Quick Guide to Writing Plugins in ImageJ

Jay Unruh

Why create plugins in ImageJ?

The key here is flexibility. If you understand the mathematical models, you can do anything to your images you want. You can also set up automation that other software packages do not allow. In addition, the input/output capabilities of ImageJ are extensive allowing operation on almost all image file types. Finally, Java is a rather efficient programming language especially compared to interpreted languages such as visual basic, Matlab, and IDL.

Logistics

The easiest way to create plugins in ImageJ quickly is to use the built in plugin editor: Plugins—New. There are different types of plugins (macro, plugin filter,ect), but we will just cover the basic plugin. After writing the plugin, you can save it to the plugins folder and compile and run it. This only works if you have installed ImageJ bundled with the jre (see ImageJ website).

Introduction to Java and ImageJ

This introduction will be brief. Hopefully the concepts will make sense when they are used in practice. For more details, see the Sun Microsystems Java tutorial.

Java is an object oriented language. Objects are declared in classes or subclasses of another class. For example, in ImageJ, a ColorProcessor is a subclass of an ImageProcessor. Classes contain constructors that create the object and methods that perform functions as well as member variables. In addition, you have interfaces which define a certain type of behavior for a class. Each class that implements that interface must contain the same methods as the interface. For example, if you want a plugin to respond to mouse clicks, you can implement the interface MouseListener. Your plugin must then contain the method mouseClicked(MouseEvent) which defines the action to be taken when the mouse is clicked. All ImageJ plugins implement the plugin interface which has the method run(String). This method runs the plugin as though it were part of the ImageJ code.

Construction of an object must be defined in the object's class file. The constructor looks just like a method, but it has the same name as the object (class and object can be used interchangeably). There can be multiple constructors for different default behaviors, but they must have different parameter lists. You can access the ImageJ source code through the web from Help—source code. Look in the package ij for the class ImagePlus. You will see multiple "public ImagePlus(…" sections that are constructors for the ImagePlus object. ImagePlus is ImageJ's overall object for an image or stack of images. For example, if you want to open a file and make an ImagePlus out of its image, you would use the following code:

ImagePlus imp=new ImagePlus(“filename”);

imp is now an ImagePlus object that contains the data for your file. Note that all java commands must be terminated with a semi-colon.

An object also contains methods. These are essentially functions to do things to the object or get information about it. You can look and find out how wide your image is in pixels by the following:

int width=imp.getWidth();

Here we stored the width of the image in the width variable. If you look at the ImagePlus source code, you will find the method “public int getWidth()”. The public means that it is available for you to use. The int means that it returns an integer value. If you go towards the top of the ImagePlus class, you will find “protected int width”. This is where the width variable is stored. You don’t have access to it because it is protected and the creators of ImageJ don’t want you to change it on a whim without making the appropriate changes to the image that need to accompany that change. Instead you have to access it through the getWidth() method.

The width is an example of a member variable for an object. The width member variable happened to be protected, but if it was not, you could access it as follows:

int width=imp.width;

You might have noticed all of the curly brackets around things ({}). These brackets surround different pieces of code and are mostly optional. Nevertheless, they are useful for finding the beginning and end of sections of code and therefore recommended for enclosing classes, constructors, methods, and other constructs.

Variables and arrays are just objects like anything else and have their own special constructors. For a floating point number, we have:

float temp=3.0f;

This constructs the floating point object, temp, and also initializes it. For an array we would have declared a float[] object:

float[] temp={3.0f,4.0f,5.0f};

If we just wanted an array filled with zeros of a specified size, we would have used:

float[] temp=new float[10];

Here 10 is the size of the array. Note that you can only declare arrays this way. Number variables (e.g. int, float) cannot be declared with the new … methodology.

