



cvsCamCtrl Programmer's Guide

1. Overview.....	1
2. Definitions and Constants.....	1
2.1 Event Definitions.....	1
2.1.1 Grab Event.....	1
2.1.2 Device Event.....	1
2.2 Error Code Definitions.....	2
2.3 Device Information Definitions.....	3
2.4 XML Version.....	3
2.5 Parameter Visibility.....	4
3. Key Structure Definitions.....	4
3.1 CVS_IMAGE Structure.....	4
3.2 CVS_BUFFER Structure.....	4
3.3 CVS_EVENT Structure.....	4
4. Color Conversion (enum ConvertColor).....	5
5. Functions.....	5
5.1 Callback Function Definitions.....	5
5.1.1 Grab Event Callback.....	5
5.1.2 Device Event Callback.....	6
5.2 System Management.....	6
5.2.1 ST_InitSystem.....	6
5.2.2 ST_FreeSystem.....	7
5.2.3 ST_IsInitSystem.....	7
5.3 Device Discovery and Information Retrieval.....	8
5.3.1 ST_GetAvailableCameraNum.....	8
5.3.2 ST_UpdateDevice.....	8
5.3.3 ST_GetEnumDeviceID.....	9
5.3.4 ST_GetEnumDeviceInfo.....	10
5.4 Device Connection and Control.....	10
5.4.1 ST_IsConnectable.....	10
5.4.2 ST_OpenDevice.....	11
5.4.3 ST_IsOpenDevice.....	12
5.4.4 ST_CloseDevice.....	12
5.4.5 ST_GetInterface.....	13
5.5 Image Acquisition and Configuration.....	14
5.5.1 ST_AcqStart.....	14
5.5.2 ST_AcqStop.....	14
5.5.3 ST_DoAbortGrab.....	15
5.5.4 ST_SetGrabTimeout.....	15
5.5.5 ST_GetGrabTimeout.....	16
5.5.6 ST_SetBufferCount.....	17
5.5.7 ST_GetBufferCount.....	17
5.6 Image Processing and Status Check.....	18
5.6.1 ST_GrabImage.....	18
5.6.2 ST_SingleGrabImage.....	19
5.6.3 ST_GetImageAvailable.....	19
5.6.4 ST_GetGrabCount.....	20

5.6.5 ST_GetFrameRate.....	21
5.6.6 ST_GetBandwidth.....	21
5.6.7 ST_GetLastError.....	22
5.6.8 ST_GetLastErrorDescription.....	22
5.7 Buffer Management.....	23
5.7.1 ST_InitBuffer.....	23
5.7.2 ST_FreeBuffer.....	23
5.8 Image Color Conversion.....	24
5.8.1 ST_CvtColor.....	24
5.9 Log and Configuration Management.....	25
5.9.1 ST_SetDetailedLog.....	25
5.9.2 ST_GetDetailedLog.....	26
5.10 Register Control.....	26
5.10.1 ST_SetIntReg.....	26
5.10.2 ST_GetIntReg.....	27
5.10.3 ST_SetFloatReg.....	28
5.10.4 ST_GetFloatReg.....	28
5.10.5 ST_SetEnumReg.....	29
5.10.6 ST_GetEnumReg.....	29
5.10.7 ST_SetStrReg.....	30
5.10.8 ST_GetStrReg.....	31
5.10.9 ST_SetCmdReg.....	31
5.11 Register Information Query.....	32
5.11.1 ST_GetIntRegRange.....	32
5.11.2 ST_GetFloatRegRange.....	33
5.11.3 ST_GetEnumEntrySize.....	34
5.11.4 ST_GetEnumEntryIntValue.....	34
5.11.5 ST_GetEnumEntryValue.....	35
5.12 Parameter Read and Write.....	36
5.12.1 ST_ReadParam.....	36
5.12.2 ST_WriteParam.....	37
5.13 Event and Callback Management.....	38
5.13.1 ST_RegisterGrabCallback.....	38
5.13.2 ST_UnregisterGrabCallback.....	39
5.13.3 ST_RegisterEventCallback.....	39
5.13.4 ST_UnregisterEventCallback.....	40
5.14 XML and JSON File Management.....	41
5.14.1 ST_ImportXML.....	41
5.14.2 ST_ExportXML.....	41
5.14.3 ST_ImportJson.....	42
5.14.4 ST_ExportJson.....	42
6. Cautions.....	43
7. License and Copyright.....	43

1. Overview

cvscamCtrl is a virtual frame grabber library provided by Crevis. This manual provides explanations of the functions, error codes, and data structures available within the library.

2. Definitions and Constants

2.1 Event Definitions

2.1.1 Grab Event

Grab events are triggered automatically during the image acquisition process from the camera. These events occur when a new image is received, an error occurs, or a timeout happens. You can detect these events and handle them within a callback function.

Event Name	Value	Description
EVENT_NEW_IMAGE	0x0	A new image has been successfully received.
EVENT_GRAB_ERROR	0x1	An error occurred during the grab process.
EVENT_GRAB_TIMEOUT	0x2	The grab operation timed out.

2.1.2 Device Event

Device events are automatically triggered when the camera's device status changes. These events allow for real-time monitoring of important information like device connection status, exposure completion, and temperature.

Note: Some events may only be supported by **GEV**(GigEVision) or **U3V**(USB3Vision), and the occurrence of events may vary depending on the interface being

Event Name	Value	Description
EVENT_DEVICE_DISCONNECT	0x0010	The device has been disconnected.
EVENT_EXPOSURE_END	0x0014	The sensor's exposure process is complete.
EVENT_ACQUISITION_TRANSFER_START	0x0018	Image data transmission has started.
EVENT_ACQUISITION_TRANSFER_END	0x001C	Image data transmission is complete.
EVENT_OVER_TRIGGER	0x0020	An excessive trigger signal has been detected.
EVENT_WARNING_TEMPERATURE_STATE	0x0030	The device's temperature has reached a warning level.
EVENT_CRITICAL_TEMPERATURE_STATE	0x0034	The device's temperature has reached a critical level.

2.2 Error Code Definitions

Error codes represent various issues that can occur. A value of 0 indicates success, while a negative value indicates an error.

Error Code	Value	Description
MCAM_ERR_OK	0	OK
MCAM_ERR_GENERIC_ERROR	-1001	Generic error
MCAM_ERR_NOT_INITIALIZED	-1002	Not initialized
MCAM_ERR_NOT_IMPLEMENTED	-1003	Not implemented
MCAM_ERR_BUSY	-1004	Device is in use
MCAM_ERR_STATE_ERROR	-1005	State error
MCAM_ERR_NOT_CONNECTED	-1006	Not connected
MCAM_ERR_NOT_FOUND	-1007	Not found
MCAM_ERR_NO_MORE_ITEM	-1008	No more items
MCAM_ERR_INVALID_PARAMETER	-1009	Invalid parameter
MCAM_ERR_FILE_ERROR	-1010	File error
MCAM_ERR_TIMEOUT	-1011	Timeout
MCAM_ERR_ABORTED	-1012	Operation aborted
MCAM_ERR_BUFFER_TOO_SMALL	-1013	Buffer size is too small
MCAM_ERR_CANNOT_OPEN_FILE	-1014	Failed to open file
MCAM_ERR_THREAD_ERROR	-1015	Thread error
MCAM_ERR_INVALID_DATA_FORMAT	-1016	Invalid data format
MCAM_ERR_NOT_ENOUGH_MEMORY	-1017	Not enough memory
MCAM_ERR_CANCEL	-1018	Canceled
MCAM_ERR_PENDING	-1019	Pending
MCAM_ERR_NO_LICENSE	-1020	No license
MCAM_ERR_CANT_READ_MANIFEST	-1021	Failed to read manifest
MCAM_ERR_NOT_SUPPORTED	-1022	Not supported
MCAM_ERR_ERR_OVERFLOW	-1023	Overflow error
MCAM_ERR_IMAGE_ERROR	-1024	Image error
MCAM_ERR_MISSING_PACKETS	-1025	Missing packets
MCAM_ERR_TOO_MANY_RESENDS	-1026	Exceeded maximum resends
MCAM_ERR_RESENDS_FAILURE	-1027	Resend failure
MCAM_ERR_TOO_MANY_CONSECUTIVE_RESENDS	-1028	Exceeded consecutive resends
MCAM_ERR_AUTO_ABORTED	-1029	Auto aborted
MCAM_ERR_BAD_VERSION	-1030	Bad version

MCAM_ERR_NO_MORE_ENTRY	-1031	No more entries
MCAM_ERR_NO_AVAILABLE_DATA	-1032	No available data
MCAM_ERR_NETWORK_ERROR	-1033	Network error
MCAM_ERR_RESYNC	-1034	Resynchronization needed
MCAM_ERR_CORRUPTED_DATA	-1035	Corrupted data
MCAM_ERR_GENICAM_XML_ERROR	-1036	GenICam XML error
MCAM_ERR_NO_DEVICE	-1037	No device
MCAM_ERR_NO_SYSTEM	-1038	No system
MCAM_ERR_NOT_OPEN_SYSTEM	-1039	System not open

2.3 Device Information Definitions

Device Information Definitions are parameters that provide the fundamental properties of the camera. This information is used to identify and manage the device and supports querying property values for a specific device. It includes various details such as the device model name, serial number, and network information (MAC address, IP address), and the information supported may vary depending on the interface.

