



# ImageFlip C++ library

---

v1.0.0

## Table of contents

---

- [Overview](#)
- [Versions](#)
- [Library files](#)
- [ImageFlip class description](#)
  - [Class declaration](#)
  - [getVersion method](#)
  - [initVFilter method](#)
  - [setParam method](#)
  - [getParam method](#)
  - [getParams method](#)
  - [executeCommand method](#)
  - [processFrame method](#)
  - [setMask method](#)
  - [encodeSetParamCommand method of VFilter class](#)
  - [encodeCommand method of VFilter class](#)
  - [decodeCommand method of VFilter class](#)
  - [decodeAndExecuteCommand method](#)
- [Data structures](#)
  - [VFilterCommand enum](#)
  - [VFilterParam enum](#)
- [VFilterParams class description](#)
  - [Class declaration](#)
  - [Serialize VFilter params](#)
  - [Deserialize VFilter params](#)
  - [Read params from JSON file and write to JSON file](#)
- [Build and connect to your project](#)
- [Example](#)

# Overview

The **ImageFlip** C++ library implements simple image flip function based on [OpenCV](#) and compatible with [VFilter](#) to be implemented in video processing pipelines. The library supports left-right, up-down and left-right-up-down image flip. The library depends on [VFilter](#) library (provides interface for video filter, source code included, Apache 2.0 license) and [OpenCV](#) library (version  $\geq 4.5$ , linked, Apache 2.0 license). The library uses C++17 standard.

## Versions

**Table 1** - Library versions.

Version	Release date	What's new
1.0.0	27.02.2024	First version of the library.

## Library files

The library supplied by source code only. The user would be given a set of files in the form of a CMake project (repository). The repository structure is shown below:

```
CMakeLists.txt ----- Main CMake file of the library.
3rdparty ----- Folder with third-party libraries.
  CMakeLists.txt ----- CMake file which includes third-party libraries.
  VFilter ----- Source code of the VFilter interface library.
src ----- Folder with source code of the library.
  CMakeLists.txt ----- CMake file of the library.
  ImageFlip.cpp ----- Source code file of the library.
  ImageFlip.h ----- Header file which includes ImageFlip class declaration.
  ImageFlipVersion.h ----- Header file which includes version of the library.
  ImageFlipVersion.h.in ---- CMake service file to generate version file.
test ----- Folder for test application.
  CMakeLists.txt ----- CMake file for test application.
  main.cpp ----- Source code file of test application.
```

## ImageFlip class description

# Class declaration

The **ImageFlip** class declared in **ImageFlip.h** file. Class declaration:

```
class ImageFlip : public VFilter
{
public:

    /// Get the version of the ImageFlip class.
    static std::string getVersion();

    /// Initialize image flip.
    bool initVFilter(VFilterParams& params) override;

    /// Set VFilter parameter.
    bool setParam(VFilterParam id, float value) override;

    /// Get the value of a specific VFilter parameter.
    float getParam(VFilterParam id) override;

    /// Get the structure containing all VFilter parameters.
    void getParams(VFilterParams& params) override;

    /// Execute a VFilter action command.
    bool executeCommand(VFilterCommand id) override;

    /// Process frame.
    bool processFrame(cr::video::Frame& frame) override;

    /// Set mask for filter.
    bool setMask(cr::video::Frame mask) override;

    /// Decode and execute command.
    bool decodeAndExecuteCommand(uint8_t* data, int size) override;
};
```

## getVersion method

The **getVersion()** static method returns string of current class version. Method declaration:

```
static std::string getVersion();
```

Method can be used without **ImageFlip** class instance:

```
std::cout << "ImageFlip version: " << cr::video::ImageFlip::getVersion();
```

Console output:

```
ImageFlip version: 1.0.0
```

## initVFilter method

The **initVFilter(...)** method initializes image flip. Method uses only parameter **mode** from [VFilterParams](#) class. Method declaration:

```
bool initVFilter(VFilterParams& params) override;
```

Parameter	Value
params	<a href="#">VFilterParams</a> class object. Method uses only parameter <b>mode</b> from <a href="#">VFilterParams</a> class. Mode: <b>0</b> - Off. <b>1</b> - Left-Right. <b>2</b> - Up-Down. <b>3</b> - Both.

