

# HIGHGUI. HIGH-LEVEL GUI AND MEDIA I/O

While OpenCV was designed for use in full-scale applications and can be used within functionally rich UI frameworks (such as Qt\*, WinForms\*, or Cocoa\*) or without any UI at all, sometimes there it is required to try functionality quickly and visualize the results. This is what the HighGUI module has been designed for.

It provides easy interface to:

- Create and manipulate windows that can display images and “remember” their content (no need to handle repaint events from OS).
- Add trackbars to the windows, handle simple mouse events as well as keyboard commands.
- Read and write images to/from disk or memory.
- Read video from camera or file and write video to a file.

## 4.1 User Interface

### createTrackbar

Creates a trackbar and attaches it to the specified window.

**C++:** `int createTrackbar(const string& trackbarname, const string& winname, int* value, int count, TrackbarCallback onChange=0, void* userdata=0)`

**C:** `int cvCreateTrackbar(const char* trackbarName, const char* windowName, int* value, int count, CvTrackbarCallback onChange)`

**Python:** `cv.CreateTrackbar(trackbarName, windowName, value, count, onChange) → None`

#### Parameters

**trackbarname** – Name of the created trackbar.

**winname** – Name of the window that will be used as a parent of the created trackbar.

**value** – Optional pointer to an integer variable whose value reflects the position of the slider. Upon creation, the slider position is defined by this variable.

**count** – Maximal position of the slider. The minimal position is always 0.

**onChange** – Pointer to the function to be called every time the slider changes position. This function should be prototyped as `void Foo(int, void*)`; , where the first parameter is the

trackbar position and the second parameter is the user data (see the next parameter). If the callback is the NULL pointer, no callbacks are called, but only `value` is updated.

**userdata** – User data that is passed as is to the callback. It can be used to handle trackbar events without using global variables.

The function `createTrackbar` creates a trackbar (a slider or range control) with the specified name and range, assigns a variable `value` to be a position synchronized with the trackbar and specifies the callback function `onChange` to be called on the trackbar position change. The created trackbar is displayed in the specified window `winname`.

---

**Note:** [Qt Backend Only] `winname` can be empty (or NULL) if the trackbar should be attached to the control panel.

---

Clicking the label of each trackbar enables editing the trackbar values manually.

## getTrackbarPos

Returns the trackbar position.

**C++:** `int getTrackbarPos(const string& trackbarname, const string& winname)`

**Python:** `cv2.getTrackbarPos(trackbarname, winname) → retval`

**C:** `int cvGetTrackbarPos(const char* trackbarName, const char* windowName)`

**Python:** `cv.GetTrackbarPos(trackbarName, windowName) → None`

### Parameters

**trackbarname** – Name of the trackbar.

**winname** – Name of the window that is the parent of the trackbar.

The function returns the current position of the specified trackbar.

---

**Note:** [Qt Backend Only] `winname` can be empty (or NULL) if the trackbar is attached to the control panel.

---

## imshow

Displays an image in the specified window.

**C++:** `void imshow(const string& winname, InputArray image)`

**Python:** `cv2.imshow(winname, image) → None`

**C:** `void cvShowImage(const char* winname, const CvArr* image)`

**Python:** `cv.ShowImage(winname, image) → None`

### Parameters

**winname** – Name of the window.

**image** – Image to be shown.

The function `imshow` displays an image in the specified window. If the window was created with the `CV_WINDOW_AUTOSIZE` flag, the image is shown with its original size. Otherwise, the image is scaled to fit the window. The function may scale the image, depending on its depth:

- If the image is 8-bit unsigned, it is displayed as is.

- If the image is 16-bit unsigned or 32-bit integer, the pixels are divided by 256. That is, the value range [0,255\*256] is mapped to [0,255].
- If the image is 32-bit floating-point, the pixel values are multiplied by 255. That is, the value range [0,1] is mapped to [0,255].

## namedWindow

Creates a window.

**C++:** void `namedWindow`(const string& **winname**, int **flags**)

**Python:** `cv2.namedWindow`(winname[, flags]) → None

**C:** int `cvNamedWindow`(const char\* **name**, int **flags**)

**Python:** `cv.NamedWindow`(name, flags=CV\_WINDOW\_AUTOSIZE) → None

### Parameters

**name** – Name of the window in the window caption that may be used as a window identifier.

**flags** – Flags of the window. Currently the only supported flag is CV\_WINDOW\_AUTOSIZE. If this is set, the window size is automatically adjusted to fit the displayed image (see `imshow()`), and you cannot change the window size manually.

The function `namedWindow` creates a window that can be used as a placeholder for images and trackbars. Created windows are referred to by their names.

If a window with the same name already exists, the function does nothing.