Note: Some device information may only be available on specific interfaces (**GEV** or **U3V**).

Definition	Value	Description	Remarks
MCAM_DEVICEINFO_USER_ID	10000	User ID	
MCAM_DEVICEINFO_MODEL_NAME	10001	Model name	
MCAM_DEVICEINFO_SERIAL_NUMBER	10002	Serial number	
MCAM_DEVICEINFO_DEVICE_VERSION	10003	Device version	
MCAM_DEVICEINFO_MAC_ADDRESS	10004	MAC address	GEV Only
MCAM_DEVICEINFO_IP_ADDRESS	10005	IP address	GEV Only
MCAM_DEVICEINFO_CURRENT_SPEED	10006	Current speed	U3V Only

2.4 XML Version

The XML version is used when saving or loading device settings through the ST_ImportXML and ST_ExportXML functions.

Definition	Value	Description
XML_VERSION_1_0	0	GenICam XML 1.0 version (default support)
XML_VERSION_1_1	1	GenICam XML 1.1 version (includes enhanced features)

2.5 Parameter Visibility

Parameter Visibility determines the level of settings visible to the user in the device configuration. Various levels are provided, from beginner to expert, and certain settings can be hidden.

Definition	Value	Description
VISIBILITY_BEGINNER	0	Beginner level - displays only basic settings
VISIBILITY_EXPERT	1	Expert level - displays advanced settings
VISIBILITY_GURU	2	Guru level - displays all settings
VISIBILITY_INVISIBLE	3	Invisible

3. Key Structure Definitions

3.1 CVS_IMAGE Structure

Stores image data properties like width, height, and channels.

```
typedef struct _cvImage {  
    int32_t width;      // Image width (in pixels)  
    int32_t height;     // Image height (in pixels)  
    int32_t step;       // Bytes per line (Stride)  
    int32_t channels;    // Number of image channels (e.g., 3 = RGB)  
    void* pImage;       // Pointer to the image data  
} CVS_IMAGE;
```

3.2 CVS_BUFFER Structure

Contains information about a single frame or image buffer.

```
typedef struct _cvBuffer {  
    uint64_t blockID;   // Unique ID of the image block  
    uint64_t timestamp; // Timestamp of image acquisition (in microseconds)  
    uint32_t size;      // Image buffer size (in bytes)  
    CVS_IMAGE image;    // Image data structure  
} CVS_BUFFER;
```

3.3 CVS_EVENT Structure

The CVS_EVENT structure stores event information that occurs in the camera system. It includes the event ID, occurrence time, and related data.

```
typedef struct _cvsEvent {
    int id; // Event ID (e.g., device disconnection, exposure end, etc.)
    uint64_t timestamp; // Time of the event occurrence (in microseconds)
    void* data; // Additional data related to the event
    uint32_t dataLength; // Size of the additional data (in bytes)
} CVS_EVENT;
```

4. Color Conversion (enum ConvertColor)

The ConvertColor enumeration defines various options for image color conversion. These values are used to convert one color space to another during image processing.

```
enum ConvertColor {
    CVP_BGR2RGB = 4, // Convert BGR to RGB
    CVP_RGB2BGR = 4, // Convert RGB to BGR (same value used)
    CVP_BayerBG2RGB = 48, // Convert BayerBG pattern to RGB
    CVP_BayerGB2RGB = 49, // Convert BayerGB pattern to RGB
    CVP_BayerRG2RGB = 46, // Convert BayerRG pattern to RGB
    CVP_BayerGR2RGB = 47, // Convert BayerGR pattern to RGB
    CVP_YUV2RGB_UYVY = 107, // Convert YUV (UYVY) format to RGB
    CVP_YUV2BGR_UYVY = 108, // Convert YUV (UYVY) format to BGR
    CVP_YUV2RGB_YVYU = 117, // Convert YUV (YVYU) format to RGB
    CVP_YUV2BGR_YVYU = 118, // Convert YUV (YVYU) format to BGR
    CVP_YUV2RGB_YUYV = 115, // Convert YUV (YUYV) format to RGB
    CVP_YUV2BGR_YUYV = 116 // Convert YUV (YUYV) format to BGR
};
```

5. Functions

5.1 Callback Function Definitions

You can register callback functions to handle Grab and device events in the API. This section explains the definitions and parameters of the callback functions that are called when an event occurs.

5.1.1 Grab Event Callback

This callback function is automatically called when a Grab event occurs. Through the callback, you can check if a new image has been received or if an error has occurred.

```
typedef void (*GrabCallback)(
    int32_t eventID,
```



```
const CVS_BUFFER* pBuffer,
void* pUserDefine
);
```

- **eventID** (IN): The ID of the Grab event that occurred.
- **pBuffer** (IN): The buffer containing the grabbed image data.
- **pUserDefine** (IN): User-defined data.

5.1.2 Device Event Callback

This callback function is called when a device event occurs. Through the callback, you can detect changes in the device status.

```
typedef void(*EventCallback)(
    const CVS_EVENT* pEvent,
    void* pUserDefine
);
```

- **pEvent** (IN): A structure containing device event information.
- **pUserDefine** (IN): User-defined data.

5.2 System Management

To use the API, you must first initialize the system and then clean it up after use. This section provides functions for system initialization, de-initialization, and status checking.

5.2.1 ST_InitSystem

Initializes the system. This function must be called before starting to use the API. If called successfully, the system is ready to operate normally.

```
CVS_IMPORT_EXPORT CVS_ERROR WINAPI ST_InitSystem();
```

- Parameters: None
- Return Value:
 - MCAM_ERR_OK (Success)
 - Error code (Failure)
- Example Code:

```
CVS_ERROR result = ST_InitSystem();

if (result != MCAM_ERR_OK)
{
    printf("Failed to InitSystem : %d\n", result);
}
else
{
    printf("System initialized successfully.\n");
}
```

5.2.2 ST_FreeSystem

Frees resources allocated by the system. This function must be called after you have finished using the API and serves to clean up the initialized system (ST_InitSystem). Calling this function will disconnect all devices and free internal memory.

```
CVS_IMPORT_EXPORT CVS_ERROR WINAPI ST_FreeSystem();
```

- Parameters: None
- Return Value:
 - MCAM_ERR_OK (Success)
 - Error code (Failure)
- Example Code:

```
CVS_ERROR result = ST_FreeSystem();

if (result != MCAM_ERR_OK)
{
    printf("Failed to FreeSystem : %d\n", result);
}
else
{
    printf("System resources freed successfully.\n");
}
```

5.2.3 ST_IsInitSystem

Checks whether the system has been initialized. You can use this to confirm if ST_InitSystem() was called successfully.

```
CVS_IMPORT_EXPORT CVS_ERROR WINAPI ST_IsInitSystem(
    bool* pFlag
);
```

- Parameters:
 - pFlag (OUT): A variable to store the initialization status (true = initialized, false = not initialized).
- Return Value:
 - MCAM_ERR_OK (Success)
 - Error code (Failure)
- Example Code:

```
bool isInitialized = false;

CVS_ERROR result = ST_IsInitSystem(&isInitialized);

if (result != MCAM_ERR_OK)
{
    printf("Failed to ST_IsInitSystem : %d\n", result);
}
else
{
}
```

```
printf("System is %s\n", isInitialized ? "initialized" : "not initialized");
}
```

5.3 Device Discovery and Information Retrieval

You can search for camera devices connected to the system and query the ID and detailed information of each device.

