Built-in Macro Functions

[A][B][C][D][E][F][G][H][I][J][K][L][M] [N][O][P] **Print List**
[Q][R][S][T][U][V][W][X][Y][Z]

A [Top]

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

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

Array Functions
These functions operate on arrays. Refer to the ArrayFunctions macro for examples.

Array.concat(array1, array2) - Returns a new array created by joining two or more arrays or values (examples).
Array.copy(array) - Returns a copy of *array*.
Array.deleteValue(array, value) - Returns a version of *array* where all numeric or string elements in the array that contain *value* have been deleted (examples).
Array.deleteIndex(array, index) - Returns a version of *array* where the element with the specified index has been deleted.
Array.fill(array, value) - Assigns the specified numeric value to each element of *array*.
Array.filter(array, filter) Returns an array containing the elements of 'array' that contain 'filter', where 'array' is an array of strings. Enclose the filter in parans to do regular expression matching. Requires 1.53f.
Array.findMaxima(array, tolerance) - Returns an array holding the peak positions (sorted with descending strength). 'Tolerance' is the minimum amplitude difference needed to separate two peaks. With v1.51n and later, there is an optional 'edgeMode' argument: 0=include edges, 1=exclude edges(default), 2=circular array. Examples.
Array.findMinima(array, tolerance) - Returns an array holding the minima positions.
Array.fourier(array, windowType) - Calculates and returns the Fourier amplitudes of *array*. *WindowType* can be "none", "Hamming", "Hann", or "flat-top", or may be omitted (meaning "none"). See the TestArrayFourier macro for an example and more documentation.
Array.getSequence(n) - Returns an array containing the numeric sequence 0,1,2...n-1.
Array.getStatistics(array, min, max, mean, stdDev) - Returns the *min*, *max*, *mean*, and *stdDev* of *array*, which must contain all numbers.
Array.print(array) - Prints the array on a single line.
Array.rankPositions(array) - Returns, as an array, the rank position indexes of *array*, starting with the index of the smallest value (example).
Array.resample(array, len) - Returns an array which is linearly resampled to a different length.
Array.reverse(array) - Reverses (inverts) the order of the elements in *array*.
Array.show(array) - Displays the contents of *array* in a window.

Array.show("title", array1, array2, ...) - Displays one or more arrays in a Results window (examples). If *title* (optional) is "Results", the window will be the active Results window, otherwise, it will be a dormant Results window (see also IJ.renameResults). If *title* ends with "(indexes)", a 0-based Index column is shown. If *title* ends with "(row numbers)", the row number column is shown.
Array.slice(array, start, end) - Extracts a part of an array and returns it. (examples).
Array.sort(array) - Sorts *array*, which must contain all numbers or all strings. String sorts are case-insensitive.
Array.sort(array1, array2, array3...) - Sorts multiple arrays, where all the arrays adopt the sort order of *array1* (example).
Array.trim(array, n) - Returns an array that contains the first *n* elements of *array*.
Array.rotate(array, d) - Rotates the array elements by 'd' steps (positive 'd' = rotate right). Requires 1.51n. Examples.
Array.getVertexAngles(xArr, yArr, arm) - From a closed contour given by 'xArr', 'yArr', an array is returned holding vertex angles in degrees (straight=0, convex = positive if contour is clockwise). Set 'arm'=1 to calculate the angle from the closest vertex points left and right, or use arm>1 for more distant neighbours and smoother results. Requires 1.51n. Examples.

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

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

atan2(y, x)
Calculates the inverse tangent 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 [Top]

beep()
Emits an audible beep.

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

C [Top]

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 the CallJavaDemo macro and the ImpProps plugin for examples.

call("class.method")
Calls a public static no-argument method in a Java class and returns a string.

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. In ImageJ 1.52d or later, use changeValues(NaN, NaN, value) to replaces NaN values.

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. Use *run("Close")* to close non-image windows.

close(pattern)
Closes windows whose title matches 'pattern', which can contain the wildcard characters '*' (matches any character sequence) and '?' (matches single character). For example, close("Histo*") could be used to dispose all histogram windows. Non-image windows like "Roi Manager" have to be specified without wildcards. For text windows, wildcards are allowed if 'pattern' ends with ".txt", ".ijm", ".js" etc. Use close("*") to close all image windows. Use close(pattern, "keep") to not close text or image windows with changes. If 'pattern' is "\\Others", all images except the front image are closed. The most recent macro window is never closed.

close("*")
Closes all image windows.

close("\\Others")
Closes all images except for the front image.

Color Functions
Functions for working with colors, available in ImageJ 1.53h and later.

> **Color.set(string)**
> Sets the drawing color, where 'string' is "red", "blue", etc., or a hex value like "#ff0000".
> **Color.set(value)**
> Sets the drawing color. For 8 bit images, 0<=*value*<=255. For 16 bit images, 0<=*value*<=65535. With RGB images, use hex constants (e.g., 0xff0000 for red).
> **Color.setForeground(string)**
> Sets the foreground color, where 'string' is "red", "blue", etc., or a hex value like "#ff0000".
> **Color.setForegroundValue(value)**

Sets the foreground color to grayscale, where 'value' is a number between 0 and 255.

Color.setBackground(string)
Sets the background color, where 'string' is "red", "blue", etc., or a hex value like "#ff0000".

Color.setBackgroundValue(value)
Sets the background color to grayscale, where 'value' is a number between 0 (black) and 255 (white).

Color.setForeground(r, g, b)
Sets the foreground color, where *r*, *g*, and *b* are >= 0 and <= 255.

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

Color.foreground
Returns the foreground color as a string, for example "red" or "#ff0000".

Color.background
Returns the background color as a string, for example "cyan" or "#00ffff".

Color.toString(r, g, b)
Converts an r,g,b color to a string.

Color.toArray(string)
Converts a color (e.g., "red" or "#ff0000") to a three element array.

Color.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.

Color.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.

Color.wavelengthToColor(wavelength)
Converts a wavelength (380-750 nm) into a color in string format (e.g., "#0031ff"). See also: Help>Examples>Plots>Plot With Spectrum.

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

D [Top]

d2s(n, decimalPlaces)
Converts the number *n* into a string using the specified number of decimal places. Uses scientific notation if 'decimalPlaces is negative. Note that d2s stands for "double to string".

debug(arg)
Calls the macro debugger, where 'arg' is "break" (the default), "run", "trace", "fast-trace", "dump" or "throw". Call debug("dump") to display the contents of the symbol table, the tokenized macro code and the variable stack. Call debug("throw") to display an example "Exception" window.

Dialog.create("Title")
Creates a modal dialog box with the specified title, or use *Dialog.createNonBlocking("Title")* to create a non-modal dialog. 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.createNonBlocking("Title") - Creates a non-modal dialog box with the specified title.
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.addMessage(string, fontSize, fontColor) - Adds a message to the dialog using a specified font size and color (example). The message can be broken into multiple lines by inserting new line characters ("\n") into the string. The 'fontSize' and 'fontColor' arguments are optional.
Dialog.addString(label, initialText) - Adds a text field to the dialog, using the specified label and initial text.
Dialog.addString(label, initialText, columns) - Adds a text field to the dialog, where *columns* specifies the field width in characters.
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.addSlider(label, min, max, default) - Adds a slider controlled numeric field to the dialog, using the specified label, and min, max and default values (example). Values with decimal points are used when (max-min)<=5 and min, max or default are non-integer.
Dialog.addCheckbox(label, default) - Adds a checkbox to the dialog, using the specified label and default state (true or false).
Dialog.addCheckboxGroup(rows, columns, labels, defaults) - Adds a *rows*x*columns* grid of checkboxes to the dialog, using the specified labels and default states (example).
Dialog.addRadioButtonGroup(label, items, rows, columns, default) - Adds a group of radio buttons to the dialog, where 'label' is the group label, 'items' is an array containing the button labels, 'rows' and 'columns' specify the grid size, and 'default' is the label of the default button. (example).
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.addDirectory(label, defaultPath) - Adds a directory field and "Browse" button. The field width is determined by the length of 'defaultPath', with a minimum of 25 columns. Use Dialog getString to retrieve the directory path. Use File.setDefaultDir() to set the default directory used when the user clicks on "Browse". Requires 1.53d.
Dialog.addFile(label, defaultPath) - Adds a file field and "Browse" button. The field width is determined by the length of 'defaultPath', with a minimum of 25 columns. Use Dialog getString to retrieve the file path. Requires 1.53d.
Dialog.addImage(pathOrURL) - Adds an image to the dialog. The 'pathOrURL' argument can be a file path (e.g., "/Users/wayne/Downloads/blobs.tif") or a URL (e.g., "https://imagej.net/images/blobs.gif").
Dialog.addImageChoice(label) - Adds a popup menu that lists the currently open images. Use Dialog getImageChoice() to retrieve the title

of the selected image. The macro aborts if there are no open images. Requires 1.53d.
Dialog.addImageChoice(label, defaultImage) - This a version of addImageChoice() with a second argument for specifying the selected image in the dropdown menu. Requires 1.53s.
Dialog.addHelp(url) - Adds a "Help" button that opens the specified URL in the default browser. This can be used to supply a help page for this dialog or macro. With v1.46b or later, displays an HTML formatted message if 'url' starts with "<html>" (example).
Dialog.addToSameRow() - Makes the next item added appear on the same row as the previous item. May be used for addNumericField, addSlider, addChoice, addCheckbox, addString, addMessage, and before the showDialog() method. In the latter case, the buttons appear to the right of the previous item. Note that *addMessage* (and *addString* if a width of more than 8 is specified) use the remaining width, so it must be the last item of a row. Requires 1.51r.
Dialog.setInsets(top, left, bottom) - Overrides the default insets (margins) used for the next component added to the dialog.
Default insets:
addMessage: 0,20,0 (empty string) or 10,20,0
addCheckbox: 15,20,0 (first checkbox) or 0,20,0
addCheckboxGroup: 10,0,0
addNumericField: 5,0,3 (first field) or 0,0,3
addStringField: 5,0,5 (first field) or 0,0,5
addChoice: 5,0,5 (first field) or 0,0,5
Dialog.setLocation(x, y) - Sets the screen location where this dialog will be displayed.
Dialog.getLocation(x, y) - Returns the screen location of this dialog. Requires 1.54g.
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, directory or file 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.
Dialog.getRadioButton() - Returns the selected item (a string) from the next radio button group.
Dialog.getImageChoice() - Returns the title of the image selected in the next image choice 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.

doWand(x, y, tolerance, mode)
Traces the boundary of the area with pixel values within 'tolerance' of the value of the

pixel at (x, y). 'mode' can be "4-connected", "8-connected" or "Legacy". "Legacy" is for compatibility with previous versions of ImageJ; it is ignored if 'tolerance' > 0. If 'mode' contains 'smooth', the contour is smoothed by interpolation (example).

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. See also: Overlay.drawLine.

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

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

drawString("text", x, y)
Draws text at the specified location. The first character is drawn obove and to the right of (x, y). Call setFont() to specify the font. Call setJustification() to have the text centered or right justified. Call getStringWidth() to get the width of the text in pixels. Refer to the TextDemo macro for examples and to DrawTextWithBackground to see how to draw text with a background.

drawString("text", x, y, background)
Draws text at the specified location with a filled background (examples).

