

ImageJ Macro Language

- [Introduction](#)
- [Hello World" Example](#)
- [Running ImageJ commands](#)
- [Command Recorder](#)
- [Macro Sets](#)
- [Keyboard Shortcuts](#)
- [Tool Macros](#)
 - [Image Tools](#)
 - [Action Tools](#)
 - [Menu Tools](#)
 - [Tool Sets](#)
 - [Icons](#)
- [Variables](#)
- [Operators](#)
- [if/else Statements](#)
- [for, while and do...while Statements](#)
- [Functions](#)
- [Working with Strings](#)
- [Extending the Macro Language](#)
- [Running from Command Line](#)
- [Debugging Macros](#)
- [Built-in Functions](#)

Introduction

A macro is a simple program that automates a series of ImageJ commands. The easiest way to create a macro is to record a series of commands using the [command recorder](#). A macro is saved as a text file and executed by selecting a menu command, by pressing a key or by [clicking on an icon in the ImageJ toolbar](#).

There are more than [300 example macros](#) on the ImageJ Web site. To try one, open it in a browser window, copy it to the clipboard (ctrl-a, ctrl-c), switch to ImageJ, open an editor window (ctrl-shift-n), paste (ctrl-v), then run it using the editor's *Macros>Run Macro* command (ctrl-r). Most of the example macros are also available in the macros folder, inside the ImageJ folder.

"Hello World" Example

As an example, we will create, run and install a one line *Hello World* macro. First, open an editor window using *Plugins>New>Macro* (or press shift-n). In the editor window, enter the following line:

```
print("Hello world");
```

To test the macro, use the editor's *Macros>Run Macro* command (or press ctrl-r). To save it, use the editor's *File>Save As* command. In the Save As dialog box, enter "Hello_World.txt" as file name, then click "Save". The macro will be automatically installed as a "Hello World" command in the Plugins menu when you restart ImageJ, assuming the file name has an underscore in it and the macro was saved in the plugins folder or a subfolder. You can run this macro by pressing a single key by creating a shortcut using *Plugins>Shortcuts>Create Shortcut*.

To re-open a macro, use the *File>Open* or *Plugins>Macros>Edit* commands, or drag and drop it on the "ImageJ" window.

Using run() to execute ImageJ commands

Use the macro language's built-in [run\(command, options\)](#) function to run any of the 400+ commands in the ImageJ menu bar. The first argument (*command*) is any ImageJ

menu command (e.g., "Measure" and "Close All") and the second argument (*options*), which is optional, is a string containing parameter values. The run() function works both with built-in commands and with user-written plugins that use ImageJ's GenericDialog and OpenDialog classes to display dialog boxes.

Use the Command Recorder (*Plugins>Macros>Record*) to generate run() function calls. For example, this code was recorded by running the *Process>Find Maxima...* command:

```
run("Find Maxima...", "noise=50 output=[Point Selection] light");
```

This code runs *Find Maxima...* with "Noise tolerance" set to 50, "Output type" set to "Point Selection" and "Light background" enabled. String parameters containing spaces, such as "Point Selection", must be enclosed in brackets.

To test it, copy it to the clipboard, switch to ImageJ, press shift-b (*File>Open Samples>Blobs*), press shift-n (*File>New>Text Window*), press ctrl-v (*Edit>Paste*), then run it by pressing ctrl+r (*File>Macros>RunMacro*).

The input fields of a dialog can be set to the contents of a macro variables by using string concatenation:

```
noise = 50;
output = "Point Selection";
run("Find Maxima...", "noise="+noise+" output=["+output+"] light");
```

With ImageJ 1.43 and later, there is an easier method that only requires adding "&" to the variable name in the options string:

```
noise = 50;
output = "Point Selection";
run("Find Maxima...", "noise=&noise; output=&output; light");
```

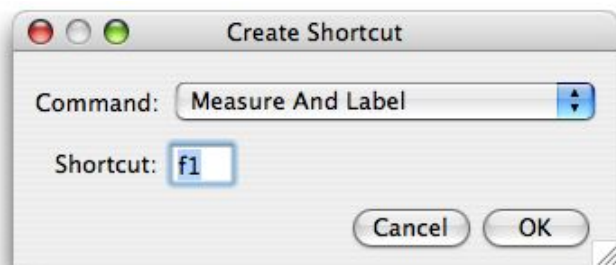
String variables passed using this method do not have to be enclosed in brackets. For more examples, see the [ArgumentPassingDemo](#) macro.

