# Table of Contents

1. **OVERVIEW** ................................................................................. 5
   1.1. NOTE ABOUT THE WORD “TRIGGER” ........................................... 6
   1.2. DIRECTORIES STRUCTURE .......................................................... 7
   1.3. REDISTRIBUTABLE FILES ............................................................. 8
   1.4. SUPPORTED FRAME GRABBERS ................................................... 8
   1.5. CAMERA CALIBRATION FILE DISTRIBUTION ................................. 9
   1.6. TOOLS ......................................................................................... 10
       1.6.1. Camera Link Serial DLL Enumerator ........................................ 10
       1.6.2. Camera Control ..................................................................... 11

2. **USING THE MOTIONSCOPE M™ MV SDK** ................................. 12
   2.1. OVERVIEW .................................................................................. 12
   2.2. PROGRAMMING LANGUAGES ....................................................... 13
   2.3. LOAD/UNLOAD THE DRIVER ....................................................... 14
   2.4. READ/WRITE SYSTEM PARAMETERS ......................................... 14
   2.5. ENUMERATE/OPEN A CAMERA .................................................. 15
   2.6. CAMERA INFORMATION ............................................................. 16
   2.7. CAMERA PARAMETERS ............................................................... 17
   2.8. REGION OF INTEREST (ROI) ....................................................... 18
   2.9. MULTIPLE REGION OF INTEREST (MROI) ................................... 19
   2.10. CAMERA COMMANDS ............................................................... 20
   2.11. BACKGROUND, PIXEL SENSITIVITY AND SHUTTER LINE .......... 21
       2.11.1. Background ........................................................................ 22
       2.11.2. Background (M5 camera with Orion II sensor) ....................... 23
       2.11.3. Pixel Sensitivity Correction (PSC) ........................................ 24
       2.11.4. Shutter line (M5 camera) ..................................................... 25
   2.12. AUTOMATIC CORRECTION OF THE IMAGE ............................... 26
   2.13. MANUAL CORRECTION OF THE IMAGE ..................................... 27
   2.14. READ THE CAMERA STATUS .................................................. 28
   2.15. 64 BIT PROGRAMMING ............................................................. 28

3. **MOTIONSCOPE M™ MV SDK REFERENCE** ............................... 29
   3.1. INITIALIZATION FUNCTIONS ..................................................... 29
       3.1.1. Overview: Initialization functions .......................................... 29
       3.1.2. MVGetVersion ................................................................. 30
       3.1.3. MVLoadDriver ................................................................. 31
       3.1.4. MVUnloadDriver ............................................................... 32
       3.1.5. MVGetSystemParameter ..................................................... 33
       3.1.6. MVSetSystemParameter ..................................................... 34
       3.1.7. MVEnumCameras ............................................................... 35
       3.1.8. MVOpenCamera ............................................................... 36
       3.1.9. MVCloseCamera ............................................................... 37
   3.2. CONFIGURATION FUNCTIONS ................................................... 38
       3.2.1. Overview: Configuration functions ....................................... 38
       3.2.2. MVGetCameraInfo ............................................................ 39
       3.2.3. MVGetParameterAttribute .................................................. 40
       3.2.4. MVGetParameter ............................................................... 41
       3.2.5. MVSetParameter ............................................................... 42
   3.3. CAMERA CONTROL FUNCTIONS .............................................. 43
3.3.1. Overview: Camera Control functions ............................................................. 43
3.3.2. MV_SendCommand ....................................................................................... 44
3.3.3. MV_GetStatus .............................................................................................. 45
3.4. IMAGE CORRECTION FUNCTIONS ............................................................... 46
3.4.1. Overview: Image Corrections functions ..................................................... 46
3.4.2. MV_ImageOpen ............................................................................................ 47
3.4.3. MV_ImageClose .......................................................................................... 48
3.4.4. MV_ImageCorrectData ............................................................................... 49
3.4.5. MV_ImageGetCorrectionBuffer ................................................................. 51
4. SERIAL COMMAND PROTOCOL ..................................................................... 52
4.1. OVERVIEW ...................................................................................................... 52
4.2. BASE ROUTINES: READ AND WRITE REGISTERS ....................................... 53
4.3. READ CAMERA INFORMATION ..................................................................... 54
4.4. INIT THE CAMERA ......................................................................................... 56
4.4.1. M3 camera .................................................................................................. 56
4.4.2. M5 camera .................................................................................................. 57
4.5. WRITE PERIOD AND EXPOSURE ................................................................. 58
4.6. WRITE SENSOR GAIN .................................................................................... 60
4.7. OTHER PARAMETERS .................................................................................... 61
4.8. SEND COMMANDS ........................................................................................ 62
4.9. CHECK OVERLAP AND NON OVERLAP (M3 CAMERA) ............................... 63
4.10. HOW TO COMPUTE THE MAXIMUM FRAME RATE ..................................... 64
5. APPENDIX .......................................................................................................... 65
5.1. APPENDIX A - RETURN CODES ................................................................. 65
5.2. APPENDIX B – CAMERA INFO PARAMETERS .............................................. 66
5.3. APPENDIX C – CAMERA PARAMETERS ....................................................... 67
5.4. APPENDIX D – CAMERA COMMANDS ....................................................... 68
5.5. APPENDIX E – DATA TYPES ........................................................................ 69
5.5.1. MV_BAUD_RATE ......................................................................................... 69
5.5.2. MV_CAM_MODEL ...................................................................................... 69
5.5.3. MV_SNS_MODEL ....................................................................................... 69
5.5.4. MV_CFA_PATTERN .................................................................................. 69
5.5.5. MV_FG_TYPE ............................................................................................ 70
5.5.6. MV_REC_MODE ........................................................................................ 70
5.5.7. MV_GAIN .................................................................................................. 70
5.5.8. MV_SYNCIN_CFG ..................................................................................... 70
5.5.9. MV_TRIGIN_CFG ...................................................................................... 71
5.5.10. MV_ROI_MODE .......................................................................................... 71
5.5.11. MV_EXP_MODE ......................................................................................... 71
5.5.12. MV_PIXEL_GAIN ..................................................................................... 71
5.5.13. MV_ERROR ................................................................................................ 72
5.5.14. MV_INFO .................................................................................................. 72
5.5.15. MV_PARAM .............................................................................................. 72
5.5.16. MV_COMMAND ....................................................................................... 72
5.6. APPENDIX F – STRUCTURES ....................................................................... 73
5.6.1. MV_ENUMITEM .......................................................................................... 73
5.7. APPENDIX G – CAMERA LINK DIRECTORY .............................................. 75
1. Overview

The on-line documentation of the Machine Vision (MV) Software Development Kit (SDK) and its components is divided into the following parts:

**Using the MotionScope M™ MV SDK**

This section shows how to use the SDK.

**MotionScope M™ MV SDK Reference**

This section contains a detailed description of the SDK functions.

**Appendix**

This section provides additional information about data structures, parameters and return codes.
1.1. Note about the word “Trigger”

In machine vision applications, the word “trigger” usually refers to the camera external synchronization signal (usually a square wave signal).

In this manual the word “trigger” refers to an external event (usually a single pulse) that ends the acquisition in a “circular” or “round-robin” array of buffers. The external square wave source of synchronization is referred as “Sync In”.

The MotionScope M camera has three external BNC connectors (See the image below).

![Image of MotionScope M camera showing external connectors: Trigger (TRG), Sync In (IN), Sync Out (OUT)]
1.2. Directories structure

The default installation directory of the SDK is “C:\Program Files\IDT\MVXSDK”. Under this directory a set of sub-directories is created:

**BIN32**: it contains the 32-bit files (executables and DLLs). They can be distributed with the camera and your 32-bit application.

**BIN64**: it contains the 64-bit versions of the driver and the utilities. They can be distributed with the camera and your 64-bit application.

**CFGFILES**: it contains the camera configuration files for Coreco Sapera CamExpert (.CCF), National Instruments Measurement and Automation Explorer (.ICD) Matrox MIL Intellicam (.DCF) and BitFlow CiView (.R64).

**DOCS**: it contains the SDK documentation.

**INCLUDE**: it contains the SDK header files (H, CS, VB) and the source code of the classes that implements the serial command protocol.

**LIB**: it contains the SDK lib files.

**SOURCE**: it contains the Visual C++ SDK examples.
1.3. Redistributable Files

This section outlines the options available to third-party vendors for distributing M camera drivers for Windows 2000/XP/Vista. The files that can be redistributed are in the BIN32, BIN64 and in the CFGFILES subdirectories of the installation directory (C:\Program Files\IDT\MVXSDK).

