ImageJ Macro Language

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 with the mouse.

There are more than 200 example macros on the ImageJ Web site. To try one, open it in a browser window, copy it to the clipboard (crtl-a, ctrl-c), switch to ImageJ, open an editor window (ctrl-shift-n), paste (ctrl-v), then run it using the editor's File/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 (shift-n). In the dialog box, enter "Hello_World" as the Name, select "Macro" as the Type, then click "OK". (The underscore in the name is how ImageJ decides, when it is starting up, which ".txt" files in the plugins folder are to be installed as macros.)

In the editor window, enter the following line:

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

To test the macro, use the editor's File/Run Macro command (ctrl-R or cmd-R). To save it, use the editor's File/Save As command. 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/Edit commands, or drag and drop it on the "ImageJ" window (Windows) on the the ImageJ icon (Mac).

204 more Hello World programs are available on the ACM Hello World! Web site.

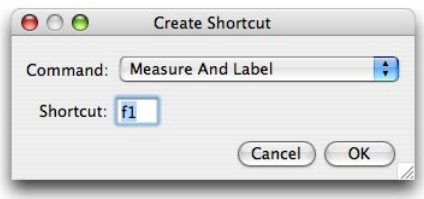
Using the Command Recorder to Generating Macros

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 Macro/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("This is Macro 2");
}
```

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 are are added to the ImageJ tool bar and executed when the user selects the tool and clicks on the image. Here is an example tool that displays the mouse click coordinates:

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

To install this tool, open an editor window (Plugins/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 six tool macros and any number of non-tool macros. Macro files must have a ".txt" extension. The cryptic hex code at the end of the macro name, which defines the tool icon, is described below.

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 - C00cO11cc" {
	getCursorLoc(x, y, z, flags);
	makeOval(x-radius, y-radius, radius*2, radius*2);
}

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

Tool macros with names ending in "Action Tool" perform an action when you click on their icon in the tool bar. 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?" {
	requires("1.37t");
	run("About ImageJ...");
}
```

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.

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
`Crgb`	set color
`Bxy`	set base location (default is (0,0))
`Rxywh`	draw rectangle
`Fxywh`	draw filled rectangle
`Oxywh`	draw oval
`oxywh`	draw filled oval
`Lxyxy`	draw line
`Dxy`	draw dot (1.32g or later)
`Pxyxy...xy0`	draw polyline
`Txyssc`	draw character

Where x (x coordinate), y (y coodinate), 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.

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);
}
```

Note that variable names are case-sensitive. "Name" and "name" are different variables.

Operators

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

Operator	Precendence	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

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 repeatadly 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);
```

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 display true.

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

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(argument1, argument2, etc) {
	statement(s)
}
```

Functions can use the return statement to return a value.

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

Functions defined with multiple arguments must be passed the same number of arguments. The sum() function has two arguments so it must called with two arguments.

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

A function with no arguments must include the parentheses

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

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

