

Alexey Podlasov

apodla@cs.joensuu.fi

Eugene Ageenko

ageenko@cs.joensuu.fi

Working and Development with ImageJ

A student reference

Department of Computer Science.

University of Joensuu.

2003

Introduction

1.1 *About this paper*

Although ImageJ is a well-known Java image processing program and a big variety of literature devoted to is available in Internet, this paper was considered as a kind of compilation where the mostly common aspects of using and development with ImageJ would be gathered together from different publication. Also, this paper was supposed to be a kind of tutorial to help students to become familiar with ImageJ environment – as a user and as a programmer – and let them concentrate on the image processing topics instead of technical details or retrieving information. The most of material in Chapters 2 and 3 was taken from ImageJ official site <http://rsb.info.nih.gov/ij/> from [1] and [2]. At the same time a practical experience of development with ImageJ was used as well.

1.2 *About ImageJ*

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 in TIFF, GIF, JPEG, BMP, DICOM, FITS or "raw" formats. 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.
- 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.

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.

1.3 *Brief description of the following chapters*

Chapter 2 is devoted to getting familiar with ImageJ GUI, tools and other features. Chapter 3 is devoted to development with ImageJ. In subsection 1 we describe its class structure and functionality. Subsections from 2 to 4 are devoted to plugins in common, the concept of image in ImageJ, utility methods provided by ImageJ and to plugin parameterization via building ImageJ GUI.

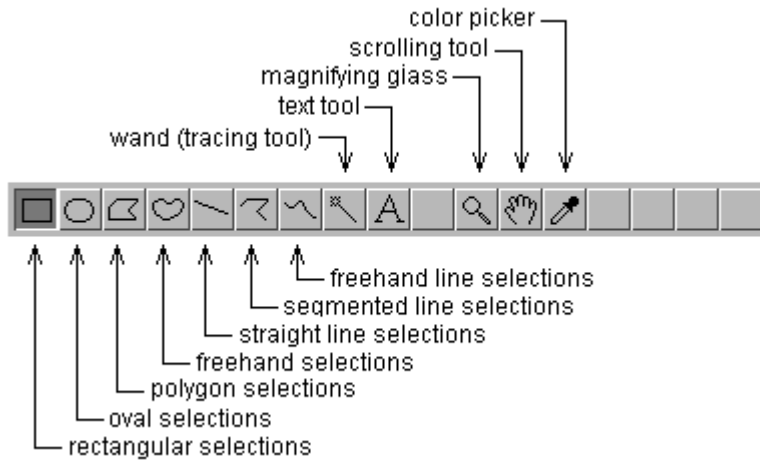
Using ImageJ – A Brief User Guide

1.4 *Basic concepts*

The "ImageJ" window contains a menu bar, tool bar, status bar, and a progress bar. Images, histograms, line profile, etc. are displayed in additional windows. Measurement results are displayed in the "Results" window. Histograms and plots are ordinary image windows that can be copied (to the internal clipboard), edited, printed and saved.

Tool Bar

The tool bar contains tools for making selections, for zooming and scrolling images, and for changing the drawing color. Click on a tool and a description of that tool is displayed in the status bar.



Status Bar



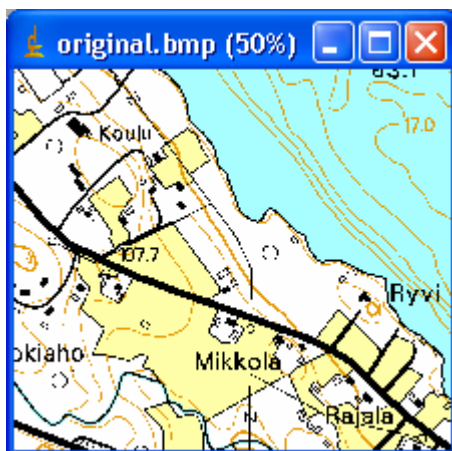
The status bar, when the cursor is over an image, displays pixel coordinates and values. After running a filter, it displays the elapsed time and processing rate in pixels/second.