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

E [Top]

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

eval(macro)
Evaluates (runs) one or more lines of macro code. An optional second argument can be used to pass a string to the macro being evaluated. See also: EvalDemo macro and runMacro function.

eval("script", javascript)
Evaluates the JavaScript code contained in the string *javascript* and returns, as a string, the value of the last statement executed. For example *eval("script", "IJ.getInstance().setAlwaysOnTop(true);")*. See also: runMacro(path, arg).

eval("js", script)
Evaluates the JavaScript code contained in the string *script* and returns, as a string, the value of the last statement executed. For example *eval("js", "Prefs.blackBackground")* returns either "true" or "false".

eval("bsh", script)
Evaluates the BeanShell code contained in the string *script*.

eval("python", script)
Evaluates the Python code contained in the string *script*.

exec(string or strings)
Executes a native command and returns the output of that command as a string. Also opens Web pages in the default browser and documents in other applications (e.g., Excel). With commands with multiple arguments, each argument should be passed as a separate string. For example
exec("open", "/Users/wayne/test.jpg", "-a", "/Applications/Gimp.app");
Refer to the ExecExamples macro for examples.

exit
Terminates execution of the macro.

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

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

Ext (Macro Extension) Functions
These are functions that have been added to the macro language by plugins using the MacroExtension interface. The Image5D_Extensions plugin, for example, adds functions that work with Image5D. The Serial Macro Extensions plugin adds functions, such as Ext.open("COM8", 9600, "") and Ext.write("a"), that talk to serial devices.

F [Top]

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. Note that these functions return a string, with the exception of *File.length*, *File.exists*, *File.isDirectory*, *File.rename* and *File.delete* when used in an assignment statement, for example "length=File.length(path)". The FileDemo macro demonstrates how to use these functions. See also: getDirectory and getFileList.

File.append(string, path) - Appends *string* to the end of the specified file.
File.close(f) - Closes the specified file, which must have been opened using File.open().
File.copy(path1, path2) - Copies a file.
File.dateLastModified(path) - Returns the date and time the specified file was last modified.
File.delete(path) - Deletes the specified file or directory. With v1.41e or later, returns "1" (true) if the file or directory was successfully deleted. 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.directory - The directory path of the last file opened using a file open dialog, a file save dialog, drag and drop, open(path) or runMacro(path).
File.exists(path) - Returns "1" (true) if the specified file exists.
File.getName(path) - Returns the file name (e.g., "blobs.tif") from *path*.
File.getNameWithoutExtension(path) - Returns the name (e.g., "blobs") from *path* without the extension.
File.getDirectory(path) - Returns the directory (e.g., "C:/Users/wayne/images/") from *path*.
File.getDefaultDir - Returns the default directory used by file open dialogs or the current working directory if called from a command line macro and File.setDefaultDir() has not been called.

File.setDefaultDir(directoryPath) - Sets the default directory used by file open dialogs.
File.getParent(path) - Returns the parent of the file specified by *path*.
File.isFile(path) - Returns "1" (true) if the specified file is not a directory.
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 as a string, or as a number when used in an assignment statement, for example "length=File.length(path)".
File.makeDirectory(path) - Creates a directory.
File.name - The name of the last file opened using a file open dialog, a file save dialog, drag and drop, or the open(path) function.
File.nameWithoutExtension - The name of the last file opened with the extension removed.
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. Currently, only one file can be open at a time. 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. See also: File.openUrlAsString.
File.openAsRawString(path) - Opens a file and returns up to the first 5,000 bytes as a string. Returns all the bytes in the file if the name ends with ".txt". Refer to the First10Bytes and ZapGremlins macros for examples.
File.openAsRawString(path, count) - Opens a file and returns up to the first *count* bytes as a string.
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. With v1.41i and later, returns "<Error: message>" if there any error, including host or file not found.
File.openSequence(path, options) - Opens the images in the specified directory as a stack. Uses a virtual stack if the 'options' string contains 'virtual'. A filter, image type, start, step, count and scale can also be set, for example "filter=abc bitdepth=32 start=10 step=2 count=10 scale=50". Requires 1.53p.
File.openDialog(title) - Displays a file open dialog and returns the path to the file chosen by the user (example). The macro exits if the user cancels the dialog.
File.rename(path1, path2) - Renames, or moves, a file or directory. Returns "1" (true) if successful.
File.saveString(string, path) - Saves *string* as a file.
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.

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

Fit Functions
These functions do curve fitting. The CurveFittingDemo macro demonstrates how to use them.

Fit.doFit(equation, xpoints, ypoints) - Fits the specified equation to the points defined by *xpoints*, *ypoints*. *Equation* can be either the equation name or an index. The equation names are shown in the drop down menu in the *Analyze>Tools>Curve Fitting* window. With ImageJ 1.42f or later, *equation* can be a string containing a user-defined equation (example).
Fit.doFit(equation, xpoints, ypoints, initialGuesses) - Fits the specified equation to the points defined by *xpoints*, *ypoints*, using initial parameter values contained in *initialGuesses*, an array equal in length to the number of parameters in *equation* (example).
Fit.doWeightedFit(equation, xpoints, ypoints, weights, initialGuesses) - Fits the specified equation to the points defined by 'xpoints', 'ypoints', with the weight of each data point given by the 'weights' array. If the error bars of the individual data points are known, the weights should be proportional to $1/error^2$. The sum of squared residuals is calculated by multiplying each individual $(y - fit)^2$ value with the corresponding weight. The 'initialGuesses' array is optional.
Fit.rSquared - Returns R^2=1-SSE/SSD, where SSE is the sum of the squares of the errors and SSD is the sum of the squares of the deviations about the mean.
Fit.p(index) - Returns the value of the specified parameter.
Fit.nParams - Returns the number of parameters.
Fit.f(x) - Returns the y value at *x* (example).
Fit.nEquations - Returns the number of equations.
Fit.getEquation(index, name, formula) - Returns the name and formula of the specified equation.
Fit.getEquation(index, name, formula, macroCode) - Returns the name, formula and macro code of the specified equation.
Fit.plot - Plots the current curve fit.
Fit.logResults - Causes doFit() to write a description of the curve fitting results to the Log window.
Fit.showDialog - Causes doFit() to display the simplex settings dialog.

floodFill(x, y)
Fills, with the foreground color, pixels that are connected to, and the same color as, the pixel at *(x, y)*. 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: Math.ceil and round.

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

G [Top]

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".

getBoolean(message, yesLabel, noLabel)
Displays a dialog box containing the specified message and buttons with custom labels.

getBoundingRect(x, y, width, height)
Replace 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. Use toScaled(x, y) to scale the coordinates. For examples, see the GetCursorLocDemo and the GetCursorLocDemoTool macros.

getDateAndTime(year, month, dayOfWeek, dayOfMonth, hour, minute, second, msec)
Returns the current date and time. Note that 'month' and 'dayOfWeek' are zero-based indexes. For an example, refer to the GetDateAndTime macro. See also: getTime.

getDimensions(width, height, channels, slices, frames)
Returns the dimensions of the current image.

getDirectory(string)
Displays a "choose directory" dialog and returns the selected directory, or returns the path to a specified directory, such as "plugins", "home", etc. The returned path ends with the file separator (either "/" or "\" depending on the OS). Returns an empty string if the specified directory is not found or aborts the macro if the user cancels the "choose directory" dialog box. For examples, see the GetDirectoryDemo and ListFilesRecursively macros. See also: getFileList and the File functions.

getDir(string)
An alias of getDirectory since 1.49q.
Displays a "choose directory" dialog and returns the selected directory, or returns the path to a specified directory, such as "plugins", "home", etc. The returned path ends with the file separator (either "/" or "\" depending on the OS). Returns an empty string if the specified directory is not found or aborts the macro if the user cancels the "choose directory" dialog box. For examples, see the GetDirectoryDemo and ListFilesRecursively macros. See also: getFileList and the File functions.

> **getDir("Choose a Directory")** - Displays a file open dialog, using the argument as a title, and returns the path to the directory selected by the user.
> **getDir("downloads")** - Returns the path to the Downloads directory.
> **getDir("file")** - Returns the path most recently used to open or save a file.
> **getDir("home")** - Returns the path to the Home directory.
> **getDir("image")** - Returns the path to the directory that the active image was loaded from.
> **getDir("imagej")** - Returns the path to the ImageJ directory.

getDir("luts") - Returns the path to the luts directory.
getDir("macros") - Returns the path to the macros directory.
getDir("plugins") - Returns the path to the plugins directory.
getDir("startup") - Returns the path to the directory that ImageJ was launched from.
getDir("temp") - Returns the path to the temporary directory (/tmp on Linux and Mac OS X).
getDir("cwd") - Returns the path to the current working directory.
getDir("preferences") - Returns the path to the directory containing "IJ_Prefs.txt".

getDisplayedArea(x, y, width, height)
Returns the pixel coordinates of the actual displayed area of the image canvas. For an example, see the Pixel Sampler Tool.

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.

getFontList()
Returns an array containing the names of available system fonts (example).

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, the histogram range can be specified using optional *histMin* and *histMax* arguments. See also: getStatistics, HistogramLister, HistogramPlotter, StackHistogramLister and CustomHistogram.

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 and Property.getInfo.

getInfo("command.name")
Returns the name of the most recently invoked command. The names of commands invoked using keyboard shortcuts are preceded by "^" (example).

getInfo(DICOM_TAG)
Returns the value of a DICOM tag in the form "xxxx,xxxx", e.g. getInfo("0008,

0060"). Returns an empty string if the current image is not a DICOM or if the tag was not found.

getInfo("font.name")
Returns the name of the current font.

getInfo("image.description")
Returns the TIFF image description tag, or an empty string if this is not a TIFF image or the image description is not available.

getInfo("image.directory")
Returns the directory that the current image was loaded from, or an empty string if the directory is not available.

getInfo("image.filename")
Returns the name of the file that the current image was loaded from, or an empty string if the file name is not available.

getInfo("image.title")
Returns the title of the current image. Same as getTitle.

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.

getInfo("log")
Returns the contents of the Log window, or "" if the Log window is not open.

getInfo("macro.filepath")
Returns the filepath of the most recently loaded macro or script. Use *File.getName(getInfo("macro.filepath"))* to get the file name and *File.getDirectory(getInfo("macro.filepath"))* to get the directory.

getInfo("micrometer.abbreviation")
Returns "µm", the abbreviation for micrometer.

getInfo("os.name")
Returns the OS name ("Mac OS X", "Linux" or "Windows").

getInfo("overlay")
Returns information about the current image's overlay.

getInfo("selection.name")
Returns the name of the current selection, or "" if there is no selection or the selection does not have a name. The argument can also be "roi.name". See also: Roi.getName.

getInfo("selection.color")
Returns the color of the current selection. See also: Roi.getStrokeColor.

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.

getInfo("threshold.method")
Returns the current thresholding method ("IsoData", "Otsu", etc).

getInfo("threshold.mode")
Returns the current thresholding mode ("Red", "B&W" or"Over/Under").