5.3.1 ST_GetAvailableCameraNum

Returns the number of available cameras currently connected to the system.

```
CVS_IMPORT_EXPORT CVS_ERROR WINAPI ST_GetAvailableCameraNum(
    uint32_t* pNum
);
```

- Parameters:
 - pNum (OUT): A variable to store the number of available cameras.
- Return Value:
 - MCAM_ERR_OK (Success)
 - Error code (Failure)
- Example Code:

```
uint32_t cameraCount = 0;

CVS_ERROR result = ST_GetAvailableCameraNum(&cameraCount);

if (result != MCAM_ERR_OK)
{
    printf("Failed to GetAvailableCameraNum : %d\n", result);
}
else
{
    printf("Number of available cameras: %d\n", cameraCount);
}
```

5.3.2 ST_UpdateDevice

Refreshes the device list. This can be used to update the device list to the latest status if a camera has been newly connected or removed.

```
CVS_IMPORT_EXPORT CVS_ERROR WINAPI ST_UpdateDevice(
    uint32_t uTimeout = 500
);
```

- Parameters:
 - uTimeout (IN): Timeout for device discovery (default: 500ms, allowed range: 100~60000ms).
- Return Value:

- MCAM_ERR_OK (Success)
 - Error code (Failure)
- Example Code:

```
CVS_ERROR result = ST_UpdateDevice(1000);

if (result != MCAM_ERR_OK) {
    printf("Failed to UpdateDevice : %d\n", result);
}
else
{
    printf("Device list updated successfully.\n");
}
```

5.3.3 ST_GetEnumDeviceID

Queries the unique ID of the specified device. Returns the ID of the device based on its index (EnumNum).

```
CVS_IMPORT_EXPORT CVS_ERROR WINAPI ST_GetEnumDeviceID(
    uint32_t EnumNum,
    char* pDeviceID,
    uint32_t* pSize
);
```

- Parameters:
 - EnumNum (IN): The index of the device to query.
 - pDeviceID (OUT): A string buffer to store the device ID.
 - pSize (IN/OUT): Buffer size. Updated to the actual size after the function call.
- Return Value:
 - MCAM_ERR_OK (Success)
 - Error code (Failure)
- Example Code:

```
uint32_t size = 128;

char deviceID[128];

CVS_ERROR result = ST_GetEnumDeviceID(0, deviceID, &size);

if (result != MCAM_ERR_OK)
{
    printf("Failed to GetEnumDeviceID : %d\n", result);
}
else
{
    printf("Device ID: %s\n", deviceID);
}
```

5.3.4 ST_GetEnumDeviceInfo

Queries the information of the specified device. You can retrieve information such as the device's model name, serial number, and MAC address.

```
CVS_IMPORT_EXPORT CVS_ERROR WINAPI ST_GetEnumDeviceInfo(
    uint32_t EnumNum,
    int32_t DeviceInfoCmd,
    char* pDeviceInfo,
    uint32_t* pSize
);
```

- Parameters:
 - EnumNum (IN): The index of the device to query.
 - DeviceInfoCmd (IN): The device information command to request (e.g., MCAM_DEVICEINFO_MODEL_NAME).
 - pDeviceInfo (OUT): A string buffer to store the device information.
 - pSize (IN/OUT): Buffer size. Updated to the actual size after the function call.
- Return Value:
 - MCAM_ERR_OK (Success)
 - Error code (Failure)
- Example Code:

```
uint32_t size = 128;

char deviceInfo[128];

CVS_ERROR result = ST_GetEnumDeviceInfo(0, MCAM_DEVICEINFO_MODEL_NAME,
deviceInfo, &size);
if (result != MCAM_ERR_OK)
{
    printf("Failed to GetEnumDeviceInfo : %d\n", result);
}
else
{
    printf("Device Model: %s\n", deviceInfo);
}
```

5.4 Device Connection and Control

These functions provide the ability to connect or disconnect a camera device and check its current connection status.

5.4.1 ST_IsConnectable

Checks if the specified device is in a connectable state. It verifies if the device is connected to the network or can operate normally.

```
CVS_IMPORT_EXPORT CVS_ERROR WINAPI ST_IsConnectable(
    uint32_t EnumNum,
    bool* pFlag
```

```
);
```

- Parameters:
 - EnumNum (IN): The index of the device to query.
 - pFlag (OUT): A pointer to a boolean variable. true if the device is connectable, false otherwise.
- Return Value:
 - MCAM_ERR_OK (Success)
 - Error code (Failure)
- Example Code:

```
bool isConnectable = false;

CVS_ERROR result = ST_IsConnectable(0, &isConnectable);

if (result != MCAM_ERR_OK)
{
    printf("Failed to IsConnectable : %d\n", result);
}
else
{
    printf("Device is %s\n", isConnectable ? "connectable" : "not
connectable");
}
```

5.4.2 ST_OpenDevice

Opens the specified device and prepares it for use. If called successfully, a device handle (hDevice) is returned, which is then used to control the device.

```
CVS_IMPORT_EXPORT CVS_ERROR WINAPI ST_OpenDevice(
    uint32_t EnumNum,
    int32_t* hDevice,
    bool isDetailedLog = false
);
```

- Parameters:
 - EnumNum (IN): The index of the device to connect.
 - hDevice (OUT): The handle of the connected device.
 - isDetailedLog (IN, optional): Whether to enable detailed logs for debugging (default is false).
- Return Value:
 - MCAM_ERR_OK (Success)
 - Error code (Failure)
- Example Code:

```
int32_t hDevice= 0;

CVS_ERROR result = ST_OpenDevice(0, &hDevice, false);
```

```

if (result != MCAM_ERR_OK)
{
    printf("Failed to OpenDevice : %d\n", result);
}
else
{
    printf("Device opened successfully. Handle: %d\n", hDevice);
}

```

5.4.3 ST_IsOpenDevice

Checks if the specified device is currently open. You can use this to verify if the device was opened successfully after calling ST_OpenDevice().

```

CVS_IMPORT_EXPORT CVS_ERROR WINAPI ST_IsOpenDevice(
    int32_t hDevice,
    bool* pFlag
);

```

- Parameters:
 - hDevice (IN): The handle of the device to check.
 - pFlag (OUT): A pointer to a boolean variable. true if the device is open, false otherwise.
- Return Value:
 - MCAM_ERR_OK (Success)
 - Error code (Failure)
- Example Code:

```

bool isOpen = false;

CVS_ERROR result = ST_IsOpenDevice(hDevice, &isOpen);

if (result != MCAM_ERR_OK)
{
    printf("Failed to IsOpenDevice : %d\n", result);
}
else
{
    printf("Device is %s\n", isOpen ? "open" : "not open");
}

```

5.4.4 ST_CloseDevice

Closes the specified device and releases it from the system. This must be called when the device is no longer in use.

```

CVS_IMPORT_EXPORT CVS_ERROR WINAPI ST_CloseDevice(
    int32_t hDevice
);

```

- Parameters:
 - hDevice (IN): The handle of the device to close.
- Return Value:
 - MCAM_ERR_OK (Success)
 - Error code (Failure)
- Example Code:

```
CVS_ERROR result = ST_CloseDevice(hDevice);

if (result != MCAM_ERR_OK)
{
    printf("Failed to CloseDevice : %d\n", result);
}
else
{
    printf("Device closed successfully.\n");
}
```

5.4.5 ST_GetInterface

Queries the interface information of the specified device. The interface information indicates how the device is connected (e.g., **GEV**, **U3V**).

```
CVS_IMPORT_EXPORT CVS_ERROR WINAPI ST_GetInterface(
    int32_t hDevice,
    char* pInterface,
    uint32_t* pSize
);
```

- Parameters:
 - hDevice (IN): The handle of the device to query.
 - pInterface (OUT): A string buffer to store the interface information.
 - pSize (IN/OUT): The size of the buffer. It will be updated to the actual size after the function call.
- Return Value:
 - MCAM_ERR_OK (Success)
 - Error code (Failure)
- Example Code:

```
uint32_t size = 128;

char interfaceInfo[128];

CVS_ERROR result = ST_GetInterface(hDevice, interfaceInfo, &size);

if (result != MCAM_ERR_OK)
{
    printf("Failed to GetInterface : %d\n", result);
}
else
```



```
{
    printf("Device Interface: %s\n", interfaceInfo);
}
```

5.5 Image Acquisition and Configuration

These functions manage settings related to camera image acquisition (Grab). They provide functions to start and stop image acquisition, set Grab timeout, and force an abortion.