<table>
<thead>
<tr>
<th>File</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>MVXDriver.dll</td>
<td>SDK main interface driver (32 and 64-bit)</td>
</tr>
<tr>
<td>M3.CCF</td>
<td>Configuration files for Coreco Sapera CamExpert</td>
</tr>
<tr>
<td>M5.CCF</td>
<td></td>
</tr>
<tr>
<td>IDT MotionScope M3.icd</td>
<td>Configuration Files for NI M&amp;A Explorer</td>
</tr>
<tr>
<td>IDT MotionScope M5.icd</td>
<td></td>
</tr>
<tr>
<td>IDT-M3-FreeRun.r64</td>
<td>Configuration Files for BitFlow SysReg</td>
</tr>
<tr>
<td>IDT-M5-FreeRun.r64</td>
<td></td>
</tr>
<tr>
<td>IDT MotionScope M3.dcf</td>
<td>Configuration Files for Matrox MIL Intellicam</td>
</tr>
<tr>
<td>IDT MotionScope M5.dcf</td>
<td></td>
</tr>
</tbody>
</table>

1.4. Supported Frame grabbers

The frame grabbers currently supported and tested are:

- Dalsa Coreco X64 Xcelera-CL PX4.
- National Instruments PCIe-1429.
- BitFlow Karbon-CL.
- Matrox Helios/Solios.
1.5. Camera calibration file distribution

Each camera CD contains the camera calibration file (CameraFiles subdirectory of the CD).

The default location of the camera calibration file is the C:\Common Files\IDT\CameraFiles directory. If an older version of MotionStudio is installed, the default camera calibration folder is the Windows\System32 directory.

The user may change the default directory by adding a specific key to the system registry.

HKEY_LOCAL_MACHINE\SOFTWARE\IDT\MotionProX

Then add a string value (CalibrationFileDirectory) containing the path to the directory where the calibration file is stored.

This registry value will override any further installation of Motion Studio.
1.6. Tools

The SDK package provides two tools:

1. **CameraLink Serial Enumerator**: this utility enumerates the clserxxx.dll files installed in the computer and displays information about them. It identifies the Camera Link Serial Directory and loads the clserxxx.dll files stored in that directory.

2. **MVXControl**: the utility configures the MotionScope camera parameters and starts/stops a free run acquisition.

1.6.1. Camera Link Serial DLL Enumerator

The Camera Link serial DLL Enumerator is a tool that detects and lists all the clserxxx.dll files installed on your computer in the Camera Link serial directory. For further information about the Camera Link Serial API and directory, please refer to the Camera Link AIA specifications.
1.6.2. Camera Control

The Camera Control Tool lists the M-Series cameras connected to the computer and shows a set of configuration parameters to modify.

The user may open the camera, change exposure, rate, gain and ROI, as well as synchronization source. Then the camera can start acquiring the images.
2. Using the MotionScope M™ MV SDK

2.1. Overview

The MotionScope MV SDK provides a consistent programmatic interface for camera control. The figure below illustrates the software layers that support your application when using the SDK. The SDK sends camera control commands to the camera by way of the standard CameraLink “clserxxx.dll” file specified in the CameraLink standard and provided by the frame grabber vendor. For image acquisition via CameraLink your application will directly call the frame grabber vendor’s support DLL (frame grabber’s API).

As an option, the user can implement the source code to send commands to the camera via the serial port. The Include subdirectory contains some example C++ files that show how to do it (please refer to the Serial Command Protocol chapter).
## 2.2. Programming Languages

A C/C++ header file is included in the SDK (MVX_API.h file in the Include sub-directory).

Most compiled languages can call functions; you will need to write your own header/import/unit equivalent based on the C header file.

The Windows driver is a DLL (MVXDriver.dll) that may be found also in the Bin32 sub-directory (Bin64 for the 64-bit version).

**MS Visual C++™**: A Visual C++ 6.0 stub COFF library is provided (MVXDriver.lib in the Lib sub-directory); if you are using Visual C++, link to MVXDriver.lib. The DLL uses Windows standard calling conventions (_stdcall). The 64-bit version of the stub lib is MVXDriver64.lib in the same directory.

**Borland C++ Builder™**: the MVXDriver.lib file is in COFF format. Borland C++ Builder requires the OMF format. To convert the library into to OMF format, use the IMPLIB Borland tool with the following syntax: “IMPLIB MVXDriver.lib MVXDriver.dll”.

**MS Visual C#**: the MvCamera.cs file has been added to the include folder. It wraps the APIs into a C# class. Just include the file into your C# project and call the “MvCamera” class members.

**MS Visual Basic**: Visual Basic modules are included in the Include subdirectory (MVX_API.vb and MVX_API.bas).

**Other compilers**: the Most other compilers can create a stub library for DLLs. The DLL uses Windows standard calling conventions (_stdcall).
2.3. Load/Unload the Driver

The first call into the MV driver must be **MVLoadDriver**. Call **MVUnloadDriver** when you are finished.

2.4. Read/Write System parameters

Some configuration parameters, called system parameters, should be configured before opening a camera. The system parameters are listed below:

**MVP_SYS_BAUDRATE**: it controls the serial port speed (baud rate). The values are enumerated in the MV_BAUD_RATE type.

The example below shows how to set the baud rate to 57,600

```c
// Load the driver
MVLoadDriver();

// Set the baud rate to 57,600
MVSeSystemParameter( MVP_SYS_BAUDRATE, MV_BAUDRATE_57600, 0 );

// Unload the driver
MVUnloadDriver();
```
2.5. Enumerate/Open a camera

To get the list of available cameras, call `MVEnumCameras`. The `lpSzCLSerialDir` parameter may be NULL (in that case the serial DLLs are loaded from the default Camera Link Serial Directory) or a specific directory (for instance, C:\Windows\System32). Once the cameras are enumerated, use the `nCamID` field of the camera list in your call to `MVOpenCamera`. Here is a simple example of opening the first available camera:

```c
MV_ENUMITEM mv1[10];
unsigned long nListLen = sizeof(mv1)/sizeof(MV_ENUMITEM);

// Load the driver
MVLoadDriver();

// nListLen is the length of your MV_ENUMITEM array
MVEnumCameras( NULL, &mv1[0], &nListLen );
// nListLen is now the number of cameras available. It may be
// larger than your MV_ENUMITEM array length!
if (( nListLen > 0 ) && (mv1[0].bIsOpen == FALSE ))
{
   MV_HANDLE hCam;
   // Open the first camera in the list.
   MVOpenCamera(mv1[0].nCamID, &hCam );
   // Do something...
   ...
   // Close the camera.
   MVCloseCamera( hCam );
}

// Unload the driver
MVUnloadDriver();
```

The camera list contains a unique ID which identifies each particular camera. Many developers use the unique ID to recall previous settings, or associate a meaningful name string with a camera.
2.6. Camera information

The information values that can be read from the camera are listed below:

- **Camera ID**: it's a unique number that identifies the camera. It may change if other frame grabbers and cameras are installed.
- **Camera Model**: the camera model (M3 or M5).
- **Color**: it indicates if the camera is monochrome or color.
- **Serial Number**: it's a 10 digit number that uniquely identifies the camera.
- **Hardware revision**: the camera hardware revision
- **Frame Grabber Type**: it shows
- **Sensor width and height**: the sensor size in pixels
- **Firmware Version**: the version of the firmware running in the camera.
- **Calibration File**: it indicates if the camera calibration file is installed
- **Multi-ROI support**: it indicates if the multiple-ROI capability is supported.
2.7. Camera Parameters

The parameters that can be configured in the camera are described below.

**Exposure**: The amount of time that light reaches the image sensor. It’s known also as integration time.

**Exposure mode**: the camera can acquire one (single exposure, default mode) or two images (double exposure). In double exposure mode, the exposure of the second image is fixed and equal to the read-out time.

**Period (Rate)**: the period is the inverse of the acquisition rate.

**Gain**: the camera has four sensor gain values. Larger values of gain will produce brighter images.

**Pixel Gain**: M cameras acquire 10 bit images. The user may select to read 8 bit out of 10 in three different ways: upper bits (default), middle bits, or lower bits.

**Record Mode**: the camera may stream the images to an array of buffers. If the mode is set to normal, the camera fills the buffers and stops streaming. If the mode is set to circular, the camera grabs round-robin into the buffers array, until an event trigger stops it. See the frame grabber documentation to configure a “circular” or “round-robin” acquisition in an array of buffers.

**Sync In Configuration**: the camera sync source can be the internal camera clock or an external waveform. If the source is external the acquisition of each frame can be done on the leading edge (edge-high), on the falling edge (edge-low), during the high level (pulse-high) or during the low level (pulse-low) of the sync signal.

**Region of Interest (ROI)**: see the topics below.
2.8. Region of Interest (ROI)

The M-series cameras support the region of interest feature (ROI). The sensor supports vertical windowing and the user can set the start row (Y min) and the stop row (Y max). See the picture below.

The picture shows the full sensor area (black rectangle) and the ROI area configured by the user (red rectangle).
2.9. Multiple region of Interest (MROI)

M3 cameras with firmware version 8 and above allow multiple region of interest. The user may select single rows from the sensor with a special Boolean table.

