleConversions)
```

```
public static boolean getDoScaling()
```

returns `true` if scaling is enabled.

`ImageConverter` does not convert stacks, you can use `StackConverter` for this purpose. An instance of this class can be created using

```
StackConverter(ImagePlus img)
```

It has the following methods:

```
void convertToGray16()
```

Converts this `Stack` to 16-bit grayscale.

```
void convertToGray32()
```

Converts this `Stack` to 32-bit (float) grayscale.

```
void convertToGray8()
```

Converts this `Stack` to 8-bit grayscale.

```
void convertToRGB()
```

Converts the `Stack` to RGB.

6 Windows

By default, plugins work with `ImagePlus` objects displayed in `ImageWindows`. They can output information to the `ImageJ` window but they cannot control a window. Sometimes

this can be necessary, especially for getting user input.

6.1 PlugInFrame

A `PlugInFrame` is a subclass of an AWT frame that implements the `PlugIn` interface. Your plugin will be implemented as a subclass of `PlugInFrame`.

There is one constructor for a `PlugInFrame`. It receives the title of the window as argument:

```
PlugInFrame(java.lang.String title)
```

As this class is a plugin, the method

```
void run(java.lang.String arg)
```

declared in the `PlugIn` interface is implemented and can be overwritten by your plugin's `run` method.

Of course all methods declared in `java.awt.Frame` and its superclasses can be overwritten. For details consult the Java AWT API documentation.

6.2 GenericDialog

In Section 5.3 we saw a very simple way of getting user input. If you need more user input than just one string or number, `GenericDialog` helps you build a *modal* AWT dialog, i. e. the programs only proceeds after the user has answered the dialog. The `GenericDialog` can be built on the fly and you don't have to care about event handling.

There are two constructors:

```
GenericDialog(java.lang.String title)
```

Creates a new `GenericDialog` with the specified title.

```
GenericDialog(java.lang.String title, java.awt.Frame parent)
```

Creates a new `GenericDialog` using the specified title and parent frame (e. g. your plugin class, which is derived from `PlugInFrame`). The ImageJ frame can be retrieved using `IJ.getInstance()`.

The dialog can be displayed using

```
void showDialog()
```

6.2.1 Adding controls

`GenericDialog` offers several methods for adding standard controls to the dialog:

```
void addCheckbox(java.lang.String label, boolean defaultValue)
```

Adds a checkbox with the specified label and default value.

```
public void addCheckboxGroup(int rows, int columns, java.lang.String[] labels, boolean[] defaultValues)
```

Adds a group of checkboxes using a grid layout with the specified number of rows and columns. The arrays contain the labels and the default values of the checkboxes.

```
void addChoice(java.lang.String label, java.lang.String[] items, java.lang.String defaultItem)
```

Adds a drop down list (popup menu) with the specified label, items and default value.

```
void addMessage(java.lang.String text)
```

Adds a message consisting of one or more lines of text.

```
void addNumericField(java.lang.String label, double defaultValue, int digits)
```

Adds a numeric field with the specified label, default value and number of digits.

```
void addStringField(java.lang.String label, java.lang.String defaultText)
```

Adds a 8 column text field with the specified label and default value.

```
void addStringField(java.lang.String label, java.lang.String defaultText, int columns)
```

Adds a text field with the specified label, default value and number of columns.

```
void addTextAreas(java.lang.String text1, java.lang.String text2, int rows, int columns)
```

Adds one or two text areas (side by side) with the specified initial contents and number of rows and columns. If `text2` is `null`, the second text area will not be displayed.

6.2.2 Getting Values From Controls

After the user has closed the dialog window, you can access the values of the controls with the methods listed here. There is one method for each type of control. If the dialog contains more than one control of the same type, each call of the method will return the value of the next control of this type in the order in which they were added to the dialog.

```
boolean getNextBoolean()
```

Returns the state of the next checkbox.

```
java.lang.String getNextChoice()
```

Returns the selected item in the next drop down list (popup menu).

```
int getNextChoiceIndex()
```

Returns the index of the selected item in the next drop down list (popup menu).

```
double getNextNumber()
```

Returns the contents of the next numeric field.

```
java.lang.String getNextString()
```

Returns the contents of the next text field.

