



# Mamba Realtime Python Reference - Windows

Automatically generated using pydoc

## Contents

<b>1</b>	<b>Introduction</b>	<b>3</b>
<b>2</b>	<b>Getting started : a small example</b>	<b>3</b>
<b>3</b>	<b>Realtime commands</b>	<b>4</b>
<b>4</b>	<b>API Reference</b>	<b>5</b>
4.1	Classes	5
4.1.1	MambaRealtimeError(exceptions.Exception)	5
4.2	Functions	5
4.2.1	activateRealtime()	5
4.2.2	addProcessRealtime(process, *args, **kwargs)	5
4.2.3	deactivateRealtime()	5
4.2.4	getErrorRealtime()	5
4.2.5	getSizeRealtime()	5
4.2.6	initializeRealtime(device, devType, seqlength=10)	6
4.2.7	isActivatedRealtime()	6
4.2.8	launchRealtime(device, devType, seqlength=10)	6
4.2.9	resetPaletteRealtime()	6
4.2.10	resetProcessRealtime()	6
4.2.11	setFrequencyRealtime(f)	6
4.2.12	setPaletteRealtime(palette)	6
4.2.13	setProcessRealtime(process, type, *args, **kwargs)	6
4.2.14	startRecordingRealtime(path)	6
4.2.15	stopRecordingRealtime()	6
4.2.16	takePictureRealtime(path)	7

## List of Figures

1	The Realtime window looking at herself (full display) . . . . .	4
---	---	---

## 1 Introduction

The Mamba Realtime module is an extension to the Mamba library for Python that allows you to test your algorithms in realtime. Images can be acquired dynamically from your webcam (or any video capture device supported) using the directshow API on Windows or Video4Linux2 (with a very limited support for V4L) on Linux. Alternatively you can use mambaRealtime to read any supported movie file and process it with your algorithm. The module then displays the result on your screen using the SDL library.

This document gives you the Python function reference along with the list of dynamic commands available in the realtime window.

Be warned that although we tried to make the functions system independent, there are still some differences between Windows and Linux implementations of this module (mainly with the initializing function). Make sure the documentation you are currently reading refers to the system you planned to target, the title should provide this information.

This document applies to version 0.2 .

## 2 Getting started : a small example

To start using the realtime, import the module:

```
#A from-import is the best way to use the realtime module.
#import works but is really painful to use since the name of the
# module is this long
from mambaRealtime import *
```

The second step is to start the image acquisition and display by either capturing them with a supported device or extracting them out of a movie file.

On Windows, any directshow API compatible device will be supported. The directshow API is part of directX and should be present by default on any modern updated Windows (XP, Vista, ...). Supported movie files are extremely various but the most commons are likely to be supported.

To initialize your directshow device:

```
# Directshow gives a number to each device , starting with 0. If you have
# many devices , there is actually no possibility to identify it clearly by a
# name (mambaRealtime lacks it) so you will have to try each number until you
# reach the correct one. Please also note that unplugging one of your device
# may have the consequence to change the order ... (Yeah I know it sucks).
# The number must be written inside a string.
launchRealtime("0", DSHOW)
```

To start playing a movie:

```
# The tag is AVC because the FFmpeg library is called libavcodec
# You should give a valid path to your movie file as first argument
launchRealtime("c:/path/to/your/movie", AVC)
```

At this point the mambaRealime module is now working.

This should have opened a new window like **1** in which you can see what your device is actually acquiring such as your smiling face (come on!) in front of your webcam.

As you can see, there is no color. Just like Mamba, the realtime module only works with greyscale images. This is not a limitation as mathematical morphology goes so you shouldn't be disturbed by it.

Once the realtime is active, you can specify a process to apply to the acquired images. Processes are Python functions with a specific prototype. Refer to the documentation of the function `setProcessRealtime()` (see [4.2.13](#)). Here are some examples using the mambaComposed gradient function:

```
# first import the gradient function
from mambaComposed import gradient

# Setting up a simple gradient as the realtime process
setProcessRealtime(gradient , INSTANT)
```

Here the gradient function was used in a very simple manner similar to this :

```
# A simple gradient
gradient(imIn , imOut)
```



Figure 1: The Realtime window looking at herself (full display)

But we know that the gradient function can take an optional argument modifying its thickness, like this :

```
# A 2 pixels thick gradient
gradient(imIn, imOut, 2)
```

In the mambaRealtime module you can specify that optional argument as well :

```
# Setting up a thick gradient as the realtime process
setProcessRealtime(gradient, INSTANT, 2)
```

Now you have a 2-pixels thick gradient applied to every image extracted from your selected source. Let's say you want to apply another algorithm to the result, like `whiteTopHatFilter` with 2 as size parameter (not sure it does anything interesting but it's for the sake of demonstration):

```
# Setting up a whiteTopHatFilter on top of the gradient (assuming
# the function was imported previously)
addProcessRealtime(whiteTopHatFilter, 2)
```

For more info on this function see [4.2.2](#).

At one point you are likely to give the mambaRealtime module wrong process functions or unsupported ones. In this case, the process will be dismissed and a small warning logo will be displayed in the top left corner. You can obtain the error info using the function:

```
# Prints the last error that occurred in the realtime
print getErrorRealtime()
```

Eventually, you can close the realtime module by either pressing escape inside the display windows or by calling the function:

```
# Close the realtime
deactivateRealtime()
```

At this point you can call back the launch function with other parameters or acquisition devices/files and it will start again.

