home | news | docs | download | plugins | resources | list | links

# Built-in Macro Functions

[ A ][ B ][ C ][ D ][ E ][ F ][ G ][ H ][ I ][ J ][ K ][ L ][ M ]
[ N ][ O ][ P ][ 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.copy(array)** - Returns a copy of *array*.
> **Array.fill(array, value)** - Assigns the specified numeric
> value to each element of *array*.
> **Array.getStatistics(array, min, max, mean, stdDev)** -
> Returns the *min*, *max*, *mean*, and *stdDev* of *array*, which
> must contain all numbers.
> **Array.invert(array)** - Inverts the order of the elements in
> *array*. Requires 1.43h.
> **Array.sort(array)** - Sorts *array*, which must contain all
> numbers or all strings. String sorts are case-insensitive in
> v1.44i or later.
> **Array.rankPositions(array)** - Returns, as an array, the
> rank positions of *array*, which must contain all numbers
> or all strings (example). Requires 1.44k.
> **Array.trim(array, n)** - Returns an array that contains the
> first *n* elements of *array*.

**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. Note that the call() function does not work when ImageJ is running as an unsigned applet.

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

**charCodeAt(string, index)**
Returns the Unicode value of the character at the specified index in *string*. Index values can range from 0 to lengthOf(*string*). Use

the fromCharCode() function to convert one or more Unicode characters to a string.

**close()**
Closes the active image. This function has the advantage of not closing the "Log" or "Results" window when you meant to close the active image. Use *run("Close")* to close non-image windows.

**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".

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

> **Dialog.addMessage(string)** - Adds a message to the dialog. The message can be broken into multiple lines by inserting new line characters ("\n") into the string.
> **Dialog.addString("Label", "Initial text")** - Adds a text field to the dialog, using the specified label and initial text.
> **Dialog.addString("Label", "Initial text", 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.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.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.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.

**Dialog.show()** - Displays the dialog and waits until the user clicks "OK" or "Cancel". The macro terminates if the user clicks "Cancel".

**Dialog.getString()** - Returns a string containing the contents of the next text field.

**Dialog.getNumber()** - Returns the contents of the next numeric field.

**Dialog.getCheckbox()** - Returns the state (true or false) of the next checkbox.

**Dialog.getChoice()** - Returns the selected item (a string) from the next popup menu.

## doCommand("Command")

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

## doWand(x, y)

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

## 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.

**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. 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.

**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*,
for example
*eval("script","IJ.getInstance().setAlwaysOnTop(true);")*. Mac users,
and users of Java 1.5, must have a copy of JavaScript.jar in the
plugins folder.

**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). Refer to the ExecExamples macro for examples.

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

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

**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. The FileDemo macro demonstrates how to use them. 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.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 open(), saveAs(), File.open() or File.openAsString().
> **File.exists(path)** - Returns "1" (true) if the specified file exists.

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

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

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

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

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

**File.makeDirectory(path)** - Creates a directory.

**File.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. Requires 1.42m.

**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.

**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.openDialog(title)** - Displays a file open dialog and returns the path to the file choosen 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.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)** - Gets the name and formula 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. Requires 1.42f.

**Fit.showDialog** - Causes doFit() to display the simplex settings dialog. Requires 1.42f.

**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: 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".

**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. 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 a file separator, either "\" (Windows) or "/". Returns an empty string if the specified directory is not found or aborts the macro if the user cancels the "choose directory" dialog box. For examples, see the GetDirectoryDemo and ListFilesRecursively macros. See also: getFileList and the File functions.

> **getDirectory("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.
> **getDirectory("plugins")** - Returns the path to the plugins directory.
> **getDirectory("macros")** - Returns the path to the macros directory.
> **getDirectory("luts")** - Returns the path to the luts directory.
> **getDirectory("image")** - Returns the path to the directory that the active image was loaded from.
> **getDirectory("imagej")** - Returns the path to the ImageJ directory.
> **getDirectory("startup")** - Returns the path to the directory that ImageJ was launched from.
> **getDirectory("home")** - Returns the path to users home directory.
> **getDirectory("temp")** - Returns the path to the temporary directory (/tmp on Linux and Mac OS X).

**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. Requires 1.44k.

**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.

**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.

**getInfo("font.name")**
Returns the name of the current font. Requires 1.43f.

**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. Requires 1.43k.

**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. Requires 1.43h.

**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. Requires
1.43h.

**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("micrometer.abbreviation")**
Returns "µm", the abbreviation for micrometer. Requires 1.43d.