The table is an array of 1024 bytes; if a value is 0, the corresponding row is skipped, otherwise it is selected. The example below shows how to select rows 0, 2, 4, etc and acquire an image with height = 512.

```
unsigned char abyTbl[1024];
int  i,j;

// prepare the table
for(i=0,j=0; i<512; i++)
{
    abyTbl[j++] = 1;
    abyTbl[j++] = 0;
}

// change to multi roi mode
MVSetParameter(MVP_ROI_MODE, MV_ROI_MULTI, 0);

// send the table
MVSetParameter(MVP_ROI_TABLE, (unsigned long)&abyTbl[0], 0);
```

The user may select a “simple” ROI or a “multiple” ROI with the MVP_ROI_MODE parameter. Then, if the ROI mode is set to MV_ROI_MULTI, the table will be set, as in the example above. If the mode is set to MV_ROI_SIMPLE, the user will set the MVP_ROI_Y_MIN and MVP_ROI_Y_MAX parameters.

Each time the ROI mode is changed, the table or the Y min and Y max parameters need to be set.

Multiple ROI is not currently supported by M5 cameras.
2.10. Camera Commands

Once the camera parameters are configured the user can send the camera the following commands:

**MVCMD_RECORD**: the camera starts streaming images.

**MVCMD_STOP**: the camera stops streaming images.

**MVCMD_TRIGGER**: the camera receives a software trigger and ends the circular acquisition.

**MVCMD_RESET**: the camera is reset.

**NOTE**: most of the camera parameters can be changed while the camera is streaming.
2.11. Background, Pixel Sensitivity and shutter line

The images acquired from the M cameras require some post-processing correction because the architecture of the CMOS sensor introduces noise and non uniformity. To reduce those effects a calibration file is produced and distributed with the camera.

In addition to this, M5 camera images may have a horizontal line, called "shutter line".
2.11.1. Background

Background correction is required because the images acquired with the lens cap on are not perfectly black. The background images are stored in the calibration file and have been taken in different operating conditions.

The images should be subtracted to the acquired images to remove the dark background. Then the pixel should be rescaled to the current range like in the formulas below:

\[
n\text{Pixel} = n\text{Pixel} - n\text{Bkg}
\]

\[
n\text{Pixel} = \left( n\text{Pixel} \times 256 \right) / \left( 256 - n\text{Bkg} \right)
\]

The current value of index of the background buffer is returned with a call to MVGetParameter with the argument MVP_BKG_INDEX. The parameter is read-only. The value should be stored somewhere after the acquisition and used in a subsequent call to MVImageCorrectData for the correction of background.

The example below shows how to correct the background on an image buffer. The buffer pBkg has been previously acquired with the camera lens cap on.

```c
void CorrectBackground(unsigned char *pImg,
    unsigned char *pBkg,
    int nImgSize)
{
    int i, nData, nBkg;

    for (i=0; i<nImgSize; i++)
    {
        nData = *pImg;
        nBkg = *pBkg++;

        if(nBkg>=nData) nData = 0;
        else
        {
            // rescale for background
            nData = (nData * 256)/(256 - nBkg);
            if( nData>255 ) nData=255;
        }

        *pImg++ = (unsigned char)nData;
    }
}
```
2.11.2. Background (M5 camera with Orion II sensor)

The Orion II sensor has 32 dark columns that can be used to reduce the horizontal temporal noise. Not all the 32 column value can be used for that purpose:

- Columns 0 to 11: dark columns values from left side of the sensor.
- Columns 12 to 23: dark columns from right side of the sensor.

The procedure for the subtraction of the dark columns and the background is shown below.

1. Extract 2288 image values from the 2320 row buffer.

2. Compute the average values of dark columns on left and right side of the sensor: nDKL and nDkR. Skip the columns that are closer to the visible part of the sensor (10, 11, 12 and 13).

\[
\text{nDkL} = \text{avg} \{ \text{img}[0], \text{img}[1], \text{img}[2], \text{img}[3], \text{img}[4], \text{img}[5], \text{img}[6], \text{img}[7], \text{img}[8], \text{img}[9]\}
\]
\[
\text{nDkR} = \text{avg} \{ \text{img}[14], \text{img}[15], \text{img}[16], \text{img}[17], \text{img}[18], \text{img}[19], \text{img}[20], \text{img}[21], \text{img}[22], \text{img}[23]\}
\]

3. To avoid that the image values after background correction go “below zero” when we apply dark columns we need to add a constant offset to the image. The same value is included in background images, and then it is removed when the background is applied. Use the values of nDkL and nDkR to compute the value of nDkVal in each position, and then subtract dark column and background. After dark column and background subtraction the value is multiplied by a constant gain (1060). The code below is applied to each row of the image.

```c
for (j=0; j<W; j++)
{
    \text{nDkVal} = \text{nDkL} + j * (\text{nDkR} - \text{nDkL}) / (\text{MaxW} - 1);

    \text{Pix} = \text{IMG}[j] + \text{nOffs} - \text{nDkVal};
    \text{Bkg} = \text{BKG}[\text{nCol}];
    \text{Pix} = \text{Pix} - \text{Bkg};
    \text{Pix} = (\text{Pix} * \text{nGain}) >> 10;
    \text{if} (\text{Pix} > 255) \text{Pix} = 255;
    \text{PIX}[i] = \text{Pix};
}
```

**Where:**
- IMG = source image row buffer
- BKG = background row buffer
- PIX = destination image row buffer
- nOffs = 48
- nGain = 1060
- nMaxW = 2288
2.11.3. **Pixel Sensitivity Correction (PSC)**

PSC images are acquired with a uniform light source and stored in the calibration file. They are used to compensate the non-uniformity in different areas of the sensor (especially borders). PSC images are made of 16 bit coefficients that are multiplied to the pixel values. A value of 1024 indicates that the pixel does not change. The formula below shows how to compensate the value “nPixel”.

\[ n_{\text{Pixel}} = (n_{\text{Pixel}} \times n_{\text{Coeff}}) >> 10; \]

The current value of index of the PSC buffer is returned with a call to MVGetParameter with the argument **MVP_PSC_INDEX**. The parameter is read-only. The value should be stored somewhere after the acquisition and used in a subsequent call to MVImageCorrectData for the correction of PSC.

The example below shows how to correct the pixel sensitivity on an image buffer. We assume that the buffer pPSC has been previously acquired without the camera lens and with a source of constant light.

```c
void CorrectPixelSensitivity(unsigned char *pImg,
                            unsigned char *pPsc,
                            int nImgSize)
{
    int i, nData, nPsc;
    for (i=0; i<nImgSize; i++)
    {
        nData = *pImg;
        nPsc = *pPsc++;
        nData = ((nData*nPsc)>>10);
        if( nData>255 ) nData=255;
        *pImg++ = (unsigned char)nData;
    }
}
```
2.11.4. Shutter line (M5 camera)

The “shutter line” is a bright horizontal line that appears in M5 images only if the exposure and the acquisition period values satisfy the formula below:

$$T_{\text{exp}} + T_{\text{ro}} > T$$

Where:

- $T_{\text{exp}}$ = exposure time
- $T_{\text{ro}}$ = Readout time. It’s the inverse of the maximum rate (at full resolution is about 5.88 ms).
- $T$ = period.

The difference between the average intensity of the upper and lower areas of the image is about 3 or 4 counts. The lower area is usually darker (see the picture below). In cameras with Orion II sensor there is no difference in intensity between upper and lower areas.

![Shutter Line Diagram](image)

The current value of the shutter line position is returned with a call to MVGetParameter with the argument `MVP_SHUTTER_LINE`. The parameter is read-only. If the shutter line is not present and the image does not need correction, the MVGetParameter routine returns `MV_E_NO_LINE`.

The value should be stored somewhere after the acquisition and used in a subsequent call to MVImageCorrectData for the correction.
2.12. Automatic Correction of the image

The values read before the acquisition with the parameters arguments MVP_BKG_INDEX, MVP_PSC_INDEX and MVP_SHUTTER_LINE can be used to automatically correct the acquired images.

The procedure is below.

1. When the camera is online and before starting to acquire, call the MVGetParameter with the argument MVP_BKG_INDEX to read the background buffer index (nBKG), the argument MVP_PSC_INDEX to read the PSC buffer index (nPSC) and MVP_SHUTTER_LINE to read the shutter line position if the camera is an M5 (nSHL). Then store the nBKG, nPSC and nSHL values in variables that will be used later.

2. Open the image correction engine (MVImageOpen) with the camera serial number and sensor model as arguments. These values are required to identify the camera calibration file and read the correct background and PSC buffers. The camera can be disconnected when this routine is called.

3. Call the MVImageCorrectData routine with the values of nBKG, nPSC and nSHL, the dimensions of the input image, the correction flag and the shutter line offset. The correction flag can be a combination of MV_IMGC_BKG bit (for background), MV_IMGC_PSC bit (for PSC) and MV_IMGC_SHL bit (for shutter line). The value MV_IMGC_ALL includes all the above values. A positive offset will be added to the area of the image that is below the shutter line, a negative offset will be subtracted to the same area.