```
hello();
```

ImageJ - Built-in Macro Functions

A

abs(n)
Returns the absolute value of *n*.

acos(n)
Returns the arc cosine (in radians) of *n*.

asin(n)
Returns the arc sine (in radians) of *n*.

atan(n)
Calculates the inverse tangent (arc tangent) of *n*. Returns a value in the range -PI/2 through PI/2.

atan2(y, x)
Calculates the arctangent of *y/x* and returns an angle in the range -PI to PI, using the signs of the arguments to determine the quadrant. Multiply the result by 180/PI to convert to degrees.

autoUpdate(boolean)
If *boolean* is true, the display is refreshed each time lineTo(), drawLine(), drawString(), etc. are called, otherwise, the display is refreshed only when `updateDisplay()` is called or when the macro terminates.

B

beep()
Emits an audible beep.

bitDepth()
Returns the bit depth of the active image: 8, 16, 24 (RGB) or 32 (float).

C

calibrate(value)
Uses the calibration function of the active image to convert a raw pixel value to a density calibrated value. The argument must be an integer in the range 0-255 (for 8-bit images) or 0-65535 (for 16-bit images). Returns the same value if the active image does not have a calibration function.

call("class.method", arg1, arg2, ...)
Calls a public static method in a Java class, passing an arbitrary number of string arguments, and returning a string. Refer to CallJavaDemo for an example.

changeValues(v1, v2, v3)
Changes pixels in the image or selection that have a value in the range *v1-v2* to *v3*. For example, `changeValues(0,5,5)` changes all pixels less than 5 to 5, and `changeValues(0x0000ff,0x0000ff,0xff0000)` changes all blue pixels in an RGB image to red.

charCodeAt(string, index)
Returns the Unicode value of the character at the specified index in *string*. Index values can range from 0 to `lengthOf(`*`string`*`)`. Use the `fromCharCode()` function to convert one or more Unicode characters to a string.

close()
Closes the active image. This function has the advantage of not closing the "Log" or "Results" window when you meant to close the active image.

cos(angle)
Returns the cosine of an angle (in radians).

D

d2s(n, decimalPlaces)
Converts the number *n* into a string using the specified number of decimal places. Note that d2s stands for "double to string". This function will probably be replaced by one with a better name.

Dialog.create("Title")
Creates a dialog box with the specified title. Call `Dialog.addString()`, `Dialog.addNumber()`, etc. to add components to the dialog. Call `Dialog.show()` to display the dialog and `Dialog.getString()`, `Dialog.getNumber()`, etc. to retrieve the values entered by the user. Refer to the DialogDemo macro for an example.

> **Dialog.addMessage(string)**
> Adds a message to the dialog. The message can be broken into multiple lines by inserting new line characters ("\n") into the string.
>
> **Dialog.addString("Label", "Initial text")**
> Adds a text field to the dialog, using the specified label and intial text.
>
> **Dialog.addNumber("Label", default)**
> Adds a numeric field to the dialog, using the specified label and default value.
>
> **Dialog.addNumber("Label", default, decimalPlaces, columns, units)**
> Adds a numeric field, using the specified label and default value. *DecimalPlaces* specifies the number of digits to right of decimal point, *columns* specifies the the field width in characters and units is a string that is displayed to the right of the field.
>
> **Dialog.addCheckbox("Label", default)**
> Adds a checkbox to the dialog, using the specified label and default state (true or false).
>
> **Dialog.addChoice("Label", items)**
> Adds a popup menu to the dialog, where *items* is a string array containing the menu items.
>
> **Dialog.addChoice("Label", items, default)**
> Adds a popup menu, where *items* is a string array containing the choices and *default* is the default choice.
>
> **Dialog.show()**
> Displays the dialog and waits until the user clicks "OK" or "Cancel". The macro terminates if the user clicks "Cancel".
>
> **Dialog.getString()**
> Returns a string containing the contents of the next text field.
>
> **Dialog.getNumber()**
> Returns the contents of the next numeric field.
>
> **Dialog.getCheckbox()**
> Returns the state (true or false) of the next checkbox.
>
> **Dialog.getChoice()**
> Returns the selected item (a string) from the next popup menu.

doCommand("Command")
Runs an ImageJ menu command in a separate thread and returns immediately. As an example, `doCommand("Start Animation")` starts animating the current stack in a separate thread and the macro continues to execute. Use `run("Start Animation")` and the macro hangs until the user stops the animation.

doWand(x, y)
Equivalent to clicking on the current image at (x,y) with the wand tool. Note that some objects, especially one pixel wide lines, may not be reliably traced unless they have been thresholded (highlighted in red) using setThreshold.

drawLine(x1, y1, x2, y2)
Draws a line between (x1, y1) and (x2, y2). Use `setColor()` to specify the color of the line and `setLineWidth()` to vary the line width.

drawOval(x, y, width, height)
Draws the outline of an oval using the current color and line width. See also: `fillOval`, `setColor`, `setLineWidth`, `autoUpdate`. Requires 1.37e.

drawRect(x, y, width, height)
Draws the outline of a rectangle using the current color and line width. See also: `fillRect`, `setColor`, `setLineWidth`, `autoUpdate`. Requires 1.37e.

drawString("text", x, y)
Draws text at the specified location. Call setFont() to specify the font. Call `setJustification()` to have the text centered or right justified. Refer to the TextDemo macro for examples.

dump()
Writes the contents of the symbol table, the tokenized macro code and the variable stack to the "Log" window.

E

endsWith(string, suffix)
Returns *true* (1) if *string* ends with *suffix*. See also: startsWith, indexOf, substring.

eval(macro)
Evaluates (runs) one or more lines of macro code. See also: EvalDemo macro and `runMacro` function.

exit() or exit("error message")
Terminates execution of the macro and, optionally, displays an error message.

exp(n)
Returns the exponential number e (i.e., 2.718...) raised to the power of *n*.

F

File Functions
These functions allow you to get information about a file, read or write a text file, create a directory, or to delete, rename or move a file or directory. The FileDemo macro demonstrates how to use them. See also: `getDirectory` and `getFileList`.

File.close(f)
Closes the specified file, which must have been opened using `File.open()`.

File.dateLastModified(path)
Returns the date and time the specified file was last modified.

File.delete(path)