```
java.lang.String getNextText()
```

Returns the contents of the next text area.

The method

```
boolean wasCanceled()
```

returns `true`, if the user closed the dialog using the “Cancel” button, and `false`, if the user clicked the “OK” button.

If the dialog contains numeric fields, use

```
boolean invalidNumber()
```

to check if the values in the numeric fields are valid numbers. This method returns `true` if at least one numeric field does not contain a valid number.

`GenericDialog` extends `java.awt.Dialog`, so you can use any method of `java.awt.Dialog` or one of its superclasses. For more information consult the Java AWT documentation.

6.3 FrameDemo PlugIn (Example)

This demo shows the usage of `GenericDialog` and `PlugInFrame`. It displays a dialog that lets the user specify the width and height of the `PlugInFrame` that will be displayed after closing the dialog.

We import the `ij` and `ij.process` package, the `ij.gui` package, where `GenericDialog` is located and the classes `PlugInFrame` and `AWT label`.

```
import ij.*;
import ij.gui.*;
import ij.plugin.frame.PlugInFrame;
import java.awt.Label;
```

Our plugin is a subclass of `PlugInFrame` which implements the `PlugIn` interface, so we don't have to implement an interface here.

```
public class FrameDemo_ extends PlugInFrame {
```

We overwrite the default constructor of the new class. If we wouldn't do that, the superclass' default constructor `PlugInFrame()` would be called, which does not exist. So we have to call the superclass' constructor and specify a title for the new frame.

```
    public FrameDemo_() {
        super("FrameDemo");
    }
```

In the `run` method we create a `GenericDialog` with the title "FrameDemo settings". Then we add two 3 digit numeric fields with a default value of 200.

```
    public void run(String arg) {
        GenericDialog gd = new GenericDialog("FrameDemo settings");
        gd.addNumericField("Frame width:",200.0,3);
        gd.addNumericField("Frame height:",200.0,3);
```

We display the dialog. As it is modal, the program is stopped until the user closes the dialog. If the user clicks "Cancel" we display an error message and leave the `run` method.

```
        gd.showDialog();
        if (gd.wasCanceled()) {
            IJ.error("PlugIn canceled!");
            return;
        }
```

Here we get the values of the numeric fields with two calls of `getNextNumber()`. We set the size of the `FrameDemo` window to these values and add a centered AWT label with the text "PlugInFrame demo". Finally we show the frame.

```
        this.setSize((int) gd.getNextNumber(),(int) gd.getNextNumber());
        this.add(new Label("PlugInFrame demo",Label.CENTER));
        this.show();
    }
}
```

6.4 ImageWindow

An `ImageWindow` is a frame (derived from `java.awt.Frame`) that displays an `ImagePlus`. The frame contains an `ImageCanvas` on which the image is painted and a line of information text on top. Each `ImagePlus` is associated with an `ImageWindow`, which is created when the image's `show()` method is called for the first time. `ImageWindows` can also be created using one of the constructors:

```
ImageWindow(ImagePlus imp)
```

Creates a new `ImageWindow` that contains the specified image.

```
ImageWindow(ImagePlus imp, ImageCanvas ic)
```

Creates a new `ImageWindow` that contains the specified image which will be painted on the specified canvas.

`ImageJ` maintains the list of open windows using the `WindowManager` class. When the constructor of `ImageWindow` is called, the window is added to the list of open windows.

```
boolean close()
```

Closes the window and removes it from the list. If the image has been changed, this method will ask the user whether the image displayed in this window shall be saved. If the user wants to save the image the method returns `false`. Otherwise it returns `true` and the image is deleted.

```
boolean isClosed()
```

Returns `true` if `close()` has already been called, `false` otherwise.

The image displayed in an `ImageWindow` and the canvas on which the image is drawn can be accessed using

```
ImagePlus getImagePlus()
```

```
ImageCanvas getCanvas()
```

`ImageWindow` provides methods for the cut, copy and paste command:

```
void copy(boolean cut)
```

Copies the current ROI (which has to be rectangular) to the clipboard. If the argument `cut` is `true` the ROI is cut and not copied.

```
void paste()
```

Pastes the content of the clipboard into the current image. The content of the clipboard may not be larger than the current image and must be the same type.

Like an `ImagePlus` an `ImageWindow` has a method

```
void mouseMove(int x, int y)
```

This method displays the specified coordinates and the pixel value of the image in this window in the status bar of the `ImageJ` window.

`ImageWindow` has also a useful public boolean variable called `running`, which is set to `false` if the user clicks in the window, presses escape or closes the window. This can be used in a plugin like shown in the following fragment to give the user a possibility to interrupt a plugin.