5.5.1 ST_AcqStart

Starts image acquisition on the specified device. The device begins continuously acquiring images.

```
CVS_IMPORT_EXPORT CVS_ERROR WINAPI ST_AcqStart(
    int32_t hDevice
);
```

- Parameters:
 - hDevice (IN): The handle of the device to start image acquisition on.
- Return Value:
 - MCAM_ERR_OK (Success)
 - Error code (Failure)
- Example Code:

```
CVS_ERROR result = ST_AcqStart(hDevice);

if (result != MCAM_ERR_OK)
{
    printf("Failed to AcqStart : %d\n", result);
}
else
{
    printf("Acquisition started successfully.\n");
}
```

5.5.2 ST_AcqStop

Stops image acquisition on the specified device. It halts the continuous image acquisition started with ST_AcqStart().

```
CVS_IMPORT_EXPORT CVS_ERROR WINAPI ST_AcqStop(
    int32_t hDevice
);
```

- Parameters:
 - hDevice (IN): The handle of the device to stop image acquisition on.
- Return Value:
 - MCAM_ERR_OK (Success)
 - Error code (Failure)

- Example Code:

```
CVS_ERROR result = ST_AcqStop(hDevice);

if (result != MCAM_ERR_OK)
{
    printf("Failed to AcqStop : %d\n", result);
}
else
{
    printf("Acquisition stopped successfully.\n");
}
```

5.5.3 ST_DoAbortGrab

Forcefully aborts the Grab process currently in progress on the specified device. If image acquisition is ongoing, it will be stopped immediately.

```
CVS_IMPORT_EXPORT CVS_ERROR WINAPI ST_DoAbortGrab(
    int32_t hDevice
);
```

- Parameters:
 - hDevice (IN): The handle of the device whose Grab process is to be aborted.
- Return Value:
 - MCAM_ERR_OK (Success)
 - Error code (Failure)
- Example Code:

```
CVS_ERROR result = ST_DoAbortGrab(hDevice);

if (result != MCAM_ERR_OK)
{
    printf("Failed to DoAbortGrab : %d\n", result);
}
else
{
    printf("Grab process aborted successfully.\n");
}
```

5.5.4 ST_SetGrabTimeout

Sets the Grab timeout for the specified device. You can set it to return a timeout error if a Grab does not complete within a certain time.

```
CVS_IMPORT_EXPORT CVS_ERROR WINAPI ST_SetGrabTimeout(
    int32_t hDevice,
    uint32_t Timeout
);
```

- Parameters:

- hDevice (IN): The handle of the device to set the timeout for.
 - Timeout (IN): The Grab timeout value (in milliseconds).
- Return Value:
 - MCAM_ERR_OK (Success)
 - Error code (Failure)
- Example Code:

```
CVS_ERROR result = ST_SetGrabTimeout(hDevice, 3000);

if (result != MCAM_ERR_OK)
{
    printf("Failed to SetGrabTimeout : %d\n", result);
}
else
{
    printf("Grab timeout set to 3000 ms.\n");
}
```

5.5.5 ST_GetGrabTimeout

Queries the Grab timeout value of the specified device.

```
CVS_IMPORT_EXPORT CVS_ERROR WINAPI ST_GetGrabTimeout(
    int32_t hDevice,
    uint32_t* pTimeout
);
```

- Parameters:
 - hDevice (IN): The handle of the device to query the timeout for.
 - pTimeout (OUT): The currently set Grab timeout value (in milliseconds).
- Return Value:
 - MCAM_ERR_OK (Success)
 - Error code (Failure)
- Example Code:

```
uint32_t timeout = 0;

CVS_ERROR result = ST_GetGrabTimeout(hDevice, &timeout);

if (result != MCAM_ERR_OK)
{
    printf("Failed to GetGrabTimeout : %d\n", result);
}
else
{
    printf("Current grab timeout: %d ms\n", timeout);
}
```

5.5.6 ST_SetBufferCount

Sets the number of image buffers to be used by the device. Increasing the buffer count reduces the possibility of dropped frames but increases memory usage.

```
CVS_IMPORT_EXPORT CVS_ERROR WINAPI ST_SetBufferCount(
    int32_t hDevice,
    uint32_t BufferCount
);
```

- Parameters:
 - hDevice (IN): Device handle.
 - BufferCount (IN): The number of buffers to set.
- Return Value:
 - MCAM_ERR_OK (Success)
 - Error code (Failure)
- Example Code:

```
uint32_t bufferCount = 10;

CVS_ERROR result = ST_SetBufferCount(hDevice, bufferCount);

if (result != MCAM_ERR_OK)
{
    printf("Failed to SetBufferCount : %d\n", result);
}
else
{
    printf("Buffer count set to: %d\n", bufferCount);
}
```

5.5.7 ST_GetBufferCount

Queries the current number of buffers set on the device.

```
CVS_IMPORT_EXPORT CVS_ERROR WINAPI ST_GetBufferCount(
    int32_t hDevice,
    uint32_t* pBufferCount
);
```

- Parameters:
 - hDevice (IN): Device handle.
 - pBufferCount (OUT): A variable to store the buffer count.
- Return Value:
 - MCAM_ERR_OK (Success)
 - Error code (Failure)
- Example Code:

```
uint32_t bufferCount = 0;
```

```

CVS_ERROR result = ST_GetBufferCount(hDevice, &bufferCount);

if (result != MCAM_ERR_OK)
{
    printf("Failed to GetBufferCount : %d\n", result);
}
else
{
    printf("Current buffer count: %d\n", bufferCount);
}

```

5.6 Image Processing and Status Check

These functions are for checking the status and querying information related to an acquired image.

5.6.1 ST_GrabImage

Grabs images continuously from the device. The ST_AcqStart function must be called first to begin image acquisition, and ST_AcqStop stops it. The ST_GrabImage function stores the image in a buffer when ST_AcqStart is active, and the acquired image is saved in pBuffer.

```

CVS_IMPORT_EXPORT CVS_ERROR WINAPI ST_GrabImage(
    int32_t hDevice,
    CVS_BUFFER* pBuffer
);

```

- Parameters:
 - hDevice (IN): Device handle.
 - pBuffer (OUT): A pointer to the CVS_BUFFER structure where the grabbed image data will be stored.
- Return Value:
 - MCAM_ERR_OK (Success)
 - Error code (Failure)
- Example Code:

```

CVS_BUFFER buffer;

CVS_ERROR result = ST_GrabImage(hDevice, &buffer);

if (result != MCAM_ERR_OK)
{
    printf("Failed to GrabImage : %d\n", result);
}
else
{
    printf("Image grabbed successfully.\n");
}

```

5.6.2 ST_SingleGrabImage

Grabs a single frame image. It can be executed independently without calling ST_AcqStart and is used when continuous image acquisition is not needed. The acquired image is saved in pBuffer. Generally, ST_GrabImage is for continuous shooting, while ST_SingleGrabImage is suitable for single-shot image capture.

```
CVS_IMPORT_EXPORT CVS_ERROR WINAPI ST_SingleGrabImage(
    int32_t hDevice,
    CVS_BUFFER* pBuffer
);
```

- Parameters:
 - hDevice (IN): Device handle.
 - pBuffer (OUT): A pointer to the CVS_BUFFER structure where the acquired image data will be stored.
- Return Value:
 - MCAM_ERR_OK (Success)
 - Error code (Failure)
- Example Code:

```
CVS_BUFFER buffer;

CVS_ERROR result = ST_SingleGrabImage(hDevice, &buffer);

if (result != MCAM_ERR_OK)
{
    printf("Failed to SingleGrabImage : %d\n", result);
}
else
{
    printf("Single image grabbed successfully.\n");
}
```

5.6.3 ST_GetImageAvailable

Checks if an image is available for acquisition on the device. It can be used to check if image acquisition has been completed.

```
CVS_IMPORT_EXPORT CVS_ERROR WINAPI ST_GetImageAvailable(
    int32_t hDevice,
    uint32_t* pFlag
);
```

- Parameters:
 - hDevice (IN): Device handle.
 - pFlag (OUT): A variable to store the status (1 = image available, 0 = no image available).
- Return Value:
 - MCAM_ERR_OK (Success)
 - Error code (Failure)