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.

getInfo("window.title")
Returns the title of the front-most window. Use getTitle to get the title of the current batch mode image.

getInfo("window.type")
Returns the type ("Image", "Text", "ResultsTable", "Editor", "Plot", "Histogram", etc.) of the front window.

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").

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. See also: makeLine.

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

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

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

getList("threshold.methods")
Returns a list of the available automatic thresholding methods (example).

getList("LUTs")
Returns, as an array, a list of the LUTs in the *Image>Lookup Tables* menu (example).

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,

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("Info")
Returns the metadata (a string) from the "Info" property of the current image. With DICOM images, this is the information (tags) in the DICOM header. See also: setMetadata and Property.getInfo.

getMetadata("Label")
Returns the current slice label. The first line of the this label (up to 60 characters) is displayed as part of the image subtitle. With DICOM stacks, returns the metadata (tags) from the DICOM header. See also: Property.getSliceLabel and Property.setSliceLabel.

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 raw value of the pixel at *(x, y)*. Uses bilinear interpolation if 'x' or 'y' are not integers. Use getValue(x, y) to get calibrated pixel values from 8 and 16 bit images and intensity values from RGB images. 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. See also: Plot.getValues().

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 and List.setMeasurements

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.) Omit the second argument and the row defaults to nResults-1 (the last row in the results table). See also: nResults, setResult, isNaN, getResultLabel.

getResultString("Column", row)
Returns a string from the ImageJ results table or "null" if the specified column is not

found. The first argument specifies a column in the table. The second specifies the row, where 0<=*row*<nResults.

getResultLabel(row)
Returns the label of the specified row in the results table, or an empty string if *Display Label* is not checked in *Analyze>Set Measurements*.

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(xpoints, ypoints)
Returns two arrays containing the X and Y coordinates, in pixels, 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 image, an integer between 1 and nSlices. Returns 1 if the active image is not a stack. See also: setSlice, Stack.getPosition.

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 and List.setMeasurements

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.

getStringWidth(string)
Returns the width in pixels of the specified string. See also: getValue("font-height"), String.setFontSize, setFont and drawString.

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.

getValue(x, y)
Returns calibrated pixel values from 8 and 16 bit images and intensity values from RGB images. Uses bilinear interpolation if 'x' or 'y' are not integers. Use getPixel(x, y) to read raw pixel values.

getValue("color.foreground")
Returns the current foreground color as a value that can be passed to the setColor(value) function. The value returned is the pixel value used by the *Edit>Fill* command and by drawing tools.

getValue("color.background")
Returns the current background color as a value that can be passed to the setColor(value) function. The value returned is the pixel value used by the *Edit>Clear* command.

getValue("rgb.foreground")
Returns the current foreground color as an RGB pixel value (example).

getValue("rgb.background")
Returns the current background color as an RGB pixel value.

getValue("font.size")
Returns the size, in points, of the current font.

getValue("font.height")
Returns the height, in pixels, of the current font.

getValue("selection.size")
Returns the number of points in the current selection, or 0 if there is no selection. See also: Roi.size.

getValue("selection.width")
Returns the stroke width of the current selection. See also: Roi.getStrokeWidth.

getValue("results.count")
Returns the number of lines in the current results table. Unlike nResults, works with tables that are not named "Results".

getValue("rotation.angle")
Returns the current *Image>Transform>Rotate* angle.

getValue("image.size")
Returns the size of the current image in bytes.

getValue("Mean, Median, Feret, etc.")
Returns a measurement result, where the argument is "Area", "Mean", "StdDev", "Mode", "Min", "Max", "X", "Y", "XM", "YM", "Perim.", "BX", "BY", "Width", "Height", "Major", "Minor", "Angle", "Circ.", "Feret", "IntDen", "Median", "Skew", "Kurt", "%Area", "RawIntDen", "Ch", "Slice", "Frame", "FeretX", "FeretY", "FeretAngle", "MinFeret", "AR", "Round", "Solidity", "MinThr", "MaxThr" or "Length". Add " raw" to the argument to disable calibration, for example getValue("Mean raw"). Add " limit" to the argument to enable the "limit to threshold" option.

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: IJ.getFullVersion.

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 [Top]

IJ Functions
These functions provide access to miscellaneous methods in ImageJ's IJ class.

IJ.deleteRows(index1, index2) - Deletes rows *index1* through *index2* in the results table.
IJ.getToolName() - Returns the name of the currently selected tool. See also: setTool.
IJ.getFullVersion - Returns the ImageJ version and build number as a string (e.g., "1.52d11").
IJ.freeMemory() - Returns the memory status string (e.g., "2971K of 658MB (<1%)") that is displayed when the users clicks in the ImageJ status bar.
IJ.checksum("MD5 string", string) - Returns the MD5 (or SHA-256) checksum from a string.
IJ.checksum("MD5 file", filepath) - Returns the MD5 (or SHA-256) checksum from a file. If 'filepath' is a directory or invalid, "0" is returned.
IJ.currentMemory() - Returns, as a string, the amount of memory in bytes currently used by ImageJ.
IJ.log(string) - Displays *string* in the Log window.
IJ.maxMemory() - Returns, as a string, the amount of memory in bytes available to ImageJ. This value (the Java heap size) is set in the *Edit>Options>Memory & Threads* dialog box.
IJ.pad(n, length) - Pads 'n' with leading zeros and returns the result (example). Note that 'n' can be either a number or a string. See also: String.pad.
IJ.redirectErrorMessages() - Causes next image opening error to be redirected to the Log window and prevents the macro from being aborted (example).
IJ.renameResults(name) - Changes the title of the "Results" table to the string *name*.
IJ.renameResults(oldName, newName) - Changes the title of a results table from *oldName* to *newName*.

Image Functions
Functions for working with the active image, available in ImageJ 1.53f and later.

Image.title
The title of the active image. Requires 1.53h.
Image.width
The width of the active image.

Image.height
The height of the active image.
Image.copy
Copies the contents of the current selection, or the entire image if there is no selection, to the internal clipboard.
Image.paste(x, y)
Inserts the contents of the internal clipboard at the specified location in the active image.
Image.paste(x, y, mode)
Inserts the contents of the internal clipboard at *x,y* using the specified transfer mode ("Copy", "Blend", "Average", "Difference", "Transparent", "Transparent2", "AND", "OR", "XOR", "Add", "Subtract", "Multiply", or "Divide""Copy", "Blend", "Average", "Difference", "Transparent", "Transparent2", "AND", "OR", "XOR", "Add", "Subtract", "Multiply", or "Divide").

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, matches.

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

is("animated")
Returns *true* if the current image is an animated stack.

is("applet")
Returns *true* if ImageJ is running as an applet.

is("area")
Returns *true* if the current image has an area selection.

is("Batch Mode")
Returns *true* if the macro interpreter is running in batch mode.

is("binary")
Returns *true* if the current image is binary (8-bit with only 0 and 255 values).

is("Caps Lock Set")
Returns *true* if the caps lock key is set. Always return *false* on some platforms.

is("changes")
Returns *true* if the current image's 'changes' flag is set.

is("composite")
Returns *true* if the current image is a a multi-channel stack that uses the CompositeImage class.

is("FFT")
Returns *true* if the current image is a Fourier transform image, an 8-bit image showing the logarithm of the power spectrum, with the full Fourier transform data in the background (so you can do an inverse transform).

is("global scale")
Returns *true* if there is global spatial calibration.

is("grayscale")
Returns *true* if the current image is grayscale, or an RGB image with identical R, G and B channels.

is("Inverting LUT")
Returns *true* if the current image is using an inverting (monotonically decreasing) lookup table.

is("InvertY")
Returns *true* if the 'invertY' property of the active image is enabled.

is("line")
Returns *true* if the current image has a line selection.

is("locked")
Returns *true* if the current image is locked.

is("Virtual Stack")
Returns *true* if the current image is a virtual stack.

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. Comparison with a NaN always returns *false* so "if (n!=n)" is equivalent to (isNaN(n))". Note that the numeric constant NaN is predefined in the macro language. The NaNs macro demonstrates how to remove NaNs from an image.

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 [Top]

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. Can be replaced with str.length (1.52t or later) or arr.length, where is 'str' is a string variable and 'arr' is an array variable.

lineTo(x, y)
Draws a line from current location to (*x, y*) . See also: Overlay.lineTo.

List Functions
These functions work with a list of key/value pairs. The ListDemo macro demonstrates how to use them.

List.set(key, value) - Adds a key/value pair to the list.
List.get(key) - Returns the string value associated with *key*, or an empty string if the key is not found.
List.getValue(key) - When used in an assignment statement, returns the value associated with *key* as a number. Aborts the macro if the value is not a number or the key is not found.
List.size - Returns the size of the list.
List.clear() - Resets the list.
List.setList(list) - Loads the key/value pairs in the string *list*.
List.getList - Returns the list as a string.
List.setMeasurements - Measures the current image or selection and loads the resulting keys (Results table column headings) and values into the list. Use *List.setMeasurements("limit")* to measure using the "Limit to threshold" option. All parameters listed in the *Analyze>Set Measurements* dialog box are measured, including those that are unchecked. Use List.getValue() in an assignment statement to retrieve the values. See the DrawEllipse macro for an example. With ImageJ 1.52p or later, it is easier to use the getValue() function to get a single measurement value. Examples: *getValue("Mean")*, *getValue("Mean limit")* and *getValue("Mean raw")*.
List.setMeasurements("limit") - This is a version of List.setMeasurements that enables the "Limit to threshold" option.
List.setCommands - Loads the ImageJ menu commands (as keys) and the plugins that implement them (as values).
List.toArrays(keys, values) - Retrieves keys and values as a pair of string arrays, sorted alphabetically for keys.
List.fromArrays(keys, values) - Creates the List from keys and values arrays.
List.indexOf(key) - Returns the alphabetic position of the specified key, or -1 if not found. Note that this function, as well as List.size, returns a string.

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

M [Top]

makeArrow(x1, y1, x2, y2, style)
Creates an arrow selection, where 'style' is a string containing "filled", "notched", "open", "headless" or "bar", plus the optionial modifiers "outline", "double", "small",

"medium" and "large" (example). See also: Roi.setStrokeWidth and Roi.setStrokeColor.

makeEllipse(x1, y1, x2, y2, aspectRatio)
Creates an elliptical selection, where *x1,y1,x2,y2* specify the major axis of the ellipse and *aspectRatio* (<=1.0) is the ratio of the lengths of minor and major axis.

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. You can create segmented line selections by specifying more than two coordinate pairs, for example *makeLine(25, 34, 44, 19, 69, 30, 71, 56)*. After creating the selection, use Roi.setStrokeWidth to set the width and Roi.setStrokeColor to set the color.

makeLine(x1, y1, x2, y2, lineWidth)
Creates a straight line selection with the specified width. See also: getLine.

makeLine(x1, y1, x2, y2, x3, y3, ...)
Creates a segmented line selection. After creating the selection, use Roi.setStrokeWidth to set the width and Roi.setStrokeColor to set the color. See also: makeSelection("polyline", xpoints, ypoints).