**Returns:** TRUE if the video filter initialized or FALSE if not.

## setParam method

The **setParam (...)** method sets new parameters value. The library provides thread-safe **setParam(...)** method call. This means that the **setParam(...)** method can be safely called from any thread. Method declaration:

```
bool setParam(VFilterParam id, float value) override;
```

Parameter	Description
id	Parameter ID according to <a href="#">VFilterParam</a> enum. Method supports only parameters: <b>MODE</b> . Mode: <b>0</b> - Off. <b>1</b> - Left-Right. <b>2</b> - Up-Down. <b>3</b> - Both.
value	Mode value.

**Returns:** TRUE if the parameter was set or FALSE if not.

## getParam method

The **getParam(...)** method returns parameter value. The library provides thread-safe **getParam(...)** method call. This means that the **getParam(...)** method can be safely called from any thread. Method declaration:

```
float getParam(VFilterParam id) override;
```

Parameter	Description
id	Parameter ID according to <a href="#">VFilterParam</a> enum. Method supports only parameters: <b>MODE</b> and <b>PROCESSING_TIME_MCSEC</b> . For other parameters the method will return <b>-1</b> .

**Returns:** parameter value or **-1** if the parameters doesn't exist (not supported in particular implementation).

## getParams method

The **getParams(...)** method is designed to obtain all image flip params. The library provides thread-safe **getParams(...)** method call. This means that the **getParams(...)** method can be safely called from any thread. Method declaration:

```
void getParams(VFilterParams& params) override;
```

Parameter	Description
params	Reference to <a href="#">VFilterParams</a> object to store params. Method will fill parameters: <b>mode</b> and <b>processingTimeMcSec</b> . Other parameters will be set to <b>-1.0</b> .

## executeCommand method

The **executeCommand(...)** method executes video filter action command. Not supported by **ImageFlip** class. Method declaration:

```
bool executeCommand(VFilterCommand id) override;
```

Parameter	Description
id	Command ID according to <a href="#">VFilterCommand</a> enum. Not supported by <b>ImageFlip</b> class.

**Returns:** TRUE if the command executed or FALSE if not.

## processFrame method

The **processFrame(...)** method designed to process frame (image flip). The library provides thread-safe **processFrame(...)** method call. This means that the **processFrame(...)** method can be safely called from any thread. Method declaration:

```
bool processFrame(cr::video::Frame& frame) override;
```

Parameter	Description
frame	Reference to <a href="#">Frame</a> object.

**Returns:** TRUE if frame processed (image flip) or FALSE if not. If filter disabled the method will return TRUE without video frame processing.

## setMask method

The **setMask(...)** method designed to set video filter mask. Not supported by **ImageFlip** class. Method declaration:

```
bool setMask(cr::video::Frame mask) override;
```

Parameter	Description
mask	Filter mask is <a href="#">Frame</a> object with GRAY pixel format. Mask <b>not supported by ImageFlip</b> class.

**Returns:** TRUE in any cases.

## encodeSetParamCommand method of VFilter class

The **encodeSetParamCommand(...)** static method encodes command to change any **VFilter** parameter value. To control any video filter remotely, the developer has to design his own protocol and according to it encode the command and deliver it over the communication channel. To simplify this, the **VFilter** class contains static methods for encoding the control command. The **VFilter** class provides two types of commands: a parameter change command (SET\_PARAM) and an action command (COMMAND).

**encodeSetParamCommand(...)** designed to encode SET\_PARAM command. Method declaration:

```
static void encodeSetParamCommand(uint8_t* data, int& size, VFilterParam id, float value);
```

Parameter	Description
data	Pointer to data buffer for encoded command. Must have size >= 11.
size	Size of encoded data. Will be 11 bytes.
id	Parameter ID according to <a href="#">VFilterParam</a> enum.
value	Parameter value.

**SET\_PARAM** command format:

Byte	Value	Description
0	0x01	SET_PARAM command header value.

Byte	Value	Description
1	Major	Major version of VFilter class.
2	Minor	Minor version of VFilter class.
3	id	Parameter ID <b>int32_t</b> in Little-endian format.
4	id	Parameter ID <b>int32_t</b> in Little-endian format.
5	id	Parameter ID <b>int32_t</b> in Little-endian format.
6	id	Parameter ID <b>int32_t</b> in Little-endian format.
7	value	Parameter value <b>float</b> in Little-endian format.
8	value	Parameter value <b>float</b> in Little-endian format.
9	value	Parameter value <b>float</b> in Little-endian format.
10	value	Parameter value <b>float</b> in Little-endian format.