**getInfo("overlay")**
Returns information about the current image's overlay. Requires 1.43r.

**getInfo("log")**
Returns the contents of the Log window, or "" if the Log window is not open. Requires 1.44l.

**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". Requires 1.42m.

**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("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(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("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).

**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. Requires 1.40b.

**getMetadata("Label")**
Returns the current slice label. The first line of the this label (up to 60 characters) is display as part of the image subtitle. With DICOM stacks, returns the metadata from the DICOM header. See also: setMetadata. Requires 1.40b.

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

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

**getPixel(x, y)**
Returns the value of the pixel at *(x,y)*. Note that pixels in RGB

images contain red, green and blue components that need to be extracted using shifting and masking. See the Color Picker Tool macro for an example that shows how to do this.

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

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

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

### getResultLabel(row)
Returns the label of the specified row in the results table, or an empty string if *Display Label* is not checked in *Analalyze>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(xCoordinates, yCoordinates)**
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.

**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: setFont, 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("color.foreground")**
Returns the integer value of the current foregound color (example).

**getValue("color.background")**
Returns the integer value of the current background color.

**getValue("font.size")**
Returns the size, in points, of the current font. Requires 1.43f.

**getValue("font.height")**
Returns the height, in pixels, of the current font. Requires 1.43f.

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

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

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

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

**I** [ Top ]

**IJ Functions**
These functions provide access to miscellaneous methods in ImageJ's IJ class. Requires 1.43l.

> **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.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.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.redirectErrorMessages()** - Causes next image opening error to be redirected to the Log window and prevents the macro from being aborted (example). Requires 1.43n.

**IJ.renameResults(name)** - Changes the title of the Results table to the string *name*. Requires 1.44c.

## 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. Requires 1.44g.

## is("applet")

Returns *true* if ImageJ is running as an applet.

## 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). Requires v1.42n.

**is("Caps Lock Set")**

Returns *true* if the caps lock key is set. Always return *false* on some platforms. Requires v1.42e.

**is("changes")**

Returns *true* if the current image's 'changes' flag is set. Requires v1.42m.

**is("composite")**

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

**is("Inverting LUT")**

Returns *true* if the current image is using an inverting lookup table.

**is("hyperstack")**

Returns *true* if the current image is a hyperstack.

**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. This function is required because (n==NaN) is always false. Note that the numeric constant NaN is predefined in the macro language.

**isOpen(id)**

Returns *true* if the image with the specified ID is open.

**isOpen("Title")**

Returns *true* if the window with the specified title is open.

**L** [ 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.

**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. Requires v1.42i.
> **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 parameter names (as keys) and values. All parameters listed in the *Analyze>Set Measurements* dialog box are measured. Use List.getValue() in an assignment statement to retrieve the values. See the DrawEllipse macro for an example. Requires v1.42i.
> **List.setCommands** - Loads the ImageJ menu commands (as keys) and the plugins that implement them (as values). Requires v1.43f.

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

**M** [ Top ]

**makeEllipse(x1, y1, x2, y2, aspectRatio)**

Creates an elliptical selection, where *x1,y1,x2,y2* specify the major axis of the ellipse and *aspectRatio* is the ratio of the lengths of minor and major axis. Requires 1.44k.

**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)*.

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

**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)**
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. At least three coordinate pairs must be specified, but not more than 200. As an example, *makePolygon(20,48,59,13,101,40,75,77,38,70)* creates a polygon selection with five sides.

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

**makeRectangle(x, y, width, height, arcSize)**
Creates a rounded rectangular selection using the specified corner arc size. Requires 1.43n.

**makeSelection(type, xcoord, ycoord)**
Creates a selection from a list of XY coordinates. The first argument should be "polygon", "freehand", "polyline", "freeline", "angle" or "point", or the numeric value returned by selectionType. The

*xcoord* and *ycoord* arguments are numeric arrays that contain the X and Y coordinates. See the MakeSelectionDemo macro for examples.

**makeText(string, x, y)**
Creates a text selection at the specified coordinates. The selection will use the font and size specified by the last call to setFont(). The CreateOverlay macro provides an example. Requires 1.43h.

**matches(string, regex)**
Returns *true* if *string* matches the specified regular expression. See also: startsWith, endsWith, indexOf, replace.