Progress Bar



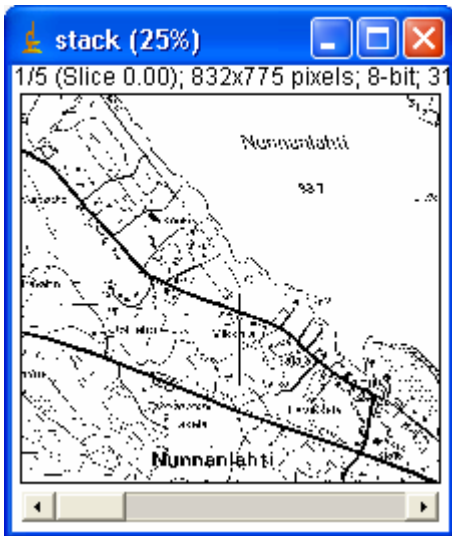
The progress bar, located to the right of the status bar, shows the progress of time-consuming operations. It will not appear if the operation requires less than approximately one second.

Images



ImageJ allows multiple images to be displayed on the screen at one time. The active window has its title bar highlighted. All operations will be performed on the active image. ImageJ supports 8-bit, 16-bit and 32-bit (real) grayscale images and 8-bit and 32-bit color images.

Stacks



ImageJ can display multiple spatially or temporally related images in a single window. These image sets are called stacks. The images that make up a stack are called slices. All the slices in a stack must be the same size and bit depth. A scroll bar provides the ability to move through the slices.

Selections



Selections are user defined areas or lines within an image. Only one selection can be active at a time. Area selections are created using the rectangular, oval, polygonal and freehand selection tools.

File Formats

The *File/Open* command opens TIFF (uncompressed), GIF, JPEG, DICOM, BMP and FITS images. It also opens lookup tables and selections (*ROIs*).

Plugins

Plugins are loadable code modules that extend the capabilities of ImageJ. Using plugins you can implement any needed algorithm concentrating on implementation of how actually image has to be processed. All background work when reading, displaying, magnifying, applying ROI, etc. will be handled by ImageJ. Plugins can be developed using any Java IDE but the easiest way to create a plugin is to open the command recorder, record a series of commands, and then click *Create Plugin*. You can use such plugin as a basis for further development for example using *Plugins/Edit* menu command.

1.5 Installation and update guide

For running ImageJ you need ImageJ class and configuration files, a Java Runtime Environment (*JRE*) and 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.

1.5.1 Installing ImageJ

The latest distribution of ImageJ can be downloaded from <http://rsb.info.nih.gov/ij/download.html>. 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 and unpack installation archive to a dedicated folder (e.g. `c:\ImageJ`)
- Run ImageJ using one of provided shortcuts or via command line:
`javaw.exe -classpath ij.jar;jimi.jar; ij.ImageJ`

1.5.2 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.

Development with ImageJ – A Brief Developer Guide

1.6 *ImageJ class structure*

This is an overview of the class structure in ImageJ. Only basic and most important for getting started classes are listed and briefly described. For complete description reader should refer to [3].

1.6.1 Class structure

`ij`

`Executer`

Runs menu commands in a separate thread. Useful for using already existing plugins inside the new one.

`IJ`

A Class containing a lot of useful utility methods.

`ImagePlus`

One of the most important classes in ImageJ representing an image itself. Based on `ImageProcessor`.

`ImageStack`

Represents the stack of images (see 1.4).

`ij.gui`

Contains classes for implementing GUI elements and different types of ROI. Classes of our interest are:

`GenericDialog`

This class implements customizable modal dialog box which also used for transferring parameters from macros into plugins.

`NewImage`

New image dialog box plus several useful static utility methods for creating new images.

`ij.plugin`

Most of ImageJ commands are implemented as plugins and could be found in that package or one of its subpackages.

`PlugIn`

This interface has to be implemented by plugins that require more complicated input than just an input image.

`ij.plugin.filter`

`PlugInFilter`

This interface has to be implemented by plugins that require just an input image.

`ij.plugin.frame`

`PlugInFrame`

This is a closable window that plugins can extend.

`ij.process`

This package contains image processors classes which are the main tool to operate with images when developing with ImageJ.

