Custom toolbars and mini applications with Action Bar

Jérôme Mutterer

CNRS, Institut de biologie moléculaire des plantes, 12 rue du général Zimmer, Strasbourg, France

jerome.mutterer@ibmp.fr

ABSTRACT

The Action Bar ImageJ plugin facilitates creation of additional custom toolbars, using simple macro-like toolbar description file. Participants will learn how to setup a basic action bar, and how to use the latest features of this plugin. Button actions are basically macro programs, so action bars can interact with all standard ImageJ components. Primary use of these custom toolbars is to keep at hand often used commands, plugins and macros, but toolbars sets can be turned into mini applications with minimal effort. With these highly task-oriented applications, end users can focus on just using a simple system with absolutely no prior knowledge of ImageJ. This workshop is aimed at ImageJ users with some previous experience with the macro language.

Keywords: User-interface, Macros, Toolbars, Mini-applications.

1. INTRODUCTION

Setting up an operational image processing environment with ImageJ is often merely putting existing bits together. The large number of available plugins or macros developed by image processing specialists in their fields can provide most new users with robust solutions to their needs.

One common pitfall however is that a new user has to spend some time learning the software's philosophy. This will include learning the menu structure to some extend, spotting commands that he will have to use to complete his imaging tasks. The minimalist look of ImageJ's graphical user interface (GUI) hides a relatively large number of sub menus and menus items. For the beginner user most of them will be useless and maybe even destabilizing. The amount of available plugins and macros adds to this feeling.

Not every user, however, has a clear idea of programming, and most of them just aren't interested in the details of how a software is built, or in the difference between a macro and a plugin. A user wants working software, with a fast learning curve.

From theses remarks came the idea that essential and needed functions should be exposed to the new user's sight as much as possible. To achieve this, action tools macros placed in ImageJ standard toolbar are very useful GUI elements. The flexibility of macro language behind these tool buttons allows the calling of specific menu commands to create simple commands shortcuts, or larger macro code, including call to user plugins to achieve more complex tasks. A toolbar buttons set, or toolset, will only hold eight buttons. Multiple toolsets can be installed, but can not be exposed simultaneously.

The Action Bar plugin provides a very simple way to extend ImageJ's GUI with custom button bars. We will call these GUI elements "action bars". They will be a convenient place to install frequently used items like built-in commands, macros, or user plugins, to provide users with a good looking, task-oriented and easy to use environment In this workshop, we will demonstrate the use of action bars. Figure 1 shows some example action bars created with the plugin.

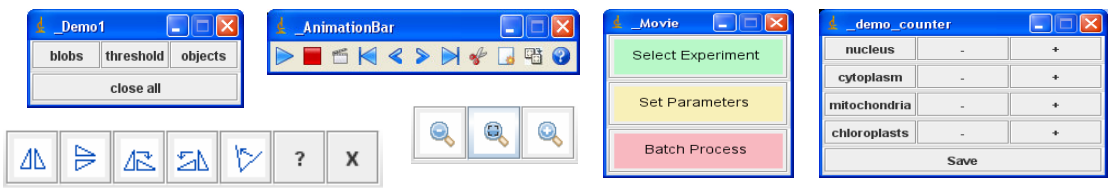


Figure 1. Some example toolbars created with Action Bar.

2. PRINCIPLE OF OPERATION

Action Bar is a standard ImageJ plugin that was designed to read a plain text configuration file containing a description of the bar's organization in terms of buttons, lines of buttons, labels, bar or button properties, and finally buttons actions. Figure 2 shows the general operating mode of the action bar plugin.