**encodeSetParamCommand(...)** is static and used without **VFilter** class instance. This method used on client side (control system). Command encoding example:

```
// Buffer for encoded data.
uint8_t data[11];
// Size of encoded data.
int size = 0;
// Random parameter value.
float outValue = static_cast<float>(rand() % 20);
// Encode command.
VFilter::encodeSetParamCommand(data, size, VFilterParam::LEVEL, outValue);
```

## encodeCommand method of VFilter class

The **encodeCommand(...)** static method encodes command for **VFilter** remote control. To control any video filter remotely, the developer has to design his own protocol and according to it encode the command and deliver it over the communication channel. To simplify this, the **VFilter** class contains static methods for encoding the control command. The **VFilter** class provides two types of commands: a parameter change command (SET\_PARAM) and an action command (COMMAND). **encodeCommand(...)** designed to encode COMMAND command (action command). Method declaration:

```
static void encodeCommand(uint8_t* data, int& size, VFilterCommand id);
```

Parameter	Description
data	Pointer to data buffer for encoded command. Must have size >= 7.
size	Size of encoded data. Will be 7 bytes.
id	Command ID according to <a href="#">VFilterCommand</a> enum.

**COMMAND** format:

Byte	Value	Description
0	0x00	COMMAND header value.
1	Major	Major version of VFilter class.
2	Minor	Minor version of VFilter class.
3	id	Command ID <b>int32_t</b> in Little-endian format.
4	id	Command ID <b>int32_t</b> in Little-endian format.
5	id	Command ID <b>int32_t</b> in Little-endian format.
6	id	Command ID <b>int32_t</b> in Little-endian format.

**encodeCommand(...)** is static and used without **VFilter** class instance. This method used on client side (control system). Command encoding example:

```
// Buffer for encoded data.
uint8_t data[7];
// Size of encoded data.
int size = 0;
// Encode command.
VFilter::encodeCommand(data, size, VFilterCommand::RESTART);
```

## decodeCommand method of VFilter class

The **decodeCommand(...)** static method decodes command on image filter side. Method declaration:

```
static int decodeCommand(uint8_t* data, int size, VFilterParam& paramId, VFilterCommand&
commandId, float& value);
```

Parameter	Description
data	Pointer to input command.
size	Size of command. Must be 11 bytes for SET_PARAM and 7 bytes for COMMAND.
paramId	VFilter parameter ID according to <a href="#">VFilterParam</a> enum. After decoding SET_PARAM command the method will return parameter ID.
commandId	VFilter command ID according to <a href="#">VFilterCommand</a> enum. After decoding COMMAND the method will return command ID.
value	VFilter parameter value (after decoding SET_PARAM command).

**Returns:** **0** - in case decoding COMMAND, **1** - in case decoding SET\_PARAM command or **-1** in case errors.



# decodeAndExecuteCommand method

The **decodeAndExecuteCommand(...)** method decodes and executes command encoded by [encodeSetParamCommand\(...\)](#) and [encodeCommand\(...\)](#) methods on video filter side. The library provides thread-safe **decodeAndExecuteCommand(...)** method call. This means that the **decodeAndExecuteCommand(...)** method can be safely called from any thread. Method declaration:

```
bool decodeAndExecuteCommand(uint8_t* data, int size) override;
```

Parameter	Description
data	Pointer to input command.
size	Size of command. Must be 11 bytes for SET_PARAM or 7 bytes for COMMAND.

**Returns:** TRUE if command decoded (SET\_PARAM or COMMAND) and executed (action command or set param command).

## Data structures

### VFilterCommand enum

Enum declaration:

```
enum class VFilterCommand
{
    /// Reset image filter algorithm.
    RESET = 1,
    /// Enable filter.
    ON,
    /// Disable filter.
    OFF
};
```

**Table 2** - Action commands description.

Command	Description
RESET	Not supported by ImageFlip class.
ON	Not supported by ImageFlip class.
OFF	Not supported by ImageFlip class.