Deletes the specified file. If the file is a directory, it must be empty. The file must be in the user's home directory, the ImageJ directory or the temp directory.

File.exists(path)
Returns "1" (true) if the specified file exists.

File.getAbsolutePath(path)
Returns the full path of the file specified by *path*.

File.getName(path)
Returns the last name in *path*'s name sequence.

File.getParent(path)
Returns the parent of the file specified by *path*.

File.isDirectory(path)
Returns "1" (true) if the specified file is a directory.

File.lastModified(path)
Returns the time the specified file was last modified as the number of milliseconds since January 1, 1970.

File.length(path)
Returns the length in bytes of the specified file.

File.makeDirectory(path)
Creates a directory.

File.open(path)
Creates a new text file and returns a file variable that refers to it. To write to the file, pass the file variable to the print function. Displays a file save dialog box if *path* is an empty string. The file is closed when the macro exits. For an example, refer to the SaveTextFileDemo macro.

File.openAsString(path)
Opens a text file and returns the contents as a string. Displays a file open dialog box if *path* is an empty string. Use `lines=split(str,"\n")` to convert the string to an array of lines.

File.openUrlAsString(url)
Opens a URL and returns the contents as a string. Returns an emptly string if the host or file cannot be found. Requires 1.37q.

File.rename(path1, path2)
Renames, or moves, a file or directory. Returns "1" (true) if successful. The file or directory must be in the user's home directory, the ImageJ directory or the temp directory. Requires 1.38b.

File.separator
Returns the file name separator character ("/" or "\").

fill()
Fills the image or selection with the current drawing color.

fillOval(x, y, width, height)
Fills an oval bounded by the specified rectangle with the current drawing color. See also: `drawOval`, `setColor`, `autoUpdate`. Requires 1.37e.

fillRect(x, y, width, height)
Fills the specified rectangle with the current drawing color. See also: `drawRect`, `setColor`, `autoUpdate`.

floodFill(x, y)
Fills, with the foreground color, pixels that are connected to, and the same color as, the pixel at *(x, y)*. With 1.37e or later, does 8-connected filling if there is an optional string argument containing "8", for example `floodFill(x, y, "8-connected")`. This function is used to implement the flood fill (paint bucket) macro tool.

floor(n)
Returns the largest value that is not greater than *n* and is equal to an integer. See also: `round`.

fromCharCode(value1,...,valueN)
This function takes one or more Unicode values and returns a string.

G

getArgument()
Returns the string argument passed to macros called by `runMacro(macro, arg)`, `eval(macro)`, `IJ.runMacro(macro, arg)` or `IJ.runMacroFile(path, arg)`.

getBoolean("message")
Displays a dialog box containing the specified message and "Yes", "No" and "Cancel" buttons. Returns *true* (1) if the user clicks "Yes", returns *false* (0) if the user clicks "No" and exits the macro if the user clicks "Cancel".

getBoundingRect(x, y, width, height)
Replaced by `getSelectionBounds`.

getCursorLoc(x, y, z, modifiers)
Returns the cursor location in pixels and the mouse event modifier flags. The *z* coordinate is zero for 2D images. For stacks, it is one less than the slice number. For examples, see the GetCursorLocDemo and the GetCursorLocDemoTool macros.

getDateAndTime(year, month, dayOfWeek, dayOfMonth, hour, minute, second, msec)
Returns the current date and time. For an example, refer to the GetDateAndTime macro. See also: `getTime`.

getDirectory(title)
Returns the path to a specified directory. If *title* is "startup", returns the path to the directory that ImageJ was launched from (usually the ImageJ directory). If it is "plugins" or "macros", returns the path to the plugins or macros folder. If it is "image", returns the path to the directory that the active image was loaded from. If it is "home", returns the path to users home directory. If it is "temp", returns the path to the /tmp directory. Otherwise, displays a dialog (with *title* as the title), and returns the path to the directory selected by the user. Note that the path returned by getDirectory() ends with a file separator, either "\" (Windows) or "/". Returns an empty string if the specified directory is not found or aborts the macro if the user cancels the dialog box. For examples, see the GetDirectoryDemo and ListFilesRecursively macros.

getFileList(directory)
Returns an array containing the names of the files in the specified directory path. The names of subdirectories have a "/" appended. For an example, see the ListFilesRecursively macro.

getImageID()
Returns the unique ID (a negative number) of the active image. Use the *selectImage(id)*, *isOpen(id)* and *isActive(id)* functions to activate an image or to determine if it is open or active.

getImageInfo()
Returns a string containing the text that would be displayed by the *Image>Show Info* command. To retrieve the contents of a text window, use `getInfo("window.contents")`. For an example, see the ListDicomTags macros. See also: `getMetadata`.

getInfo("window.contents")
If the front window is a text window, returns the contents of that window. If the front window is an image, returns a string containing the text that would be displayed by *Image>Show Info*. Note that getImageInfo() is a more reliable way to retrieve information about an image. Use `split(getInfo(),'\n')` to break the string returned by this function into separate lines. Replaces the `getInfo()` function. Requires 1.38m.

getInfo("image.subtitle")
Returns the subtitle of the current image. This is the line of information displayed above the image and below the title bar. Requires 1.38m.

getInfo("slice.label")
Return the label of the current stack slice. This is the string that appears in parentheses in the subtitle, the line of information above the image. Returns an empty string if the current image is not a stack or the current slice does not have a label. Requires 1.38m.

getInfo(key)
Returns the Java property associated with the specified key (e.g., "java.version", "os.name", "user.home", "user.dir", etc.). Returns an empty string if there is no value associated with the key. See also: `getList("java.properties")`. Requires 1.38m.

getLine(x1, y1, x2, y2, lineWidth)
Returns the starting coordinates, ending coordinates and width of the current straight line selection. The coordinates and line width are in pixels. Sets x1 = -1 if there is no line selection. Refer to the GetLineDemo macro for an example.

getList("window.titles")
Returns a list (array) of non-image window titles. For an example, see the DisplayWindowTitles macro. Requires 1.38m.