```
...
win.running = true;
while (win.running) {
    // do computation
}
```

6.5 ImageCanvas

Each `ImageWindow` has an `ImageCanvas` on which the image is drawn. This is a subclass of `java.awt.Canvas` and also implements a `MouseListener` and a `MouseMotionListener` (for more information see the Java API documentation, package `java.awt.event`). It can therefore be useful for event handling, e.g. by subclassing it. Additionally it can be used to get information on how the image is displayed and to modify this. Some useful methods of `ImageCanvas` are listed here:

```
java.awt.Point getCursorLoc()
```

Returns the current cursor location.

```
double getMagnification()
```

Returns the current magnification factor of the image.

```
java.awt.Rectangle getSrcRect()
```

The surrounding rectangle of the image with current magnification.

```
int offScreenX(int x)
```

Converts a screen x-coordinate to an offscreen x-coordinate.

```
int offScreenY(int y)
```

Converts a screen y-coordinate to an offscreen y-coordinate.

```
int screenX(int x)
```

Converts an offscreen x-coordinate to a screen x-coordinate.

```
int screenY(int y)
```

Converts an offscreen y-coordinate to a screen y-coordinate.

```
void setCursor(int x, int y)
```

Sets the cursor based on the current tool and cursor location.

```
void setImageUpdated()
```

`ImagePlus.updateAndDraw` calls this method to get the `paint` method to update the image from the `ImageProcessor`.

```
void setMagnification(double magnification)
```

Sets new magnification factor for image.

```
void zoomIn(int x, int y)
```

Zooms in by making the window bigger.

```
void zoomOut(int x, int y)
```

Zooms out by making `srcRect` bigger.

6.6 Subclasses of ImageWindow

6.6.1 StackWindow

A `StackWindow` is a frame for displaying `ImageStacks`. It is derived from `ImageWindow` and has a horizontal scrollbar to navigate within the stack.

```
void showSlice(int index)
    Displays the specified slice and updates the stack's scrollbar.

void updateSliceSelector()
    Updates the stack's scrollbar.
```

6.6.2 Histogram Window

`HistogramWindow` is a subclass of `ImageWindow` designed to display histograms. There are two constructors:

```
HistogramWindow(ImagePlus imp)
    Displays a histogram (256 bins) of the specified image. The window has the title
    "Histogram".

HistogramWindow(java.lang.String title, ImagePlus imp, int bins)
    Displays a histogram of the image, using the specified title and number of bins.

void showHistogram(ImagePlus imp, int bins)
    Displays the histogram of the image using the specified number of bins in the
    HistogramWindow.
```

6.6.3 PlotWindow

This is a subclass of `ImageWindow` designed for displaying plots in a (x,y) -plane

```
PlotWindow(java.lang.String title, java.lang.String xLabel,
java.lang.String yLabel, float[] xValues, float[] yValues)
    Constructs a new plot window with specified title, labels for x- and y- axis and adds
    points with specified (x,y)-coordinates.

void addLabel(double x, double y, java.lang.String label)
    Adds a new label with the specified text at position (x,y).

void addPoints(float[] x, float[] y, int shape)
void addPoints(double[] x, double[] y, int shape)
    These two methods add points with specified  $(x,y)$  coordinates to the plot. The
    number of points is given by the length of the array. The argument shape determines
    the shape of a point. Currently only circles are supported, which is specified by
    passing the constant PlotWindow.CIRCLE.

void setLimits(double xMin, double xMax, double yMin, double yMax)
    Sets the limits of the plotting plane.
```

6.7 Event Handling (Example)

`ImageWindow` and `ImageCanvas` are derived from the AWT classes `Frame` and `Canvas` and therefore support event handling. This is especially useful to get user input via mouse and keyboard events.

Event handling in Java AWT is based on interfaces called listeners. There is a listener interface for each type of event. A class implementing a listener interface is able to react on a certain type of event. The class can be added to a component's list of listeners and will be notified when an event that it can handle occurs.

A plugin that has to react on a certain type of event can implement the appropriate interface. It can access the window of the image it works on and the canvas on which the image is painted. So it can be added as a listener to these components.

For example, we want to write a plugin that reacts to mouse clicks on the image it works on. The listener interfaces are defined in `java.awt.event`, so we import this package.