There are two basic control structures in java. The first is the for loop for looping through arrays:

```
for(int i=0;i<10;i++){
        temp[i]=(float)i;
}
```

Here we have filled the temp array with the floating point version of i. There are several things to note here. First, we declared the integer variable, i, within the loop. Note that i does not exist outside this loop. Alternatively we could have declared it before hand:

```
int i;
for(i=0;i<10;i++){
        temp[i]=(float)i;
}
```

Here i continues to be defined after our loop. This loop starts with i=0 and ends with i=9. All java arrays are indexed starting at zero and end at length-1. You can find the length of an array by:

int length=temp.length; (note that length is a member variable of the float[] class)

Also note the i++ that we have used. This increments an integer upwards by 1. We could have also used i+=1 to accomplish this same goal. If you wanted to set every other value in temp to its index you could use the following:

```
for(int i=0;i<10;i+=2){
        temp[i]=(float)i;
}
```

Finally note that we couldn't just set temp[i] equal to i. This is because i is an integer and temp[i] is a float value. Therefore we have to type-cast the integer into a float using (float).

A second widely used control structure is the if-then-else loop. We could use this to threshold the temp array into binary.

```
float threshval=5.0f;
for(int i=0;i<10;i++){
        if(temp[i]>=threshval){
                temp[i]=1.0f;
        }
        else{
```

```
                temp[i]=0.0f;
        }
}
```

Note that >= means greater than or equal to.

Example

Lets pick and example of an operation on an image. I want to take an image and set its maximum and minimum values and then spit it back out as a new image.

First create a new plugin entitled set_min_max.java. Note that you must include an underscore in the plugin name in order for it to be automatically incorporated into the plugins list when you restart ImageJ. The compiled version will be called set_min_max.class. The plugin is as follows:

```
ImagePlus imp=WindowManager.getCurrentImage();
int width=imp.getWidth();
int height=imp.getHeight();
ImageProcessor ip=imp.getProcessor();
float[] pixels=(float[])ip.getPixels();
float max=10.0f;
float min=0.0f;
float[] newpixels=new float[width*height];
System.arraycopy(pixels,0,newpixels,0,width*height);
for(int i=0;i<width*height;i++){
        if(newpixels[i]>max){
                newpixels[i]=max;
        }
        else{
                if(newpixels[i]<min){
                        newpixels[i]=min;
                }
        }
}
ImageProcessor ip2=new FloatProcessor(width,height,newpixels,null);
ImagePlus imp2=new ImagePlus("Min_Max_Image",ip2);
imp2.show();
```

Here lets explain a couple of things. First, we have to get the currently selected image. For this we use the window manager class of ImageJ. This is already running as a part of ImageJ, so we don't have to construct it. It has a getCurrentImage() method that gives us the ImagePlus of the currently selected image. In order to get access to the pixel values, we have to get the ImageProcessor of that ImagePlus which we do using the getProcessor() method. Note that there are different subclasses of ImageProcessor (ColorProcessor, ByteProcessor, FloatProcessor, etc). Our ImageProcessor hopefully

will be a FloatProcessor because we are trying to type-cast the pixels array as float[]. Therefore if the image is not 32-bit (floating point), we will get an error. We have to use type-casting here because getPixels returns an object with unspecified type. Since float[] is a subclass of object, we can used type-casting, but the object must be a float[] object or an appropriate type-casting substitute or this will not work. While you can typecast an int into a float, you cannot typecast an int[] into a float[] (what works with single variables doesn't necessarily work with arrays). You can convert all grayscale images into 32-bit by using Image-Type-32-bit. This will preserve all of the nuances in the data (see below for more on this). You can also convert your image into a FloatProcessor before getting the pixels using the convertToFloat() method in the ImageProcessor class.

Note that pixels is the actual data presented in our original image (unless of course we use the convertToFloat() method on a different type of image). If we change this data, the image will be changed. Since we want our changes to be in a second image, we have to copy the original into a new array. ImageJ images are one dimensional arrays that represent a raster scanned image with each row presented after the last. We could use a for loop for the copying, but instead we use the highly optimized System.arraycopy(source,start,destination,start,length) method. Then we do our min/max filter on the copied image. Finally we create a new ImageProcessor and a new ImagePlus from that ImageProcessor. The ImagePlus show() method shows the resulting image.

One random thing you may need to know: the if-then loop usually requires a boolean value. The data[i]>=max gives a boolean value. You could also use the following:

```
boolean gtval=false;
gtval= data[i]>=max;
if(gtval){data[i]=max;}
```

Now what if we wanted to do something upon the condition that a is equal to b?

```
boolean eqval = a=b;
```

This doesn't work because we are setting a boolean value equal to float value, a. Instead we need:

```
boolean eqval = a= =b;
```

therefore = = can be translated "is equal to" while = means "equals".