getList("java.properties")
Returns a list (array) of Java property keys. For an example, see the DisplayJavaProperties macro. See also: `getInfo(key)`. Requires 1.38m.

getLocationAndSize(x, y, width, height)
Returns the location and size, in screen coordinates, of the active image window. Use getWidth and getHeight to get the width and height, in image coordinates, of the active image. See also: `setLocation`,

getHeight()
Returns the height in pixels of the current image.

getHistogram(values, counts, nBins[, histMin, histMax])
Returns the histogram of the current image or selection. *Values* is an array that will contain the pixel values for each of the histogram counts (or the bin starts for 16 and 32 bit images), or set this argument to 0. *Counts* is an array that will contain the histogram counts. *nBins* is the number of bins that will be used. It must be 256 for 8 bit and RGB image, or an integer greater than zero for 16 and 32 bit images. With 16-bit images, the *Values* argument is ignored if *nBins* is 65536. With 16-bit and 32-bit images, and ImageJ 1.35a and later, the histogram range can be specified using optional *histMin* and *histMax* arguments. See also: `getStatistics`, HistogramLister, HistogramPlotter, StackHistogramLister and CustomHistogram.

getLut(reds, greens, blues)
Returns three arrays containing the red, green and blue intensity values from the current lookup table. See the LookupTables macros for examples.

getMetadata()
Returns the metadata (a string) associated with the current image or stack slice. With stacks, the first line of the metadata is the slice label. Use getInfo("slice.label") to get just the slice label. With DICOM images and stacks, returns the metadata from the DICOM header. See also: `setMetadata`, `getImageInfo`.

getMinAndMax(min, max)
Returns the minimum and maximum displayed pixel values (display range). See the DisplayRangeMacros for examples.

getNumber("prompt", defaultValue)
Displays a dialog box and returns the number entered by the user. The first argument is the prompting message and the second is the value initially displayed in the dialog. Exits the macro if the user clicks on "Cancel" in the dialog. Returns *defaultValue* if the user enters an invalid number. See also: `Dialog.create`.

getPixel(x, y)
Returns the value of the pixel at *(x,y)*. Note that pixels in RGB images contain red, green and blue components that need to be extracted using shifting and masking. See the Color Picker Tool macro for an example that shows how to do this.

getPixelSize(unit, pixelWidth, pixelHeight)
Returns the unit of length (as a string) and the pixel dimensions. For an example, see the ShowImageInfo macro. See also: `getVoxelSize`.

getProfile()
Runs *Analyze>Plot Profile* (without displaying the plot) and returns the intensity values as an array. For an example, see the GetProfileExample macro.

getRawStatistics(nPixels, mean, min, max, std, histogram)
This function is similar to getStatistics except that the values returned are uncalibrated and the histogram of 16-bit images has a bin width of one and is returned as a *max*+1 element array. For examples, refer to the ShowStatistics macro set. See also: `calibrate`.

getResult("Column", row)
Returns a measurement from the ImageJ results table or NaN if the specified column is not found. The first argument specifies a column in the table. It must be a "Results" window column label, such as "Area", "Mean" or "Circ.". The second argument specifies the row, where 0<=*row*<nResults. *nResults* is a predefined variable that contains the current measurement count. (Actually, it's a built-in function with the "()" optional.) With ImageJ 1.34g and later, you can omit the second argument and have the row default to nResults-1 (the last row in the results table). See also: `nResults`, `setResult`, `isNaN`, `getResultLabel`.

getResultLabel(row)
Returns the label of the specified row in the results table or an empty string if the row does not have a label.

getSelectionBounds(x, y, width, height)
Returns the smallest rectangle that can completely contain the current selection. *x* and *y* are the pixel coordinates of the upper left corner of the rectangle. *width* and *height* are the width and height of the rectangle in pixels. If there is no selection, returns (0, 0, ImageWidth, ImageHeight). See also: `selectionType` and `setSelectionLocation`.

getSelectionCoordinates(xCoordinates, yCoordinates)
Returns two arrays containing the X and Y coordinates of the points that define the current selection. See the SelectionCoordinates macro for an example. See also: `selectionType`, `getSelectionBounds`.

getSliceNumber()
Returns the number of the currently displayed stack slice, an integer between 1 and nSlices. Returns 1 if the active image is not a stack. See also: `setSlice`.

getString("prompt", "default")
Displays a dialog box and returns the string entered by the user. The first argument is the prompting message and the second is the initial string value. Exits the macro if the user clicks on "Cancel" or enters an empty string. See also: `Dialog.create`.

getStatistics(area, mean, min, max, std, histogram)
Returns the area, average pixel value, minimum pixel value, maximum pixel value, standard deviation of the pixel values and histogram of the active image or selection. The histogram is returned as a 256 element array. For 8-bit and RGB images, the histogram bin width is one. For 16-bit and 32-bit images, the bin width is (*max*-*min*)/256. For examples, refer to the ShowStatistics macro set. Note that trailing arguments can be omitted. For example, you can use `getStatistics(area)`, `getStatistics(area, mean)` or `getStatistics(area, mean, min, max)`. See also: `getRawStatistics`

getThreshold(lower, upper)
Returns the lower and upper threshold levels. Both variables are set to -1 if the active image is not thresholded. See also: `setThreshold`, `getThreshold`, `resetThreshold`.

getTime()
Returns the current time in milliseconds. The granularity of the time varies considerably from one platform to the next. On Windows NT, 2K, XP it is about 10ms. On other Windows versions it can be as poor as 50ms. On many Unix platforms, including Mac OS X, it actually is 1ms. See also: `getDateAndTime`.

GetTitle()
Returns the title of the current image.

getVoxelSize(width, height, depth, unit)
Returns the voxel size and unit of length ("pixel", "mm", etc.) of the current image or stack. See also: `getPixelSize`, `setVoxelSize`.

getVersion()
Returns the ImageJ version number as a string (e.g., "1.34s"). See also: `requires`.

getWidth()
Returns the width in pixels of the current image.

getZoom()