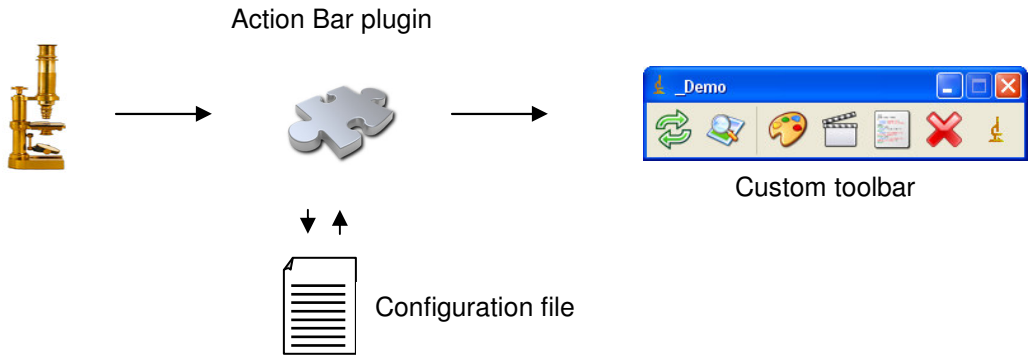


Figure 2. The Action Bar plugin reads all necessary information from a configuration file to output a custom toolbar. All bar and button properties, path to button icons and button macro action code are described in this single text file.

2.1 Installation

The latest version of Action can be downloaded from the ImageJ documention wiki at http://imagejdocu.tudor.lu/imagej-documentation-wiki/Members/jmutterer/Action_Bar/ . The plugin will be best installed in a direct subfolder of the plugins directory as follows :

```
ImageJ/plugins/ActionBar/                    main ActionBar folder
ImageJ/plugins/ActionBar/icons/              button icons folder
ImageJ/plugins/ActionBar/Action_Bar.jar      jar file with classes and source code
```

2.2 Starting with the assistant

The easiest way to create a new action bar is to manually start the plugin by choosing `ImageJ/Plugins/Action Bar/ActionBar` from the plugins menu. This will start a assistant that can help you create an action bar canvas, which you will be able to customize later on.

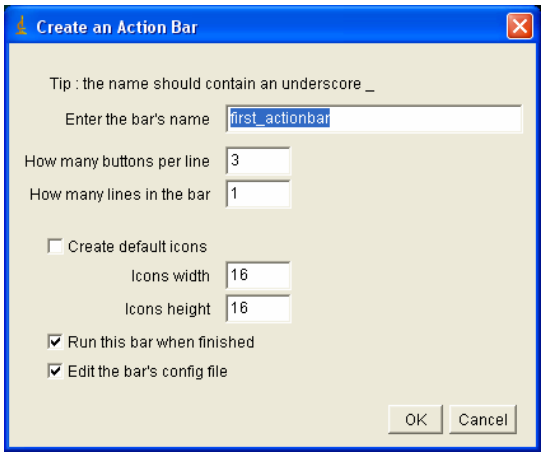


Figure 3. The Action Bar creation assistant will create a standard text canvas with a adequate structure.

Everything needed to run this configuration file is created, and this file is opened in an editor window.

```
// Action Bar description file :first_actionbar
run("Action Bar","/plugins/ActionBar/first_actionbar.txt");
exit();

<line>

<button> 1 line 1
label=button_1_1
icon=noicon
arg=<macro>
showMessage("You pressed button 1 line 1");
</macro>

<button> 2 line 1
label=button_2_1
icon=noicon
arg=<macro>
showMessage("You pressed button 2 line 1");
</macro>

<button> 3 line 1
label=button_3_1
icon=noicon
arg=<macro>
showMessage("You pressed button 3 line 1");
</macro>

</line>

// end of file
```

Figure 4. The example code needed for the design of a one line, three buttons action bar. The structure is the following : a file header that enables this text file to be run by the ImageJ macro runner. This header holds only a run("Action Bar", arg); statement that will run the Action Bar plugin against the proper configuration file (this text file itself). Bellow this header is a simple description of the button bar. The bars are organized in terms of lines of buttons. The design of each button requires three special items called the button's label, the button's icon, and the button's argument.

Running this code by hitting CTRL-R, should start your first action bar :

From this point all that will be needed will be to customize this configuration file to add, more buttons, more lines, icons, set special button or bar properties, and of course set up button action.

2.3 Setting buttons look and actions

At the core of action bars are simple buttons made of Swing JButtons. These buttons all have three special properties, a label, an icon, and an argument, and should be strictly declared in the following way :

```
<button>
label=my_label
icon=first_actionbar/B1.gif
arg=print("Hello");
```

The following rules must be followed to create new buttons.