- Example Code:

```
uint32_t isAvailable = 0;

CVS_ERROR result = ST_GetImageAvailable(hDevice, &isAvailable);

if (result != MCAM_ERR_OK)
{
    printf("Failed to GetImageAvailable : %d\n", result);
}
else
{
    printf("Image available: %d\n", isAvailable);
}
```

5.6.4 ST_GetGrabCount

Queries the number of images grabbed so far and the number of errors that have occurred on the device.

```
CVS_IMPORT_EXPORT CVS_ERROR WINAPI ST_GetGrabCount(
    int32_t hDevice,
    uint64_t* Count,
    uint64_t* ErrorCount
);
```

- Parameters:
 - hDevice (IN): Device handle.
 - Count (OUT): The number of images grabbed.
 - ErrorCount (OUT): The number of errors that occurred during the acquisition process.
- Return Value:
 - MCAM_ERR_OK (Success)
 - Error code (Failure)
- Example Code:

```
uint64_t grabCount = 0, errorCount = 0;

CVS_ERROR result = ST_GetGrabCount(hDevice, &grabCount, &errorCount);

if (result != MCAM_ERR_OK)
{
    printf("Failed to GetGrabCount : %d\n", result);
}
else
{
    printf("Grab count: %llu, Error count: %llu\n", grabCount, errorCount);
}
```

5.6.5 ST_GetFrameRate

Queries the currently set frame rate on the device. You can check how many frames per second the camera is capturing.

```
CVS_IMPORT_EXPORT CVS_ERROR WINAPI ST_GetFrameRate(  
    int32_t hDevice,  
    double* FrameRate  
);
```

- Parameters:
 - hDevice (IN): Device handle.
 - FrameRate (OUT): The current frame rate (fps).
- Return Value:
 - MCAM_ERR_OK (Success)
 - Error code (Failure)
- Example Code:

```
double frameRate = 0.0;  
  
CVS_ERROR result = ST_GetFrameRate(hDevice, &frameRate);  
  
if (result != MCAM_ERR_OK)  
{  
    printf("Failed to GetFrameRate : %d\n", result);  
}  
else  
{  
    printf("Current frame rate: %.2f fps\n", frameRate);  
}
```

5.6.6 ST_GetBandwidth

Queries the current network or interface bandwidth in use by the device.

```
CVS_IMPORT_EXPORT CVS_ERROR WINAPI ST_GetBandwidth(  
    int32_t hDevice,  
    double* Bandwidth  
);
```

- Parameters:
 - hDevice (IN): Device handle.
 - Bandwidth (OUT): The current bandwidth usage (Mbps).
- Return Value:
 - MCAM_ERR_OK (Success)
 - Error code (Failure)
- Example Code:

```
double bandwidth = 0.0;  
  
CVS_ERROR result = ST_GetBandwidth(hDevice, &bandwidth);
```



```

if (result != MCAM_ERR_OK)
{
    printf("Failed to GetBandwidth : %d\n", result);
}
else
{
    printf("Current bandwidth usage: %.2f Mbps\n", bandwidth);
}

```

5.6.7 ST_GetLastError

Queries the last error code that occurred on the device.

```

CVS_IMPORT_EXPORT CVS_ERROR WINAPI ST_GetLastError(
    int32_t hDevice
);

```

- Parameters:
 - hDevice (IN): Device handle.
- Return Value:
 - The last error code (MCAM_ERR_...).
- Example Code:

```

CVS_ERROR lastError = ST_GetLastError(hDevice);

printf("Last device error code: %d\n", lastError);

```

5.6.8 ST_GetLastErrorDescription

Returns a string description of the last error that occurred on the device. It can be used with ST_GetLastError to get detailed error information.

```

CVS_IMPORT_EXPORT const char* WINAPI ST_GetLastErrorDescription(
    int32_t hDevice
);

```

- Parameters:
 - hDevice (IN): Device handle.
- Return Value:
 - A string describing the error.
- Example Code:

```

const char* errorDescription = ST_GetLastErrorDescription(hDevice);

printf("Last device error: %s\n", errorDescription);

```

5.7 Buffer Management

These functions provide the ability to initialize and free memory buffers that store image data.

5.7.1 ST_InitBuffer

The ST_InitBuffer function initializes a CVS_BUFFER structure. It creates a buffer to store image data, and the number of channels can be specified. It must be called before calling ST_GrabImage or ST_SingleGrabImage. If the default value (0) is set, the number of channels is automatically determined based on the connected camera's settings.

```
CVS_IMPORT_EXPORT CVS_ERROR WINAPI ST_InitBuffer(
    int32_t hDevice,
    CVS_BUFFER* pBuffer,
    int32_t channels = 0
);
```

- Parameters:
 - hDevice (IN): Device handle.
 - pBuffer (OUT): A pointer to the CVS_BUFFER structure to be initialized.
 - channels (IN): The number of channels for the buffer (default 0, which uses the camera's setting).
- Return Value:
 - MCAM_ERR_OK (Success)
 - Error code (Failure)
- Example Code:

```
CVS_BUFFER buffer;

CVS_ERROR result = ST_InitBuffer(hDevice, &buffer, 0);

if (result != MCAM_ERR_OK)
{
    printf("Failed to InitBuffer : %d\n", result);
}
else
{
    printf("Buffer initialized successfully.\n");
}
```

5.7.2 ST_FreeBuffer

The ST_FreeBuffer function frees the memory of a CVS_BUFFER created with ST_InitBuffer. It frees a buffer that is no longer in use to prevent memory leaks. After calling ST_FreeBuffer, you must call ST_InitBuffer again to reuse the buffer.

```
CVS_IMPORT_EXPORT CVS_ERROR WINAPI ST_FreeBuffer(
    CVS_BUFFER* pBuffer
);
```

- Parameters:

- pBuffer (IN): A pointer to the CVS_BUFFER structure to be freed.
- Return Value:
 - MCAM_ERR_OK (Success)
 - Error code (Failure)
- Example Code:

```
CVS_ERROR result = ST_FreeBuffer(&buffer);

if (result != MCAM_ERR_OK)
{
    printf("Failed to FreeBuffer : %d\n", result);
}
else
{
    printf("Buffer freed successfully.\n");
}
```

5.8 Image Color Conversion

This section provides functionality to convert the color space of an image.

5.8.1 ST_CvtColor

The ST_CvtColor function provides the ability to convert an image from one color space to another. It converts the input image according to the conversion code (code) and stores it in the output buffer (pDest). This function is useful for post-processing images acquired with ST_GrabImage or ST_SingleGrabImage.

```
CVS_IMPORT_EXPORT CVS_ERROR WINAPI ST_CvtColor(
    CVS_BUFFER pSrc,
    CVS_BUFFER* pDest,
    int32_t code
);
```

- Parameters:
 - pSrc (IN): The CVS_BUFFER structure containing the original image to be converted.
 - pDest (OUT): A pointer to the CVS_BUFFER structure where the converted image will be stored.
 - code (IN): The color space conversion code (e.g., CVP_BayerRG2RGB).
- Return Value:
 - MCAM_ERR_OK (Success)
 - Error code (Failure)
- Example Code:

```
CVS_BUFFER destBuffer;

// Initialize destBuffer
ST_InitBuffer(hDevice, &destBuffer, 3);
```

```

// Convert from BayerRG8 to RGB
int32_t code = CVP_BayerRG2RGB;

// srcBuffer is an image buffer obtained earlier via Grab
CVS_ERROR result = ST_CvtColor(srcBuffer, &destBuffer, code);

if (result != MCAM_ERR_OK)
{
    printf("Failed to convert color : %d\n", result);
}
else
{
    printf("Color conversion successful.\n");
}

// Release destBuffer
ST_FreeBuffer(&destBuffer);

```

5.9 Log and Configuration Management

These functions provide the ability to enable or disable detailed logs for debugging.