NOTE about M5 camera with Orion II sensor: Orion II sensor has 32 dark columns. When the MVImageCorrectData routine is called, the input buffer width is 2320 pixels but the image width parameter must be 2288. The routine automatically extracts the 32 dark columns from the input buffer and generates a 2288 pixels wide output image.
2.13. Manual correction of the image

If the user wants to manually correct the images, the SDK provides a routine that returns the background and pixel sensitivity buffers.

The procedure is below.

1. When the camera is online and before starting to acquire, call the MVGetParameter with the argument MVP_BKG_INDEX to read the background buffer index (nBKG), the argument MVP_PSC_INDEX to read the PSC buffer index (nPSC) and MVP_SHUTTER_LINE to read the shutter line position if the camera is an M5 (nSHL). Then store the nBKG, nPSC and nSHL values in variables that will be used later.

2. Open the image correction engine (MVImageOpen) with the camera serial number and sensor model as arguments. These values are required to identify the camera calibration file and read the correct background and PSC buffers. The camera may be disconnected when this routine is called.

3. Allocate two buffers, one for the background and one for the PSC. The background buffer size is W x H, where W is the sensor width and H is the sensor height (the values can be read from the camera). The PSC buffer size is 2 x W x H.

4. Call the MVImageGetCorrectionBuffer routine two times. The first time use the value of nBKG as index and MV_IMG_BKG as correction flag, to read the background buffer. Then use nPSC as index and MV_IMG_PSC as correction flag, to read the PSC data.

5. Write some lines of code to correct the image for background (see topic 2.10.1 or 2.10.2), PSC (see topic 2.10.3) and shutter line.

6. Close the correction engine with a call of MVImageClose.
2.14. Read the camera status

A routine has been introduce to read the camera status (MVGetStatus). The returned status may be 0 if the camera is in idle mode or 1 if it's busy streaming. The routine may be used to detect whether an acquisition is done.

2.15. 64 Bit Programming

One of the main issues in migrating software from 32 bit to 64 bit platforms is the size of types.

An "int" and a "long" are 32-bit values in 64-bit Windows operating systems. For programs that you plan to compile for 64-bit platforms, you should be careful not to assign pointers to 32-bit variables. Pointers are 64-bit on 64-bit platforms, and you will truncate the pointer value if you assign it to a 32-bit variable.

In the MV SDK all the parameters are 64 bit values, but for most of them the most significant 32-bit part is set to 0. You should be careful if you set multiple ROI because the parameter is a table. The example below shows how to read and write the multiple ROI table in a 64-bit program.

```c
unsigned char abyTbl[1024];
unsigned long *pTbl = (unsigned long *)&abyTable[0];
unsigned long nLoValue, nHiValue;

// read the table
MVGetParameter(MVP_ROI_TABLE, pTbl, NULL);

// change 1 item of the table
abyTbl[0] = 0;

// write the table
nLoValue = (unsigned long) &abyTbl[0];
nHiValue = (unsigned long) ((&abyTbl[0]) >> 32);
MVSetParameter(MVP_ROI_TABLE, nLoValue, nHiValue);
```
3. MotionScope M™ MV SDK Reference

3.1. Initialization Functions

3.1.1. Overview: Initialization functions

Initialization functions allow the user to initialize the driver, enumerate the available cameras, open and close them.

- **MVGetVersion**: returns the DLL version numbers.
- **MVLoadDriver**: loads the driver and initializes it.
- **MVUnloadDriver**: unloads the driver.
- **MVGetSystemParameter**: reads one of the system parameters
- **MVSetSystemParameter**: configures one of the system parameters
- **MVEnumCameras**: enumerates the M-Series cameras connected to the computer.
- **MVOpenCamera**: opens a camera.
- **MVCloseCamera**: closes a camera previously open.
3.1.2. MVGetVersion

**MV_ERROR MVGetVersion (unsigned short *pVerMajor, unsigned short *pVerMinor)**

**Return values**

MV_SUCCESS if successful, otherwise

MV_E_GENERIC_ERROR if the version numbers could not be extracted from the driver.

**Parameters**

*pVerMajor*

Specifies the pointer to the variable that receives the major version number

*pVerMinor*

Specifies the pointer to the variable that receives the minor version number

**Remarks**

This function must be called to retrieve the DLL version.

**See also:**
3.1.3. MVLoadDriver

**MV_ERROR MVLoadDriver (void)**

Return values

MV_SUCCESS if successful, otherwise

MV_E_GENERIC_ERROR if any error occurs during the initialization.

**Parameters**

None

**Remarks**

The routine loads the driver DLL and initializes it. It must be called before any other routine, except MVGetVersion. If any error occurs, the routine returns MV_E_GENERIC_ERROR.

See also: MVUnloadDriver
3.1.4. MVUnloadDriver

void MVUnloadDriver (void)

Return values

None

Parameters

None

Remarks

This function must be called before terminating the application. This function frees any memory and resource allocated by the driver and unloads it.

See also: MVLoadDriver
3.1.5. MVGetSystemParameter

**MV_ERROR MVGetSystemParameter** *(MV_PARAM nParamKey, unsigned long *pLoValue, unsigned long *pHiValue)*

**Return values**

- MV_SUCCESS if successful, otherwise
- MV_E_INVALID_ARGUMENTS, if one of the arguments is not valid.
- MV_E_NOT_SUPPORTED, if the nParamKey is not supported.

**Parameters**

- **nParamKey**
  Specifies which parameter the function returns
- **pLoValue**
  Specifies the pointer to the variable that receives the LS part of the parameter's value
- **pHiValue**
  Specifies the pointer to the variable that receives the MS part of the parameter's value

**Remarks**

This function reads a system parameter.

**See also:** MVSetSystemParameter
3.1.6. MVSetSystemParameter

**MV_ERROR MVSetSystemParameter (MV_PARAM nParamKey, unsigned long nLoValue, unsigned long nHiValue)**

**Return values**
- MV_SUCCESS if successful, otherwise
- MV_E_INVALID_VALUE, if the parameter value is not valid
- MV_E_NOT_SUPPORTED, if the nParamKey is not supported.

**Parameters**

- **nParamKey**
  Specifies which parameter the function sets.
- **nLoValue**
  Specifies the LS part of the parameter's value
- **nHiValue**
  Specifies the MS part of the parameter's value

**Remarks**

This function writes a system parameter.

**See also:** MVGetSystemParameter
3.1.7. MVEnumCameras

\texttt{MV\_ERROR MVEnumCameras (const char \*lpszCLSerialDir, PMV\_ENUMITEM pItemList, unsigned long \*pItemNr)}

Return values

MV\_SUCCESS if successful, otherwise

MV\_E\_INVALID\_ARGUMENTS, if any of the parameters is not valid.

Parameters

\textit{lpszCLSerialDir}

Specifies the camera Link Serial Directory

\textit{pItemList}

Specifies the pointer to an array of MV\_ENUMITEM structures

\textit{pItemNr}

Specifies the pointer to the variable that receives the number of detected cameras

Remarks

The routine enumerates the active cameras and fills the MV\_ENUMITEM structures with information about them. The driver loads the clserxxx.dll files detected in the \textit{lpszCLSerialDir} directory or in the default Camera Link Serial directory if \textit{lpszCLSerialDir} is NULL. This routine must be called before \texttt{MVOpenCamera} to find out which cameras are available. The pItemNr variable must specify the number of structures in the pItemList array and receives the number of enumerated cameras.

See also: \texttt{MVOpenCamera}
3.1.8. MVOpenCamera

```
MV_ERROR MVOpenCamera (unsigned long nCameraId, MV_HANDLE* pHandle)
```

**Return values**

MV_SUCCESS if successful, otherwise

MV_E_INVALID_CAMERA_ID, if the camera ID is not valid.

MV_E_CAM_ALREADY_OPEN, if the camera is already open.

**Parameters**

*nCameraId*

Specifies the ID of the camera to be opened

*pHandle*

Specifies the pointer to the variable that receives the camera handle

**Remarks**

The routine opens the camera with the nCamId identifier. The value can be retrieved by calling the MVEnumCameras routine (see the MV_ENUMITEM structure).

See also: MVCloseCamera
3.1.9. MVCloseCamera

**MV_ERROR MVCloseCamera (MV_HANDLE hCamera)**

**Return values**

MV_SUCCESS if successful, otherwise

**MV_E_INVALID_HANDLE**, if the camera handle is not valid.

**Parameters**

*hCamera*

Specifies the handle to an open camera

**Remarks**

Closes an open Camera

See also: MVOpenCamera
3.2. Configuration Functions

3.2.1. Overview: Configuration functions

The configuration functions allow the user to configure the camera.