- A new button is declared with the `<button>` tag.
- The next line should start with the `label=` keyword followed by a string that will be used either for the button's label itself, or for the button tooltip.
- The third line should start with the `icon=` keyword followed by a path leading to the image used as button icon face. The path should be relative to the ImageJ/Plugins/ActionBar/icons folder. If you prefer not having an icon, use the `noicon` keyword instead of the path, and the button's label will be used as button face.
- The last line should start with the `arg=` keyword followed by the action code that will be run when the button is pressed. The argument can be of different types that will be discussed below.

➔ The standard type of button argument is an ImageJ macro string of one line.

```
arg=if (isOpen("Log")) {selectWindow("Log"); run("Close");}
```

➔ For longer macro code, or for more readability, the whole code should be declared in an `<macro></macro>` statement, like this :

```
arg=<macro>
if (isOpen("Log")) {
        selectWindow("Log");
        run("Close");
        }
</macro>
```

➔ For using ImageJ tool macros, the tool's code can be inserted inside `<tool></tool>` statement, like this :

```
arg=<tool>
          getCursorLoc(x, y, z, flags);
          print("Pixel: "+x+" "+y+" Value: "+getPixel(x,y));
</tool>
```

The tool's code will be wrapped in a standard ImageJ tool and installed in the toolbar.

➔ A special button argument can be used to toggle visibility of the main ImageJ window :

```
arg=<hide>
```

Pressing this button will hide or show the main ImageJ window.

➔ Another special button argument can be used close the current action bar :

```
arg=<close>
```

Pressing this button will close this action bar. This, however, can be achieved by clicking on the default close button in the frame's title bar, or by hitting the escape key in the case of a popup action bar.

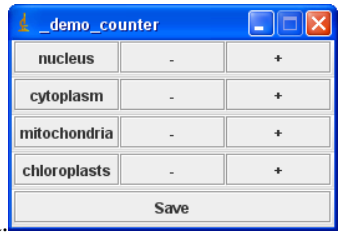
2.3 Adding more lines, and setting some bar properties

Action bars can be further customized by adding bar properties, multiple lines of buttons, adding some decorations or texts to the bar, using a different layout, or adding special code at button startup, or that is shared across buttons. I will review these features hereafter.

➔ `<line></line>` All buttons in the same line should be included inside this statement, as shown in Figure 4.

➔ `<noGrid>` tag : the default layout of action bars has been improved, compared with the previous versions. Now all buttons in a given line share the same width, and all lines have an overall same width. This is the default behaviour, unless you use the `<noGrid>` tag.

```
<noGrid>
```

a. 

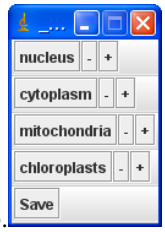
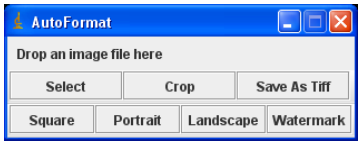
b. 

Figure 5. Example of the default layout with homogenous buttons and lines widths (a), compared with the forced no-grid layout (b). The latter can be more flexible, but the user will need to manage button icons width.

➔ `<text>` tag. This tag allows for insertion of a textual label in the action bar. The label is added between two lines of buttons. Note that the label can be html-formatted.

```
<text> Drop an image file here
```



➔ `<separator>` tag. This tag adds a space between two buttons. With the default layout, the width of this space is the width of exactly one button. To get only a tiny separator, like in the example below, use the separator tag in combination with the no-grid tag.

```
<separator>
```

a. 

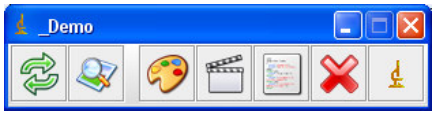
b. 

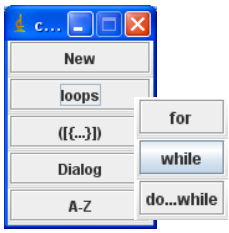
Figure 6. Example uses of the `<separator>` tag without (a) or in combination with the `<noGrid>` tag (b).