# VFilterParam enum

Enum declaration:

```
enum class VFilterParam
{
    /// Current filter mode, usually 0 - off, 1 - on.
    MODE = 1,
    /// Enhancement level for particular filter, as a percentage in range from
    /// 0% to 100%.
    LEVEL,
    /// Processing time in microseconds. Read only parameter.
    PROCESSING_TIME_MCSEC,
    /// Type of the filter. Depends on the implementation.
    TYPE,
    /// VFilter custom parameter. Custom parameters used when particular image
    /// filter has specific unusual parameter.
    CUSTOM_1,
    /// VFilter custom parameter. Custom parameters used when particular image
    /// filter has specific unusual parameter.
    CUSTOM_2,
    /// VFilter custom parameter. Custom parameters used when particular image
    /// filter has specific unusual parameter.
    CUSTOM_3
};
```

**Table 3** - Params description.

Parameter	Access	Description
MODE	read / write	Image flip mode: <b>0</b> - Off. <b>1</b> - Left-Right. <b>2</b> - Up-Down. <b>3</b> - Both.
LEVEL	read / write	<b>Not supported by ImageFlip class.</b>
PROCESSING_TIME_MCSEC	read only	Processing time in microseconds. Read only parameter. Used to check performance of image flip.
TYPE	read / write	<b>Not supported by ImageFlip class.</b>
CUSTOM_1	read / write	<b>Not supported by ImageFlip class.</b>
CUSTOM_2	read / write	<b>Not supported by ImageFlip class.</b>
CUSTOM_3	read / write	<b>Not supported by ImageFlip class.</b>

# VFilterParams class description

## Class declaration

The **VFilterParams** class is used to provide video filter parameters structure. Also **VFilterParams** provides possibility to write/read params from JSON files (**JSON\_READABLE** macro) and provides methods to encode and decode params. **VFilterParams** interface class declared in **VFilter.h** file. Class declaration:

```
class VFilterParams
{
public:
    /// Current filter mode, usually 0 - off, 1 - on.
    int mode{ 0 };
    /// Enhancement level for particular filter, as a percentage in range from
    /// 0% to 100%.
    float level{ 0 };
    /// Processing time in microseconds. Read only parameter.
    int processingTimeMcSec{ 0 };
    /// Type of the filter. Depends on the implementation.
    int type{ 0 };
    /// VFilter custom parameter. Custom parameters used when particular image
    /// filter has specific unusual parameter.
    float custom1{ 0.0f };
    /// VFilter custom parameter. Custom parameters used when particular image
    /// filter has specific unusual parameter.
    float custom2{ 0.0f };
    /// VFilter custom parameter. Custom parameters used when particular image
    /// filter has specific unusual parameter.
    float custom3{ 0.0f };

    /// Macro from ConfigReader to make params readable/writable from JSON.
    JSON_READABLE(VFilterParams, mode, level, type, custom1, custom2, custom3)

    /// operator =
    VFilterParams& operator= (const VFilterParams& src);

    /// Encode (serialize) params.
    bool encode(uint8_t* data, int bufferSize, int& size,
                VFilterParamsMask* mask = nullptr);

    /// Decode (deserialize) params.
    bool decode(uint8_t* data, int dataSize);
};
```

**Table 4 - VFilterParams** class fields description is related to [VFilterParam enum](#) description.

Field	type	Description
mode	int	Image flip mode: <b>0</b> - Off. <b>1</b> - Left-Right. <b>2</b> - Up-Down. <b>3</b> - Both.
level	float	<b>Not supported by ImageFlip class.</b>
processingTimeMcSec	int	Processing time in microseconds. Read only parameter. Used to check performance of image flip.
type	int	<b>Not supported by ImageFlip class.</b>
custom1	float	<b>Not supported by ImageFlip class.</b>
custom2	float	<b>Not supported by ImageFlip class.</b>
custom3	float	<b>Not supported by ImageFlip class.</b>

**None:** *VFilterParams* class fields listed in Table 4 **have to** reflect params set/get by methods *setParam(...)* and *getParam(...)*.