Using the Command Recorder to Generating Macros

Simple macros can be generated using the command recorder (*Plugins>Macros>Record*). For example, this macro, which measures and labels a selection,

```
run("Measure");
run("Label");
```

is generated when you use the *Analyze>Measure* and *Analyze>Label* commands with the recorder running. Save this macro in the plugins folder, or a subfolder, as "Measure_And_Label.txt", restart ImageJ and there will be a new "Measure And Label" command in the Plugins menu. Use the *Plugins>Shortcuts>Create Shortcut* command to assign this new command a keyboard shortcut.



In this example, the "Measure And Label" macro is assigned to the F1 key. Note how the underscores in the macro file name (at least one is required) are converted to spaces in the command name.

Macro Sets

A macro file can contain more than one macro, with each macro declared using the *macro* keyword.

```
macro "Macro 1" {
    print("This is Macro 1");
}

macro "Macro 2" {
    print("This is Macro 2");
}
```

In this example, two macros, "Macro 1" and "Macro 2", are defined. To test these macros, select them, *Copy* (ctrl-c), switch to ImageJ, open an editor window (ctrl-shift-n), *Paste* (ctrl-v), select the editor's *Macros>Install Macros* command, then select *Macros>Macro 1* to run the first macro or *Macros>Macros 2* to run the second.

Macros in a macro set can communicate with each other using global variables.

```
var s = "a string";

macro "Enter String..." {
    s = getString("Enter a String:", s);
}

macro "Print String" {
    print("s=" + s);
}
```

Use the editor's *File>Save As* command to create a macro file containing these two macros. Name it something like "MyMacros.txt" and save it in the macros folder inside the ImageJ folder. (Note that the ".txt" extension is required.) Then, to install the macros in the *Plugins>Macros* submenu, use the *Plugins>Macros>Install* command and select "MyMacros.txt" in the file open dialog. Change the name to "StartupMacros.txt" and ImageJ will automatically install the macros when it starts up.

Keyboard Shortcuts

A macro in a macro set can be assigned a keyboard shortcut by listing the shortcut in brackets after the macro name.

```
macro "Macro 1 [a]" {
    print("The user pressed 'a'");
}

macro "Macro 2 [1]" {
    print("The user pressed '1'");
}
```

In this example, pressing 'a' runs the first macro and pressing '1' runs the second. These shortcuts duplicate the shortcuts for *Edit>Selection>Select All* and *Analyze>Gels>Select First Lane* so you now have to hold down control (command on the Mac) to use the keyboard shortcuts for these commands.

Note that keyboard shortcuts will not work unless the macros are installed and the "ImageJ" window, or an image window, is the active (front) window and has keyboard focus. You install macros using the macro editor's *Macros>Install Macros* command or the *Plugins>Macros>Install* command. Install the two macros in the above example and you will see that the commands

```
Macro 1 [a]
Macro 2 [1]
```

get added to *Plugins>Macros* submenu. Save these macros in a file named "StartupMacros.txt" in the macros folder and ImageJ will automatically install them when it starts up.

Function keys ([f1], [f2]...[f12]) and numeric keypad keys ([n0], [n1]...[n9], [n/], [n*], [n-], [n+] or [n.]) can also be used for shortcuts. ImageJ will display an error message if a function key shortcut duplicates a shortcut used by a plugin. Numeric keypad shortcuts (available in ImageJ 1.33g or later) are only used by macros so duplicates are not possible. Note that on PCs, numeric keypad shortcuts only work when the *Num Lock* light is on. A more extensive example ([KeyboardShortcuts](#)) is available.

Tool Macros

You can define macros that create tools that get added to the ImageJ toolbar. There are three types of macro tools: image tools, action tools and menu tools. The three types can be combined to create a *tool set*. *Tool sets* can be added to the ImageJ toolbar as needed by clicking on the **>>** icon in the toolbar.