`ImageProcessor`

One of the most important classes for developing with ImageJ. Contains a big variety of standard operations. All custom operations with image should be implemented using this class.

`StackProcessor`

This class process stacks.

1.7 Plugin concept in ImageJ

Functions provided by ImageJ could be extended by user plugins. In fact, the most of ImageJ standard functions are implemented as plugins. User plugin is a Java class implementing necessary interface and placed into certain folder. Plugins could be written in ImageJ Java editor or in any existing Java environment, recorded with ImageJ macro recorder or written on ImageJ internal macro language. Plugins found by ImageJ are placed into *Plugins* menu or submenus of it.

1.7.1 Types of plugins

There are two basic types of plugins in ImageJ. They are ones which implement *PlugIn* interface and ones which implement *PlugInFilter* interface. The difference is that *PlugIn* interface receives only a string with parameters as its input and programmer has to implement access to desired image by himself. On the other hand plugins implementing *PlugInFilter* interface do not care about that and receive access to currently active image directly as parameter.

PlugIn

Interface has only one method:

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

This method runs the plugin and what you implement here is what plugin actually does. `arg` is a string argument passed to plugin.

PlugInFilter

When implementing `PlugInFilter` `setup` method is called first

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

This method sets up the plugin for use. This method returns a flag word that represents plugins capabilities. The combination of the following flags could be returned by `setup` could be seen on a table below. Actual functionality of the plugin implemented by method

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

This method runs the plugin and what you implement here is what plugin actually does. It has currently active image as its argument. `ImageProcessor` provides a lot of already developed operations over the image. It also provides direct access to image pixels to implement any custom functionality.

Flags returned by setup:

static int	DOES_16 Set this flag if the filter handles 16 bit grayscale images.
static int	DOES_32 Set this flag if the filter handles 32 bit floating point grayscale images.
static int	DOES_8C Set this flag if the filter handles 8-bit color images.
static int	DOES_8G Set this flag if the filter handles 8-bit grayscale images.
static int	DOES_ALL Set this flag if the filter handles all types of images.
static int	DOES_RGB Set this flag if the filter handles RGB images.
static int	DOES_STACKS Set this flag if the filter wants its <code>run()</code> method to be called for all the slices in a stack.
static int	DONE Set this flag and <code>run</code> method will not be called.
static int	NO_CHANGES Set this flag if the filter makes no changes to the pixel data.
static int	NO_IMAGE_REQUIRED Set this flag if the filter does not require that an image to be open.
static int	NO_UNDO Set this flag if the filter does not require undo.
static int	ROI_REQUIRED Set this flag if the filter requires an ROI.
static int	STACK_REQUIRED Set this flag if the filter requires a stack.
static int	SUPPORTS_MASKING Set this flag if the filter wants ImageJ, for non-rectangular ROIs, to restore that part of the image that's inside the bounding rectangle but outside of the ROI.

1.7.2 Installing the plugins

You can install plugin following way:

- Put plugin `.class` file into a “plugins” folder which is a subfolder of ImageJ folder. Plugins with underscore in its name (i.e. “`Inverter.class`”) will appear in Plugins menu automatically when you restart ImageJ.
- Other plugins have to be added to menu using “*Plugins/Shortcut/Install plugin...*” menu command.
- If plugin files are located in on of the subfolder of “plugins” folder they will be located in submenus with corresponding names

1.7.3 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.

1.7.4 Compiling and running plugins

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 standard ones, they have to be added into the “Target:” field of the of the “Shortcut” properties of the ImageJ shortcut as follow:

```
javaw.exe mx80m -classpath ij.jar;tools11.jar;mylib.jar ij.ImageJ
```

1.7.5 Integrating into IJ GUI

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

1.8 Image in ImageJ

In ImageJ images are represented by `ImagePlus` and `ImageProcessor` objects. In the following section we’ll take a closer look at the way images are handled.

1.8.1 Types of images

ImageJ supports following types of images:

- **8 bit grayscale image**, 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**, a pixel is represented by a `short` variable.
- **RGB color image**, a pixel is represented by an `int` variable.
- **32 bit image floating point grayscale image**, a pixel is represented by a `float` variable.