**MVGetCameraInfo** gets information from the camera, such as camera model, serial number, sensor size, etc.

**MVGetParameterAttribute** gets the parameters attributes (minimum and maximum values).

**MVGetParameter** gets a parameter from the camera.

**MVSetParameter** sets a camera parameter.
3.2.2. MVGetCameraInfo

**MV_ERROR MVGetCameraInfo** (MV_HANDLE hCamera, MV_INFO nInfoKey, unsigned long *pLoValue, unsigned long *pHiValue)

Return values

MV_SUCCESS if successful, otherwise

MV_E_INVALID_HANDLE, if the camera handle is not valid.

MV_E_INVALID_ARGUMENTS, if one of the arguments is not valid.

MV_E_NOT_SUPPORTED, if the nInfoKey is not supported.

Parameters

**hCamera**

Specifies the handle to an open camera

**nInfoKey**

Specifies which parameter the function has to return

**pLoValue**

Specifies the pointer to the variable that receives the LS part of the info value

**pHiValue**

Specifies the pointer to the variable that receives the MS part of the info value

Remarks

This function returns camera specific information, such as camera model and serial number. See the [Appendix B](#) for a list of all the available nInfoKey values.

See also:
3.2.3. MVGetParameterAttribute

MV_ERROR MVGetParameterAttribute (MV_HANDLE hCamera, MV_PARAM nParamKey, unsigned long *pMinValue, unsigned long *pMaxValue)

Return values

MV_SUCCESS if successful, otherwise
MV_E_INVALID_HANDLE, if the camera handle is not valid.
MV_E_INVALID_ARGUMENTS, if one of the arguments is not valid.
MV_E_NOT_SUPPORTED, if the nParamKey is not supported.

Parameters

hCamera
Specifies the handle to an open camera

nParamKey
Specifies which parameter the function returns.

pMinValue
Specifies the pointer to the parameter’s minimum value.

pMaxValue
Specifies the pointer to the parameter’s maximum value.

Remarks

This function reads the parameter’s minimum and maximum values.

See also: MVGetParameter
3.2.4. MVGetParameter

```
MV_ERROR MVGetParameter (MV_HANDLE hCamera, MV_PARAM nParamKey, unsigned long *pLoValue, unsigned long *pHiValue)
```

**Return values**

MV_SUCCESS if successful, otherwise

MV_E_INVALID_HANDLE, if the camera handle is not valid.

MV_E_INVALID_ARGUMENTS, if one of the arguments is not valid.

MV_E_NOT_SUPPORTED, if the nParamKey is not supported.

**Parameters**

*hCamera*

Specifies the handle to an open camera

*nParamKey*

Specifies which parameter the function returns

*pLoValue*

Specifies the pointer to the variable that receives the LS part of the parameter's value

*pHiValue*

Specifies the pointer to the variable that receives the MS part of the parameter's value

**Remarks**

This function reads a parameter from the camera.

**See also:** MVSetParameter
3.2.5. MVSetParameter

**MV_ERROR MVSetParameter (MV_HANDLE hCamera, MV_PARAM nParamKey, unsigned long nLoValue, unsigned long nHiValue)**

**Return values**

MV_SUCCESS if successful, otherwise
MV_E_INVALID_HANDLE, if the camera handle is not valid.
MV_E_INVALID_ARGUMENTS, if one of the arguments is not valid.
MV_E_NOT_SUPPORTED, if the nParamKey is not supported.
MV_E_INVALID_VALUE, if the parameter value is not valid

**Parameters**

*hCamera*

Specifies the handle to an open camera

*nParamKey*

Specifies which parameter the function sets.

*nLoValue*

Specifies the LS part of the parameter's value

*nHiValue*

Specifies the MS part of the parameter's value

**Remarks**

This function writes a parameter to the camera.

**See also:** MVGetParameter
3.3. Camera Control Functions

3.3.1. Overview: Camera Control functions

The camera control functions allow the user to send commands to the camera and retrieve the pointers to background and pixel sensitivity correction frames.

**MVSendCommand** sends a specific command to the camera.

**MVGetStatus** returns the camera current status.
3.3.2. MVSendCommand

```
MV_ERROR MVSendCommand (MV_HANDLE hCamera, MV_COMMAND nCmdKey, unsigned long nLoParam, unsigned long nHiParam)
```

**Return values**

MV_SUCCESS if successful, otherwise

MV_E_INVALID_HANDLE, if the camera handle is not valid.

MV_E_NOT_SUPPORTED, if the command key is not supported.

**Parameters**

*hCamera*

Specifies the handle to an open camera

*nCmdKey*

Specifies the command code

*nLoParam*

Specifies the LS part of the optional command parameter

*nHiParam*

Specifies the MS part of the optional command parameter

**Remarks**

This function sends a command to the camera. For a list of the commands please refer to the Appendix.

**See also:**
3.3.3. MVGetStatus

MV_ERROR MVGetStatus (MV_HANDLE hCamera, unsigned long pnStatus)

Return values

MV_SUCCESS if successful, otherwise

MV_E_INVALID_HANDLE, if the camera handle is not valid.

MV_E_INVALID_ARGUMENTS, if input parameters are not valid.

Parameters

hCamera

Specifies the handle to an open camera

pnStatus

Specifies the point to the variable where the status will be stored

Remarks

This function returns the camera current status (0: idle, 1:busy)

See also:
3.4. Image Correction Functions

3.4.1. Overview: Image Corrections functions

The image correction routines allow the user to correct the images when the camera is offline.

MVImageOpen open the image correction engine.

MVImageClose closes and image correction engine previously open.

MVImageCorrectData corrects the image with background data, pixel sensitivity data and removes the shutter line. It also compensates the difference of intensity of upper and lower areas of the image when the shutter line is present.

MVImageGetCorrectionBuffer returns a buffer of Background or PSC data.
3.4.2. MVImageOpen

MV_ERROR MVImageOpen (unsigned long nSerialNumber, unsigned long nSnsModel, MV_IMG_HANDLE *pHandle)

Return values

MV_SUCCESS if successful, otherwise
MV_E_INVALID_ARGUMENTS, if the pointer to the buffer is not valid.

Parameters

nSerialNumber
Specifies the serial number of the camera which acquired the images

nSnsModel
Specifies the sensor model of the camera which acquired the images

pHandle
Specifies the pointer to a variable that receives the handle to the correction engine

Remarks

This function opens an image correction engine and returns a handle.

See also: MVImageClose
3.4.3. MVImageClose

**MV_ERROR MVImageClose (MV_IMG_HANDLE hImage)**

**Return values**

MV_SUCCESS if successful, otherwise

MV_E_INVALID_HANDLE, if the camera handle is not valid.

**Parameters**

*hImage*

Specifies the handle to an open image engine

**Remarks**

This function closes an image correction engine previously open.

**See also:** MVImageOpen
3.4.4. MVImageCorrectData

```c
MV_ERROR MVImageCorrectData (MV_IMG_HANDLE hImage, void* pDataBufIn, void* pDataBufOut, unsigned long nBkgIndex, unsigned long nPscIndex, unsigned long nShlPos, unsigned long nRoiXo, unsigned long nRoiYo, unsigned long nRoiWidth, unsigned long nRoiHeight, unsigned long nCorrectionFlag, int nShlOffset)
```

**Return values**

MV_SUCCESS if successful, otherwise
MV_E_INVALID_HANDLE, if the camera handle is not valid.
MV_E_INVALID_ARGUMENTS, if the input arguments are not valid.

**Parameters**

- **hImage**
  Specifies the handle to an open image correction engine
- **pDataBufIn**
  Specifies the pointer to the input data buffer
- **pDataBufOut**
  Specifies the pointer to the output data buffer
- **nBkgIndex**
  Specifies the index of the background image to be used for the correction
- **nPscIndex**
  Specifies the index of the PSC image to be used for the correction
- **nShlPos**
  Specifies the position of the shutter line
- **nRoiXo, nRoiYo**
  Specifies the Origin coordinates of the input image ROI
- **nRoiWidth, nRoiHeight**
  Specifies the dimensions of the input image ROI
- **nCorrectionFlag**
  Specifies the correction flag
- **nShlOffset**
  Specifies the offset that is added (if positive) or subtracted (if negative) to the area of the image that is below the shutter line.

**Remarks**

This function corrects the image with background and pixel sensitivity data and removes the shutter line. It also compensates the difference of intensity of upper and lower areas of the image when the shutter line is present.
The operations are selected with the bits of the nCorrectionFlag variable (MV_IMGC_BKG for background, MV_IMGC_PSC for PSC and MV_IMGC_SHL for shutter line) and with the value of the nShlOffset. Other input parameters are the background index, the PSC index, and the shutter line position. Those parameters have been read from the camera before the acquisition. If the camera model is M5 and the shutter line position is inside the image, the image can be also corrected with the shutter line offset. If the offset is positive it will be added to the area of the image that is below the shutter line (lower area). If the offset is negative it will be subtracted to the same area. If the offset is zero, no correction will be done.