Image Tools

An image tool executes when the user clicks on the image with that tool. The macro that defines it must have a name that ends in "Tool - xxxx", where "xxxx" is hexadecimal code (described below) that defines the tool's icon. Here is an example image tool that displays the coordinates each time the user clicks on the image:

```
macro "Sample Tool - C0a0L18f8L818f" {
    getCursorLoc(x, y, z, flags);
    print("Sample: "+x+" "+y);
}
```

To install this tool, open an editor window (*Plugins>Macros>New*), paste in the macro, then use the editor's *Macros>Install Macros* command. Put this macro in a file named "StartupMacros.txt" in the macros folder and it will automatically be installed when ImageJ starts up. A macro file can contain up to eight tool macros and any number of non-tool macros. Macro files must have a ".txt" or ".ijm" extension.

You can also permanently install a macro tool by adding it to the ">>" menu in the toolbar. As an example, save the "Sample Tool" macro in the ImageJ/plugins/Tools folder as "Sample_Tool.txt", restart ImageJ, and select "Sample Tool" from the toolbar's ">>" menu.

A tool can display a configuration dialog box when the user double clicks on it. To set this up, add a macro that has the same name as the tool, but with " Options" added, and that macro will be called each time the user double clicks on the tool icon. In this example, the `getNumber` dialog is displayed when the users double clicks on the circle tool icon.

```
var radius = 20;

macro "Circle Tool - C00c011cc" {
    getCursorLoc(x, y, z, flags);
    makeOval(x-radius, y-radius, radius*2, radius*2);
}

macro "Circle Tool Options" {
    radius = getNumber("Radius: ", radius);
}
```

Action Tools

Tool macros with names ending in "Action Tool" perform an action when you click on their icon in the toolbar. In this example, the "About ImageJ" window is displayed when the user clicks on the tool icon (a question mark).

```
macro "About ImageJ Action Tool - C059T3e16?" {
    doCommand("About ImageJ...");
}
```

Note that action tool commands that display a dialog box may not run correctly if they are invoked using the `run()` function.

More examples are available at rsb.info.nih.gov/ij/macros/tools/ or in the ImageJ/macros/tools folder. Use *File>Open* (or drag and drop) to open one of these files and the tools defined in it will be automatically installed, or use the *Plugins>Macros>Install* command to install tools without opening an editor window.

Menu Tools

You can use the `newMenu` function to add menus to the toolbar. The `Toolbar Menu` macro demonstrates how to do this. You can also customize the contextual menu that appears when you right click on an image. The `CustomPopupMenu` macro demonstrates how to do this.

Tool Sets

A macro file can contain up to eight macro tools, along with any number of ordinary macros. A macro file (macro set) that contains macro tools is called a *tool set*. Save a *tool set* in the ImageJ/macros/toolsets folder and it will appear in `>>` menu at the right end of the toolbar. Select the *tool set* from the `>>` menu and the tools contained in it will be installed the tool bar. Restore the default tool set by selecting "Startup MacOS" from the `>>` menu.

The [/ij/macros/toolsets](http://ij/macros/toolsets) folder on the ImageJ website contains 13 example *tool sets*.

Tool Icons

Tool macro icons are defined using a simple and compact instruction set consisting of a one letter commands followed by two or more lower case hex digits.

Command	Description
Bxy	set base location (default is 0,0)
Crgb	set color
Dxy	draw dot
Fxywh	draw filled rectangle
Gxyxy...xy00	draw polygon (requires 1.48j)
Hxyxy...xy00	draw filled polygon (requires 1.48k)
Lxyxy	draw line
Oxywh	draw oval
Pxyxy...xy0	draw polyline
Rxywh	draw rectangle
Txyssc	draw character
Vxywh	draw filled oval (requires 1.48j)

Where x (x coordinate), y (y coordinate), w (width), h (height), r (red), g (green) and b (blue) are lower case hex digits that specify a values in the range 0-15. When drawing a character (T), ss is the decimal font size in points (e.g., 14) and c is an ASCII character.

Variables

The ImageJ macro language is "typeless". Variables do not need to be declared and do not have explicit data types. They are automatically initialized when used in an assignment statement. A variable can contain a number, a string or an array. In fact, the

same variable can be any of these at different times. Numbers are stored in 64-bit double-precision floating point format.

Variable names are case-sensitive. "Name" and "name" are different variables.