1.8.2 ImagePlus class

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`:

- `ImagePlus.COLOR_256` **8 bit color image with a look-up table.**
- `ImagePlus.COLOR_RGB` **RGB color image.**
- `ImagePlus.GRAY16` **16 bit grayscale image.**
- `ImagePlus.GRAY32` **A 32 bit floating point grayscale image.**
- `ImagePlus.GRAY8` **A 8 bit grayscale image.**

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

There are different ways to construct an `ImagePlus` object:

- **Using one of the available constructors:**

```
ImagePlus image = new ImagePlus();
```

Default constructor, creates a new empty `ImagePlus`

```
ImageProcessor ip = new ImageProcessor(width, height);
```

```
ImagePlus image = new ImagePlus(title, ip);
```

Constructs a new `ImagePlus` based on given `ImageProcessor ip`.

```
ImageStack stack = new ImageStack(width, height)
```

```
ImagePlus image = new ImagePlus(title, stack)
```

Constructs a new `ImagePlus` based on given `ImageStack stack`.

- **Using `ImagePlus.createImagePlus()`**

```
ImagePlus givenImage = ...
```

```
ImagePlus image = givenImage.createImagePlus();
```

To create a new image with this image properties (e.g. dimension and color model)

- **The class `NewImage` offers some useful static methods for creating new images of a certain type. For these methods the title, width, height, number of slices and filling are specified as parameters.**

`NewImage.createByteImage` - creates a new 8 bit grayscale or color;

`NewImage.createFloatImage` - creates a new 32 bit floating point image.

`NewImage.createRGBImage` - Creates a new RGB image.

`NewImage.createShortImage` - a new 16 bit grayscale

The type of an `ImagePlus` can be retrieved using

```
int getType()
```

Similar methods exist for getting such image properties like dimension, title, 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()
```

1.8.3 ImageProcessor class

Each image is based on the 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 one of its subclasses. There are:

- ByteProcessor Used for 8 bit grayscale and color images.
- ShortProcessor Used for 16 bit grayscale images.
- ColorProcessor Used for 32 bit RGB images.
- FloatProcessor Used for 32 bit floating point images.

1.8.4 Accessing pixel values

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. Notice that we receive a one-dimensional array. It contains the image pixel values scanline by scanline. To convert a position in this array to a (x,y) coordinate in an image, we need to know image's dimensions. The width and height of an ImageProcessor can be retrieved using:

```
int getHeight()
```

```
int getWidth()
```

Reading pixels from ByteProcessor, ShortProcessor and from ColorProcessor need more explanation.

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. Again, we can be sure that sign bit is eliminated by using 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 stored in 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. Therefore any modifications affect the `ImageProcessor` immediately. However, you can set any appropriate array to be `ImageProcessor`'s pixel array using

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

You do not always have to retrieve or set the whole pixel array. `ImageProcessor` offers alternative methods for accessing 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.

NOTE: developer should avoid methods described above when modifying large parts of the image because of its slow speed. It is much faster to work with the pixel array.

1.8.5 Regions of interest

A plugin filter does not always have to work on the whole image. `ImageJ` supports regions of interest (or selections). The bounding rectangle of the current ROI can be retrieved from the `ImageProcessor` using

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

This makes possible just to 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)
```

More methods for working with ROIs can be found in `ImagePlus`:

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

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

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

Creates a rectangular selection.

```
void ImagePlus.setRoi(Roi roi)
```

Creates a selection based on the specified ROI object.

```
Roi ImagePlus.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`

1.8.6 Displaying images

`ImageJ` uses a class called `ImageWindow` to display `ImagePlus` images. `ImagePlus` contains everything that is necessary for updating or showing newly created images. Typically you will need just to show a newly created image using `void show()` or update modified using `void updateAndDraw()`.

1.8.7 Sample plugin

With the knowledge of previous sections we can write our own plugin. Let's modify standard `Inverter_` plugin to demonstrate already known features: Getting an image, accessing pixel values, using ROI and creating and showing new resulting image.

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

Our plugin must have an underscore in its name. It needs an image as its input, so it has to implement `PlugInFilter`.

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

First we need to set up a plugin using `setup` method. The return value tells the `ImageJ` that our plugin will handle 8 bit grayscale images and stacks with masking. If one would try to apply that plugin to another image, `ImageJ` will handle it and responds about error automatically. If string argument is "about" then plugin shows about box and do nothing by returning `DONE`.