```
import ij.*;
import ij.plugin.filter.PlugInFilter;
import ij.process.*; import ij.gui.*;
import java.awt.event.*;
```

We have to access the image and the canvas in more than one method, so we declare them as instance variables:

```
ImagePlus img;
ImageCanvas canvas;
```

The plugin has to implement the appropriate Interface:

```
public class Mouse_Listener implements PlugInFilter, MouseListener {
    ...
```

In the `setup` method we have access to the `ImagePlus` so we save it to the instance variable. We also set the plugin's capabilities.

```
public int setup(String arg, ImagePlus img) {
    this.img = img;
    return DOES_ALL+NO_CHANGES;
}
```

In the `run` method we get the `ImageWindow` that displays the image and the canvas on which it is drawn. We want the plugin to be notified when the user clicks on the canvas so we add the plugin to the canvas' `MouseListeners`.

```
public void run(ImageProcessor ip) {
    ImageWindow win = img.getWindow();
    canvas = win.getCanvas();
    canvas.addMouseListener(this);
}
```

To implement the interface we have to implement the five methods it declares. We only want to react on clicks so we can leave the others empty. We get the coordinates of the point of the mouse click from the event object that is passed to the method. The image could be scaled in the window so we use the `offScreenX()` and `offScreenY()` method of `ImageCanvas` to receive the true coordinates.

```
public void mouseClicked(MouseEvent e) {
    int x = e.getX();
    int y = e.getY();
    int offscreenX = canvas.offScreenX(x);
    int offscreenY = canvas.offScreenY(y);
    IJ.write("mousePressed: "+offscreenX+", "+offscreenY);
```

```
    }  
  
    public void mousePressed(MouseEvent e) {}  
  
    public void mouseReleased(MouseEvent e) {}  
  
    public void mouseEntered(MouseEvent e) {}  
  
    public void mouseExited(MouseEvent e) {}
```

A more advanced mouse listener (which avoids assigning the listener to the same image twice) and a similar example that reacts on keyboard events can be found at the ImageJ plugins page (cf. Section 10).

Like mouse and key listeners a plugin can implement any event listener, e.g. a mouse motion listener. For adding a mouse motion listener, the following changes of the mouse listener plugin are necessary:

The class has to implement the event listener interface:

```
    public class Mouse_Listener implements PlugInFilter, MouseListener,  
                                           MouseMotionListener {
```

In the `setup` method, we add the plugin as listener to the image canvas.

```
        canvas.addMouseMotionListener(this);
```

Of course we have to implement the methods defined in the interface:

```
    public void mouseDragged(MouseEvent e) {  
        IJ.write("mouse dragged: "+e.getX()+" "+e.getY());  
    }  
  