In the following example, a number, a string and an array are assigned to the same variable.

```
v = 1.23;
print(v);
v = "a string";
print(v);
v = newArray(10, 20, 50);
for (i=0; i<v.length; i++) print(v[i]);
```

You can run this code by selecting it, copying it to the clipboard (ctrl-C), switching to ImageJ, opening an editor window (*Edit>New*), pasting (ctrl-V), then pressing ctrl-R. (Note: on the Mac, use the apple key instead of the control key.)

Global variables should be declared before the macros that use them using the 'var' statement. For example:

```
var x=1;

macro "Macro1..." {
    x = getNumber("x:", x);
}

macro "Macro2" {
    print("x="+x);
}
```

The 'var' statement should not be used inside macro or function code blocks. Using 'var' in a macro or function may cause it to fail.

Operators

The ImageJ macro language supports almost all of the standard Java operators but with fewer precedence levels.

Operator	Prec.	Description
++	1	pre or post increment
--	1	pre or post decrement
-	1	unary minus
!	1	boolean complement
~	1	bitwise complement
*	2	multiplication
/	2	division
%	2	remainder
&	2	bitwise AND
	2	bitwise OR
^	2	bitwise XOR
<<, >>	2	left shift, right shift
+	3	addition or string concatenation
-	3	subtraction
<, <=	4	less than, less than or equal
>, >=	4	greater than, greater than or equal
==, !=	4	equal, not equal

&&	5	boolean AND
	5	boolean OR
=	6	assignment
+=, -=, *=, /=	6	assignment with operation

For numeric expressions, parentheses can be used to define the order of operations, e.g. $a*(b+c)$. Parentheses should not be used for string expressions (currently, expressions with parentheses are always interpreted as numeric).

if/else Statements

The *if* statement conditionally executes other statements depending on the value of a boolean expression. It has the form:

```
if (condition) {
    statement(s)
}
```

The condition is evaluated. If it is true, the code block is executed, otherwise it is skipped. If at least one image is open, this example prints the title of the active image, otherwise it does nothing.

```
if (nImages>0) {
    title = getTitle();
    print("title: " + title);
}
```

An optional *else* statement can be included with the *if* statement:

```
if (condition) {
    statement(s)
} else {
    statement(s)
}
```

In this case, the code block after the *else* is executed if the condition is false. If no images are open, this example prints "No images are open", otherwise it prints the title of the active image.

```
if (nImages==0)
    print("No images are open");
else
    print("The image title is " + getTitle());
```

Note that the "==" operator is used for comparisons and the "=" operator is used for assignments. The braces are omitted in this example since they are not required for code blocks containing a single statement.

The macro language does not have a *switch* statement but it can be simulated using *if/else* statements. Here is an example:

```
type = selectionType();
if (type==-1)
    print("no selection");
else if ((type>=0 && type<=4) || type==9)
    print("area selection");
else if (type==10)
    print("point selection");
else
    print("line selection");
```

Looping Statements (for, while and do...while)

Looping statements are used to repeatedly run a block of code. The ImageJ macro language has three looping statements:

- **for** - runs a block of code a specified number of times

- **while** - repeatedly runs a block of code while a condition is true
- **do...while** - runs a block of code once then repeats while a condition is true

The **for** statement has the form:

```
for (initialization; condition; increment) {
    statement(s)
}
```

The *initialization* is a statement that runs once at the beginning of the loop. The *condition* is evaluated at top of each iteration of the loop and the loop terminates when it evaluates to false. Finally, *increment* is a statement that runs after each iteration through the loop.

In this example, a *for* loop is used to print the values 0, 10, 20...90.

```
for (i=0; i<10; i++) {
    j = 10*i;
    print(j);
}
```

The braces can be omitted if there is only one statement in the loop.

```
for (i=0; i<=90; i+=10)
    print(i);
```

The **while** statement has the form:

```
while (condition) {
    statement(s)
}
```

First, the condition is evaluated. If it is true, the code block is executed and the *while* statement continues testing the condition and executing the code block until the condition becomes false.

In this example, a *while* loop is used to print the values 0, 10, 20...90.

```
i = 0;
while (i<=90) {
    print(i);
    i = i + 10;
}
```

The **do...while** statement has the form:

```
do {
    statement(s)
} while (condition);
```

Instead of evaluating the condition at the top of the loop, *do-while* evaluates it at the bottom. Thus the code block is always executed at least once.