Returns the magnification of the active image, a number in the range 0.03125 to 32.0 (3.1% to 3200%).

I

imageCalculator(operator, img1, img2)
Runs the *Process>Image Calculator* tool, where *operator* ("add","subtract","multiply","divide", "and", "or", "xor", "min", "max", "average", "difference" or "copy") specifies the operation, and *img1* and *img2* specify the operands. *img1* and *img2* can be either an image title (a string) or an image ID (an integer). The *operator* string can include up to three modifiers: "create" (e.g., "add create") causes the result to be stored in a new window, "32-bit" causes the result to be 32-bit floating-point and "stack" causes the entire stack to be processed. See the ImageCalculatorDemo macros for examples.

indexOf(string, substring)
Returns the index within *string* of the first occurrence of *substring*. See also: `lastIndexOf`, `startsWith`, `endsWith`, `substring`, `toLowerCase`, `replace`.

indexOf(string, substring, fromIndex)
Returns the index within *string* of the first occurrence of *substring*, with the search starting at *fromIndex*.

isActive(id)
Returns *true* if the image with the specified ID is active.

isKeyDown(key)
Returns *true* if the specified key is pressed, where *key* must be "shift", "alt" or "space". See also: `setKeyDown`.

isNaN(n)
Returns *true* if the value of the number *n* is NaN (Not-a-Number). A common way to create a NaN is to divide zero by zero. This function is required because (n==NaN) is always false. Note that the numeric constant NaN is predefined in the macro language.

isOpen(id)
Returns *true* if the image with the specified ID is open.

isOpen("Title")
Returns *true* if the window with the specified title is open.

L

lastIndexOf(string, substring)
Returns the index within *string* of the rightmost occurrence of *substring*. See also: `indexOf`, `startsWith`, `endsWith`, `substring`.

lengthOf(str)
Returns the length of a string or array.

lineTo(x, y)
Draws a line from current location to (*x,y*) .

log(n)
Returns the natural logarithm (base e) of *n*. Note that log10(n) = log(n)/log(10).

M

makeLine(x1, y1, x2, y2)
Creates a new straight line selection. The origin (0,0) is assumed to be the upper left corner of the image. Coordinates are in pixels. With ImageJ 1.35b and letter, you can create segmented line selections by specifying more than two coordinate, for example `makeLine(25,34,44,19,69,30,71,56)`.

makeOval(x, y, width, height)
Creates an elliptical selection, where (*x,y*) define the upper left corner of the bounding rectangle of the ellipse.

makePolygon(x1, y1, x2, y2, x3, y3, ...)
Creates a polygonal selection. At least three coordinate pairs must be specified, but not more than 200. As an example, `makePolygon(20,48,59,13,101,40,75,77,38,70)` creates a polygon selection with five sides.

makeRectangle(x, y, width, height)
Creates a rectangular selection. The *x* and *y* arguments are the coordinates (in pixels) of the upper left corner of the selection. The origin (0,0) of the coordinate system is the upper left corner of the image.

makeSelection(type, xcoord, ycoord)
Creates a selection from a list of XY coordinates. The first argument should be "polygon", "freehand", "polyline", "freeline", "angle" or "point". In ImageJ 1.32g or later, it can also be the numeric value returned by selectionType. The *xcoord* and *ycoord* arguments are numeric arrays that contain the X and Y coordinates. See the MakeSelectionDemo macro for examples.

maxOf(n1, n2)
Returns the greater of two values.

minOf(n1, n2)
Returns the smaller of two values.

moveTo(x, y)
Sets the current drawing location. The origin is always assumed to be the upper left corner of the image.

N

newArray(size)
Returns a new array containing *size* elements. You can also create arrays by listing the elements, for example `newArray(1,4,7)` or `newArray("a","b","c")`. For more examples, see the Arrays macro. The ImageJ macro language does not directly support 2D arrays. As a work around, either create a blank image and use `setPixel()` and `getPixel()`, or create a 1D array using `a=newArray(xmax*ymax)` and do your own indexing (e.g., `value=a[x+y*xmax]`).

newImage(title, type, width, height, depth)
Opens a new image or stack using the name *title*. The string *type* should contain "8-bit", "16-bit", "32-bit" or "RGB". In addition, it can contain "white", "black" or "ramp" (the default is "white"). As an example, use "16-bit ramp" to create a 16-bit image containing a grayscale ramp. *Width* and *height* specify the width and height of the image in pixels. *Depth* specifies the number of stack slices.

newMenu(macroName, stringArray)
Defines a menu that will be added to the toolbar when the menu tool named *macroName* is installed. Menu tools are macros with names ending in "Menu Tool". *StringArray* is an array containing the menu commands. Returns a copy of *stringArray*. For an example, refer to the Menus toolset. Requires v1.38b or later.

nImages
Returns number of open images. The parentheses "()" are optional.

nResults
Returns the current measurement counter value. The parentheses "()" are optional.

nSlices
Returns the number of slices in the current stack. Returns 1 if the current image is not a stack. The parentheses "()" are optional. See also: `getSliceNumber`.

O

open(path)
Opens and displays a tiff, dicom, fits, pgm, jpeg, bmp, gif, lut, roi, or text file. Displays an error message and aborts the macro if the specified file is not in one of the supported formats, or if the file is not found. Displays a file open dialog box if *path* is an empty string or if there is no argument. Use the *File>Open* command with the command recorder running to generate calls to this function.

P

parseFloat(string)
Converts the string argument to a number and returns it. Returns NaN (Not a Number) if the string cannot be converted into a number. Use the isNaN() function to test for NaN. For examples, see ParseFloatIntExamples.

parseInt(string)
Converts *string* to an integer and returns it. Returns NaN if the string cannot be converted into a integer.

parseInt(string, radix)
Converts *string* to an integer and returns it. The optional second argument (*radix*) specifies the base of the number contained in the string. The radix must be an integer between 2 and 36. For radixes above 10, the letters of the alphabet indicate numerals greater than 9. Set *radix* to 16 to parse hexadecimal numbers. Returns NaN if the string cannot be converted into a integer. For examples, see ParseFloatIntExamples.

Plot.create("Title", "X-axis Label", "Y-axis Label", xValues, yValues)