There is a bunch of other functions, for recording, palette display and so on. Check the reference at the end of this document.

### 3 Realtime commands

Once the realtime window is active, you have access to commands that will display information, activate palette coloring, activate the process or allow you to close the window.

- `<esc>` closes the realtime acquisition. Once pressed you must reinitialize and reactivate it with the correct functions.

- **p** toggles the color palette. You should first specify one using the appropriate function.
- **o** toggles on/off the computation process. You should first specify one using the appropriate function.
- **r** displays framerate information in a white bar at the bottom of the display. If the bar is full, the framerate is equal to the one required (10 by default), half the bar is filled, then the framerate is half the one required and so on... You can see an example in figure 1
- **h** displays histogram of what is actually displayed. You can see an example in figure 1
- **f** toggles on/off the fullscreen display.
- **<pause>** toggles on/off the pause.

## 4 API Reference

This module provides an interface to the realtime module of Mamba. It provides functions to open the realtime thread and communicate with it.

### 4.1 Classes

#### 4.1.1 MambaRealtimeError(exceptions.Exception)

Mamba Realtime basic exception. Occurs when improper call to a function is made.

Method resolution order: MambaRealtimeError exceptions.Exception exceptions.BaseException \_\_builtin\_\_.object

### 4.2 Functions

#### 4.2.1 activateRealtime()

Activates the realtime module. Opens the acquisition device and creates the display in a separate thread. The thread ensures the acquisition and display of an image at a frequency of 10Hz with no treatment applied.

#### 4.2.2 addProcessRealtime(process, \*args, \*\*kwargs)

Adds the given 'process' (treatment) to the list of process to apply to the acquired image. This function will append the process to the list of processes currently applied.

'process' must be a function/method whom prototype must be process(imIn, imOut, ...) where imIn and imOut are mamba.imageMb objects. imIn is given by the acquisition device and imOut will be displayed.

Others arguments can be passed through 'args' and 'kwargs'.

If process is not compliant with this, the treatment will be deactivated in realtime.

Take also care to remove any display activation (showDisplay method).

#### 4.2.3 deactivateRealtime()

Deactivates the realtime module. Closes the acquisition device and the display.

#### 4.2.4 getErrorRealtime()

Returns the last error that occurred in the realtime thread. The thread does not produce exceptions so if your request did not work, use this function to get the error message.

The function will return an empty string if no error occurred.

#### 4.2.5 getSizeRealtime()

Returns a tuple containing the size of the images acquired and displayed by the realtime module.

If the realtime module is not properly initialized the returned size will be negative.

#### 4.2.6 initializeRealtime(device, devType, seqlength=10)

Initializes the realtime module using 'device' for the acquisition. 'devType' indicates the type of the device (either DSHOW or AVC). This function must be called before any other.

'seqlength' controls the length of the image sequence the thread is filling with the acquisition image. This sequence can be used in computation as an input.

#### 4.2.7 isActivatedRealtime()

Returns True if the realtime thread is active and alive.

#### 4.2.8 launchRealtime(device, devType, seqlength=10)

Initializes and activates the realtime module using 'device' for the acquisition.

The options are similar to the ones used in function initializeRealtime.

This function is similar to calling successively initializeRealtime and activateRealtime.

#### 4.2.9 resetPaletteRealtime()

Resets the palette used to display the acquired images in the realtime thread.

#### 4.2.10 resetProcessRealtime()

Resets the realtime thread so that no process (treatment) is applied to the images acquired and displayed.

#### 4.2.11 setFrequencyRealtime(f)

Sets the frequency 'f' of acquisition and display in the realtime thread. This is strongly limited by the acquisition module and the selected process.

The frequency is nonetheless limited inside the [1.0, 50.0] fps range by the realtime module.

#### 4.2.12 setPaletteRealtime(palette)

Sets the palette used to display the acquired images in the realtime thread. 'palette' can be any palette as defined in the mamba module.

#### 4.2.13 setProcessRealtime(process, type, \*args, \*\*kwargs)

Changes the 'process' (treatment) applied to the images acquired and displayed by the realtime thread. This function will replace any process or chains of processes that may be currently active.

if 'type' is set to INSTANT, 'process' must be a function/method whom prototype must be process(imIn, imOut, ...) where imIn and imOut are mamba.imageMb object. imIn is given by the acquisition device and imOut will be displayed.

if 'type' is set to SEQUENTIAL, 'process' must be a function/method whom prototype must be process(seqIn, seqIndex, imOut, ...) where imOut is a mamba.imageMb object. seqIn is a sequence as defined in module mambaComposed.sequence that contains the last 10 images obtained from the acquisition device. seqIndex indicates the index of the most recent. imOut will be displayed.

Others arguments can be passed through 'args' and 'kwargs'.

If process is not compliant with this, the treatment will be deactivated in realtime.

Take also care to remove any display activation (showDisplay method).

#### 4.2.14 startRecordingRealtime(path)

Starts the recording of the realtime footage into file 'path'. The created file will be a video encoded using MPEG2 codec (DVD format).

#### 4.2.15 stopRecordingRealtime()

Stops the recording of the realtime footage.

#### 4.2.16 `takePictureRealtime(path)`

Takes a picture of the current image displayed (at the moment the function is called) and puts it into file 'path'.