In this example, a *do...while* loop is used to print the values 0, 10, 20...90.

```
i = 0;
do {
    print(i);
    i = i + 10;
} while (i<=90);
```

User-defined Functions

A function is a callable block of code that can be passed values and can return a value. The ImageJ macro language has two kinds of functions: **built-in** and user-defined. A user-defined function has the form:

```
function myFunction(arg1, arg2, arg3) {
    statement(s)
```

```
}
```

Functions can use the return statement to return a value.

```
function sum(a, b) {  
    return a + b;  
}
```

The number of arguments given when calling a function must correspond to the number of arguments in the function definition. The `sum()` function has two arguments so it must be called with two arguments.

```
print(sum(1, 2));
```

A function definition with no arguments must include the parentheses

```
function hello() {  
    print("Hello");  
}
```

and, unlike built-in functions, must be called with parentheses.

```
hello();
```

Basic data types (strings and numbers) are passed to a function by value; arrays are passed by reference.

Working with Strings

Use the `indexOf()` function to test to see if one string is contained in another. Use the `startsWith()` and `endsWith()` functions to see if a string starts with or ends with another string. Use the `substring()` function to extract a portion of string. Use the `lengthOf()` function to determine the length of a string.

Use the `"=="`, `"!="`, `">"` and `"<"` operators to compare strings. Comparisons are case-insensitive. For example, the following code displays *true*.

```
ans = "Yes";  
if (ans=="yes")  
    print ("true");  
else  
    print("false");
```

Extending the Macro Language

There are two ways to add functions implemented as Java code to the macro language. One is to use the `call` function to call static methods defined in a plugin. The other is to write a plugin that implements the `MacroExtension` interface. The `ImpProps` plugin demonstrates how to use the `call` function to get and set image properties and the `Image5D_Extensions` plugin demonstrates how to add `MacroExtension` (Ext) functions that work with the `Image5D` plugins.

Running Macros from the Command Line

You can run a macro from the command line and pass a string argument using the `-macro` or `-batch` command line options. As an example, this macro opens an image in the 'images' directory in the user's home directory:

```
name = getArgument;  
if (name=="") exit ("No argument!");  
path = getDirectory("home")+"images"+File.separator+name;  
setBatchMode(true);  
open(path);  
print(getTitle+": "+getWidth+"x"+getHeight);
```

Assume it is named 'OpenImage.txt' and it is located in the macros folder. Run the command

```
java -jar ij.jar -macro OpenImage blobs.tif
```

and ImageJ will launch and "blobs.tif: 256x254" is displayed in the Log window. Note that ImageJ assumed the '.txt' extension and the ImageJ/macros directory. Or run

```
java -jar ij.jar -batch OpenImage blobs.tif
```

and ImageJ does not launch and "blobs.tif: 256x254" is displayed in the terminal window.

A description of all the ImageJ command line options is available at rsb.info.nih.gov/ij/docs/install/linux.html#options.

Debugging Macros

You can debug a macro using the commands in the *Debug* menu, which was added to the macro editor in ImageJ 1.42. You start a debugging session by pressing ctrl-d (*Debug Macro*). You can then single step through the macro code by repeatedly pressing ctrl-e (*Step*).

There are seven commands in the *Debug* menu:

- **Debug Macro** - Starts running the macro in debug mode and opens the "Debug" window, which initially displays the memory usage, number of open images, and the active image's title. The macro stops running at the first executable line of code, which is highlighted. Use one of the following commands to continue execution.
- **Step** - Executes the highlighted statement and advances to the next. The variable names and values in the "Debug" window are updated.
- **Trace** - Runs the macro, displaying variable names and values in the "Debug" window as they are encountered.
- **Fast Trace** - Same as above, but faster.
- **Run** - Runs the macro to completion at normal speed.
- **Run to Insertion Point** - Runs the macro to a statement that was previously defined by clicking the mouse on an executable line of code.
- **Abort** - Exits the macro.

Use the [TestArrayInspector](#) macro to practice using the debugger. The Array Inspector (third window in the illustration below) requires ImageJ 1.48d or later.