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

makePoint(x, y, options)
Creates a point selection at the specified location, with the type ('hybrd', 'cross', 'dot' or 'circle'), color ('red', 'white', etc.) and size ('tiny', 'small', 'medium', 'large' or 'extra large') of the point defined in the 'options' string (example). The point is added to an overlay if the options string contains 'add'.

Create a multi-point selection by using *makeSelection("point", xpoints, ypoints)*. All the makePoint() options ('cross', 'small', 'red', etc.) can be added to first argument.

makePoint(x, y)
Creates a point selection at the specified location. Create a multi-point selection by using *makeSelection("point", xpoints, ypoints)*. Use *setKeyDown("shift"); makePoint(x, y);* to add a point to an existing point selection.

makePolygon(x1, y1, x2, y2, x3, y3, ...)
Creates a polygonal selection, where at least three coordinate pairs must be specified. As an example, *makePolygon(20, 48, 59, 13, 101, 40, 75, 77, 38, 70)* creates a polygon selection with five sides. See also: makeSelection("polygon", xpoints, ypoints).

makeRectangle(x, y, width, height)
Creates a rectangular selection, where *x* and *y* 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.

makeRectangle(x, y, width, height, arcSize)
Creates a rounded rectangular selection using the specified corner arc size.

makeRotatedRectangle(x1, y1, x2, y2, width)
Creates a rotated rectangular selection, which is similar to a wide line where (x1, y1) is

the start of the line, (x2, y2) is the end of the line and 'width' is the line width.

makeSelection(type, xpoints, ypoints)
Creates a selection from a list of XY coordinates. The first argument should be "polygon", "freehand", "polyline", "freeline", "angle" or "point", or the numeric value returned by selectionType. The *xpoints* and *ypoints* arguments are numeric arrays that contain the X and Y coordinates. See the MakeSelectionDemo macro for examples.

When creating a multi-point selection and using ImageJ 1.52i or later, the first argument can include words that define the type ('hybrd', 'cross', 'dot' or 'circle'), color ('red', 'white', etc.) and size ('tiny', 'small', 'medium', 'large' or 'extra large') of the points (e.g., "point small red cross").

makeText(string, x, y)
Creates a text selection, where *x* and *y* are the coordinates (in pixels) of the upper left corner of the selection. The selection will use the font and size specified by the last call to setFont(). The CreateOverlay macro provides an example.

matches(string, regex)
Returns *true* if *string* matches the specified regular expression. Can be replaced with string.matches(regex) in ImageJ 1.52t or later. See also: startsWith, endsWith, indexOf, replace.

Math Functions
Functions for performing basic numeric operations, available in ImageJ 1.52u or later.

Math.abs(n)
Returns the absolute value of *n*.
Math.acos(n)
Returns the inverse cosine (in radians) of *n*.
Math.asin(n)
Returns the inverse sine (in radians) of *n*.
Math.atan(n)
Returns the inverse tangent (arctangent) of *n*, a value in the range -PI/2 through PI/2.
Math.atan2(y, x)
Calculates the inverse tangent 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.
Math.ceil(n)
Returns the smallest (closest to negative infinity) value that is greater than or equal to *n* and is an integer.
Math.cos(angle)
Returns the cosine of an angle (in radians).
Math.erf(x)
Returns an approximation of the error function.
Math.exp(n)
Returns the exponential number e (i.e., 2.718...) raised to the power of *n*.
Math.floor(n)
Returns the largest value that is not greater than *n* and is equal to an integer.
Math.log(n)
Returns the natural logarithm (base e) of *n*.
Math.log10(n)
Returns the base 10 logarithm of *n*.

Math.min(n1, n2)
Returns the smaller of two values.
Math.max(n1, n2)
Returns the larger of two values.
Math.pow(base, exponent)
Returns the value of *base* raised to the power of *exponent*.
Math.round(n)
Returns the closest integer to *n*.
Math.sin(angle)
Returns the sine of an angle (in radians).
Math.sqr(n)
Returns the square of *n*.
Math.sqrt(n)
Returns the square root of *n*, or NaN if *n* is less than zero.
Math.tan(n)
Returns the tangent of an angle (in radians).
Math.toRadians(degrees)
Converts an angle in degrees to an equivalent angle in radians. Requires 1.53f.
Math.toDegrees(radians)
Converts an angle in radians to an equivalent angle in degrees. Requires 1.53f.
Math.constrain(n, min, max)
Limits 'n' to the range 'min'-'max'. Requires 1.53n.
Math.map(n, low1, high1, low2, high2)
Maps the value 'n' in the input range 'low1'-'high1' to the output range 'low2'-'high2'. Requires 1.53n.

maxOf(n1, n2)
Returns the greater of two values. See also: Math.max.

minOf(n1, n2)
Returns the smaller of two values. See also: Math.min.

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

N [Top]

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.

In ImageJ 1.53g or newer, arrays automatically expand in size as needed (example). To use expandable arrays in earlier versions, call *setOption("ExpandableArrays", true)*.

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. Precede with *call("ij.gui.ImageWindow.setNextLocation", x, y)* to set the location of the new image. *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 Toolbar Menus toolset.

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

nResults
Returns the current measurement counter value. The parentheses "()" are optional. See also: getValue("results.count").

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

O [Top]

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. With 1.41k or later, opens images specified by a URL, for example *open("../../images/clown.gif")*. With 1.49v or later, opens a folder of images as a stack. Use open("path/to/folder", "virtual") to open a folder of images as a virtual stack.

open(path, n)
Opens the *n*th image in the TIFF stack specified by *path*. For example, the first image in the stack is opened if *n*=1 and the tenth is opened if *n*=10.

Overlay Functions
Use these functions to create and manage non-destructive graphic overlays. For an exmple, refer to the OverlayPolygons macro. See also: setColor, setLineWidth and setFont.

Overlay.moveTo(x, y)
Sets the current drawing location.
Overlay.lineTo(x, y)
Draws a line from the current location to (*x, y*) .
Overlay.drawLine(x1, y1, x2, y2)
Draws a line between (*x1, y1*) and (*x2, y2*).
Overlay.add
Adds the drawing created by Overlay.lineTo(), Overlay.drawLine(), etc. to the overlay without updating the display.
Overlay.setPosition(n)
Sets the stack position (slice number) of the last item added to the overlay (example). Set the position to 0 and the item will be displayed on all stack

slices.

Overlay.setPosition(c, z, t)
Sets the hyperstack position (channel, slice, frame) of the last item added to the overlay. Set a position to 0 and the item will be displayed on all the corresponding channels, slices or frames.

Overlay.drawRect(x, y, width, height)
Draws a rectangle, where (*x, y*) specifies the upper left corner.

Overlay.drawEllipse(x, y, width, height)
Draws an ellipse, where (*x, y*) specifies the upper left corner of the bounding rectangle.

Overlay.drawString("text", x, y)
Draws text at the specified location and adds it to the overlay. The first character is drawn obove and to the right of (x, y). Use setFont() to specify the font and setColor to set specify the color (example).

Overlay.drawString("text", x, y, angle)
Draws text at the specified location and angle and adds it to the overlay (example).

Overlay.show
Displays the current overlay.

Overlay.hide
Hides the current overlay.

Overlay.hidden
Returns *true* if the active image has an overlay and it is hidden.

Overlay.remove
Removes the current overlay.

Overlay.clear
Resets the overlay without updating the display.

Overlay.size
Returns the size (selection count) of the current overlay. Returns zero if the image does not have an overlay.

Overlay.addSelection
Adds the current selection to the overlay.

Overlay.addSelection(strokeColor)
Sets the stroke color ("red", "green", "ff8800", etc.) of the current selection and adds it to the overlay.

Overlay.addSelection(strokeColor, strokeWidth)
Sets the stroke color ("blue", "yellow", "ffaa77" etc.) and stroke width of the current selection and adds it to the overlay.

Overlay.addSelection("", 0, fillColor)
Sets the fill color ("red", "green", etc.) of the current selection and adds it to the overlay.

Overlay.activateSelection(index)
Activates the specified overlay selection.

Overlay.activateSelectionAndWait(index)
Activates the specified overlay selection and waits for the display to finish updating.

Overlay.removeSelection(index)
Removes the specified selection from the overlay.

Overlay.update(index)
Replaces the overlay selection at 'index' with the current selection. Analogous to roiManager("update").

Overlay.moveSelection(index, x, y)
Moves the specified selection to the specified location.

Overlay.indexAt(x, y)

Returns the index of the last selection in the overlay containing the point (x, y), or an empty string if no selection contains the point. Requires 1.53b.

Overlay.getBounds(index, x, y, width, height)
Returns the bounding rectangle of the selection in the overlay specified by 'index'. Requires 1.53b.

Overlay.removeRois(name)
Removes the selections named 'name' from the overlay. Requires 1.53b.

Overlay.copy
Copies the overlay on the current image to the overlay clipboard.

Overlay.paste
Copies the overlay on the overlay clipboard to the current image.

Overlay.drawLabels(boolean)
Enables/disables overlay labels.

Overlay.setLabelFontSize(size, options)
Sets the label font size. The options string can contain 'scale' (enlarge labels when image zoomed), 'bold' (display bold labels) or 'back' (display labels with contrasting background color.

Overlay.setLabelColor(color)
Sets the color used to draw labels.

Overlay.useNamesAsLabels(boolean)
Controls whether ROI names or index numbers are used as labels.

Overlay.selectable(false)
Prevents the selections in this overlay from being activated by clicking on their labels or by long clicking. Requires 1.51r.

Overlay.measure
Measures all the selections in the current overlay.

Overlay.setStrokeColor(color)
Sets the stroke color all the selections in the current overlay.

Overlay.setStrokeWidth(width)
Sets the stroke width all the selections in the current overlay.

Overlay.flatten
Creates a new RGB image that has the overlay rendered as pixel data.

Overlay.cropAndSave(dir, format)
Saves the contents of the overlay selections as separate images, where 'dir' is the directory path and 'format' is "tif", "png" or "jpg". Set 'format' to "show" and the images will be displayed in a stack and not saved. Requires 1.53d.

Overlay.fill(color)
Fills all the selections in the overlay with 'color'. Requires 1.53h.

Overlay.fill(color1, color2)
Fills all the selections in the overlay with 'color1' after clearing the image to 'color2'. Requires 1.53h.

Overlay.xor(array)
Creates a selection that is the result of XORing all the selections in the overlay that have an index in 'array'. Requires 1.53h.

P [Top]

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.

parseFloat("Infinity")
Returns the special floating point value representing +Infinity.

parseFloat("-Infinity")
Returns the special floating point value representing -Infinity.

parseFloat("NaN")
Returns the special floating point value representing NaN (not a number).

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.

PI
Returns the constant π (3.14159265), the ratio of the circumference to the diameter of a circle.

Plot Functions
Use these functions to generate and display plots. For examples, check out the Example Plot, More Example Plots, AdvancedPlots, Semi-log Plot and Arrow Plot macros.