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

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

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

**N** [ 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.

The ImageJ macro language does not directly support 2D arrays. As a work around, either create a blank image and use setPixel() and getPixel(), or create a 1D array using a=newArray(xmax*ymax) and do your own indexing (e.g., value=a[x+y*xmax]).

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

**newMenu(macroName, stringArray)**
Defines a menu that will be added to the toolbar when the menu tool named *macroName* is installed. Menu tools are macros with

names ending in "Menu Tool". *StringArray* is an array containing the menu commands. Returns a copy of *stringArray*. For an example, refer to the Toolbar Menus toolset.

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

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

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

## 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("http://imagej.nih.gov/ij/images/clown.gif").*

### 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. Requires v1.44e.

**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.drawRect(x, y, width, height)**
Draws a rectangle, where (*x,y*) specifies the upper left corner. Requires 1.44f.

**Overlay.drawEllipse(x, y, width, height)**
Draws an ellipse, where (*x,y*) specifies the upper left corner of the bounding rectangle. Requires 1.44f.

**Overlay.drawString("text", x, y)**
Draws text at the specified location. Call setFont() to specify the font.

**Overlay.show**
Displays the current overlay.

**Overlay.hide**
Hides the current overlay.

**Overlay.remove**
Removes the current overlay.

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

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

**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.

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

**Plot.setLimits(xMin, xMax, yMin, yMax)**
Sets the range of the x-axis and y-axis of plots created using *Plot.create()*. Must be called immediately after Plot.create().

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

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

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

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

**Plot.setJustification("center")**

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

**Plot.drawLine(x1, y1, x2, y2)**
Draws a line between *x1,y1* and *x2,y2*, using the coordinate system defined by *Plot.setLimits()*. Requires 1.42k.

**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).

**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.

## R [ Top ]

**random**
Returns a random number between 0 and 1.

**random("seed", seed)**
Sets the seed (a whole number) used by the *random()* function.

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

**replace(string, old, new)**
Returns the new string that results from replacing all occurrences of *old* in *string* with *new*, where *old* and *new* are single character strings. If *old* or *new* are longer than one character, each substring of *string* that matches the regular expression *old* is replaced with *new*. 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.

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

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

### roiManager(cmd)
Runs an ROI Manager command, where *cmd* must be "Add", "Add & Draw", "Update", "Delete", "Deselect", "Measure", "Draw", "Fill", "Label", "Combine", "Split", "Sort", "Reset" or "Multi Measure". The ROI Manager is opened if it is not already open. Use *roiManager("reset")* to delete all items on the list. Use setOption("Show All", boolean) to enable/disable "Show All" mode. For examples, refer to the RoiManagerMacros, ROI Manager Stack Demo and RoiManagerSpeedTest macros.

### roiManager(cmd, name)
Runs an ROI Manager I/O command, where *cmd* is "Open", "Save or "Rename", and *name* is a file path or name. The "Save" option ignores selections and saves all the ROIs as a ZIP archive. It displays a file save dialog if *name* is "". You can get the selection name using *call("ij.plugin.frame.RoiManager.getName", index)*. The ROI Manager is opened if it is not already open.

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

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

### 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.

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

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

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

### S [ 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", "measurements", "xy Coordinates" or "text". Use *saveAs(format)* to have a "Save As" dialog displayed.

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

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

**screenWidth**
Returns the screen width in pixels.

**selectionContains(x, y)**
Returns *true* if the point *x,y* is inside the current selection. Aborts the macro if there is no selection. Requires 1.44g.

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

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

**selectImage(id)**
Activates the image with the specified ID (a negative number). If *id* is greater than zero, activates the *id*th image listed in the Window menu. 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. Concatenate " dark" to the method name if the image has a dark background. For an example, see the AutoThresholdingDemo macro. Requires 1.42p.

**setBackgroundColor(r, g, b)**

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

### setBatchMode(arg)
If *arg* is *true*, the interpreter enters batch mode and images are not displayed, allowing the macro to run up to 20 times faster. If *arg* is *false*, exits batch mode and displays the active image in a window. ImageJ exits batch mode when the macro terminates if there is no *setBatchMode(false)* call. Note that a macro should not call *setBatchMode(true)* more than once.

Set *arg* to "*exit and display*" to exit batch mode and display all open batch mode images. Here are five example batch mode macros: BatchModeTest, BatchMeasure, BatchSetScale, ReplaceRedWithMagenta.