➔ `<onTop>` tag. Setting up an action bar with this tag will allow this bar to always stay on top of all images, including the current active image.

```
<onTop>
```

➔ `<popup>` tag. With this tag, the generated toolbar will have no window decoration, i.e. a simple frame with only the buttons will be displayed. These special toolbars cannot be moved around the screen , but will popup at the mouse current location. Popup toolbars could be useful as sub-action bars, if started from a button inside another bar. See also the image popup bar example. A popup bar is closed when one of its button or the escape key is pressed.

```
<popup>
```



➔ `<startupAction>` `</startupAction>`. The code inside these markers will be run when the action bar is displayed. This could be useful to setup the environment in which the bar is supposed to work. A application I could think about was to start other action bars. This way you can call a 'master' bar that will run other bars when started.

```
<startupAction>
   print ("action bar started");
   run("Record...");
   newImage("test","8-bitWhite",800,600, 1);
</startupAction>
```

An action bar carrying this startup action would acknowledge it was started to the Log window, start the command recorder, and set up a new test image.

➔ `<codeLibrary>` `</codeLibrary>`. The code inside these markers will be append to the macro actions of each button, before the latter is run. This makes it a good place to add functions that need to be shared across buttons. It is some sort of a duplicate of the function library that is appended to every macro file.

```
<codeLibrary>
   function mean (a) {
   sum = 0; for (i=0;i<a.length;i++) sum+=a[i];
   return (sum/i);
   }
</codeLibrary>
```

Now every button in this bar can use the `mean(array)` function.

2.4 Turning an action bar (or an action bar set) into a small application

➔ `<main>` tag. By setting this bar property, the action bar will act as the application main window. This essentially means that ImageJ will quit if the action bar is closed. Also, the main ImageJ window will be temporarily hidden.

```
<main>
```

Remember to add a button that will restore ImageJ visibility if you think you need it !

➔ `<DnD>` tag. By setting this bar property, you'll be able to drag and drop all files supported by ImageJ or by an helper I/O plugin, provided it is properly configured to work with ImageJ. This is simply achieved by capturing ImageJ drag and drop behavior and applying it to the current action bar.

```
<DnD>
```

Together with the `<main>` tag, this will turn your action bar into a mini application that now supports all kind of image formats, by just dragging the files onto the button bar. The macro-coded button actions give you access to the whole range of ImageJ functions.

2.5 How to start an action bar

Once you have successfully created and customized your action bar, you will be wanting to integrate it in your ImageJ installation.

➔ Simple call of the `ImageJ/Plugins/Action Bar/ActionBar` menu item. This will start the action bar creation assistant, as documented in section 2.2.

➔ Action Bar can be installed as a regular plugin shortcut with the `ImageJ/Plugins/Shortcuts/Install Plugin…` menu entry. The path to the action bar's configuration file should be given as argument. The action bar will then be available from the specified menu entry.

➔ The recommended way to call an action bar is from a macro language `run(plugin, argument)` statement. This is the method used in the header of the bar's configuration files created by the assistant.

```
run("Action Bar","plugins/ActionBar/_Demo1.txt");
```

Note that the path should be relative to ImageJ startup directory. The `"Autorun"` macro located in the `StartupMacros.txt` file is a good place to launch action bars from, so that they will show up every time ImageJ is started.

➔ Simply put the action bar's configuration file in the plugins folder, or direct subfolder. At ImageJ startup time, all macros at these locations will be installed in the plugins menu, if they have an underscore `( _ )` in their file name.

ACKNOWLEDGMENTS

I would like to acknowledge existing users that helped me fix many problems in the first releases of the Action Bar plugin, and that also suggested improvements and new features for this plugin. The original idea of a customizable toolbar arose from Patrick Pirrotte's ContextualImageTypeModifier plugin. Of course this software only takes advantage of ImageJ's existing features and fantastic flexibility, so credit essentially goes to Wayne Rasband.