    public void mouseMoved(MouseEvent e) {  
        IJ.write("mouse moved: "+e.getX()+" "+e.getY());  
    }  
    %
```

For details about listener interfaces, their methods and the events passed please see the Java AWT documentation.

7 Advanced Topics

7.1 Data Import/Export

7.1.1 Movies

Because of its capability to handle image stacks ImageJ can be used to process movies. Import and export from and to common movie formats is possible.

QuickTime import and export plugins are available on the ImageJ plugins page. They are based on Apple's QuickTime for Java library. To use these plugins, QuickTime for Java has to be installed (which should be the case on Mac OS 9.04 or higher and Mac OS X). On older Macs and under Windows QuickTime (available at <http://www.apple.com/quicktime/>

download) has to be installed performing a custom installation and selecting QuickTime for Java which will install the file `QTJava.zip`.

Under Windows `QTJava.zip` has to be included in the classpath (`-cp` option) of the Java virtual machine used for running ImageJ.

For AVI files, a writer plugin and a reader plugin for uncompressed AVIs are available. It does not require any further libraries.

ImageJ can also handle image sequences, i. e. numbered files in the same directory having any type that is supported by ImageJ. They can be loaded using the “File / Import / Image Sequence” command. The File Opener plugin also allows to load a number of files listed in a text file.

For all movie plugins it is recommended to increase the amount of available memory of the JVM used for running ImageJ (cf. Section 8).

7.2 Using the ImageJ Library outside ImageJ

The ImageJ classes form an image processing library which can be used in other Java applications and applets and also on the server side in servlets or Java Server Pages (JSP). The following section outlines the use of the ImageJ library in such projects.

7.2.1 Why use the ImageJ library in your Java project?

Java 1.1.

Java 2 introduced many improvements concerning image processing with Java. But most users still use browsers that only have a Java 1.1 virtual machine and only a minority uses Sun’s Java plugin. Java 2 support may also be not available on less widespread platforms. ImageJ is based on Java 1.1 and is therefore a good choice especially for applets.

ImagePlus as internal image format.

You will probably need an internal image representation format for your application. It is convenient to use `ImagePlus` and `ImageProcessor` for this purpose, as a lot of basic functionality (reading/writing pixel values, scaling, etc.) is already available.

Plugins.

If you decide to use `ImagePlus` as your internal image format you can also use all plugins from the ImageJ distribution as well as all other ImageJ plugins.

File I/O.

You can use the ImageJ file input/output plugins for reading and writing files in a variety of formats.

Of course there are some other useful Java based imaging toolkits and libraries besides ImageJ. The ImageJ links page at <http://rsb.info.nih.gov/ij/links.html> lists some of them.

7.2.2 Applications and Applets

To use the ImageJ library in your Java application, just import the necessary ImageJ packages (e. g. `ij.process`) in your classes. To compile and run your application you have to add `ij.jar` (if it’s not in the application directory you also have to specify the path) to the classpath. In an application, you could also use dynamic class loading as it is implemented in ImageJ for accessing user plugins.

Using the ImageJ library in an applet is quite similar: include the import statement in your classes and add `ij.jar` to the classpath for compiling the applet. The Java security model requires all libraries used by an unsigned applet to be located on the same host and you have to include `ij.jar` in the archive list of the applet.

Assume your applet's code is located in `myapplet.jar`, the applet class is `MyApplet.class`, it uses the ImageJ library and both JAR files are located in the same directory as the HTML file that embeds the applet. The applet tag in the HTML page would look like

```
<APPLET CODE="MyApplet.class" ARCHIVE="myapplet.jar, ij.jar"
        WIDTH=400 HEIGHT=400>
</APPLET>
```

7.2.3 Servlets and JSP

Web applications often require modifying or generating images on the fly, e.g. stock charts. In Java based server side solutions the ImageJ library can be used for image processing very easily. If you are using servlets, just add the appropriate `import` statement for an ImageJ package to your servlet and include `ij.jar` in the classpath. In JSP applications, ImageJ can either be used "behind the curtain" inside Java Beans (where you just import it as in any other kind of Java class) or directly in a JSP page using e.g.

```
<%\@ page import="ij.process.*" \%>
```

As display format you often have only the choice between JPEG, GIF or PNG in web applications. The output will not be written to a file but to the response stream of the servlet/JSP. You could use ImageJ's file encoders (an example can be found at <http://rsb.info.nih.gov/ij/applets.html>) for this purpose or use e.g. Sun's JPEG encoder. The following servlet sample code illustrates how to load a file (in any format that can be read by ImageJ) and send it as a JPEG stream to the user's browser. The name of the image will be specified as parameter `image` of a GET request. A call of the servlet could look like

```
http://www.myserver.com/servlet/ShowImage?image=/images/picture.tif
```

We assume in this example that the image loaded is a RGB color image. Here is the complete code (requires Java 2):

First we import the servlet packages, the AWT image subpackage, the required ImageJ packages and the Sun JPEG encoder:

```
import javax.servlet.*;
import javax.servlet.http.*;
import java.io.*;
import java.awt.image.*;
import ij.*;
import ij.io.*;
import ij.process.*;
import com.sun.image.codec.jpeg.*;

public class ShowImage extends HttpServlet
```

We implement the method for handling a GET request, which gets the HTTP request and response as parameters. First we read the parameter containing the image URL from the request and open the ImageJ using ImageJ's Opener. As we assumed to open a color image, we can now get its ColorProcessor, create a new BufferedImage and store the image's pixel array in it.

