ImageJ / Fiji
Short Introduction into Macro and Plugin Programming
ZOI Friday seminar

B. Kovář

Department of Image Processing
Institute of Information Theory and Automation

21. února 2014
Contents

1 Introduction

2 GUI

3 Macro Language

4 Plugin Programming

5 Conclusions
**ImageJ**: successor of a software from the National Institutes of Health called NIH Image (Pascal-based, Mac only, early 90's)

- Public Domain – source code available,
- Java based – Win, Mac, Linux, . . . ,
- Macros & plug-ins,
- Huge potential, **very active community**,  
- Technical design limitations (15 yrs. old design),
- \( \Rightarrow \) **ImageJ v2**

ImageJ requires a Java runtime installed, can update itself and is shipped with small, generic set of Plugins and Macros...

**ImageJ basically can do everything**, it’s just sometimes very hard to find out how...
Fiji – Fiji is just ImageJ – ”ImageJ (batteries included)”

- Fiji is an ImageJ distribution intended for Life sciences,
- Huge set of Plugins,
- Powerful update mechanism,
- Extensive documentation, tutorials, etc.,
- But: shares ImageJ’s limitations.
ImageJ

- Generic image processing framework - unfocused

FiJi

- Aimed for Life Sciences,
- Emphasis on Registration, Segmentation & Volume Data,
- Community development effort,
- Quality mechanisms for plugins,
- Support for more additional languages: Python, Ruby, Scala...
- Script editor to develop plugins and macros.
Contents

1. Introduction
2. GUI
3. Macro Language
4. Plugin Programming
5. Conclusions
The main window has three parts:

1. **Menu bar**
2. **Tools**
3. **Status line**
   - ImageJ & Java version
   - Memory architecture & current usage
   - Pixel positions and intensities while hovering on images
   - Descriptions of toolbar entries
   - ...

![ImageJ GUI](image.jpg)
Macros can be used to

- automate repetitive tasks
- document what you did
- share common procedures
- provide consistency in your analysis
- scripting languages (build-in, javascript, python, ...)

A Macro is a program that automates a series of ImageJ commands. ImageJ has a dedicated programming language for this.
The obligatory "Hello World" code:

```
macro "hello" {
    print("Hello World!");
}
```

Running this opens a new window "Log" showing the message: "Hello World!"

`print()` is useful for debugging.
Variables & Strings

The most important concept: a named piece of memory!

- Name and value
- Value can be numeric or text
- Can be used in arithmetic expressions or string combinations

```java
x = 2;
y = 3;
result = x * x + y + y;

name = "Bob";
msg = "Those days are over, " + name;
print(msg);

amount = amount * 2;
counter = counter + 1;

counter++;
```
Parameter input by user

getNumber() can be used to request a numerical input by the user, getString() for text.

```java
macro "input_print_out_calc" { 
  a = getNumber("a?", 10); 
  b = getNumber("b?", 5); 
  c = a*b; 
  print("\\Clear"); 
  print(c); 
}
```
Running ImageJ commands

Crucial element for macros: run ImageJ commands

1. Almost all commands can be run from macros
2. A few Plugins are not macro-ready

Most convenient way to find the corresponding macro commands is to use the built-in Macro Recorder:
Batch processing

To apply a macro or multiple macro commands to many files, you could add file-accessing and looping commands or use the Batch Process tool.
Loops are required to iterate a certain part of our macro.

```javascript
for (i=0; i<5; i+=1) {
    print(i + " : " + msg);
}
```
Stack-processing with loops

One very powerful application of loops is stack processing since many ImageJ commands work just on a single slice. To work on all slices the “nSlices” and “setSlice” commands are used:

```macro "Measure Avg Intensity Stack" {
    frames=nSlices;
    run("Set Measurements...", " mean redirect=None decimal=4");
    run("Clear Results");
    for(i=0; i<frames; i++) {
        currentslice=i+1;
        setSlice(currentslice);
        run("Measure");
    }
}
```
Conditional statements

If–else statements can be used to run a certain part only if one or more conditions are met. Note the "==" in the condition (a comparison operator), this is different from a single "=" (assignment). Other comparison operators are: <, <=, >, >=, !=

```java
num = getNumber("Number:", 3);
if (num == 3) {
    print(num + ": default value");
} else {
    print(num + ": value changed");
}
```
User-defined functions