A word about data types:

Below are the allowed image data types for ImageJ:

Binary: ones and zeros (actually stored as 0 and 255)
Byte (also known as 8 bit): numbers between 0 and 255
Short (also known as 16 bit): numbers between 0 and 65535

Float (also known as 32 bit): numbers stored in exponential format
RGB: each pixel has a red, green, and blue 8-bit number

Converting to the float data type will never decrease resolution—all data will be exactly preserved. Once the image is in the float data type, its brightness and contrast can be adjusted as desired without changing the original data—this only changes the appearance. Nevertheless, once the data are converted back to 8-bit or 16-bit, the data will be rescaled with the maximum displayed value set at the maximum of the selected scale (255 or 65535) and the minimum displayed value set to zero. I would recommend always writing plugins initially to operate on 32-bit float images. Later the functionality can be added to support 16 and 8-bit images if desired.

Writing GUI's in ImageJ

GUI stands for graphical user interface. This just means that there is communication between the user and the code. ImageJ provides a generic interface class called GenericDialog. It also provides file opening and closing dialogs that I won't discuss here. The source code is listed under package IJ.gui. Lets make a simple gui for our set_min_max plugin. Note everything to the right of // is not compiled—this is a comment. If you would like to use multiple lines for comments, start with /* and end with */.

```
GenericDialog gd=new GenericDialog("Options"); //here we construct the dialog
float min=0.0f;
gd.addNumericField("Minimum Value",min,3); /*this method calls for a label, a default
        value, and then the number of decimal places*/
float max=10.0f;
gd.addNumericField("Maximum Value",max,3);
boolean newimage=false;
gd.addCheckbox("Create New Image?",newimage); //again, label, then default value
gd.showDialog(); //show the dialog box
if(gd.wasCanceled()){return;} //see if it was cancelled (return exits the plugin)
min=(float)gd.getNextNumber(); //find out what was chosen by the user
max=(float)gd.getNextNumber();
newimage=gd.getNextBoolean();
ImagePlus imp=WindowManager.getCurrentImage();
int width=imp.getWidth();
int height=imp.getHeight();
ImageProcessor ip=imp.getProcessor();
float[] pixels=(float[])ip.getPixels();
float[] newpixels=new float[width*height];
if(newimage){
        System.arraycopy(pixels,0,newpixels,0,width*height);
        for(int i=0;i<width*height;i++){
                if(newpixels[i]>max){
                        newpixels[i]=max;
```

```
                }
                else{
                        if(newpixels[i]<min){
                                newpixels[i]=min;
                        }
                }
        }
        ImageProcessor ip2=new ImageProcessor(width,height,newpixels,null);
        ImagePlus imp2=new ImagePlus("Min_Max_Image",ip2);
        imp2.show();
} else {
        for(int i=0;i<width*height;i++){
                if(pixels[i]>max){
                        pixels[i]=max;
                }
                else{
                        if(pixels[i]<min){
                                pixels[i]=min;
                        }
                }
        }
        imp.updateAndDraw();
}
```

Obviously, the possibilities are endless, but hopefully this gives you a start. The ImageJ source code files may seem complicated at times, but they are quite readable compared to other programs. In this way, most of what you need can be learned by looking at the source code files and seeing how others have done the task you need.

Image Stacks in ImageJ

One often encounters the need to use an image stack. For example, time series', spectral series', and depth series' are represented as image stacks. Once these are imported into ImageJ, it is possible to scroll through the stack and observe behavior. The stack data and information are located in the ImageStack object inside the ImagePlus object.