```
public void doGet (HttpServletRequest request,
                  HttpServletResponse response)
    throws ServletException, IOException {

    String inputURL=request.getParameter("image");
    Opener opener = new Opener();
    ImagePlus image = opener.openURL(inputURL);

    ColorProcessor cp = (ColorProcessor) image.getProcessor();
    int[] pixels = (int[]) cp.getPixels();
    BufferedImage bimg = new BufferedImage(cp.getWidth(),
                                         cp.getHeight(), BufferedImage.TYPE_INT_RGB);
    bimg.setRGB(0,0,cp.getWidth(),cp.getHeight(),
               pixels,0,cp.getWidth());
}
```

As we want to return a JPEG image, we set the appropriate MIME type for the HTTP response. We get the response's binary output stream and open a JPEG encoder on it. To get best quality, we disable subsampling and set the JPEG quality parameters to the maximum. Finally we encode the image with the specified parameters and clean up by flushing and closing the output stream.

```
    response.setContentType("image/jpeg");

    OutputStream outstr = response.getOutputStream();
    JPEGImageEncoder jie = JPEGCodec.createJPEGEncoder(outstr);

    JPEGEncodeParam jep = jie.getDefaultJPEGEncodeParam(bimg);
    jep.setQuality(1.0f,false);
    jep.setHorizontalSubsampling(0,1);
    jep.setHorizontalSubsampling(1,1);
    jep.setHorizontalSubsampling(2,1);
    jep.setVerticalSubsampling(0,1);
    jep.setVerticalSubsampling(1,1);
    jep.setVerticalSubsampling(2,1);
    jie.encode(bimg,jep);
    outstr.flush();
    outstr.close();
}
}
```

A big advantage of JSP is the separation of implementation (which can be wrapped into Java Beans) and the page layout. A sample that shows how a JSP based image processing

system could look like, can be found at the original location of this tutorial (see title page). It consists of a Java Bean that wraps the whole ImageJ functionality and also supports dynamic loading of user plugins (although plugins must not require user input except for the argument string).

8 Troubleshooting

8.1 ImageJ runs out of memory.

This can be solved by making more memory available to the Java Runtime Environment. As virtual memory is significantly slower than real RAM, you should try to assign not more than 2/3 of your real RAM to the Java virtual machine ³.

Windows

To increase the amount of memory available to ImageJ, edit the `-mx` option in the “Target:” field of the of the “Shortcut” properties of the ImageJ shortcut. The default is 80MB. For example, to increase the available memory from 80MB to 340MB, change the “Target” line to

```
C:\ImageJ\jre\bin\jrew.exe -mx340m -classpath ij.jar;tools11.jar ij.ImageJ
```

MacOS

Java applications allocate memory from the System heap so there is usually no need to increase the value of “Preferred Size” in ImageJ’s “Get Info” dialog. Strangely enough, allocating more memory to ImageJ reduces the amount of memory available for loading images! It may, however, be necessary to allocate more memory to ImageJ to avoid error messages with plugins that use QuickTime for Java. The Finder’s “About this Computer” window is a good way to monitor ImageJ’s memory usage.

Mac OS X

To make more memory available to ImageJ, edit the line

```
com.apple.mrj.application.vm.options=-Xmx128M
```

in the `ImageJ.app/Contents/Resources/MRJApp.properties` file, where `ImageJ.app` is the directory (bundle) that the Finder displays as the ImageJ application. You can view the files and folders in the ImageJ bundle by control-clicking on the ImageJ application and selecting “Show Package Contents”. Another way to make more memory available to ImageJ is by running from the command line and using the `-mx` option.

Linux, Unix, other OS

You can set the amount of memory available for the Java runtime environment by using the `mx` switch followed by the amount of memory. For example, to make 256MB available for the JVM, call it using:

```
./jre/bin/jre -mx256m -cp ij.jar:tools.jar ij.ImageJ
```

If you want to use this setting all the time, make the modification in the run script.

³The operating system specific descriptions listed here are taken from <http://rsb.info.nih.gov/ij/install>.

8.2 A plugin is not displayed in ImageJ's plugins menu.

This may have several reasons:

- The plugin name does not contain an underscore.
- The plugin may not be in the plugin directory or one of its subdirectories.
- If you did not compile the plugin inside ImageJ, make sure that the compilation was successful and a class file has been created.

8.3 When you call the “Plugins/Compile and Run ...” menu, you get the message: “This JVM appears not to include the javac compiler. [...]”

If you are using Mac OS, you need the MRJ SDK in addition to the MRJ (Macintosh Runtime for Java). Consult Section 1.3.2 for further reference.

If you experience this problem when using the Windows or Linux distribution including a Java compiler, make sure

- that the tools library (`tools11.jar` if you are using JRE/JDK 1.1, `tools.jar` if you are using Java 2 (JRE/JDK 1.2 or higher)) is in the classpath, and
- that you are using the right Java environment if you have more than one installed. Specify the path to the Java Virtual Machine you want to use explicitly.

8.4 ImageJ throws an exception when you use it in a web application running on a Unix/Linux server.

Many ImageJ classes are based on Java AWT. But the initialization of AWT will fail—usually during initializing the default toolkit, which is required for loading images via AWT—on “headless” servers, i. e. servers with no Xserver installed. With JDK 1.4 or later, you can use the headless option

```
java -cp ij.jar;. -Djava.awt.headless=true ij.ImageJ
```

9 Frequently Asked Questions

9.1 How to change the URL for the sample images (menu “File/Open Samples”) in order to access local copies of the files?

The URL is set using the `images.location` value in the file `IJ_Props.txt` which is located in `ij.jar`. The URL must include a trailing `/`.

You can edit `ij.jar` with a program that reads ZIP files. Some of them (e.g. WinZip) support editing a file from the archive directly and will update the archive after closing the modified file. Otherwise it is necessary to extract `IJ_Props.txt` from the archive and add it again after editing.

Example:

```
images.location=http://www.mymirror.com/ij/images/
```

9.2 How to include user plugins when running ImageJ as applet?

When running ImageJ as an unsigned applet, the class loader that loads user plugins will not work as the plugins folder is not in the code base. Add a package statement such as `package ij.plugins` to the plugin code and insert the compiled class into `ij.jar` (be sure to include it into the right folder). To make the plugin appear in the plugins menu add a line like

```
plug-in08="Plugin",ij.plugin.Plugin_
```

to the plugins section in `IJ_Props.txt`, which is also located in `ij.jar`.

You can edit `ij.jar` with a program that reads ZIP files. Some of them (e.g. WinZip) support editing a file from the archive directly and will update the archive after closing the modified file. Otherwise it is necessary to extract `IJ_Props.txt` from the archive and add it again after editing.

10 Further Resources

10.1 API Documentation, Source Code

The ImageJ API documentation is available online at <http://rsb.info.nih.gov/ij/docs/api/index.html>.

API documentation and source code are available for download at <http://rsb.info.nih.gov/ij/download.html>.

10.2 Plugins Page

Many ImageJ plugins (with source code) are available at <http://rsb.info.nih.gov/ij/plugins/index.html>.

10.3 ImageJ Mailing List

For questions concerning ImageJ that are not answered by the documentation consult the ImageJ mailing list.

A complete archive can be found at <http://list.nih.gov/archives/imagej.html>.

For information about subscribing see <http://rsb.info.nih.gov/ij/list.html>.

10.4 Java Resources

10.4.1 Online Resources

Java API documentation and many tutorials are available from Sun Microsystems at <http://java.sun.com/> under “Docs & Training”. Other online Java resources are:

- A comprehensive collection of Java resources (books, tutorials, FAQs, tools): <http://www.apl.jhu.edu/~hall/java>.
- Exploring Java: <http://www.ooi.com/exploringjava/>
- Java Programmer’s FAQ: <http://www.afu.com/>
- O’Reilly Java Center: <http://java.oreilly.com/>

Version 1.6

- JavaWorld: <http://www.javaworld.com/>
- DocJava: <http://www.DocJava.com/>
- Java Developers Journal: <http://www.sys-con.com/java/>
- JARS: http://www.jars.com/jars_resources_java.html

10.4.2 Books

Java in a Nutshell: A Desktop Quick Reference (Java Series)

by David Flanagan
648 pages, 3rd edition (November 1999)
O'Reilly & Associates
ISBN: 1565924878

Java Examples in a Nutshell

by David Flanagan
500 pages, 2nd edition (September 2000)
O'Reilly & Associates
ISBN: 0596000391

The first third of this book is interesting for someone who wants to get into Java programming, the other chapters cover more advanced topics.

The Sun Java Series

Detailed information can be found at <http://java.sun.com/docs/books/>.