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.
Plot.create("Title", "{cat1, cat2, cat3}", "Y-axis Label")
Draws category labels instead of x-axis numbers 0, 1, 2.
Plot.add(type, 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 (*type*) can be "line", "connected circle", "filled", "bar", "separated bar", "circle", "box", "triangle", "diamond", "cross", "x", "dot", "error bars" (in y direction) or "xerror bars". Run *Help>Examples>Plots>Plot Styles* to see examples. Error bars (supplied by an optional fourth argument) apply to the last dataset provided by *Plot.create* or *Plot.add*.
Plot.add(type, xValues, yValues, label)
Adds a curve or set of points to the plot and sets the dataset label used by Plot.showValuesWithLabels.
Plot.addHistogram(values, binWidth, binCenter)
Creates a 'staircase' histogram from array 'values'. If 'binWidth' is 0, automatic binning is applied. 'binCenter' is optional with default=0. BinCenter can, for example, be set to an expected symmetry point for avoiding artificial asymmetry.

Plot.drawVectors(xStarts, yStarts, xEnds, yEnds)
Draws arrows from the starting to ending coordinates contained in the arrays.
Plot.drawLine(x1, y1, x2, y2)
Draws a line between *x1,y1* and *x2,y2*, using the coordinate system defined by *Plot.setLimits()*.
Plot.drawNormalizedLine(x1, y1, x2, y2)
Draws a line using a normalized 0-1, 0-1 coordinate system, with (0, 0) at the top left and (1, 1) at the lower right corner.
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.setLimits(xMin, xMax, yMin, yMax)
Sets the range of the x-axis and y-axis of plots. With version 1.50g and later, when 'NaN' is used as a limit, the range is calculated automatically from the plots that have been added so far.
Plot.getLimits(xMin, xMax, yMin, yMax)
Returns the current axis limits. Note that min>max if the axis is reversed.
Plot.setLimitsToFit()
Sets the range of the x and y axes to fit all data.
Plot.setColor(color)
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", "yellow", or a hex value like "#ff00ff". Note that the curve specified in *Plot.create()* is drawn last.
Plot.setColor(color1, color2)
This is a two argument version of Plot.setColor, where the second argument is used for filling symbols or as the line color for connected points.
Plot.setBackgroundColor(color)
Sets the background color of the plot frame (example).
Plot.setLineWidth(width)
Specifies the width of the line used to draw a curve. Symbols (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 displayed or updated.
Plot.setJustification("center")
Specifies the justification used by *Plot.addText()*. The argument can be "left", "right" or "center". The default is "left".
Plot.setLegend("label1\tlabel2...", "options")
Creates a legend for each of the data sets added with *Plot.create* and *Plot.add*. In the first argument, the labels for the data sets should be separated with tab or newline characters. The optional second argument can contain the legend position ("top-left", "top-right", "bottom-left", "bottom-right"; default is automatic positioning), "bottom-to-top" for reversed sequence of the labels, and "transparent" to make the legend background transparent.
Plot.setFrameSize(width, height)
Sets the plot frame size in pixels, overriding the default size defined in the *Edit>Options>Profile Plot Options* dialog box.
Plot.getFrameBounds(x, y, width, height)
Returns the plot frame bounds.

Plot.setLogScaleX(boolean)
Sets the x axis scale to Logarithmic, or back to linear if the optional boolean argument is false. In versions up to 1.49s, it must be called immediately after *Plot.create* and before *Plot.setLimits*. See the LogLogPlot macro for an example.
Plot.setLogScaleY(boolean)
Sets the y axis scale to Logarithmic, or back to linear if the optional boolean argument is false.
Plot.setXYLabels("x Label", "y Label")
Sets the axis labels.
Plot.setFontSize(size, "options")
Sets the default font size in the plot (otherwise specified in *Profile Plot Options*), used e.g. for axes labels. Can be also called prior to *Plot.addText*. The optional second argument can include "bold" and/or "italic".
Plot.setAxisLabelSize(size, "options")
Sets the fort size of the axis labels. The optional second argument can include "bold" and/or "italic".
Plot.setFormatFlags("11001100001111")
Controls whether to draw axes labels, grid lines and ticks (major/minor/ticks for log axes). Use the macro recorder and *More>>Axis Options* in any plot window to determine the binary code.
Plot.setStyle(index, styleString)
Set the style of the plot object (curve, label, etc.) with the specified index, using a comma-delimiterd string ("color1, color2, line width, symbol"). For an example, run the *Help>Examples>Plots>Plot Styles* command.
Plot.freeze(boolean)
Freezes or unfreezes the plot. In the frozen state, the plot cannot be resized or updated. In the unfrozen state, the plot cannot be tiled.
Plot.setOptions(string)
Sets options for the plot. Multiple options may be separated by whitespace or commas. Note that whitespace surrounding the '=' characters is not allowed. Currently recognized options are:
"addhspace=10 addvspace=5" - Increases the left and right or top and bottom margins by the given number of pixels.
"xinterval=30 yinterval=90" - Sets interval between numbers, major ticks and grid lines (default intervals are used if the custom intervals would be too dense or too sparse)
"xdecimals=2 ydecimals=-1" - Sets the minimum number of decimals; use negative numbers for scientific notation.
"msymbol=' \\u00d7" - Sets multiplication symbol for scientific notation, here a cross with spaces.
For an example, run the *Help>Examples>Plots>Plot Results* command. Requires 1.53a.
Plot.replace(index, type, xValues, yValues)
Replaces the plot object (curve, Label, etc.) with the specified index by points or a curve as specified in Plot.add(type, xValues, yValues).
Plot.useTemplate("plot name" or id)
Transfers the formatting of an open plot window to the current plot.
Plot.makeHighResolution(factor)
Creates a high-resolution image of the plot enlarged by *factor*. Add the second argument "disable" to avoid antialiased text. **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.
Plot.getValues(xpoints, ypoints)
Returns the values displayed by clicking on "List" in a plot or histogram window (example).
Plot.showValues()
Displays the plot values in the Results window using simple "X , Y, X1, Y1, .." column headings. Must be preceded by *Plot.show*.
Plot.showValuesWithLabels()
Displays the plot values in the Results window using the dataset labels, for example the labels supplied by Plot.setLegend. Must be preceded by *Plot.show*.
Plot.drawGrid()
Redraws the grid above previous plots.
Plot.drawShapes("rectangles", lefts, tops, rights, bottoms)
Draws one or more rectangles. The four arguments (values or arrays) hold rectangle coordinates.
Plot.drawBoxes("boxes width=30", x, y1, y2, y3, y4, y5)
Draws a boxplot, where 'width' is in pixels, array 'x' holds x-positions and arrays 'y1'..'y5' hold the quartile borders in ascending order. Secondary color will fill the box. For horizontal boxes, use "boxesx width=30" instead.
Plot.removeNaNs()
Removes NaN values from the plot. This has two effects: a) Numerical output via "List" or macro will be free of NaNs; b) Line plots that were interrupted by NaNs are now connected.
Plot.eneableLive(boolean)
Enables and disables the "Live" mode of the active plot. Requires 1.54e.

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. The print() function 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).

Numeric expressions are automatically converted to strings using four decimal places, or use the d2s function to specify the decimal places. For example, *print(2/3)* outputs "0.6667" but *print(d2s(2/3, 1))* outputs "0.7".

The print() function accepts commands such as *"\\Clear"*, *"\\Update:<text>"* and *"\\Update<n>:<text>"* (for n<26) that clear the "Log" window and update its contents. For example, *print("\\Clear")* erases the Log window, *print("\\Update:new text")* replaces the last line with "new text" and *print("\\Update8:new 8th line")* replaces the 8th line with "new 8th line". Refer to the LogWindowTricks macro for an example.

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.

Property Functions
Functions for reading and writing image properties, available in ImageJ 1.53a or later.

Property.get(key)
Returns the image property (a string) associated with *key*, or an empty string if *key* is not found.
Property.getNumber(key)
Returns the image property associated with *key*, as a number, or returns NaN if *key* is not found or the property cannot be converted into a number. Requires 1.53b.
Property.set(key, property)
Adds a key-value pair to the property list of the current image. *key* must be a string but *property* can be either a string or a number. The key-value pair is removed if *property* is an empty string. Add a "ShowInfo" property (e.g. Property.set("ShowInfo", "true")) and the property list will be displayed by the *Image>Show Info* command (requires 1.53b). The property list is persistent when the image is saved in TIFF format.
Property.getInfo()
Returns the image "info" property string (e.g. DICOM tags or Bio-Formats metadara), or an empty string if there is no "Info" property.
Property.getSliceLabel
Returns the current slice label. The first line of the label (up to 60 characters) is displayed as part of the image subtitle. With DICOM stacks, returns the DICOM header metadata (tags).
Property.setSliceLabel(string, slice)
Sets *string* as the label of the specified stack slice, were 1<=*slice*<=nslices. The first 60 characters, up to the first newline, of the label are displayed as part of the image subtitle. Slice labels are saved in the TIFF header.
Property.setSliceLabel(string)
Sets *string* as the label of the current stack slice.
Property.getList
Returns the properties as a string (e.g. "key1=value1\nkey2=value2"). Requires 1.53b.
Property.setList(string)
Sets the properties from the key/value pairs in 'string'. Requires 1.53b.

R [Top]

random
Returns a uniformly distributed pseudorandom number between 0 and 1.

random("gaussian")
Returns a Gaussian ("normally") distributed pseudorandom number with mean 0.0 and standard deviation 1.0.

random("seed", seed)
Sets the seed (a whole number) used by the *random()* and *random("gaussian")* functions. Set a specific seed to get a reproducible pseudorandom sequence.

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

replace(string, old, new)
Returns a string that results from replacing all occurrences of *old* in *string* with *new*, where *old* is a single character string. If *old* is longer than one character, each substring of *string* that matches the regular expression *old* is replaced with *new*. When doing a simple string replacement, and *old* contains regular expression metacharacters ('.', '[', ']', '^', '$', etc.), you must escape them with a "\\". For example, to replace "[xx]" with "yy", use *string=replace(string, "\\[xx\\]", "yy")*. Can be replaced with string.replace(old, new) in ImageJ 1.52t or later. See also: matches.

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
With 16-bit and 32-bit images, resets the minimum and maximum displayed pixel values (display range) to be the same as the current image's minimum and maximum pixel values. With 8-bit images, sets the display range to 0-255. With RGB images, does nothing. See the DisplayRangeMacros for examples.

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

restoreSettings
Restores *Edit>Options* submenu settings saved by the saveSettings function.

ROI Functions
Use these functions to get information about the current selection or to get or set selection properties. Refer to the RoiFunctionsDemo macro for examples.

Roi.size
Returns the size of the current selection in points.
Roi.contains(x, y)
Returns *true* if the point *x,y* is inside the current selection. Aborts the macro if there is no selection. See also: Roi.getContainedPoints and selectionContains.
Roi.getBounds(x, y, width, height)
Returns the location and size of the selection's bounding rectangle.
Roi.getFloatBounds(x, y, width, height)
Returns the location and size of the selection's bounding rectangle as real numbers.
Roi.getCoordinates(xpoints, ypoints)
Returns, as two arrays, the x and y coordinates that define this selection.
Roi.getContainedPoints(xpoints, ypoints)
Returns, as two arrays, the x and y coordinates of the pixels inside the current selection. Aborts the macro if there is no selection.
Roi.getDefaultColor
Returns the current default selection color as a string.
Roi.getStrokeColor

Returns the selection stroke color as a string.