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

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

### setFont(name, size[, style])
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". 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"). Requires 1.43f.

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

**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. Requires
v1.40b.

**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: getMetadata. Requires v1.40b.

**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. Requires v1.42d.

**setOption(option, boolean)**
Enables or disables an ImageJ option, where *option* is one of the following options and *boolean* is either *true* or *false*.

"*DisablePopupMenu*" enables/disables the the menu displayed when you right click on an image.

"*Show All*" enables/disables the the "Show All" mode in the ROI Manager.

"*Changes*" sets/resets the 'changes' flag of the current image. Set this option *false* to avoid "Save Changes?" dialog boxes when closing images.

"*DebugMode*" enables/disables the ImageJ debug mode. ImageJ displays information, such as TIFF tag values, when it is in debug mode.

"*OpenUsingPlugins*" controls whether standard file types (TIFF, JPEG, GIF, etc.) are opened by external plugins or by ImageJ (example).

"*QueueMacros*" 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. Note that "QueueMacros" does not work with macros using function key shortcuts in ImageJ 1.41g and earlier.

"*DisableUndo*" 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.

*"Display Label"*, *"Limit to Threshold"*, *"Area"*, *"Mean"* and *"Std"*, added in v1.41, enable/disable the corresponding *Analyze>Set Measurements* options.

*"ShowMin"* 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.

*"BlackBackground"* enables/disables the *Process>Binary>Options* "Black background" option.

*"Bicubic"* provides a way to force commands like *Edit>Selection>Straighten*, that normally use bilinear interpolation, to use bicubic interpolation.

*"Loop"* enables/disables the *Image>Stacks>Tools>Animation Options* "Loop back and forth" option. Requires v1.44n.

### 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. 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. For an example, see the MeasureStack macros. See also: getSliceNumber, nSlices.

**setThreshold(lower, upper)**
Sets the lower and upper threshold levels. The values are uncalibrated except for 16-bit images (e.g., unsigned 16-bit images). There is an optional third argument that can be "red", "black & white", "over/under" or "no color". 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=spare, 11=zoom, 12=hand, 13=dropper, 14=angle, 15...21=spare. See also: toolID.

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

**setVoxelSize(width, height, depth, unit)**
Defines the voxel dimensions and unit of length ("pixel", "mm", etc.) for the current image or stack. A "um" unit will be converted to "µm" (requires v1.43). The *depth* argument is ignored if the current image is not a stack. See also: getVoxelSize.

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

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

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

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

**showStatus("message")**

Displays a message in the ImageJ status bar.

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

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

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

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

**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)** - Sets the position.
> **Stack.getFrameRate()** - Returns the frame rate (FPS).
> **Stack.setFrameRate(fps)** - Sets the frame rate.
> **Stack.setTUnit(string)** - Sets the time unit. Requires v1.42k.
> **Stack.setZUnit(string)** - Sets the Z-dimension unit.

Requires v1.42k.
**Stack.setDisplayMode(mode)** - Sets the display mode, where *mode* is "composite", "color" or "grayscale". Requires a multi-channel stack and v1.40a or later.
**Stack.getDisplayMode(mode)** - Sets the string *mode* to the current display mode. Requires v1.40a.
**Stack.setActiveChannels(string)** - Sets the active channels in a composite color image, where *string* is a list of ones and zeros that specify the channels to activate. For example, "101" activates channels 1 and 3.
**Stack.getActiveChannels(string)** - Returns a string that represents the state of the channels in a composite color image, where '1' indicates an active channel and '0' indicates an inactive channel. Requires v1.43d.
**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. Requires v1.40c.
**Stack.getStatistics(voxelCount, mean, min, max, stdDev)** - Calculates and returns stack statistics. Requires v1.42m.

### 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. See also: endsWith, indexOf, lastIndexOf, lengthOf, startsWith and substring.

**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 to the clipboard.
**String.paste** - Returns the contents of the clipboard.

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

### substring(string, index)

Returns a substring of *string* that begins at *index* and extends to the end of *string*.

**T** [ Top ]

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

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

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

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

**toolID**
Returns the ID of the currently selected tool. See also: setTool, IJ.getToolName.

**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.

**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". 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 dispayed in the dialog.

top | home | news | docs | download | plugins | resources | list | links
Last updated 2010/01/14