5.9.1 ST_SetDetailedLog

The ST_SetDetailedLog function provides the ability to enable or disable the detailed logging feature for a specific device (hDevice). Enabling detailed logs allows for more detailed recording of events and errors that occur during the device's operation. This feature is useful for troubleshooting and debugging.

```

CVS_IMPORT_EXPORT CVS_ERROR WINAPI ST_SetDetailedLog(
    int32_t hDevice,
    bool Flag
);

```

- Log file storage location:
 - Windows: C:\ProgramData\CREVIS\Logs\Device-specific
 - Linux: ~/CREVIS/Device-specific
- Log management policy:
 - If there are more than 500 files in each device's folder, or
 - If the folder size exceeds 5MB,
 - The oldest files are automatically deleted to free up storage space.
- Parameters:
 - hDevice (IN): The handle of the device to set the log for.
 - Flag (IN): Whether to enable detailed logs (true = enabled, false = disabled).
- Return Value:
 - MCAM_ERR_OK (Success)
 - Error code (Failure)
- Example Code:

```

CVS_ERROR result = ST_SetDetailedLog(hDevice, true);

if (result != MCAM_ERR_OK)
{
    printf("Failed to enable detailed log : %d\n", result);
}
else
{
    printf("Detailed log enabled successfully. Logs saved in:\n");
}

```

5.9.2 ST_GetDetailedLog

The ST_GetDetailedLog function provides the ability to check if the detailed logging feature is currently enabled for a specific device (hDevice). It returns the current detailed log status.

```

CVS_IMPORT_EXPORT CVS_ERROR WINAPI ST_GetDetailedLog(
    int32_t hDevice,
    bool* pFlag
);

```

- Parameters:
 - hDevice (IN): The handle of the device to check the log status for.
 - pFlag (OUT): A variable to store the detailed log status (true = enabled, false = disabled).
- Return Value:
 - MCAM_ERR_OK (Success)
 - Error code (Failure)
- Example Code:

5.10 Register Control

These functions provide the ability to read and write values for various device properties (Nodes or Registers). This allows for dynamic control of the camera's features, such as exposure time, gain, and pixel format.

5.10.1 ST_SetIntReg

Sets the integer (Integer) register value of the specified device.

```

CVS_IMPORT_EXPORT CVS_ERROR WINAPI ST_SetIntReg(
    int32_t hDevice,
    const char* NodeName,
    int64_t val
);

```

- Parameters:
 - hDevice (IN): The device handle.

- **NodeName (IN):** The node name of the register to set.
 - **val (IN):** The integer value to set.
- **Return Value:**
 - **MCAM_ERR_OK (Success)**
 - **Error code (Failure)**

- **Example Code:**

```
CVS_ERROR result = ST_SetIntReg(hDevice, "Width", 1000);

if (result != MCAM_ERR_OK)
{
    printf("Failed to set integer register: %d\n", result);
}
else
{
    printf("Successfully set integer register.\n");
}
```

5.10.2 ST_GetIntReg

Queries the integer (Integer) register value of the specified device.

```
CVS_IMPORT_EXPORT CVS_ERROR WINAPI ST_GetIntReg(
    int32_t hDevice,
    const char* NodeName,
    int64_t* pVal
);
```

- **Parameters:**
 - **hDevice (IN):** The device handle.
 - **NodeName (IN):** The node name of the register to query.
 - **pVal (OUT):** A pointer to the variable that will store the queried integer value.
- **Return Value:**
 - **MCAM_ERR_OK (Success)**
 - **Error code (Failure)**
- **Example Code:**

```
int64_t value = 0;

CVS_ERROR result = ST_GetIntReg(hDevice, "Width", &value);

if (result != MCAM_ERR_OK)
{
    printf("Failed to get integer register: %d\n", result);
}
else
{
    printf("Width: %lld\n", value);
}
```

5.10.3 ST_SetFloatReg

Sets the floating-point (Float) register value of the specified device.

```
CVS_IMPORT_EXPORT CVS_ERROR WINAPI ST_SetFloatReg(
    int32_t hDevice,
    const char* NodeName,
    double fVal
);
```

- Parameters:
 - hDevice (IN): The device handle.
 - NodeName (IN): The node name of the register to set.
 - fVal (IN): The floating-point value to set.
- Return Value:
 - MCAM_ERR_OK (Success)
 - Error code (Failure)
- Example Code:

```
CVS_ERROR result = ST_SetFloatReg(hDevice, "ExposureTime", 1000.0);

if (result != MCAM_ERR_OK)
{
    printf("Failed to set float register: %d\n", result);
}
else
{
    printf("Successfully set float register.\n");
}
```

5.10.4 ST_GetFloatReg

Queries the floating-point (Float) register value of the specified device.

```
CVS_IMPORT_EXPORT CVS_ERROR WINAPI ST_GetFloatReg(
    int32_t hDevice,
    const char* NodeName,
    double* pFval
);
```

- Parameters:
 - hDevice (IN): The device handle.
 - NodeName (IN): The node name of the register to query.
 - pFval (OUT): A pointer to the variable that will store the queried floating-point value.
- Return Value:
 - MCAM_ERR_OK (Success)
 - Error code (Failure)
- Example Code:

```
double value = 0.0;
```

```

CVS_ERROR result = ST_GetFloatReg(hDevice, "ExposureTime", &value);

if (result != MCAM_ERR_OK)
{
    printf("Failed to get float register: %d\n", result);
}
else
{
    printf("Exposure Time: %.2f\n", value);
}

```

5.10.5 ST_SetEnumReg

Sets the enumeration (Enum) register value of the specified device.

```

CVS_IMPORT_EXPORT CVS_ERROR WINAPI ST_SetEnumReg(
    int32_t hDevice,
    const char* NodeName,
    char* val
);

```

- Parameters:
 - hDevice (IN): The device handle.
 - NodeName (IN): The node name of the register to set.
 - val (IN): The enumeration value to set.
- Return Value:
 - MCAM_ERR_OK (Success)
 - Error code (Failure)
- Example Code:

```

CVS_ERROR result = ST_SetEnumReg(hDevice, "TriggerMode", "On");

if (result != MCAM_ERR_OK)
{
    printf("Failed to set enum register: %d\n", result);
}
else
{
    printf("Successfully set enum register.\n");
}

```

5.10.6 ST_GetEnumReg

Queries the enumeration (Enum) register value of the specified device.

```

CVS_IMPORT_EXPORT CVS_ERROR WINAPI ST_GetEnumReg(
    int32_t hDevice,
    const char* NodeName,
    char* pInfo,
    uint32_t* pSize
);

```



```
);
```

- Parameters:
 - hDevice (IN): The device handle.
 - NodeName (IN): The node name of the register to query.
 - pInfo (OUT): The string buffer to store the queried enumeration value.
 - pSize (IN/OUT): The size of the buffer. It will be updated to the actual size after the function call.
- Return Value:
 - MCAM_ERR_OK (Success)
 - Error code (Failure)
- Example Code:

```
char triggerMode[128];

uint32_t size = sizeof(triggerMode);

CVS_ERROR result = ST_GetEnumReg(hDevice, "TriggerMode", triggerMode, &size);

if (result != MCAM_ERR_OK)
{
    printf("Failed to get enum register: %d\n", result);
}
else
{
    printf("Trigger Mode: %s\n", triggerMode);
}
```

5.10.7 ST_SetStrReg

Sets the string (String) register value of the specified device.

```
CVS_IMPORT_EXPORT CVS_ERROR WINAPI ST_SetStrReg(
    int32_t hDevice,
    const char* NodeName,
    char* val
);
```

- Parameters:
 - hDevice (IN): The device handle.
 - NodeName (IN): The node name of the register to set.
 - val (IN): The string value to set.
- Return Value:
 - MCAM_ERR_OK (Success)
 - Error code (Failure)
- Example Code:

```
CVS_ERROR result = ST_SetStrReg(hDevice, "DeviceUserID", "Camera_01");

if (result != MCAM_ERR_OK)
```

```

{
    printf("Failed to set string register: %d\n", result);
}
else
{
    printf("Successfully set string register.\n");
}

```

5.10.8 ST_GetStrReg

Queries the string (String) register value of the specified device.

```

CVS_IMPORT_EXPORT CVS_ERROR WINAPI ST_GetStrReg(
    int32_t hDevice,
    const char* NodeName,
    char* pInfo,
    uint32_t* pSize
);

```

- Parameters:
 - hDevice (IN): The device handle.
 - NodeName (IN): The node name of the register to query.
 - pInfo (OUT): The buffer to store the queried string value.
 - pSize (IN/OUT): The size of the buffer. It will be updated to the actual size after the function call.
- Return Value:
 - MCAM_ERR_OK (Success)
 - Error code (Failure)
- Example Code:

```

char deviceID[128];

uint32_t size = sizeof(deviceID);

CVS_ERROR result = ST_GetStrReg(hDevice, "DeviceUserID", deviceID, &size);

if (result != MCAM_ERR_OK)
{
    printf("Failed to get string register: %d\n", result);
}
else
{
    printf("Device User ID: %s\n", deviceID);
}

```

5.10.9 ST_SetCmdReg

Executes the command (Command) register of the specified device. For example, it can be used to execute commands like a software trigger.

```
CVS_IMPORT_EXPORT CVS_ERROR WINAPI ST_SetCmdReg(
    int32_t hDevice,
    const char* NodeName
);
```

- Parameters:
 - hDevice (IN): The device handle.
 - NodeName (IN): The node name of the command register to execute.
- Return Value:
 - MCAM_ERR_OK (Success)
 - Error code (Failure)
- Example Code:

```
CVS_ERROR result = ST_SetCmdReg(hDevice, "TriggerSoftware");

if (result != MCAM_ERR_OK)
{
    printf("Failed to execute command register: %d\n", result);
}
else
{
    printf("Successfully executed command register.\n");
}
```

5.11 Register Information Query

These functions provide the ability to query the value range and status information of registers.