Roi.getFillColor
Returns the selection fill color as a string.

Roi.getName
Returns the selection name or an empty string if the selection does not have a name.

Roi.getProperty(key)
Returns the value (a string) associated with the specified key or an empty string if the key is not found.

Roi.setProperty(key, value)
Adds the specified key and value pair to the selection properties. Assumes a value of "1" (true) if there is only one argument.

Roi.getProperties
Returns all selection properties or an empty string if the selection does not have properties.

Roi.getSplineAnchors(x, y)
Returns the x and y coordinates of the anchor points of a spline fitted selection (example).

Roi.setPolygonSplineAnchors(x, y)
Creates or updates a spline fitted polygon selection (example).

Roi.setPolylineSplineAnchors(x, y)
Creates or updates a spline fitted polyline selection (example).

Roi.getType
Returns, as a string, the type of the current selection.

Roi.move(x, y)
Moves the selection to the specified location.

Roi.translate(dx, dy)
Translates the selection by 'dx' and 'dy'. Requires 1.53v.

Roi.setStrokeColor(color)
Sets the selection stroke color ("red", "5500ff00". etc.).

Roi.setStrokeColor(red, green, blue)
Sets the selection stroke color, where 'red', 'green' and 'blue' are integers (0-255).

Roi.setStrokeColor(rgb)
Sets the selection stroke color, where 'rgb' is an integer.

Roi.setFillColor(color)
Sets the selection fill color ("red", "5500ff00". etc.). With 1.54e and later, set the stroke color after setting the fill color to both fill and outline the selection.

Roi.setFillColor(red, green, blue)
Sets the selection fill color, where 'red', 'green' and 'blue' are integers (0-255). With 1.54e and later, set the stroke color after setting the fill color to both fill and outline the selection.

Roi.setFillColor(rgb)
Sets the selection fill color, where 'rgb' is an integer. With 1.54e and later, set the stroke color after setting the fill color to both fill and outline the selection.

Roi.setAntiAlias(boolean)
Controls whether anti-aliasing is used when drawing selections (example).

Roi.setName(name)
Sets the selection name.

Roi.setStrokeWidth(width)
Sets the selection stroke width.

Roi.setUnscalableStrokeWidth(width)

Sets the width of the line used to draw this selection and prevents the width from increasing when the image is zoomed. Requires 1.53p.
Roi.getStrokeWidth
Returns the stroke width of the current selection.
Roi.setGroup(group) - Sets the group (a positive number) of the current selection. Zero sets the group to "none". See also: RoiManager.selectGroup and RoiManager.setGroup.
Roi.getGroup - Returns the group (a positive number) of the current selection, or zero if the selection is not in a group.
Roi.setDefaultGroup(group) - Sets the default group (a positive number) of the current selection. Zero sets the default group to "none".
Roi.getDefaultGroup - Returns the default group (a positive number) of the current selection, or zero if the selection is not in a group.
Roi.setPosition(slice) - Sets the selection position. Requires 1.53b.
Roi.setPosition(channel, slice, frame) - Sets the selection position.
Roi.getPosition(channel, slice, frame) - Returns the selection position.
Roi.getPointPosition(index)
Returns the stack position of the point with the specified index in the current point selection.
Roi.setJustification(str) - Sets the justification ("left", "center" or "right") of the current text selection. Requires 1.53b.
Roi.setFontSize(size) - Sets the font size (in points) of the current text selection. Requires 1.53b.
Roi.getGroupNames - Returns the group names as a comma-delimeted string. Requires 1.53b.
Roi.setGroupNames(string) - Sets the group names from a comma-delimeted string. Requires 1.53b.
Roi.remove
Deletes the selection, if any, from the active image.
Roi.selectNone
Removes the current selection. Requires 1.53v15.
Roi.copy
Copies the selection on the active image to the selection clipboard. Requires 1.53u.
Roi.paste
Copies the selection on the selection clipboard to the active image. Requires 1.53u.
Roi.getFeretPoints(xpoints, ypoints)
Creates new x and y arrays with the end points of the MaxFeret in array elements [0],[1] and MinFeret in [2],[3] (example).

ROI Manager Functions

These function run ROI Manager commands. The ROI Manager is opened if it is not already open. Use *roiManager("reset")* to delete all ROIs on the list. Use setOption("Show All", boolean) to enable/disable "Show All" mode. Additional ROI Manager Functions. For examples, refer to the RoiManagerMacros, ROI Manager Stack Demo and RoiManagerSpeedTest macros.

roiManager("and")
Uses the conjunction operator on the selected ROIs, or all ROIs if none are selected, to create a composite selection.

roiManager("add")
Adds the current selection to the ROI Manager.

roiManager("add & draw")
Outlines the current selection and adds it to the ROI Manager.

roiManager("combine")
Uses the union operator on the selected ROIs to create a composite selection. Combines all ROIs if none are selected.

roiManager("count")
Returns the number of ROIs in the ROI Manager list. See also: RoiManager.size and RoiManager.selected.

roiManager("delete")
Deletes the selected ROIs from the list, or deletes all ROIs if none are selected.

roiManager("deselect")
Deselects all ROIs in the list. When ROIs are deselected, subsequent ROI Manager commands are applied to all ROIs.

roiManager("draw")
Draws the selected ROIs, or all ROIs if none are selected, using the equivalent of the *Edit>Draw* command.

roiManager("fill")
Fills the selected ROIs, or all ROIs if none are selected, using the equivalent of the *Edit>Fill* command.

roiManager("index")
Returns the index of the currently selected ROI on the list, or -1 if the list is empty or no ROIs are selected. Returns the index of the first selected ROI if more than one is selected

roiManager("measure")
Measures the selected ROIs, or if none is selected, all ROIs on the list.

roiManager("multi measure")
Measures all the ROIs on all slices in the stack, creating a Results Table with one row per slice.

roiManager("multi-measure append")
Measures all the ROIs on all slices in the stack, appending the measurements to the Results Table, with one row per slice.

roiManager("multi-measure one")
Measures all the ROIs on all slices in the stack, creating a Results Table with one row per measurement.

roiManager("multi plot")
Plots the selected ROIs, or all ROIs if none are selected, on a single graph.

roiManager("open", file-path)
Opens a .roi file and adds it to the list or opens a ZIP archive (.zip file) and adds all the selections contained in it to the list.

roiManager("remove slice info")
Removes the information in the ROI names that associates them with

particular stack slices.

roiManager("rename", name)
Renames the selected ROI. You can get the name of an ROI on the list using RoiManager.getName.

roiManager("reset")
Deletes all ROIs on the list.

roiManager("save", file-path)
Saves all the ROIs on the list in a ZIP archive.

roiManager("save selected", file-path)
Saves the selected ROI as a .roi file.

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")*. Note that macros that use this function sometimes run orders of magnitude faster in batch mode. Use *roiManager("deselect")* to deselect all items on the list. For an example, refer to the ROI Manager Stack Demo macro. See also: RoiManager.select and RoiManager.selectByName.

roiManager("select", indexes)
Selects multiple items in the ROI Manager list, where *indexes* is an array of integers, each of which must be greater than or equal to 0 and less than the value returned by *roiManager("count")*. The selected ROIs are not highlighted in the ROI Manager list and are no longer selected after the next ROI Manager command is executed.

roiManager("show all")
Displays all the ROIs as an overlay.

roiManager("show all with labels")
Displays all the ROIs as an overlay, with labels.

roiManager("show all without labels")
Displays all the ROIs as an overlay, without labels.

roiManager("show none")
Removes the ROI Manager overlay.

roiManager("size")
Returns the number of ROIs in the ROI Manager list.

roiManager("sort")
Sorts the ROI list in alphanumeric order.

roiManager("split")
Splits the current selection (it must be a composite selection) into its component parts and adds them to the ROI Manager.

roiManager("update")
Replaces the selected ROI on the list with the current selection.

Additional ROI Manager Functions

These functions for working with the ROI Manager are available in ImageJ 1.52u and later.

RoiManager.multiCrop(dir, options)
If 'options' contains "save", saves the contents of the selected ROIs in TIFF format as separate images, where 'dir' is the directory path. Add " png" or " jpg" to save in PNG or JPEG format. Add ' show' and the images will be displayed in a stack. Requires 1.53d.

RoiManager.getName(index)
Returns the name of the selection with the specified index, or an empty string if the selection does not have a name.

RoiManager.getIndex(name)
Returns the index of the first selection with the specified name, or -1 if no selection has that name. Requires 1.53s.

RoiManager.rotate(angle)
rotates the selected selections by 'angle'. Requires 1.53s.

RoiManager.rotate(angle, xcenter, ycenter)
Rotates the selected selections by 'angle', around 'xcenter' and 'ycenter'. Requires 1.53s.

RoiManager.scale(xscale, yscale, centered)
Scales the selected selections by 'xscale' and 'yscale'. Uses the center of the image as the origin if 'centered' is true. Requires 1.53s.

RoiManager.select(index)
Activates the selection at the specidied index.

RoiManager.selectByName(name)
Activates the selection with the specified name. Requires 1.53s.

RoiManager.selected
Returns the number of selected ROIs in the ROI Manager list. Requires 1.53e. See also: RoiManager.size

RoiManager.selectGroup(group)
Selects all ROIs in the ROI Manager that belong to *group*.

RoiManager.selectPosition(c, z, t)
Selects all ROIs in the ROI Manager that are at the specified position. As an example, to select all ROIs in a stack positioned at slice 5, use RoiManager.selectPosition(0, 5, 0). Requires 1.54d.

RoiManager.setGroup(group)
Sets the group of the selected ROIs. See also: Roi.setGroup, Roi.getGroup, Roi.setDefaultGroup and Roi.getDefaultGroup.

RoiManager.setPosition(slice)
Sets the position of the selected selections.

RoiManager.setPosition(channel, slice, frame)
Sets the position of the selected selections. Requires 1.54g.

RoiManager.size
Returns the number of ROIs in the ROI Manager list. See also: RoiManager.selected

RoiManager.translate(dx, dy)
Translates the selected selections by 'dx' and 'dy'. Requires 1.53s.

RoiManager.associateROIsWithSlices(boolean)
Sets the "Associate 'Show all' ROIs with slices" option. Requires 1.54a.

RoiManager.restoreCentered(boolean)
Sets the "Restore ROIs centered" option. Requires 1.54a.

RoiManager.useNamesAsLabels(boolean)
Sets the "Use ROI names as labels" option. Requires 1.54a.