## Serialize VFilter params

[VFilterParams](#) class provides method **encode(...)** to serialize VFilter params. Serialization of **VFilterParams** is necessary in case when image filter parameters have to be sent via communication channels. Method provides options to exclude particular parameters from serialization. To do this method inserts binary mask (1 byte) where each bit represents particular parameter and **decode(...)** method recognizes it. Method declaration:

```
bool encode(uint8_t* data, int bufferSize, int& size, VFilterParamsMask* mask = nullptr);
```

Parameter	Value
data	Pointer to data buffer. Buffer size must be >= 48 bytes.
bufferSize	Data buffer size. Buffer size must be >= 48 bytes.
size	Size of encoded data.
mask	Parameters mask - pointer to <b>VFilterParamsMask</b> structure. <b>VFilterParamsMask</b> (declared in <i>VFilter.h</i> file) determines flags for each field (parameter) declared in <a href="#">VFilterParams class</a> . If the user wants to exclude any parameters from serialization, he can put a pointer to the mask. If the user wants to exclude a particular parameter from serialization, he should set the corresponding flag in the <b>VFilterParamsMask</b> structure.

**Returns:** TRUE if params encoded (serialized) or FALSE if not.

**VFilterParamsMask** structure declaration:

```

struct VFilterParamsMask
{
    bool mode{ true };
    bool level{ true };
    bool processingTimeMcSec{ true };
    bool type{ true };
    bool custom1{ true };
    bool custom2{ true };
    bool custom3{ true };
};

```

Example without parameters mask:

```

// Prepare parameters.
cr::video::VFilterParams params;
params.level = 80.0f;

// Encode (serialize) params.
int bufferSize = 128;
uint8_t buffer[128];
int size = 0;
params.encode(buffer, bufferSize, size);

```

Example with parameters mask:

```

// Prepare parameters.
cr::video::VFilterParams params;
params.level = 80.0;

// Prepare mask.
cr::video::VFilterParams mask;
// Exclude level.
mask.level = false;

// Encode (serialize) params.
int bufferSize = 128;
uint8_t buffer[128];
int size = 0;
params1.encode(buffer, bufferSize, size, &mask);

```

## Deserialize VFilter params

[VFilterParams](#) class provides method **decode(...)** to deserialize params. Deserialization of VFilterParams is necessary in case when it is needed to receive params via communication channels. Method automatically recognizes which parameters were serialized by **encode(...)** method. Method declaration:

```

bool decode(uint8_t* data, int dataSize);

```

Parameter	Value
data	Pointer to data buffer with serialized params.

Parameter	Value
dataSize	Size of command data.

**Returns:** TRUE if params decoded (deserialized) or FALSE if not.

Example:

```
// Prepare parameters.
cr::video::VFilterParams params1;
params1.level = 80;

// Encode (serialize) params.
int bufferSize = 128;
uint8_t buffer[128];
int size = 0;
params1.encode(buffer, bufferSize, size);

// Decode (deserialize) params.
cr::video::VFilterParams params2;
params2.decode(buffer, size);
```

## Read params from JSON file and write to JSON file

**VFilter** depends on open source [ConfigReader](#) library which provides method to read params from JSON file and to write params to JSON file. Example of writing and reading params to JSON file:

```
// write params to file.
cr::utils::ConfigReader inConfig;
cr::video::VFilterParams in;
inConfig.set(in, "vFilterParams");
inConfig.writeToFile("VFilterParams.json");

// Read params from file.
cr::utils::ConfigReader outConfig;
if(!outConfig.readFromFile("VFilterParams.json"))
{
    cout << "Can't open config file" << endl;
    return false;
}
```

**VFilterParams.json** will look like:

```

{
  "vFilterParams":
  {
    "level": 50,
    "mode": 2,
    "type": 1,
    "custom1": 0.7f,
    "custom2": 12.0f,
    "custom3": 0.61f
  }
}

```

## Build and connect to your project

Typical commands to build **ImageFlip**:

```

cd ImageFlip
git submodule update --init --recursive
mkdir build
cd build
cmake ..
make

```

If you want connect **ImageFlip** to your CMake project as source code you can make follow. For example, if your repository has structure:

```

CMakeLists.txt
src
  CMakeList.txt
  yourLib.h
  yourLib.cpp

```

Create folder **3rdparty** in your repository and copy **ImageFlip** repository folder there. New structure of your repository:

```

CMakeLists.txt
src
  CMakeList.txt
  yourLib.h
  yourLib.cpp
3rdparty
  ImageFlip

```

Create CMakeLists.txt file in **3rdparty** folder. CMakeLists.txt should contain:

```

cmake_minimum_required(VERSION 3.13)

#####
## 3RD-PARTY
## dependencies for the project