You can call `destroyWindow()` or `destroyAllWindows()` to close the window and de-allocate any associated memory usage. For a simple program, you do not really have to call these functions because all the resources and windows of the application are closed automatically by the operating system upon exit.

---

**Note:** Qt backend supports additional flags:

- **CV\_WINDOW\_NORMAL** or **CV\_WINDOW\_AUTOSIZE**: CV\_WINDOW\_NORMAL enables you to resize the window, whereas CV\_WINDOW\_AUTOSIZE adjusts automatically the window size to fit the displayed image (see `imshow()`), and you cannot change the window size manually.
- **CV\_WINDOW\_FREERATIO** or **CV\_WINDOW\_KEEPRATIO**: CV\_WINDOW\_FREERATIO adjusts the image with no respect to its ratio, whereas CV\_WINDOW\_KEEPRATIO keeps the image ratio.
- **CV\_GUI\_NORMAL** or **CV\_GUI\_EXPANDED**: CV\_GUI\_NORMAL is the old way to draw the window without statusbar and toolbar, whereas CV\_GUI\_EXPANDED is a new enhanced GUI.

By default, flags == CV\_WINDOW\_AUTOSIZE | CV\_WINDOW\_KEEPRATIO | CV\_GUI\_EXPANDED

---

## destroyWindow

Destroys a window.

**C++:** void `destroyWindow`(const string& **winname**)

**Python:** `cv2.destroyWindow`(winname) → None

**C:** void `cvDestroyWindow`(const char\* **name**)

**Python:** `cv.DestroyWindow`(name) → None

#### Parameters

**winname** – Name of the window to be destroyed.

The function `destroyWindow` destroys the window with the given name.

## destroyAllWindows

Destroys all of the HighGUI windows.

**C++:** `void destroyAllWindows()`

**Python:** `cv2.destroyAllWindows()` → None

**C:** `void cvDestroyAllWindows()`

**Python:** `cv.DestroyAllWindows()` → None

The function `destroyAllWindows` destroys all of the opened HighGUI windows.

## MoveWindow

Moves window to the specified position

**C:** `void cvMoveWindow(const char* name, int x, int y)`

**Python:** `cv.MoveWindow(name, x, y)` → None

#### Parameters

**name** – Window name

**x** – The new x-coordinate of the window

**y** – The new y-coordinate of the window

## ResizeWindow

Resizes window to the specified size

**C:** `void cvResizeWindow(const char* name, int width, int height)`

**Python:** `cv.ResizeWindow(name, width, height)` → None

#### Parameters

**name** – Window name

**width** – The new window width

**height** – The new window height

---

#### Note:

- The specified window size is for the image area. Toolbars are not counted.
  - Only windows created without `CV_WINDOW_AUTOSIZE` flag can be resized.
-

## SetMouseCallback

Sets mouse handler for the specified window

**C:** void **cvSetMouseCallback**(const char\* **name**, CvMouseCallback **onMouse**, void\* **param**=NULL )

**Python:** **cv.SetMouseCallback**(name, onMouse, param) → None

### Parameters

**name** – Window name

**onMouse** – Mouse callback. See OpenCV samples, such as <http://code.opencv.org/svn/opencv/trunk/opencv/samples/cpp/ffilldemo.cpp>, on how to specify and use the callback.

**param** – The optional parameter passed to the callback.

## setTrackbarPos

Sets the trackbar position.

**C++:** void **setTrackbarPos**(const string& **trackbarname**, const string& **winname**, int **pos**)

**Python:** **cv2.setTrackbarPos**(trackbarname, winname, pos) → None

**C:** void **cvSetTrackbarPos**(const char\* **trackbarName**, const char\* **windowName**, int **pos**)

**Python:** **cv.SetTrackbarPos**(trackbarName, windowName, pos) → None

### Parameters

**trackbarname** – Name of the trackbar.

**winname** – Name of the window that is the parent of trackbar.

**pos** – New position.

The function sets the position of the specified trackbar in the specified window.

---

**Note:** [Qt Backend Only] winname can be empty (or NULL) if the trackbar is attached to the control panel.

---

## waitKey

Waits for a pressed key.

**C++:** int **waitKey**(int **delay**=0)

**Python:** **cv2.waitKey**([delay]) → retval

**C:** int **cvWaitKey**(int **delay**=0 )

**Python:** **cv.WaitKey**(delay=0) → int

### Parameters

**delay** – Delay in milliseconds. 0 is the special value that means “forever”.

The function **waitKey** waits for a key event infinitely (when  $\text{delay} \leq 0$ ) or for **delay** milliseconds, when it is positive. Since the OS has a minimum time between switching threads, the function will not wait exactly **delay** ms, it will wait at least **delay** ms, depending on what else is running on your computer at that time. It returns the code of the pressed key or -1 if no key was pressed before the specified time had elapsed.