**See also:** MVImageOpen, MVImageClose
3.4.5. MVImageGetCorrectionBuffer

```
MV_ERROR MVImageGetCorrectionBuffer (MV_IMG_HANDLE hImage, 
unsigned long nIndex, unsigned long nCorrectionFlag, void* pDataBuf, 
unsigned long nBufSize)
```

Return values

MV_SUCCESS if successful, otherwise
MV_E_INVALID_HANDLE, if the camera handle is not valid.
MV_E_INVALID_ARGUMENTS, if the input arguments are not valid.
MV_E_NO_DATA, if the correction data is not available
MV_E_BUF_TOO_SMALL, if the output buffer is too small

Parameters

`hImage`
Specifies the handle to an open image correction engine

`nIndex`
Specifies the index of the correction data

`nCorrectionFlag`
Specifies the type of buffer to copy (MV_IMGC_BKG or MV_IMGC_PSC)

`pDataBuf`
Specifies the pointer to the output data buffer

`nBufSize`
Specifies the size in bytes of the data buffer

Remarks

This function copies to the pDataBuf buffer the correction data specified by the nIndex value and the nCorrectionFlag (MV_IMGC_BKG for background data, MV_IMGC_PSC for PSC data). The data may be either background or pixel sensitivity correction data. If the data is not available or the buffer size is too small the routine returns an error code.
4. Serial Command Protocol

4.1. Overview

The serial command protocol is implemented in the C++ classes contained in some source code files in the “Include” directory of the SDK:

**CamRegisters.h**: header file that contains the definitions of internal camera registers.

**CLinkPort.cpp, CLinkPort.h**: files that implement the CCLinkPort class. It contains all the information for reading the camera EEPROM information, configuring the camera registers, and sending camera commands. In the same file, the CSerial class is used to include the base serial operations (Open, close, write, read, flush).

**CCLinkCam.cpp, CCLinkCam.h**: files that implement the CCLinkCamera class. It contains all the information to identify a camera, initialize it and start the configuration.

---

The user should provide routines from the base serial port operations (Open, Close, Read, Write, Flush, Get-Baud-Rates and Set-Baud-Rate). The code will be included in the CSerial class.
4.2. Base Routines: read and write registers

The camera internal registers are read and written with the protocol implemented in the routines below.

```c
// Read register
int ReadRegister(SER_HANDLE hSerial, unsigned char byAdd, unsigned char *pbData)
{
    unsigned long dwSize;
    unsigned char abyBuf[2];
    int nRet = -1;
    //
    abyBuf[0] = 0x01 | 0x00 | 0x00 | (byAdd & 0xf) << 1;
    abyBuf[1] = 0x01 | 0x40 | 0x20 | ((byAdd >> 4) & 0xf) << 1;
    dwSize = 2;
    FlushPort(hSerial);
    if(SerialWrite(hSerial, (char*)abyBuf, &dwSize, 100)==0)
    {
        dwSize = 1;
        nRet = SerialRead(hSerial, (char*)pbData, &dwSize, 100);
    }
    return nRet;
}

// Write register
int CCLinkPort::WriteRegister(SER_HANDLE hSerial, unsigned char byAdd, unsigned char byData)
{
    unsigned long dwSize = 4;
    unsigned char abyBuf[4];
    abyBuf[0] = 0x01 | 0x00 | 0x00 | (byAdd & 0xf) << 1;
    abyBuf[1] = 0x01 | 0x80 | 0x20 | ((byAdd >> 4) & 0xf) << 1;
    abyBuf[2] = 0x01 | 0x00 | 0x00 | (byData & 0xf) << 1;
    abyBuf[3] = 0x01 | 0x00 | 0x20 | ((byData >> 4) & 0xf) << 1;
    return SerialWrite(hSerial, (char*)abyBuf, &dwSize, 100);
}

// Write register block
int CCLinkPort::WriteRegisterBlock(SER_HANDLE hSerial, unsigned char byAdd, unsigned char *pbyData, int nSize)
{
    unsigned char abyBuf[8192];
    unsigned long dwSize;
    int i;
    byAdd |= 0x80;
    abyBuf[0] = 0x01 | 0x00 | 0x00 | (byAdd & 0xf) << 1;
    abyBuf[1] = 0x01 | 0x80 | 0x20 | ((byAdd >> 4) & 0xf) << 1;
    for( i=0; i<nSize; i++ )
    {
        abyBuf[i*2+2] = 0x01 | 0x00 | 0x00 | (pbyData[i] & 0xf) << 1;
        abyBuf[i*2+3] = 0x01 | 0x00 | 0x20 | ((pbyData[i] >> 4) & 0xf) << 1;
    }
    dwSize = 2*nSize + 2;
    return SerialWrite(hSerial, (char*)abyBuf, &dwSize, 500);
}
```
4.3. Read camera information

The camera information is stored in a VENDOR_INFO structure inside the camera EEPROM. The EEPROM can be read via the GetVendorInfo routine in the CLinkPort.cpp file.

```
typedef struct
{
    unsigned short nSize;
    char szSignature[4];
    unsigned long nSerial;
    unsigned char bColor;
    unsigned short nSnsRev;
    unsigned short nReserved1;
    unsigned short nRevision;
    unsigned short nModel;
    unsigned short nReserved2;
    unsigned short anDACs0[20];
    unsigned short anDACs1[20];
    unsigned short anDACs2[20];
    unsigned short anDACs3[20];
    unsigned short nClock;
    unsigned short nExtraRows;
    unsigned short nRCC;
    unsigned short nReserved3;
    unsigned short nReserved4;
    unsigned short nExpSteps;
    unsigned short nReserved5;
    unsigned short nRegRp1;
    unsigned short nRegRp2;
    unsigned short nRegRp3;
} VENDOR_INFO, *PVENDOR_INFO;
```

- `szSignature`: a four character string that contains “VINF”
- `nSerial`: the camera serial number
- `bColor`: 1 if the camera is color, otherwise 0.
- `nSnsRev`: 0 for M3 and M5 cameras with Orion I sensor, 1 or above for M5 cameras with Orion II sensor.
- `nRevision`: the hardware revision.
- `nModel`: 5 for M3 camera, 7 for M5 camera.
- `anDACs0, 1, 2, 3`: DACs set for gain 0, 1, 2 and 3.
- `nClock`: the camera internal clock in MHz * 100.
- `nExtraRows`: the camera extra rows (used for internal calculations)
- `nRCC`: the camera row clocks count (used for internal calculations)
- `nExpSteps`: the camera exposure steps (used for internal calculations)
- `nRegRp1, 2, 3`: internal registers for M3 camera.

The routine used to read the vendor info structure is shown below.
```c
int CCLinkPort::GetVendorInfo(SER_HANDLE hSerial, PVENDOR_INFO pInfo)
{
    unsigned char abyInfo[512] = {0};
    PVENDOR_INFO pI = (PVENDOR_INFO)pInfo;
    unsigned long dwAdd=0,dwSize=512;
    char szErrText[256]="";
    unsigned long nErrLen=256;
    int nRet = CL_ERR_GENERIC;
    //
    FlushPort(hSerial);
    WriteRegisterBlock(hSerial,CR_SP_COMM_BUFF, &dwAdd, 4);
    WriteRegisterBlock(hSerial,CR_SP_COMM_BUFF, &dwSize, 4);
    if( WriteRegister(hSerial,CR_SP_COMM, CR_SP_COMM_READ_FLASH)==0 )
    {
        Sleep(100);
        nRet = SerialRead(hSerial, (char*)abyInfo, &dwSize, 1000);
        if( nRet==CL_ERR_SUCCESS )
        {
            if( _strcmpi(pI->szSignature,VINF_SIGNATURE)==0 )
                memcpy(pInfo,pI, sizeof(VENDOR_INFO));
            else nRet = CL_ERR_GENERIC;
        }
    }
    return nRet;
}
```

The routine reads the structure and validates it with a comparison between the `szSignature` string and the "VINF" constant. After the read of the structure, the user can also read the firmware version with the routine below.

```c
void CCLinkPort::GetFirmwareVersions(SER_HANDLE hSerial, unsigned long& nVer1, unsigned long& nVer2)
{
    unsigned char byData = 0;
    nVer1 = 128;
    nVer2 = 0;
    if( ReadRegister(hSerial,SR_VERSION, &byData)==CL_ERR_SUCCESS )
        nVer1 = byData;
    byData = 0;
    if( ReadRegister(hSerial,SR_VERSION+1, &byData)==CL_ERR_SUCCESS )
        nVer2 = byData;
    if(nVer2==0) nVer2 = 1;
}
```

The `nVer1` value is usually 128; the `nVer2` value may change and may be used to detect which initialization values have to be sent to the camera.
4.4. Init the camera

After the camera detection (model, sensor, serial number) the user should send the initialization sequence to the camera. The steps are shown in the Open routine in the CLinkCam.cpp file.

- Send Reset Command (see the following topics).
- Set Camera Initialization for the configuration of some camera internal registers.
- Set Default values of configuration parameters (Rate, exposure, gain).