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

run(command)
Executes an ImageJ menu command. Use the Command Recorder (*Plugins>Macros>Record*) to generate run() function calls.

run(command, options)
Executes an ImageJ menu command with arguments. The 'options' string 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 concatenation to pass a variable as an argument. With ImageJ 1.43 and later, variables can be passed without using string concatenation by adding "&" to the variable name. For examples, see the ArgumentPassingDemo macro. Plugins can directly retrieve the 'options' string by calling Macro.getOptions().

runMacro(name)
Runs the specified macro or script, which is assumed to be in the Image macros folder. A full file path may also be used. Returns any string argument returned by the macro or the last expression evaluated in the script. For an example, see the CalculateMean macro. See also: eval.

runMacro(name, arg)
Runs the specified macro or script, which is assumed to be in the macros folder, or use a full file path. The string argument 'arg' can be retrieved by the macro or script using the getArgument() function. Returns the string argument returned by the macro or the last expression evaluated in the script. See also: getArgument.

S [Top]

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", "results", "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.

selectionContains(x, y)
Returns *true* if the point *x,y* is inside the current selection. Aborts the macro if there is no selection. See also: Roi.contains

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: Roi.getName, Roi.setName, 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. See also: Roi.getType,

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. The *id* can also be an image title (a string).

selectWindow("name")
Activates the window with the title "name".

setAutoThreshold()
Uses the "Default" method to determine the threshold. It may select dark or bright areas as thresholded, as was the case with the *Image>Adjust>Threshold* "Auto" option in ImageJ 1.42o and earlier. See also: setThreshold, getThreshold, resetThreshold.

setAutoThreshold(method)
Uses the specified method to set the threshold levels of the current image. Use the getList("threshold.methods") function to get a list of the available methods. Add " dark" to the method name if the image has a dark background. Add " 16-bit" to use the full 16-bit histogram when calculating the threshold of 16-bit images. Add " stack" to use histogram of the entire stack when calculating the threshold. For an example, see the AutoThresholdingDemo macro.

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

setBackgroundColor(rgb)
Sets the background color, where *rgb* is an RGB pixel value. See also: getValue("rgb.background").

setBatchMode(arg)
Controls whether images are visible or hidden during macro execution. If *arg* is 'true', the interpreter enters batch mode and newly opened images are not displayed. If *arg* is 'false', exits batch mode and disposes of all hidden images except for the active image, which is displayed in a window. The interpreter also exits batch mode when the macro terminates, disposing of all hidden images.

With ImageJ 1.48h or later, you can use 'show' and 'hide' arguments to individually show or hide images. By not displaying and updating images, batch mode macros run up to 20 times faster. Examples: BatchModeTest, BatchMeasure, BatchSetScale and ReplaceRedWithMagenta.

setBatchMode("exit and display")
Exits batch mode and displays all hidden images.

setBatchMode("show")
Displays the active hidden image, while batch mode remains in same state.

setBatchMode("hide")
Enters (or remains in) batch mode and hides the active image

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")*. See also: Color.set.

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")*.

setColor(string)
Sets the drawing color, where 'string' can be "black", "blue", "cyan", "darkGray", "gray", "green", "lightGray", "magenta", "orange", "pink", "red", "white", "yellow", or a hex value like "#ff0000".

setFont(name, size[, style])
Sets the font used by the drawString function. The first argument is the font name. It should be "SansSerif", "Serif" or "Monospaced". The second is the point size. The optional third argument is a string containing "bold" or "italic", or both. The third argument can also contain the keyword "antialiased". With ImageJ 1.54e and newer, text antialiasing is enabled by default and you have to use the keyword 'no-smoothing' to display non-antialiased text. For examples, run the TextDemo macro.

setFont("user")
Sets the font to the one defined in the *Edit>Options>Fonts* window. See also: getInfo("font.name"), getValue("font.size") and getValue("font.height").

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

setForegroundColor(rgb)
Sets the foreground color, where *rgb* is an RGB pixel value. See also: getValue("rgb.foreground").

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. 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 window to a new location. Use Table.setLocationAndSize(x, y, width, height, title) to set the location and size of a specified window. Use *call("ij.gui.ImageWindow.setNextLocation", x, y)* to set the location of the next opened window. See also: getLocationAndSize, Table.setLocationAndSize, screenWidth, screenHeight.

setLocation(x, y, width, height)
Moves and resizes the active image window. The new location of the top-left corner is specified by *x* and *y*, and the new size by *width* and *height*. See also: Table.setLocationAndSize.

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("Info", string)
Assigns the metadata in *string* to the "Info" image property of the current image. This metadata is displayed by *Image>Show Info* and saved as part of the TIFF header. See also: getMetadata and Property.getInfo.

setMetadata("Label", string)
Sets *string* as the label of the current image or stack slice. The first 60 characters, or up to the first newline, of the label are displayed as part of the image subtitle. The labels are saved as part of the TIFF header. See also: Property.setSliceLabel and Property.getSliceLabel.

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

setMinAndMax(min, max, channels)
Sets the display range of specified channels in an RGB image, where 4=red, 2=green, 1=blue, 6=red+green, etc. Note that the pixel data is altered since RGB images, unlike composite color images, do not have a LUT for each channel.

setOption(option, boolean)
These functions enable or disable ImageJ options, where *boolean* is either *true* or *false*.

> **setOption(measurement, boolean)**
> Enable/disables *Analyze>Set Measurements* options where 'measurement' can be *"Display label"*, *"Limit to threshold"*, *"Area"*, *"Mean"*, *"Std"*, *"Perimeter"*, *"Stack position"* or *"Add to overlay"*.
> **setOption("AntialiasedText", boolean)**
> Controls the "Antialiased text" option in the *Edit>Options>Fonts* dialog. Requires v1.51h.
> **setOption("AutoContrast", boolean)**
> Enables/disables the *Edit>Options>Appearance* "Auto contrast stacks" option. You can also have newly displayed stack slices contrast enhanced by holding the shift key down when navigating stacks.

setOption("Bicubic", boolean)
Provides a way to force commands like *Edit>Selection>Straighten*, that normally use bilinear interpolation, to use bicubic interpolation.
setOption("BlackBackground", boolean)
Enables/disables the *Process>Binary>Options* "Black background" option.
setOption("Changes", boolean)
Sets/resets the 'changes' flag of the current image. Set this option *false* to avoid "Save Changes?" dialog boxes when closing images.
setOption("CopyHeaders", boolean)
Enables/disables the "Copy column headers" option in the *Edit>Options>Input/Output* dialog. See String.copyResults Requires v1.52p.
setOption("DebugMode", boolean)
Enables/disables the ImageJ debug mode. ImageJ displays information, such as TIFF tag values, when it is in debug mode.
setOption("DisablePopupMenu", boolean)
Enables/disables the the menu displayed when you right click on an image.
setOption("DisableUndo", boolean)
Enables/disables 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.
setOption("ExpandableArrays", boolean)
Enables/disables support for auto-expanding arrays (example).
setOption("FlipFitsImages", boolean)
Controls whether images are flipped vertically by the FITS reader.
setOption("FullRange16bitInversions", boolean)
Set to have 16-bit images inverted using the full range (0-65535). Requires 1.54d.
setOption("InterpolateLines", boolean)
Sets/resets the 'Interpolate line profiles' option in *Edit>Options>Plots*.
setOption("InvertY", boolean)
Sets/resets the 'invertY' option of the active image.
setOption("JFileChooser", boolean)
Enables/disables use of the Java JFileChooser to open and save files instead of the native OS file chooser.
setOption("Loop", boolean)
Enables/disables the *Image>Stacks>Tools>Animation Options* "Loop back and forth" option.
setOption("MonospacedText", boolean)
Enables/disables monospaced text in the "Log" window. Requires 1.54c11.
setOption("OpenUsingPlugins", boolean)
Controls whether standard file types (TIFF, JPEG, GIF, etc.) are opened by external plugins or by ImageJ (example).
setOption("QueueMacros", boolean)
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.
setOption("ScaleConversions", boolean)
Enables/disables the "Scale when converting" option in the *Edit>Options>Conversions* dialog. When this option is enabled (the

default), commands in the *Image>Type>* sub-menu scale from the min and max displayed pixel value to 0-255 when converting from 16-bit or 32-bit to 8-bit or to 0-65535 when converting from 32-bit to 16-bit. The min and max displayed pixel values can be set using the *Image>Adjust>Brightness/Contrast dialog* or the setMinAndMax() function. Call *setOption("CalibrateConversions", true)* to have conversions to 8-bit and 16-bit density calibrated. Requires v1.52k.

setOption("CalibrateConversions", boolean)
Enables/disables the "Calibrate conversions" option in the *Edit>Options>Conversions* dialog. Conversions to 8-bit and 16-bit are density calibrated when this option is enabled, so results from measurements stay the same.

setOption("Show All", boolean)
Enables/disables the the "Show All" mode in the ROI Manager.

setOption("ShowAngle", boolean)
Determines whether or not the "Angle" value is displayed in the Results window when measuring straight line lengths.

setOption("ShowMin", boolean)
Determines whether or not the "Min" value is displayed in the Results window when "Min & Max Gray Value" is enabled in the *Analyze>Set Measurements* dialog box.

setOption("WaitForCompletion", boolean)
Set *false* and the next exec() call will return null and not wait for the command being executed to finish. Requires v1.52u.

setOption("WandAllPoints", boolean)
Controls whether Wand selections with straight lines longer than one pixel should have intermediate points with single-pixel spacing. Requires v1.51q.

setPasteMode(mode)
Sets the transfer mode used by the *Edit>Paste* command, where *mode* is "Copy", "Blend", "Average", "Difference", "Transparent-white", "Transparent-zero", "AND", "OR", "XOR", "Add", "Subtract", "Multiply", "Divide, "Min" or "Max". The GetCurrentPasteMode macro demonstrates how to get the current paste mode. In ImageJ 1.42 and later, the paste mode is saved and restored by the saveSettings and restoreSettings.

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. With v1.47o or later, it can be a string. 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, ExtractRGBChannels 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. Use Stack.setPosition(channel, slice, frame) to set hyperstack positions. For an example, see the MeasureStack macros. See also: getSliceNumber, nSlices.

setThreshold(lower, upper)
Sets the lower and upper threshold levels. There is an optional third argument that can be "red", "black & white", "over/under", "no color" or "raw". With density calibrated images, *lower* and *upper* must be calibrated values unless the optional third argument is "raw". See also: setAutoThreshold, getThreshold, resetThreshold.

setTool(name)
Switches to the specified tool, where *name* is "rectangle", "roundrect", "elliptical", "brush", "polygon", "freehand", "line", "polyline", "freeline", "arrow", "angle", "point", "multipoint", "wand", "text", "zoom", "hand" or "dropper". Refer to the SetToolDemo, ToolShortcuts or ToolSwitcher, macros for examples. See also: IJ.getToolName.

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=unused, 11=zoom, 12=hand, 13=dropper, 14=angle, 15...21=custom tools. 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. A "um" unit will be converted to "μm". 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. Can display HTML formatted text (example).

showMessage("title", "message")
Displays "message" in a dialog box using "title" as the the dialog box title. Can display HTML formatted text (example).

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. See also: getBoolean.

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. Use negative values to show subordinate progress bars as moving dots (example).

showProgress(currentIndex, finalIndex)
Updates the progress bar, where the length of the bar is set to *currentIndex*/*finalIndex* of the maximum bar length. The bar is erased if *currentIndex*>*finalIndex* or *finalIndex*==0.

showStatus("message")
Displays a message in the ImageJ status bar. If *message* starts with '!', subsequent process messages are suppressed (example).

showText("string")
Displays a string in a text window.

showText("Title", "string")
Displays a string in a text window using the specified title.