Here we find the maximum z projection of a stack:

```
ImagePlus imp=WindowManager.getCurrentImage();
int width=imp.getWidth();
int height=imp.getHeight();
ImageStack stack=imp.getStack();
int slices=stack.getSize();
float[] maxpixels=new float[width*height];
for(int i=0;i<width*height;i++){
        float[] tempfloat=(float[])stack.getPixels(1);
```

```
        maxpixels[i]=tempfloat[i];
        for(int j=2;j<=slices;j++){
                tempfloat=(float[])stack.getPixels(j);
                if(tempfloat[i]>maxpixels[i]){maxpixels[i]=tempfloat[i];}
        }
}
ImageProcessor ip2=new FloatProcessor(width,height,maxpixels,null);
ImagePlus imp2=new ImagePlus("Max_Projection",ip2);
imp2.show();
```

Note that the slices in ImageStack are indexed from 1 to size, not from 0 to size-1 like normal arrays.

This might seem incredibly inefficient, since we have to get the pixel array for each slice width*height times. Nevertheless, remember that the pixel array is just an object and when we call the getPixels(#) method, we just get the reference to that object (it is not copied). Therefore the getPixels method is just giving us the location of the array of pixel values, not a copy of that array.

Plotting in ImageJ

The plotting functionality in ImageJ is severely limited. While this is understandable given that ImageJ is not a plotting utility, I feel that it could easily be augmented to allow for more advanced plotting applications. In that respect it would gain from all of the advantages that ImageJ now has over other image editing applications and would be continually improved by the addition of plugins. Nevertheless, this is not the case and I have little control over it, so I will give the simple explanation of plotting as it currently exists here.

Simple plots can be created with the PlotWindow class under package ij.gui. Here is some simple code to create a vertical line profile:

```
GenericDialog gd=new GenericDialog("Options"); //here we construct the dialog
int linenum=0;
gd.addNumericField("Line Number",linenum,0);
gd.showDialog();
if(gd.wasCanceled()){return;}
linenum=(int)gd.getNextNumber();
ImagePlus imp=WindowManager.getCurrentImage();
int width=imp.getWidth();
int height=imp.getHeight();
ImageProcessor ip=imp.getProcessor();
float[] pixels=(float[])ip.getPixels();
float[] line=new float[height];
float[] xvals=new float[height];
for(int i=0;i< height;i++){
```

```
        line[i]=pixels[i*width+linenum];
        xvals[i]=(float)i;
}
PlotWindow plot=new PlotWindow("Profile”+linenum,"intensity","pixel",xvals,line);
plot.draw();
```

Here we introduced a very nice feature of java. Strings can be mixed with numeric values which are automatically converted to strings.

```
int linenum=5;
String plotlabel=”Profile “+linenum;
```

Here plotlabel is equal to “Profile 5”. Java automatically converted the variable linenum into a string for us.

Unfortunately, once you have the plot drawn, there is little you can do except save the values or copy them to excel or another program. Be careful, the precision of the copied data can be significantly less than the displayed data. You can change the number of decimal places under Analyze-Set Measurements. Nevertheless, the total number of decimal places cannot exceed 9. Ideally ImageJ would copy the exponential notation version of the floating point number. Fortunately, ImageJ has written the plotwindow as an extension of the ImageWindow class. In order to gain access to the plot data, you have to get the ImagePlus, then get its ImageWindow, casting it into a PlotWindow object type. You then have to use the getXValues() and getYValues() methods to get the arrays of data. Unfortunately, this only works for the most recently added data set. If you add a data set to a plot, the previous data set is only available for viewing, not for editing. Here is an example of getting the most recent data set and multiplying it by a number.

```
ImagePlus imp=WindowManager.getCurrentImage();
PlotWindow plot=(PlotWindow)imp.getWindow();
float[] xvals=plot.getXValues();
float[] yvals=plot.getYValues();
int length=yvals.length;
float[] newyvals=new float[length];
float[] newxvals=new float[length];
float multiplier=10.0f;
for(int i=0;i<length;i++){
        newyvals[i]=yvals[i]*multiplier;
        newxvals[i]=xvals[i];
}
//since we can’t change the original plot, we have to create a new plot
PlotWindow plot2=new PlotWindow(“Multiplied by “+
multiplier,"x","y",newxvals,newyvals);
plot2.draw();
```

Other ImageJ Outputs

Here are a few other ways that your code can communicate with the user:
IJ.log(“output text”);
IJ.showMessage(“output text”);
IJ.showStatus(“output text”);
IJ.showProgress(int currentindex,int finalindex);