4.4.1. M3 camera

The routine can be called only once. It sets three registers with values retrieved from the VENDOR_INFO structure.

```c
void CCLinkPort::SetM3Registers(SER_HANDLE hSerial, unsigned long nReg1, unsigned long nReg2, unsigned long nReg3)
{
    WriteRegister(hSerial,CR_RP1,(unsigned char)(nReg1&0xFF));
    WriteRegister(hSerial,CR_RP2,(unsigned char)(nReg2&0xFF));
    WriteRegister(hSerial,CR_RP3,(unsigned char)(nReg3&0xFF));
}
```
4.4.2. M5 camera

The routine initializes the camera internal timing. The timing sequence changes with different firmware and it should be sent each time the camera is reset.

```c
void CCLinkPort::SetM5Registers(SER_HANDLE hSerial, unsigned long nFwVersion, unsigned long nSnsModel)
{
    unsigned short awTim1[] = { 0x0803, 0x0FFF, 0x0803, 0x0000, 0x0003 };
    unsigned short awTim2[] = { 0x0803, 0x08FF, 0x0F81, 0x0000, 0x0001 });
    unsigned char abyVal[] = {0,100,170,190,10,160,255,2,255,255,200,200};
    unsigned long nRegAM40 = 0x0322;
    unsigned short d, *pwTim;
    int i;

    pwTim = nFwVersion>10 ? awTim2 : awTim1;
    for(i=0;i<5;i++)
    {
        d = pwTim[i];
        WriteRegister(hSerial,(CR_TIMING+2*i),d&0xFF);
        WriteRegister(hSerial,(CR_TIMING+2*i+1),(d>>8)&0xFF);
    }

    // write AM40 registers
    WriteRegisterBlock(hSerial,CR_SP_COMM_BUFF,&nRegAM40, 4);
    WriteRegister(hSerial, CR_SP_COMM, CR_SP_COMM_SET_SREG);
    if(nSnsModel==MV_SM_ORION_II)
    {
        // write sensor registers
        for(i=0;i<12;i++)
            WriteRegister(hSerial,(CR_M5_REG_START+i),abyVal[i]);
    }
}
```
4.5. Write Period and Exposure

The procedure for the configuration of the camera period and exposure are listed below.

- Convert the exposure from microseconds into camera internal units: US2Steps() routine in CLinkCam.cpp file.

- Convert the period from microseconds into camera internal units and correct the value.

```c
unsigned long CCLinkCamera::ComputePeriodSteps(unsigned long nPeriodUS)
{
    unsigned long nSteps = m_Ser.US2Steps(m_Cfg.nClock,m_Cfg.nExpSteps,nPeriodUS);
    if(m_Cfg.nCamModel==MV_CM_MS_M3) nSteps -= 0;
    else
    {
        if(m_Cfg.nSnsModel==MV_SM_ORION_II) nSteps -= 25;
        else nSteps = nSteps - (m_Cfg.nFwVersion>10 ? 70 : 22);
    }
    return nSteps;
}
```

- If the exposure mode is double, compute the read out time with the number of rows and camera parameters read from the VENDOR_INFO structure (extra rows, RCC and clock). Subtract the read out from the period value.

  If the firmware version is larger than 17, the register 0x4C should be configured. The value is computed when the rate or the exposure changes and it is used to compensate the difference of intensity with low rates.

```c
void CCLinkCamera::SetPeriodAndExposure()
{
    unsigned long nPer = m_Cfg.nPeriodStp;
    unsigned long nExp = m_Cfg.nExpStp;
    unsigned long nRows = m_Cfg.nRoiY1 - m_Cfg.nRoiYo + 1;
    unsigned long nRo = READ_OUT(m_Cfg.nRCC, (nRows+m_Cfg.nExtraRows), m_Cfg.nExpSteps);
    if(m_Cfg.nExpMode==MV_EM_DOUBLE)
        nPer -= nRo;

    m_Ser.WritePeriodAndExposure(m_hSer,nPer,nExp);
    if(m_Cfg.nFwVersion>17)
    {
        int nSafeGap = 32;
        int nTX_R = (m_Cfg.nCamModel==MV_CM_MS_M5) ? 200 : 8;
        int n = (nPer - nExp)*8 - nROT;
        int nROT = (nRows + 1)*m_Cfg.nRCC + nTX_R;
        n -= nROT + nSafeGap;
        if( m_Cfg.nCamModel==MV_CM_MS_M3 || n < 0 ) n = -1;
        m_Ser.WriteRegister4C(m_hSer,(unsigned long)n);
    }
}```
• Write the period and exposure registers (see below)

```cpp
void CCLinkPort::WritePeriodAndExposure(SER_HANDLE hSerial,
unsigned long nPer, unsigned long nExp)
{
    unsigned long nData;
    int i;

    if(nExp>(nPer-2)) nExp = nPer-2;

    // exposure
    nData = (nPer - 3) - (nExp - 1);
    for( i=0; i<4; i++ )
    {
        WriteRegister(hSerial,(unsigned char)(CR_EXPOSURE_TIME+i),
(unsigned char)(nData&0xff) );
        nData >>= 8;
    }

    // period
    nData = nPer - 3;
    for( i=0; i<4; i++ )
    {
        WriteRegister(hSerial,(unsigned char)(CR_PERIOD+i), (unsigned
char)(nData&0xff));
        nData >>= 8;
    }
}
```

• If the firmware version is 18 or above, Write the register 0x4C (see below)

```cpp
void CCLinkPort::WriteRegister4C(SER_HANDLE hSerial, unsigned long
nReg)
{
    unsigned long i,nVal;
    nVal = nReg;
    for(i=0; i<3; i++)
    {
        WriteRegister(hSerial, (0x4C+i), (nVal&0xff));
        nVal >>= 8;
    }
}
```
4.6. Write sensor gain

Sensor gain is controlled by a full set of DAC voltages that are sent to the camera. The VENDOR_INFO structure (that should be read before any other operation) contains 4 sets of DAC voltages, one for each sensor gain.

When the sensor gain changes the corresponding set of DAC values should be sent to the camera.

The number of DAC values is 16 for M5 and 10 for M3. M3 camera has an additional value that is used for the overlap/non overlap mode. For further explanation please see the topic 4.9.

```c
int CCLinkPort::WriteDACs(SER_HANDLE hSerial, unsigned long *pDAC, int nIsM5)
{
    unsigned long nPtr=0, nDACCode;
    _int64 nData,nMask;
    unsigned long i, j, nx, ny, sh, nBits,nSize, nDACsRef;
    unsigned char nDataBit;
    if( nIsM5 ) { nx = 8; ny = 2; nSize=16; nDACsRef = 5000; }
    else { nx =  2; ny = 2; nSize=10; nDACsRef = 3000; }
    nBits = 16*ny;
    nMask = ( (__int64)1)<<(nBits-1);
    for( i=0; i<8; i++)
    {
        nData = 0;
        for( j=0, nData=0, sh=0; j<ny; j++ )
        {
            if( j==(ny-1) && i>=nx ) nDACCode = 0;
            else nDACCode = (1024 * pDAC[i + 8*j]) / nDACsRef;
            if (nDACCode > 1023) nDACCode = 1023;
            nData += ( __int64)(((i + 1)<<12) + (nDACCode << 2)) << sh;
            sh += 16;
        }
        // 2 values
        WriteRegister(hSerial,CR_DACS,0);
        WriteRegister(hSerial,CR_DACS,CB_DACS_CS);
        // 32 x 3 = 96 values (or 48 x 3 values)
        for (j = 0; j < nBits; j++)
        {
            nDataBit = (nData & nMask) ? CB_DACS_DATA : 0;
            nData <<= 1;
            WriteRegister(hSerial,CR_DACS, CB_DACS_CS + nDataBit);
            WriteRegister(hSerial,CR_DACS, CB_DACS_CS + nDataBit + CB_DACS_CLOCK);
            WriteRegister(hSerial,CR_DACS, CB_DACS_CS + nDataBit);
        }
        // 2 values
        WriteRegister(hSerial,CR_DACS, CB_DACS_CS);
        WriteRegister(hSerial,CR_DACS, 0);
    }
    return CL_ERR_SUCCESS;
}
```
4.7. Other parameters

Other parameters that can be set are:

**ROI**: WriteROI() routine in CLinkPort.cpp file.

**Exposure mode**: WriteExpMode() routine in CLinkPort.cpp file.

**Pixel Gain**: WritePixelGain() routine in CLinkPort.cpp file.

**“Sync In” configuration**: WriteSyncInCfg() routine in CLinkPort.cpp file.

**“Trigger In” configuration**: WriteTrigInCfg() routine in CLinkPort.cpp file.
4.8. Send Commands

The commands that can be sent to the camera are listed in the SendCommand() routine in the CLinkCam.cpp file.