Generates a plot using the specified title, axis labels and X and Y coordinate arrays. If only one array is specified it is assumed to contain the Y values and a 0..n-1 sequence is used as the X values. It is also permissible to specify no arrays and use `Plot.setLimits()` and `Plot.add()` to generate the plot. Use `Plot.show()` to display the plot in a window or it will be displayed automatically when the macro exits. For examples, check out the ExamplePlots macro file.

Plot.setLimits(xMin, xMax, yMin, yMax)
Sets the range of the x-axis and y-axis of plots created using `Plot.create()`.

Plot.setLineWidth(width)
Specifies the width of the line used to draw a curve. Points (circle, box, etc.) are also drawn larger if a line width greater than one is specified. Note that the curve specified in `Plot.create()` is the last one drawn before the plot is dispayed or updated.

Plot.setColor("red")
Specifies the color used in subsequent calls to `Plot.add()` or `Plot.addText()`. The argument can be "black", "blue", "cyan", "darkGray", "gray", "green", "lightGray", "magenta", "orange", "pink", "red", "white" or "yellow". Note that the curve specified in `Plot.create()` is drawn last.

Plot.add("circles", xValues, yValues)
Adds a curve, set of points or error bars to a plot created using `Plot.create()`. If only one array is specified it is assumed to contain the Y values and a 0..n-1 sequence is used as the X values. The first argument can be "line", "circles", "boxes", "triangles", "crosses", "dots", "x" or "error bars".

Plot.addText("A line of text", x, y)
Adds text to the plot at the specified location, where (0,0) is the upper left corner of the the plot frame and (1,1) is the lower right corner. Call `Plot.setJustification()` to have the text centered or right justified.

Plot.setJustification("center")
Specifies the justification used by `Plot.addText()`. The argument can be "left", "right" or "center". The default is "left".

Plot.show()
Displays the plot generated by `Plot.create()`, `Plot.add()`, etc. in a window. This function is automatically called when a macro exits.

Plot.update()
Draws the plot generated by `Plot.create()`, `Plot.add()`, etc. in an existing plot window. Equivalent to `Plot.show()` if no plot window is open.

pow(base, exponent)
Returns the value of *base* raised to the power of *exponent*.

print(string)
Outputs a string to the "Log" window. Numeric arguments are automatically converted to strings. Starting with ImageJ v1.34b, `print()` accepts multiple arguments. For example, you can use `print(x,y,width, height)` instead of `print(x+" "+y+" "+width+" "+height)`. If the first argument is a file handle returned by File.open(path), then the second is saved in the refered file (see SaveTextFileDemo). Starting with ImageJ 1.37j, `print()` accepts commands such as *"\\Clear"* and *"\\Update:<text>"* that clear the "Log" window and update its contents. Refer to the LogWindowTricks macro for an example. Starting with ImageJ 1.38m, the second argument to `print(arg1, arg2)` is appended to a text window or table if the first argument is a window title in brackets, for example `print("[My Window]", "Hello, world")`. With text windows, newline characters ("\n") are not automatically appended and text that starts with *"\\Update:"* replaces the entire contents of the window. Refer to the PrintToTextWindow, Clock and ProgressBar macros for examples. The second argument to `print(arg1, arg2)` is appended to a table (e.g., ResultsTable) if the first argument is the title of the table in brackets. Use the *Plugins>New* command to open a blank table. Any command that can be sent to the "Log" window (*"\\Clear"*, *"\\Update:<text>"* , etc.) can also be sent to a table. Refer to the SineCosineTable2 and TableTricks macros for examples.

R

random
Returns a random number between 0 and 1.

rename(name)
Changes the title of the active image to the string *name*.

replace(string, old, new)
Returns the new string that results from replacing all occurrences of *old* in *string* with *new*, where *old* and *new* are single character strings. With Java 1.4 and later, replaces each substring of *string* that matches the regular expression *old* with *new*.

requires("1.29p")
Display a message and aborts the macro if the ImageJ version is less than the one specified. See also: `getVersion`.

reset
Restores the backup image created by the snapshot function. Note that `reset()` and `run("Undo")` are basically the same, so only one `run()` call can be reset.

resetMinAndMax
Resets the minimum and maximum displayed pixel values (display range) to be the same as the current image's minimum and maximum pixel values. See the DisplayRangeMacros for examples.

resetThreshold
Disables thresholding. See also: `setThreshold`, `setAutoThreshold`, `getThreshold`.

restorePreviousTool
Switches back to the previously selected tool. Useful for creating a tool macro that performs an action, such as opening a file, when the user clicks on the tool icon.

restoreSettings
Restores *Edit/Options* submenu settings saved by the `saveSettings()` function.

roiManager(cmd)
Runs an ROI Manager command, where *cmd* must be "Add", "Add & Draw", "Deselect", "Measure", "Draw" or "Combine". The ROI Manager is opened if it is not already open. Use `roiManager("reset")` to delete all items on the list. With v1.37u or later, use `setOption("Show All", boolean)` to enable/disable "Show All" mode. For examples, refer to the RoiManagerMacros, RoiManagerAddParticles and ROI Manager Stack Demo macros.

roiManager(cmd, name)
Runs an ROI Manager I/O command, where *cmd* is "Open", "Save or "Rename", and *name* is a file path or name. With v1.35c or later, the "Save" option ignores selections and saves all the ROIs as a ZIP archive. The "Rename" option requires v1.37h or later. With v1.37i or later, you can get the selection name using `call("ij.plugin.frame.RoiManager.getName", index)`. The ROI Manager is opened if it is not already open.

roiManager("count")
Returns the number of items in the ROI Manager list.

roiManager("index")
Returns the index of the currently selected item on the ROI Manager list, or -1 if the list is empty, no items are selected, or more than one item is selected. Requires v1.37q.

roiManager("select", index)
Selects an item in the ROI Manager list, where *index* must be greater than or equal zero and less than the value returned by `roiManager("count")`. Use `roiManager("deselect")` to deselect all items on the list. For an example, refer to the ROI Manager Stack Demo macro.

round(n)
Returns the closest integer to *n*. See also: `floor`.

run("command"[, "options"])
Executes an ImageJ menu command. The optional second argument contains values that are automatically entered into dialog boxes (must be GenericDialog or OpenDialog). Use the Command Recorder (*Plugins>Macros>Record*) to generate run() function calls. Use string concatentation to pass a variable as an argument. For examples, see the ArgumentPassingDemo macro.