```

```
#####
project(3rdparty LANGUAGES CXX)

#####
## SETTINGS
## basic 3rd-party settings before use
#####
# To inherit the top-level architecture when the project is used as a submodule.
SET(PARENT ${PARENT}_YOUR_PROJECT_3RDPARTY)
# Disable self-overwriting of parameters inside included subdirectories.
SET(${PARENT}_SUBMODULE_CACHE_OVERWRITE OFF CACHE BOOL "" FORCE)

#####
## CONFIGURATION
## 3rd-party submodules configuration
#####
SET(${PARENT}_SUBMODULE_IMAGE_FLIP ON CACHE BOOL "" FORCE)
if (${PARENT}_SUBMODULE_IMAGE_FLIP)
    SET(${PARENT}_IMAGE_FLIP ON CACHE BOOL "" FORCE)
    SET(${PARENT}_IMAGE_FLIP_TEST OFF CACHE BOOL "" FORCE)
endif()

#####
## INCLUDING SUBDIRECTORIES
## Adding subdirectories according to the 3rd-party configuration
#####
if (${PARENT}_SUBMODULE_IMAGE_FLIP)
    add_subdirectory(ImageFlip)
endif()
```

File **3rdparty/CMakeLists.txt** adds folder **ImageFlip** to your project and excludes test application from compiling. Your repository new structure will be:

```
CMakeLists.txt
src
  CMakeList.txt
  yourLib.h
  yourLib.cpp
3rdparty
  CMakeLists.txt
  ImageFlip
```

Next you need include folder 3rdparty in main **CMakeLists.txt** file of your repository. Add string at the end of your main **CMakeLists.txt**:

```
add_subdirectory(3rdparty)
```

Next you have to include **ImageFlip** library in your **src/CMakeLists.txt** file:

```
target_link_libraries(${PROJECT_NAME} ImageFlip)
```

Done!



# Example

The example creates artificial image and provides user interface to control image flip mode:

```
#include <iostream>
#include <stdint>
#include <string.h>
#include <opencv2/opencv.hpp>
#include "ImageFlip.h"

int main(void)
{
    std::cout << "ImageFlip v" <<
    cr::video::ImageFlip::getVersion() << std::endl;

    // Prepare source artificial image.
    const int width = 1280;
    const int height = 720;
    cv::Mat srcImg = cv::Mat::zeros(height, width, CV_8UC3);
    cr::video::Frame frame(width, height, cr::video::Fourcc::BGR24);
    cv::circle(srcImg, cv::Point(250, 250),
               50, cv::Scalar(0, 0, 255), cv::FILLED, cv::LINE_AA);

    // Create image flip object and params.
    cr::video::ImageFlip imageFlip;
    cr::video::VFilterParams params;

    // Main loop.
    while (true)
    {
        // Copy source image data for Frame object.
        memcpy(frame.data, srcImg.data, frame.size);

        // Image flip.
        imageFlip.processFrame(frame);

        // Get image flip params (VFilterParams).
        imageFlip.getParams(params);

        // Draw params.
        cv::Mat resultImg(height, width, CV_8UC3, frame.data);
        std::string mode;
        switch (params.mode) {
            case 0: mode = "OFF"; break;
            case 1: mode = "LEFT-RIGHT"; break;
            case 2: mode = "UP-DOWN"; break;
            case 3: mode = "BOTH"; break;
            default: mode = "OFF"; break; }
        cv::putText(resultImg, "[1/2]Mode: " + mode,
                   cv::Point(5, 25), cv::FONT_HERSHEY_SIMPLEX, 0.7,
                   cv::Scalar(255, 255, 0), 1, cv::LINE_AA);
        cv::putText(resultImg, "Time, mcsec: " +
                   std::to_string(params.processingTimeMcSec),
```

```
        cv::Point(5, 50), cv::FONT_HERSHEY_SIMPLEX, 0.7,  
        cv::Scalar(255, 255, 0), 1, cv::LINE_AA);  
  
    // Show results.  
    cv::imshow("ImageFlip", resultImg);  
  
    // Process keyboard events.  
    switch (cv::waitKey(30))  
    {  
    case 27:  
        return 1;  
    case 49:  
        imageFlip.setParam(cr::video::VFilterParam::MODE, params.mode - 1);  
        break;  
    case 50:  
        imageFlip.setParam(cr::video::VFilterParam::MODE, params.mode + 1);  
        break;  
    }  
    }  
    return 1;  
}
```