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. Multiple delimiters in the *string* are merged (taken as one) and delimiters at the start or end of the *string* are ignored unless the delimiter is a single comma, a single semicolon or a regular expression. With ImageJ 1.49f or later, *delimiters* can be also a regular expression enclosed in parentheses, e.g. *delimiters*="(\n\n)" splits only at empty lines (two newline characters following each other). See also: String.join.

Note that split() may return empty strings when the second argument is ",", ";' or "\n". To avoid empty strings, use ",,", ";;" and "\n\n" as the second argument.

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

Stack (hyperstack) Functions
These functions allow you to get and set the position (channel, slice and frame) of a

hyperstack (a 4D or 5D stack). The HyperStackDemo demonstrates how to create a hyperstack and how to work with it using these functions

Stack.isHyperstack - Returns true if the current image is a hyperstack.
Stack.getDimensions(width, height, channels, slices, frames) Returns the dimensions of the current image.
Stack.setDimensions(channels, slices, frames) - Sets the 3rd, 4th and 5th dimensions of the current stack.
Stack.setChannel(n) - Displays channel *n* .
Stack.setSlice(n) - Displays slice *n* .
Stack.setFrame(n) - Displays frame *n* .
Stack.getPosition(channel, slice, frame) - Returns the current position.
Stack.setPosition(channel, slice, frame) - Displays the specified channel, slice and frame.
Stack.getFrameRate() - Returns the frame rate (FPS).
Stack.setFrameRate(fps) - Sets the frame rate.
Stack.getFrameInterval() - Returns the frame interval in time (T) units.
Stack.setFrameInterval(interval) - Sets the frame interval in time (T) units.
Stack.getUnits(X, Y, Z, Time, Value) - Returns the x, y, z, time and value units.
Stack.setUnits(X, Y, Z, Time, Value) - Sets the x, y, z, time and value units.
Stack.setXUnit(string) - Sets the X-dimension unit.
Stack.setYUnit(string) - Sets the Y-dimension unit.
Stack.setZUnit(string) - Sets the Z-dimension unit.
Stack.setTUnit(string) - Sets the time unit.
Stack.setDisplayMode(mode) - Sets the display mode, where *mode* is "composite", "color" or "grayscale". Requires a multi-channel stack.
Stack.getDisplayMode(mode) - Sets the string *mode* to the current display mode.
Stack.setActiveChannels(string) - Controls which channels in a composite color image are displayed, where *string* is a list of ones and zeros that specify the channels to display. For example, "101" causes channels 1 and 3 to be displayed.
Stack.getActiveChannels(string) - Returns a string that represents the state of the channels in a composite color image, where '1' indicates a displayed channel and '0' indicates an inactive channel.
Stack.toggleChannel(channel) - Switches the display state of the specified channel of a composite color image.
Stack.swap(n1, n2) - Swaps the two specified stack images, where *n1* and *n2* are integers greater than 0 and less than or equal to nSlices.
Stack.getStatistics(voxelCount, mean, min, max, stdDev) - Calculates and returns stack statistics.
Stack.setOrthoViews(x, y, z) - If an *Orthogonal Views* is active, its crosspoint is set to x, y, z (example).
Stack.getOrthoViews(x, y, z) - If an *Orthogonal Views* is active, its crosspoint is returned as x, y, z values.
Stack.getOrthoViewsID - Returns the image ID of the current *Orthogonal Views*, or zero if none is active.
Stack.getOrthoViewsIDs(XY, YZ, XZ) - If an *Orthogonal Views* is active, the three views are returned as XY, YZ, XZ image ID values.
Stack.stopOrthoViews - Stops the current *Orthogonal Views* and closes the "YZ" and "XZ" windows.

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

String Functions
These functions do string buffering and copy strings to and from the system clipboard. The CopyResultsToClipboard macro demonstrates their use.

These string functions, where 's' is a string variable, are also available in ImageJ 1.52t and newer: s.**charAt**(i), s.**contains(s2)**, s.**endsWith(s2)**, s.**indexOf(s2)**, s.**lastIndexOf(s2)**, s.**length**, s.**matches(s2)**, s.**replace(s1, s2)**, s.**startsWith(s2)**, s.**substring(i1, i2)**, s.**substring(i)**, s.**toLowerCase**, s.**toUpperCase**, s.**trim**.

String.resetBuffer - Resets (clears) the buffer.
String.append(str) - Appends *str* to the buffer.
String.buffer - Returns the contents of the buffer.
String.copy(str) - Copies *str* to the clipboard.
String.copyResults - Copies the Results table, or selected rows in the Results table, to the clipboard. Column headers are included if the "Copy column headers" option in the *Edit>Options>Input/Output* dialog is enabled. Use SetOption("CopyHeaders", boolean) to programmatically enable/disable this option.
String.getResultsHeadings - Returns the Results window headers. Example.
String.format(format, n1, n2, ...) - Returns a formatted string using the specified format and numbers.
String.pad(n, length) - Pads 'n' with leading zeros so that it is 'length' characters wide and returns the result. Note that 'n' can be either a number or a string. For an example, run the *Help>Examples>Macro>Stack Overlay* command.
String.paste - Returns the contents of the clipboard.
String.join(array) - Creates a ", " delimited string from the elements of 'array'. See also: split.
String.join(array, delimiter) - Creates a string from the elements of 'array', using the specified string delimiter.
String.trim(string) - Returns a copy of 'string' that has leading and trailing whitespace omitted. Can be replaced with string.trim() in ImageJ 1.52t and later.
String.show(str)
Displays *str* in a text window.
String.show(title, str)
Displays *str* in a text window using *title* as the title.
String.setFontSize(size)
Sets the size of the font used by drawString(). Requires 1.53v.

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. Can be replaced with str.substring(i1, i2) in ImageJ 1.52t and later. See also: indexOf, startsWith, endsWith, replace.

substring(string, index)
Returns a substring of *string* that begins at *index* and extends to the end of *string*. Can be replaced with str.substring(i) in ImageJ 1.52t and later.

T [Top]

Table Functions
These functions, added in ImageJ 1.52a, work with tables. They operate on the current (frontmost) table or, with most of the functions, an optional title argument can be provided (must be last argument). Examples: Sine/Cosine Tables, Rearrange Table and Access Tables.

Table.create(name) - Creates or resets a table. If the table exists, it is reset (cleared), otherwise a new table with the specified name is opened.
Table.reset(name) - Resets the specified table.
Table.size - The number of rows in the current table.
Table.title - The title (name) of the current table.
Table.headings - The column headings as a tab-delimited string.
Table.allHeadings - All 38 default measument headings as a tab-delimited string.
Table.get(columnName, rowIndex) - Returns the numeric value from the cell at the specified column and row.
Table.getString(columnName, rowIndex) - Returns a string value from the cell at the specified column and row.
Table.set(columnName, rowIndex, value) - Assigns a numeric or string value to the cell at the specified column and row.
Table.getColumn(columnName) - Returns the specified column as an array.
Table.setColumn(columnName, array) - Assigns an array to the specified column, where 'array' can contain either numbers and/or strings. A new column is added to the table if 'columnName' does not exist.
Table.renameColumn(oldName, newName) - Renames a column.
Table.deleteColumn(columnName) - Deletes the specified column.
Table.columnExists(columnName) - Returns 'true' if the specified column exists. Requires 1.53p.
Table.update - Updates the window displaying the current table.
Table.rename(oldName, newName) - Renames a table.
Table.setSelection(firstRowIndex, lastRowIndex) - Selects a range of rows in the current table. Use range (-1, -1) for selecting none.
Table.getSelectionStart - Returns the index of the first selected row in the current table, or - 1 if there is no selection.
Table.getSelectionEnd - Returns the index of the last selected row in the current table, or -1 if there is no selection.
Table.save(filePath) - Saves a table.
Table.open(filePath) - Opens a table.
Table.deleteRows(firstIndex, lastIndex) - Deletes specified rows.
Table.sort(column) - Sorts the table on the specified column (example).
Table.showRowNumbers(boolean) - Enable/disable display of row numbers. Default is 'false'. Does not work with the standard "Results" table, which always has row numbers.
Table.showRowIndexes(boolean) - Enable/disable display of row indexes.
Table.saveColumnHeader(boolean) - The column header row is not saved if the argument is 'false' (default is 'true').
Table.showArrays(title, array1, array2, ...) - Displays arrays in a table (same as Array.show). If *title* ends with "(indexes)", a 0-based Index column is shown. If it ends with "(row numbers)", the row number column is shown.

Table.applyMacro(code) - Applies macro code to each row of the table. Columns are assigned variable names as given by Table.headings. For columns (%Area, Perim. and Circ.), special characters are replaced by underscores (_Area, Perim_ and Circ_). New variables starting with an uppercase letter create a new column with this name. The variable 'row' (row index) is pre-defined. Currently only supports numeric values except for row labels.
Table.setLocationAndSize(x, y, width, height) - Sets the location and size of the current table. Works with other types of windows if a title is specified as the fifth argument.

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

toBinary(number)
Returns a binary string representation of *number*. Use IJ.pad to add leading zeros.

toHex(number)
Returns a hexadecimal string representation of *number*. Use IJ.pad to add leading zeros.

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, IJ.getToolName.

toScaled(x, y)
Converts unscaled pixel coordinates to scaled coordinates using the properties of the current image or plot. Also accepts arrays.

toScaled(x, y, z)
Converts unscaled (x, y, z) pixel coordinates to scaled coordinates.

toUnscaled(x, y)
Converts scaled coordinates to unscaled pixel coordinates using the properties of the current image or plot. Also accepts arrays. Refer to the AdvancedPlots macro set for examples.

toUnscaled(x, y, z)
Converts scaled (x, y, z) pixel coordinates to unscaled (raw) coordinates.

toScaled(length)
Converts (in place) a horizontal length in pixels to a scaled length using the properties of the current image. In a plot with Log scale, the scaled value is regarded as an exponent of 10.

toUnscaled(length)
Converts (in place) a scaled horizontal length to a length in pixels using the properties of the current image. In a plot with Log scale, the scaled value is regarded as an exponent of 10.

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

parseFloat and parseInt.

toString(number, decimalPlaces)
Converts *number* into a string, using the specified number of decimal places. See also: d2s.

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

U [Top]

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 [Top]

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

waitForUser(string)
Halts the macro and displays *string* in a dialog box. The macro proceeds when the user clicks "OK" or it is aborted if the user clicks on "Cancel". Unlike showMessage, the dialog box is not modal, so the user can, for example, create a selection or adjust the threshold while the dialog is open. To display a multi-line message, add newline characters ("\n") to *string*. This function is based on Michael Schmid's Wait_For_User plugin. Example: WaitForUserDemo.

waitForUser(title, message)
This is a two argument version of *waitForUser*, where *title* is the dialog box title and *message* is the text displayed in the dialog.

waitForUser
This is a no argument version of *waitForUser* that displays "Click OK to continue" in the dialog box.

top | home
Last updated 2023/10/18