runMacro(name)
Runs the specified macro file, which is assumed to be in the Image macros folder. A full file path may also be used. The ".txt" extension is optional. Returns any string argument returned by the macro. May have an optional second string argument that is passed to macro. For an example, see the CalculateMean macro. See also: `eval` and `getArgument`.

S

save(path)
Saves an image, lookup table, selection or text window to the specified file path. The path must end in ".tif", ".jpg", ".gif", ".zip", ".raw", ".avi", ".bmp", ".fits", ".png", ".pgm", ".lut", ".roi" or ".txt".

saveAs(format, path)
Saves the active image, lookup table, selection, measurement results, selection XY coordinates or text window to the specified file path. The *format* argument must be "tiff", "jpeg", "gif", "zip", "raw", "avi", "bmp", "fits", "png", "pgm", "text image", "lut", "selection", "measurements", "xy Coordinates" or "text". Use `saveAs(format)` to have a "Save As" dialog displayed.

saveSettings()
Saves most *Edit/Options* submenu settings so they can be restored later by calling `restoreSettings()`.

screenHeight
Returns the screen height in pixels. See also: `getLocationAndSize`, `setLocation`.

screenWidth
Returns the screen width in pixels.

selectionName
Returns the name of the current selection, or an empty string if the selection does not have a name. Aborts the macro if there is no selection. See also: `setSelectionName` and `selectionType`.

selectionType()
Returns the selection type, where 0=rectangle, 1=oval, 2=polygon, 3=freehand, 4=traced, 5=straight line, 6=segmented line, 7=freehand line, 8=angle, 9=composite and 10=point. Returns -1 if there is no selection. For an example, see the ShowImageInfo macro.

selectImage(id)
Activates the image with the specified ID (a negative number). If *id* is greater than zero, activates the *id*th image listed in the Window menu. With ImageJ 1.33n and later, *id* can be an image title (a string).

selectWindow("name")
Activates the image window with the title "name". Also activates non-image windows in v1.30n or later.

setAutoThreshold()
Sets the threshold levels based on an analysis of the histogram of the current image or selection. Equivalent to clicking on "Auto" in the *Image>Adjust>Threshold* window. See also: `setThreshold`, `getThreshold`, `resetThreshold`.

setBackgroundColor(r, g, b)
Sets the background color, where *r*, *g*, and *b* are >= 0 and <= 255.

setBatchMode(arg)
If *arg* is *true*, the interpreter enters batch mode and images are not displayed, allowing the macro to run up to 20 times faster. If *arg* is *false*, exits batch mode and displays the active image in a window. ImageJ exits batch mode when the macro terminates if there is no `setBatchMode(false)` call. With ImageJ 1.37j or later, set *arg* to *"exit and display"* to exit batch mode and display all open batch mode images. Here are five example batch mode macros: BatchModeTest, BatchMeasure, BatchSetScale, ExpandOrShrinkSelection and ReplaceRedWithMagenta.

setColor(r, g, b)
Sets the drawing color, where *r*, *g*, and *b* are >= 0 and <= 255. With 16 and 32 bit images, sets the color to 0 if r=g=b=0. With 16 and 32 bit images, use `setColor(1,0,0)` to make the drawing color the same is the minimum displayed pixel value. `SetColor()` is faster than `setForegroundColor()`, and it does not change the system wide foreground color or repaint the color picker tool icon, but it cannot be used to specify the color used by commands called from macros, for example *run("Fill")*.

setColor(value)
Sets the drawing color. For 8 bit images, 0<=*value*<=255. For 16 bit images, 0<=*value*<=65535. For RGB images, use hex constants (e.g., 0xff0000 for red). This function does not change the foreground color used by `run("Draw")` and `run("Fill")`.

setFont(name, size[, style])
The first argument is the font name. It should be "SansSerif", "Serif" or "Monospaced". The second argument is the point size of the font. The optional third argument is a string containing "bold" or "italic", or both. With ImageJ 1.37e or later, the third argument can also contain the keyword "antialiased". For examples, run the TextDemo macro.

setForegroundColor(r, g, b)
Sets the foreground color, where *r*, *g*, and *b* are >= 0 and <= 255. See also: `setColor`.

setJustification("center")
Specifies the justification used by `drawString()` and `Plot.addText()`. The argument can be "left", "right" or "center". The default is "left".

setKeyDown(keys)
Simulates pressing the shift, alt or space keys, where *keys* is a string containing some combination of "shift", "alt" or "space". Any key not specified is set "up". Use `setKeyDown("none")` to set all keys in the "up" position. With ImageJ 1.38e or later, call `setKeyDown("esc")` to abort the currently running macro or plugin. For examples, see the CompositeSelections, DoWandDemo and AbortMacroActionTool macros. See also: `isKeyDown`.

setLineWidth(width)
Specifies the line width (in pixels) used by `drawLine()`, `lineTo()`, `drawRect()` and `drawOval()`.

setLocation(x, y)
Moves the active image window to a new location. See also: `getLocationAndSize`, `screenWidth`, `screenHeight`.

setLut(reds, greens, blues)
Creates a new lookup table and assigns it to the current image. Three input arrays are required, each containing 256 intensity values. See the LookupTables macros for examples.

setMetadata(string)
Assigns the metadata contained in the specified string to the the current image or stack slice. With stacks, the first 15 characters, or up to the first newline, are used as the slice label. The metadata is always assigned to the "Info" image property (displayed by *Image>Show Info*) if *string* starts with "Info:". Note that the metadata is saved as part of the TIFF header when the image is saved. See also: `getMetadata`.

setMinAndMax(min, max)
Sets the minimum and maximum displayed pixel values (display range). See the DisplayRangeMacros for examples.

setOption(option, boolean)
Enables or disables an ImageJ option, where *option* is "DisablePopupMenu", "DebugMode", "Show All", "Changes", "OpenUsingPlugins" or "QueueMacros", and *boolean* is either *true* or *false*. "Show All", added in v1.37u, enables/disables the ROI Manager's "Show All" mode. "Changes", added in v1.38a, sets/resets the 'changes' flag of the current image. "OpenUsingPlugins", added in v1.38f, controls whether standard file types (TIFF, JPEG, GIF, etc.) are opened by external plugins or by ImageJ (example). The "QueueMacros" option, added in v1.38g, controls whether macros invoked using keyboard shortcuts run sequentially on the event dispatch thread (EDT) or in separate, possibly concurrent, threads (example). In "QueueMacros" mode, screen updates, which also run on the EDT, are delayed until the macro finishes. The "DisableUndo" option, added in v1.38h, disables/enables the *Edit>Undo* command. Note that a `setOption("DisableUndo",true)` call without a corresponding `setOption("DisableUndo",false)` will cause *Edit>Undo* to not work as expected until ImageJ is restarted.