```
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 as input parameter and retrieve image's dimensions, pixel values and ROI.

```

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

```

Then we create destination image processor to represent resulting image. By copying bit we construct that part of the image that will not be modified, then we get pixel array to modify area which is in ROI.

```

    ByteProcessor dip = new ByteProcessor(width,height);
    dip.copyBits(ip,0,0,Blitter.COPY);
    byte[] dstpixels = (byte[])dip.getPixels();

```

To process pixels inside ROI we implement 2 nested loops to walk through rectangle row by row and assign modified values to destination image processor .

```

    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;
            dstpixels[i] = (byte)(255-pixels[i]);
        }
    }

```

Finally, we create and show a window containing resulting image. Notice that `ImagePlus` object is created on the basis of destination image processor .

```

        new ImagePlus("Inverted",dip).show();
    }

    void showAbout() {
        IJ.showMessage("About Inverter_...",
            "Long live my genius plugin!!!");
    }
}

```

1.8.8 Stacks

ImageJ supports expandable arrays of images called image stacks, which 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()
```

The number of slices of a stack can be retrieved using

```
int getSize()
```

Pixels of each slice in a stack could be accessed using

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

or via ImageProcessor class

```
ImageProcessor getProcessor(int n)
```

To determine and change currently displayed slice use

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

You can create new stack to use it as image's stack. Create stack using ImagePlus method

```
ImageStack createEmptyStack()
```

or one of ImageStack constructors

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

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

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

There are also useful methods when working with stacks:

```
void addSlice(java.lang.String sliceLabel, ImageProcessor ip)
void addSlice(java.lang.String sliceLabel, java.lang.Object pixels)
void deleteLastSlice()
void deleteSlice(int n)
int getHeight()
int getWidth()
```

1.8.9 Additional references

For additional reference reader may refer to [3]

1.9 Utility methods

The ImageJ API contains a class called `IJ` that contains some very useful static methods.

1.9.1 Simple 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)
```

in the second case

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

You can also specify the title of the message box using

```
static void showMessage(title, msg)
```

More flexible method is

```
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()  
static void outOfMemory(java.lang.String name)  
static boolean versionLessThan(java.lang.String version)
```

1.9.2 Using main window, status and progress bars.

To display a line of text in the results window use

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

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

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

Converts a number to a formatted string using specified precision.

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)
```

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).
```

1.9.3 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.

```
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.

1.9.4 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.

1.10 Building GUI

Sometimes you will need a more complicated user input than just an input image. In that case you have to build GUI for your plugin to interact with the user. Typically you will need to create a modal dialog box to ask for additional parameters. Use `PlugInFrame` and `GenericDialog` classes for building GUI. That classes provide simple way for retrieving parameters from the user. Moreover, using those classes it is possible to set parameters in a parameter string when running plugin from a command line or macro. In that case parameters will be transferred automatically to your plugin by ImageJ and modal dialog will not appear.

1.10.1 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 should 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.

1.10.2 Generic Dialog

The `GenericDialog` class is used to build a modal AWT dialog. It 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()
```

1.10.3 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.

```
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 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.

1.10.4 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. If plugin was started from the command line, and one or all parameters were specified in a parameter string, then modal dialog will not appear and values of the parameters will be transferred automatically to retrieving methods (methods below).

```
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.

Appendices

1.11 References

- 1) IJ documentation. <http://rsb.info.nih.gov/ij/docs/index.html>
- 2) Werner Bailer. Writing IJ Plugins – A Tutorial.
<http://mtd.fh-hagenberg.at/depot/imaging/imagej/ijtutorial.pdf>
- 3) IJ API. <http://rsb.info.nih.gov/ij/developer/api/index.html>