5.11.1 ST_GetIntRegRange

Queries the value range of an integer (Integer) register for the specified device. It returns the minimum, maximum, and increment values.

```
CVS_IMPORT_EXPORT CVS_ERROR WINAPI ST_GetIntRegRange(
    int32_t hDevice,
    const char* NodeName,
    int64_t* pMin,
    int64_t* pMax,
    int64_t* pInc
);
```

- Parameters:
 - hDevice (IN): The device handle.
 - NodeName (IN): The node name of the register to query.
 - pMin (OUT): A variable to store the minimum value of the register.
 - pMax (OUT): A variable to store the maximum value of the register.
 - pInc (OUT): A variable to store the increment value of the register.
- Return Value:

- MCAM_ERR_OK (Success)
 - Error code (Failure)
- Example Code:

```
int64_t min, max, inc;

CVS_ERROR result = ST_GetIntRegRange(hDevice, "ExposureTime", &min, &max,
&inc);

if (result != MCAM_ERR_OK)
{
    printf("Failed to get integer register range: %d\n", result);
}
else
{
    printf("ExposureTime Range: Min = %lld, Max = %lld, Increment = %lld\n", min,
max, inc);
}
```

5.11.2 ST_GetFloatRegRange

Queries the value range of a floating-point (Float) register for the specified device. It returns the minimum and maximum values.

```
CVS_IMPORT_EXPORT CVS_ERROR WINAPI ST_GetFloatRegRange(
    int32_t hDevice,
    const char* NodeName,
    double* pMin,
    double* pMax
);
```

- Parameters:
 - hDevice (IN): The device handle.
 - NodeName (IN): The node name of the register to query.
 - pMin (OUT): A variable to store the minimum value of the register.
 - pMax (OUT): A variable to store the maximum value of the register.
- Return Value:
 - MCAM_ERR_OK (Success)
 - Error code (Failure)
- Example Code:

```
double min, max;

CVS_ERROR result = ST_GetFloatRegRange(hDevice, "Gain", &min, &max);

if (result != MCAM_ERR_OK)
{
    printf("Failed to get float register range: %d\n", result);
}
else
{
    printf("Gain Range: Min = %.2f, Max = %.2f\n", min, max);
}
```

```
}
```

5.11.3 ST_GetEnumEntrySize

Queries the number of entries in an enumeration (Enum) register for the specified device.

```
CVS_IMPORT_EXPORT CVS_ERROR WINAPI ST_GetEnumEntrySize(  
    int32_t hDevice,  
    const char* NodeName,  
    int32_t* pVal  
);
```

- Parameters:
 - hDevice (IN): The device handle.
 - NodeName (IN): The node name of the register to query.
 - pVal (OUT): A variable to store the number of enumeration entries.
- Return Value:
 - MCAM_ERR_OK (Success)
 - Error code (Failure)
- Example Code:

```
int32_t enumSize;  
  
CVS_ERROR result = ST_GetEnumEntrySize(hDevice, "PixelFormat", &enumSize);  
  
if (result != MCAM_ERR_OK)  
{  
    printf("Failed to get enum entry size: %d\n", result);  
}  
else  
{  
    printf("PixelFormat Enum Size: %d\n", enumSize);  
}
```

5.11.4 ST_GetEnumEntryIntValue

Queries the integer (Integer) value of a specific entry at a given index within an enumeration (Enum) register.

```
CVS_IMPORT_EXPORT CVS_ERROR WINAPI ST_GetEnumEntryIntValue(  
    int32_t hDevice,  
    const char* NodeName,  
    int32_t EntryIdx,  
    int32_t* pVal  
);
```

- Parameters:
 - hDevice (IN): The device handle.
 - NodeName (IN): The node name of the register to query.

- EntryIdx (IN): The index of the entry to query.
 - pVal (OUT): A variable to store the integer value of the entry.
- Return Value:
 - MCAM_ERR_OK (Success)
 - Error code (Failure)
- Example Code:

```
int32_t entryValue;

CVS_ERROR result = ST_GetEnumEntryIntValue(hDevice, "PixelFormat", 0,
&entryValue);

if (result != MCAM_ERR_OK)
{
    printf("Failed to get enum entry int value: %d\n", result);
}
else
{
    printf("PixelFormat[0] Enum Value: %d\n", entryValue);
}
```

5.11.5 ST_GetEnumEntryValue

Queries the string (String) value of a specific entry at a given index within an enumeration (Enum) register.

```
CVS_IMPORT_EXPORT CVS_ERROR WINAPI ST_GetEnumEntryValue(
    int32_t hDevice,
    const char* NodeName,
    int32_t EntryIdx,
    char* pInfo,
    uint32_t* pSize
);
```

- Parameters:
 - hDevice (IN): The device handle.
 - NodeName (IN): The node name of the register to query.
 - EntryIdx (IN): The index of the entry to query.
 - pInfo (OUT): The string buffer to store the queried entry value.
 - pSize (IN/OUT): The size of the buffer. It will be updated to the actual size after the function call.
- Return Value:
 - MCAM_ERR_OK (Success)
 - Error code (Failure)
- Example Code:

```
char enumEntry[128];

uint32_t size = sizeof(enumEntry);

CVS_ERROR result = ST_GetEnumEntryValue(hDevice, "PixelFormat", 0, enumEntry,
```

```

&size);

if (result != MCAM_ERR_OK)
{
    printf("Failed to get enum entry value: %d\n", result);
}
else
{
    printf("PixelFormat[0] Enum Value: %s\n", enumEntry);
}

```

5.12 Parameter Read and Write

These functions provide the ability to read and write data from or to a device's specific memory address.

5.12.1 ST_ReadParam

Reads parameter values of a specific address and length from the specified device.

```

CVS_IMPORT_EXPORT CVS_ERROR WINAPI ST_ReadParam(
    int32_t hDevice,
    unsigned char* pBuffer,
    int64_t Address,
    int64_t Length
);

```

- Parameters:
 - hDevice (IN): The device handle.
 - pBuffer (OUT): The buffer to store the read parameter data.
 - Address (IN): The starting address of the parameter to read.
 - Length (IN): The length of the data to read (in bytes).
- Return Value:
 - MCAM_ERR_OK (Success)
 - Error code (Failure)
- Example Code:

```

int64_t address = 0x1000; // Example address
int64_t length = 16;      // Read 16 bytes
unsigned char buffer[16];

CVS_ERROR result = ST_ReadParam(hDevice, buffer, address, length);

if (result != MCAM_ERR_OK)
{
    printf("Failed to read parameter: %d\n", result);
}
else
{

```

```

printf("Successfully read parameter data: ");
for (int i = 0; i < length; i++)
{
    printf("%02X ", buffer[i]);
}
printf("\n");
}

```

5.12.2 ST_WriteParam

Writes parameter values of a specific address and length to the specified device.

```

CVS_IMPORT_EXPORT CVS_ERROR WINAPI ST_WriteParam(
    int32_t hDevice,
    const unsigned char* pBuffer,
    int64_t Address,
    int64_t Length
);

```

- Parameters:
 - hDevice (IN): The device handle.
 - pBuffer (IN): The buffer containing the parameter data to write.
 - Address (IN): The starting address of the parameter to write.
 - Length (IN): The length of the data to write (in bytes).
- Return Value:
 - MCAM_ERR_OK (Success)
 - Error code (Failure)
- Example Code:

```

int64_t address = 0x1000; // Example address
int64_t length = 16;      // Write 16 bytes
unsigned char buffer[16] = {0xA1, 0xB2, 0xC3, 0xD4, 0xE5, 0xF6, 0x11, 0x22,
                             0x33, 0x44, 0x55, 0x66, 0x77, 0x88, 0x99, 0xAA};

CVS_ERROR result = ST_WriteParam(hDevice, buffer, address, length);

if (result != MCAM_ERR_OK)
{
    printf("Failed to write parameter: %d\n", result);
}
else
{
    printf("Successfully wrote parameter data.\n");
}

```


5.13 Event and Callback Management

These functions allow you to register specific callback functions to handle events that occur during device operation and image acquisition.