setPasteMode(mode)
Sets the transfer mode used by the *Edit>Paste* command, where 'mode' is "Copy", "Blend", "Average", "Difference", "Transparent", "AND", "OR", "XOR", "Add", "Subtract", "Multiply", or "Divide". In v1.37a or later, 'mode' can also be "Min" or "Max".

setPixel(x, y, value)
Stores *value* at location (*x,y*) of the current image. The screen is updated when the macro exits or call `updateDisplay()` to have it updated immediately.

setResult("Column", row, value)
Adds an entry to the ImageJ results table or modifies an existing entry. The first argument specifies a column in the table. If the specified column does not exist, it is added. The second argument specifies the row, where 0<=*row*<=nResults. (*nResults* is a predefined variable.) A row is added to the table if *row=nResults*. The third argument is the value to be added or modified. Call *setResult("Label", row, string)* to set the row label. Call *updateResults()* to display the updated table in the "Results" window. For examples, see the SineCosineTable and ConvexitySolidarity macros.

setRGBWeights(redWeight, greenWeight, blueWeight)
Sets the weighting factors used by the *Analyze>Measure*, *Image>Type>8-bit* and *Analyze>Histogram* commands when they convert RGB pixels values to grayscale. The sum of the weights must be 1.0. Use *(1/3,1/3,1/3)* for equal weighting of red, green and blue. The weighting factors in effect when the macro started are restored when it terminates. For examples, see the MeasureRGB, ShowRGBChannels and RGB_Histogram macros.

setSelectionLocation(x, y)
Moves the current selection to *(x,y)*, where *x* and *y* are the pixel coordinates of the upper left corner of the selection's bounding rectangle. The RoiManagerMoveSelections macro uses this function to move all the ROI Manager selections a specified distance. See also: `getSelectionBounds`.

setSelectionName(name)
Sets the name of the current selection to the specified name. Aborts the macro if there is no selection. See also: `selectionName` and `selectionType`.

setSlice(n)
Displays the *n*th slice of the active stack. Does nothing if the active image is not a stack. For an example, see the MeasureStack macros. See also: `getSliceNumber`, `nSlices`.

setThreshold(lower, upper)
Sets the lower and upper threshold levels. Starting with ImageJ 1.34g, there is an optional third argument that can be "red", "black & white", "over/under" or "no color". See also: `setAutoThreshold`, `getThreshold`, `resetThreshold`.

setTool(id)
Switches to the specified tool, where 0=rectangle, 1=oval, 2=polygon, 3=freehand, 4=straight line, 5=polyline, 6=freeline, 7=point, 8=wand, 9=text, 10=spare, 11=zoom, 12=hand, 13=dropper, 14=angle, 15...19=spare. See also: `toolID`.

setupUndo()
Call this function before drawing on an image to allow the user the option of later restoring the original image using *Edit/Undo*. Note that `setupUndo()` may not work as intended with macros that call the run() function. For an example, see the DrawingTools tool set.

setVoxelSize(width, height, depth, unit)
Defines the voxel dimensions and unit of length ("pixel", "mm", etc.) for the current image or stack. The *depth* argument is ignored if the current image is not a stack. See also: `getVoxelSize`.

setZCoordinate(z)
Sets the Z coordinate used by `getPixel()`, `setPixel()` and `changeValues()`. The argument must be in the range 0 to n-1, where n is the number of images in the stack. For an examples, see the Z Profile Plotter Tool.

showMessage("message")
Displays "message" in a dialog box.

showMessage("title", "message")
Displays "message" in a dialog box using "title" as the the dialog box title.

showMessageWithCancel(["title",]"message")
Displays "message" in a dialog box with "OK" and "Cancel" buttons. "Title" (optional) is the dialog box title. The macro exits if the user clicks "Cancel" button.

showProgress(progress)
Updates the ImageJ progress bar, where 0.0<=*progress*<=1.0. The progress bar is not displayed if the time between the first and second calls to this function is less than 30 milliseconds. It is erased when the macro terminates or *progress* is >=1.0.

showStatus("message")
Displays a message in the ImageJ status bar.

sin(angle)
Returns the sine of an angle (in radians).

snapshot()
Creates a backup copy of the current image that can be later restored using the reset function. For examples, see the ImageRotator and BouncingBar macros.

split(string, delimiters)
Breaks a string into an array of substrings. *Delimiters* is a string containing one or more delimiter characters. The default delimiter set " \t\n\r" (space, tab, newline, return) is used if *delimiters* is an empty string or split is called with only one argument. Returns a one element array if no delimiter is found.

sqrt(n)
Returns the square root of *n*. Returns NaN if *n* is less than zero.

startsWith(string, prefix)
Returns *true* (1) if *string* starts with *prefix*. See also: `endsWith`, `indexOf`, `substring`, `toLowerCase`.

substring(string, index1, index2)
Returns a new string that is a substring of *string*. The substring begins at *index1* and extends to the character at *index2* - 1. See also: `indexOf`, `startsWith`, `endsWith`, `replace`.

T

tan(angle)
Returns the tangent of an angle (in radians).

toBinary(number)
Returns a binary string representation of *number*.

toHex(number)
Returns a hexadecimal string representation of *number*.

toLowerCase(string)
Returns a new string that is a copy of *string* with all the characters converted to lower case.

toolID
Returns the ID of the currently selected tool. See also: `setTool`.

toString(number)
Returns a decimal string representation of *number*. See also: `toBinary`, `toHex`, `parseFloat` and `parseInt`.

toUpperCase(string)
Returns a new string that is a copy of *string* with all the characters converted to upper case.

U

updateDisplay()
Redraws the active image.

updateResults()
Call this function to update the "Results" window after the results table has been modified by calls to the `setResult()` function.

W

`wait(n)`
Delays (sleeps) for *n* milliseconds.

`write(string)`
Outputs a string to the "Results" window. Numeric arguments are automatically converted to strings.