When macros become more sophisticated, it is very likely that some parts will appear multiple times. These parts should be placed in separate functions to avoid redundancy and improve readability:

```javascript
function CheckStack() {
    if (nSlices==1) {
        exit("Image is not a stack");
    }
}

function CheckThreshold() {
    getThreshold(lower, upper);
    if ((lower==-1) && (upper==-1)) {
        exit("Must be thresholded");
    }
}

macro "threshold check" {
    CheckStack();
    CheckThreshold();
    getThreshold(lower, upper);
}
```
Return values

If your function does some calculation, you need a way to hand back the result to the code that called the function:

```javascript
function ReturnAdd(n, m) {
    p = n + m;
    return p;
}

macro "function add" {
    a = 1;
    b = 2;
    result = ReturnAdd(a, b);
    print(result);
}
```
Advanced dialogues

If multiple values and settings need to be requested from the user, it is easier to use a "Dialog" for this task:

```java
Dialog.create("input dialog");
Dialog.addMessage("my description");
Dialog.addNumber("value 1:", 0);
Dialog.addNumber("value 2:", 0);
Dialog.addCheckbox("really?", true);
Dialog.show;

val1 = Dialog.getNumber();
val2 = Dialog.getNumber();
really = Dialog.getCheckbox();
```
Installing Macros

Macros are recognized by ImageJ from the following filename suffixes:

- *.IJM (preferred)
- *.TXT (deprecated)

Code files with the txt-suffix need to have an underscore in their name, otherwise ImageJ ignores them. The macro code files need to be placed in a sub-directory inside the ImageJ base directory:

1. macros
2. plugins (preferred)
Contents

1 Introduction
2 GUI
3 Macro Language
4 Plugin Programming
5 Conclusions
Plugins: when macros are not enough ...  

Macros are quick and easy to record or write, **but**

- Sloooow,
- Waste memory for more complicated tasks,
- Cannot use the full functionality of ImageJ, only what the built-in functions offer,
- Run always in foreground, blocking the computer,

Plugins are a much more powerful concept than macros and most of ImageJ's built-in menu commands are in fact implemented as plugins. Plugins are implemented as Java classes, which means that you can use all features of the Java language, access the full ImageJ API and use all standard and third-party Java APIs in a plugin.
What does a plugin consist of?

Plugins are .jar files (really .zip files with a certain structure), containing:

- one or more Java class(es) implementing the functionality, and
- a plugins.config file defining which menu entries offer the functionality.
- optionally additional resources required by the plugin, such as images or icons.

If the plugin is implemented in one Java class, and it offers only one menu entry in the Plugins menu, it can be offered as bare .java or .class file, too.
Installing plugins

Plugins can be installed manually by copying the .jar file into the plugins/subdirectory of Fiji and by updating the menus using Help → > Update Menus.

There are two easier ways, though:

1. Plugins → > Install PlugIn...
2. Or drag ’n drop the .jar file onto the Fiji main window.
Plugin development

For writing new plugin programs, we also need text editor and Java compiler. The Java runtime environment (JRE) contains both, so no additional software is required to get started.

It’s highly recommended to use one of the freely available integrated Java programming environments, such as Eclipse, IntelliJ IDEA or NetBeans.
Types of Plugins

There are basically three types of plugins:

- those that do not require an image as input (implementing the interface `PlugIn`)
- plugin filters, that require an image as input (implementing the interface `PlugInFilter`).
- and `PlugInFrame` – a plugin that runs in its own window.
Images and Processors

While the ImageJ API makes it easy to work with images on the programming level, their internal representation is fairly complex.
• ImageJ is a particularly flexible image-processing package that is a free, open source
• The program will run on all computer platforms
• ImageJ claims to be the fastest pure Java image processing program; it is capable of filtering a 2048x2048 image in 0.1 seconds (according to the ImageJ website)
• ImageJ’s intended purpose is medical image processing, but it works very well in other context.
• The software has a particular emphasis on analysis and incorporates a number of tools for measuring images.
• As a result functions are often more mathematically “transparent” than in commercial packages such as Adobe Photoshop.
• A well-developed “plugin” interface allows for customization.
• A very active user community has contributed dozens of freely available plugins.
• A “macro” interface allows tasks to be recorded and automated.
Resources

2. ImageJ dokuwiki: http://imagejdocu.tudor.lu/
7. FiJi wiki containing documentation for plugins, tutorials, etc: http://fiji.sc/wiki/