5.13.1 ST_RegisterGrabCallback

Registers a Grab event callback function for a specified device and event ID. The registered callback function is automatically called when a Grab event occurs.

```
CVS_IMPORT_EXPORT CVS_ERROR WINAPI ST_RegisterGrabCallback(  
    int32_t hDevice,  
    int32_t Event,  
    GrabCallback grabCallback,  
    void* UserData  
);
```

- Parameters:
 - hDevice (IN): The device handle.
 - Event (IN): The Grab event ID.
 - grabCallback (IN): The callback function to register.
 - UserData (IN): User-defined data (which can be referenced from within the callback function).
- Return Value:
 - MCAM_ERR_OK (Success)
 - Error code (Failure)
- Example Code:

```
void GrabEventHandler(int32_t eventID, const CVS_BUFFER* pBuffer, void*  
pUserData)  
{  
    printf("Grab event occurred! Event ID: %d\n", eventID);  
}  
  
...  
  
CVS_ERROR result = ST_RegisterGrabCallback(hDevice, EVENT_NEW_IMAGE,  
GrabEventHandler, NULL);  
  
if (result != MCAM_ERR_OK)  
{  
    printf("Failed to register grab callback: %d\n", result);  
}  
else  
{  
    printf("Successfully registered grab callback.\n");  
}
```

5.13.2 ST_UnregisterGrabCallback

Unregisters a Grab event callback function previously registered for a specified device and event ID.

```
CVS_IMPORT_EXPORT CVS_ERROR WINAPI ST_UnregisterGrabCallback(
    int32_t hDevice,
    int32_t Event
);
```

- Parameters:
 - hDevice (IN): The device handle.
 - Event (IN): The Grab event ID.
- Return Value:
 - MCAM_ERR_OK (Success)
 - Error code (Failure)
- Example Code:

```
CVS_ERROR result = ST_UnregisterGrabCallback(hDevice, EVENT_NEW_IMAGE);

if (result != MCAM_ERR_OK)
{
    printf("Failed to unregister grab callback: %d\n", result);
}
else
{
    printf("Successfully unregistered grab callback.\n");
}
```

5.13.3 ST_RegisterEventCallback

Registers a device event callback function for a specified device and event ID. The registered callback function is automatically called when a specific device event occurs.

```
CVS_IMPORT_EXPORT CVS_ERROR WINAPI ST_RegisterEventCallback(
    int32_t hDevice,
    int32_t Event,
    EventCallback eventCallback,
    void* UserData
);
```

- Parameters:
 - hDevice (IN): The device handle.
 - Event (IN): The device event ID.
 - eventCallback (IN): The callback function to register.
 - UserData (IN): User-defined data (which can be referenced from within the callback function).
- Return Value:
 - MCAM_ERR_OK (Success)
 - Error code (Failure)
- Example Code:

```

void DeviceEventHandler(const CVS_EVENT* pEvent, void* pUserData)
{
    printf("Device event occurred! Event ID: %d\n", pEvent->eventID);
}

...

CVS_ERROR result = ST_RegisterEventCallback(hDevice, EVENT_EXPOSURE_END,
DeviceEventHandler, NULL);

if (result != MCAM_ERR_OK)
{
    printf("Failed to register device event callback: %d\n", result);
}
else
{
    printf("Successfully registered device event callback.\n");
}

```

5.13.4 ST_UnregisterEventCallback

Unregisters a device event callback function previously registered for a specified device and event ID.

```

CVS_IMPORT_EXPORT CVS_ERROR WINAPI ST_UnregisterEventCallback(
    int32_t hDevice,
    int32_t Event
);

```

- Parameters:
 - hDevice (IN): The device handle.
 - Event (IN): The device event ID.
- Return Value:
 - MCAM_ERR_OK (Success)
 - Error code (Failure)
- Example Code:

```

CVS_ERROR result = ST_UnregisterEventCallback(hDevice, EVENT_EXPOSURE_END);

if (result != MCAM_ERR_OK)
{
    printf("Failed to unregister device event callback: %d\n", result);
}
else
{
    printf("Successfully unregistered device event callback.\n");
}

```

5.14 XML and JSON File Management

These functions provide the ability to save and load camera device settings using XML or JSON files.

5.14.1 ST_ImportXML

Imports an XML configuration file for the specified device.

```
CVS_IMPORT_EXPORT CVS_ERROR WINAPI ST_ImportXML(  
    int32_t hDevice,  
    const char* FileName  
);
```

- Parameters:
 - hDevice (IN): The device handle.
 - FileName (IN): The path to the XML file to import.
- Return Value:
 - MCAM_ERR_OK (Success)
 - Error code (Failure)
- Example Code:

```
CVS_ERROR result = ST_ImportXML(hDevice, "config.xml");  
  
if (result != MCAM_ERR_OK)  
{  
    printf("Failed to import XML: %d\n", result);  
}  
else  
{  
    printf("Successfully imported XML configuration.\n");  
}
```

5.14.2 ST_ExportXML

Exports the current device settings to an XML file.

```
CVS_IMPORT_EXPORT CVS_ERROR WINAPI ST_ExportXML(  
    int32_t hDevice,  
    const char* FileName,  
    int Version = XML_VERSION_1_1,  
    int Visibility = VISIBILITY_GURU  
);
```

- Parameters:
 - hDevice (IN): The device handle.
 - FileName (IN): The path to the XML file to export.
 - Version (IN, optional): The XML version (default is XML_VERSION_1_1).
 - Visibility (IN, optional): The visibility level of the settings to export (default is VISIBILITY_GURU).
- Return Value:

- MCAM_ERR_OK (Success)
 - Error code (Failure)
- Example Code:

```
CVS_ERROR result = ST_ExportXML(hDevice, "output_config.xml");

if (result != MCAM_ERR_OK)
{
    printf("Failed to export XML: %d\n", result);
}
else
{
    printf("Successfully exported XML configuration.\n");
}
```

5.14.3 ST_ImportJson

Imports a JSON configuration file for the specified device.

```
CVS_IMPORT_EXPORT CVS_ERROR WINAPI ST_ImportJson(
    int32_t hDevice,
    const char* FileName
);
```

- Parameters:
 - hDevice (IN): The device handle.
 - FileName (IN): The path to the JSON file to import.
- Return Value:
 - MCAM_ERR_OK (Success)
 - Error code (Failure)
- Example Code:

```
CVS_ERROR result = ST_ImportJson(hDevice, "config.json");

if (result != MCAM_ERR_OK)
{
    printf("Failed to import JSON: %d\n", result);
}
else
{
    printf("Successfully imported JSON configuration.\n");
}
```

5.14.4 ST_ExportJson

Exports the current device settings to a JSON file.

```
CVS_IMPORT_EXPORT CVS_ERROR WINAPI ST_ExportJson(
    int32_t hDevice,
    const char* FileName,
    int Visibility = VISIBILITY_GURU
```

```
);
```

- Parameters:
 - hDevice (IN): The device handle.
 - FileName (IN): The path to the JSON file to export.
 - Visibility (IN, optional): The visibility level of the settings to export (default is VISIBILITY_GURU).
- Return Value:
 - MCAM_ERR_OK (Success)
 - Error code (Failure)
- Example Code:

```
CVS_ERROR result = ST_ExportJson(hDevice, "output_config.json");

if (result != MCAM_ERR_OK)
{
    printf("Failed to export JSON: %d\n", result);
}
else
{
    printf("Successfully exported JSON configuration.\n");
}
```

6. Cautions

When using the SDK/library, some functions or parameters may differ depending on the version. The supported events and parameter ranges may vary depending on the camera type, such as network cameras (**GEV**) or USB3 cameras (**U3V**).

You should always check the return value of all functions and perform proper error handling. When registering callback functions, be careful of duplicate registrations or unregistrations.

7. License and Copyright

This library is the property of ©**2000 - 2025 CREVIS CO., LTD.**

Unauthorized reproduction and redistribution may be restricted; please comply with the licensing policy.