```cpp
MV_ERROR CCLinkCamera::SendCommand(MV_COMMAND nCmdKey)
{
    switch(nCmdKey)
    {
    case MVCMD_RECORD:
        if(m_Cfg.nCamModel==MV_CM_MS_M3) CheckDACsM3();
        m_Ser.SendCmdStartStreaming(m_hSer);
        break;
    case MVCMD_STOP:
        m_Ser.SendCmdStopStreaming(m_hSer);
        break;
    case MVCMD_TRIGGER:
        m_Ser.SendCmdTrigger(m_hSer);
        break;
    case MVCMD_RESET:
        m_Ser.SendCmdReset(m_hSer);
        if(m_Cfg.nCamModel==MV_CM_MS_M5)
            m_Ser.SetM5Registers(m_hSer,m_Cfg.nFwVer,m_Cfg.nSnsModel);
        break;
    default:
        return MV_E_NOT_SUPPORTED;
    }
    return MV_SUCCESS;
}
```

**Record (Start Streaming):** before starting streaming the user should check if the camera is in overlap or non overlap mode (see next topic).

**Stop (Stop Streaming):** a specific command stops any camera streaming.

**Trigger:** send a command to issue a software trigger to the camera.

**Reset:** if the camera is M5, after the reset the user should send the timing initialization sequence.
4.9. Check Overlap and non overlap (M3 camera)

M3 camera operates in **Overlap** mode when the exposure and the acquisition period values satisfy the formula below:
\[ T_{\text{exp}} + T_{\text{ro}} > T \]

Where:
- \( T_{\text{exp}} \) = exposure time
- \( T_{\text{ro}} \) = Readout time. It's the inverse of the maximum rate (at full resolution is about 5.88 ms).
- \( T \) = period (inverse of the acquisition rate).

If the formula above is not true, the camera is in "**non-overlap**" mode. Before starting streaming, the user should check if the camera is in overlap or non overlap mode and change the value of one of the DAC voltages.

- \( \text{anDAC}[2] \) = value for non overlap mode.
- \( \text{anDAC}[10] \) = value for overlap mode.

The procedure is shown in the routine below.

```cpp
void CCLinkCamera::CheckDACsM3()
{
    unsigned long nVref3=0, *pDAC;
    unsiged long anDACs[NMAXDACS];
    int nIdx;
    if(m_Cfg.nGain==0) pDAC = m_Cfg.anDACs0;
    else if(m_Cfg.nGain==1) pDAC = m_Cfg.anDACs1;
    else if(m_Cfg.nGain==2) pDAC = m_Cfg.anDACs2;
    else if(m_Cfg.nGain==3) pDAC = m_Cfg.anDACs3;
    // DAC[2] may change if the camera is in overlap or non overlap
    if(IsOverlap())
    {
        nVref3 = pDAC[m_Cfg.nDACsCnt];
        nIdx = 1;
    }
    else
    {
        nVref3 = pDAC[2];
        nIdx = 0;
    }
    // if Vref3 has changed, send the DAC array to the camera
    if(nVref3!=m_nCurrVref3)
    {
        memcpy(anDACs,pDAC,NMAXDACS*sizeof(unsigned long));
        anDACs[2] = nVref3;
        m_Ser.WriteDACs(m_hSer,anDACs,m_Cfg.nCamModel==MV_CM_MS_M5);
        m_nCurrVref3 = nVref3;
    }
}
```
4.10. How to compute the maximum frame rate

The routine below shows how to compute the value of the minimum acquisition period (i.e. the inverse of the maximum frame rate). The minimum period is approximately equal to the read out time and it depends on some internal camera parameters (clock, RCC and extra rows) and some configuration parameters (number of rows in the ROI).

```c
unsigned long CCLinkCamera::ComputeMinPeriod()
{
    unsigned long nRows = m_Cfg.nRoiY1 - m_Cfg.nRoiYo + 1;
    double dRows = (double)(nRows + m_Cfg.nExtraRows);
    double dRCC = (double)(m_Cfg.nRCC*1000);
    double dDel = (8000000000.0/(double)m_Cfg.nClock);
    double dClk = ((double)m_Cfg.nClock)/1000000.;
    double dt; // dt is computed in ns
    if(m_Cfg.nExpMode==MV_EM_DOUBLE)
        dt = (2*dRCC*dRows)/dClk + 3*dDel;
    else
        dt = (dRCC*dRows)/dClk + 2*dDel;
    dt = (dt*100.5)/100.0; // convert to us
    return ((unsigned long)dt)/1000;
}
```
5. Appendix

5.1. Appendix A - Return Codes

The following table shows the values of the codes returned by the MV APIs. The values can be found in the MVX_API.h header file in the Include subdirectory.

<table>
<thead>
<tr>
<th>Code</th>
<th>Value</th>
<th>Notes</th>
</tr>
</thead>
<tbody>
<tr>
<td>MV_SUCCESS</td>
<td>0</td>
<td>OK – No errors</td>
</tr>
<tr>
<td>MV_E_GENERIC_ERROR</td>
<td>1</td>
<td>Generic Error</td>
</tr>
<tr>
<td>MV_E_INVALID_ARGUMENTS</td>
<td>2</td>
<td>Invalid function arguments</td>
</tr>
<tr>
<td>MV_E_INVALID_CAMERA_ID</td>
<td>3</td>
<td>Invalid camera ID used in MVOpenCamera. The ID is retrieved from the MVEnumCameras routine</td>
</tr>
<tr>
<td>MV_E_CAM_ALREADY_OPEN</td>
<td>4</td>
<td>Cannot open the camera because it’s already open.</td>
</tr>
<tr>
<td>MV_E_INVALID_HANDLE</td>
<td>5</td>
<td>Invalid MV_HANDLE camera handle</td>
</tr>
<tr>
<td>MV_E_NOT_SUPPORTED</td>
<td>6</td>
<td>The function is not supported</td>
</tr>
<tr>
<td>MV_E_INVALID_VALUE</td>
<td>7</td>
<td>Invalid parameter value</td>
</tr>
<tr>
<td>MV_E_NO_LINE</td>
<td>8</td>
<td>The image does not contain the shutter line</td>
</tr>
<tr>
<td>MV_E_NO_DATA</td>
<td>9</td>
<td>No correction data is available</td>
</tr>
<tr>
<td>MV_E_BUF_TOO_SMALL</td>
<td>10</td>
<td>The buffer is too small</td>
</tr>
</tbody>
</table>
5.2. Appendix B – Camera Info Parameters

The following table shows the values and a brief description of the parameters that can be retrieved by calling the MVGetCameraInfo routine. The numeric values of the parameters can be found in the `MVX_API.h` header file in the `Include` subdirectory.

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>MVI_CAMERA_ID</td>
<td>Camera ID (see MV_ENUMITEM structure in MVX_API.h)</td>
</tr>
<tr>
<td>MVI_CAMERA_MODEL</td>
<td>Camera Model (see MV_CAM_TYPE in MVX_API.h)</td>
</tr>
<tr>
<td>MVI_COLOR</td>
<td>The camera is color (0: monochrome – 1:color)</td>
</tr>
<tr>
<td>MVI_SERIAL</td>
<td>The camera serial number (10 digits decimal value)</td>
</tr>
<tr>
<td>MVI_REVISION</td>
<td>The camera revision</td>
</tr>
<tr>
<td>MVI_FG_TYPE</td>
<td>The Camera Link™ frame grabber model</td>
</tr>
<tr>
<td>MVI_SNS_WIDTH</td>
<td>The sensor width [pixels]</td>
</tr>
<tr>
<td>MVI_SNS_HEIGHT</td>
<td>The sensor Height [pixels]</td>
</tr>
<tr>
<td>MVI_FW_VERSION</td>
<td>The camera firmware version</td>
</tr>
<tr>
<td>MVI_CAL_FILE</td>
<td>The camera calibration file is installed</td>
</tr>
<tr>
<td>MVI_MULTI_ROI</td>
<td>The multiple ROI is supported</td>
</tr>
<tr>
<td>MVI_SNS_MODEL</td>
<td>Sensor Model (see MV_SENSOR_MODEL in MVX_API.h)</td>
</tr>
<tr>
<td>MVI_EXTRA_ROWS</td>
<td>The camera extra rows (rows that should be removed from the image)</td>
</tr>
<tr>
<td>MVI_EXTRA_COLS</td>
<td>The camera extra columns (columns that should be removed from the image)</td>
</tr>
<tr>
<td>MVI_CFA_PATTERN</td>
<td>The sensor color filter array pattern (GRBG, BGGR, RGGB or GBRG)</td>
</tr>
</tbody>
</table>
5.3. Appendix C – Camera Parameters

The following table shows the values and a brief description of the parameters that can be read and written in the camera. The numeric values of the parameters can be found in the `MVX_API.h` header file in the `Include` subdirectory.

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Rd/Wri</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>MVP_EXPOSURE</td>
<td>R/W</td>
<td>Camera exposure in microseconds [µs]</td>
</tr>
<tr>
<td>MVP_PERIOD</td>
<td>R/W</td>
<td>Acquisition period in microseconds [µs]</td>
</tr>
<tr>
<td>MVP_GAIN</td>
<td>R/W</td>
<td>Camera gain (0 to 3)</td>
</tr>
<tr>
<td>MVP_REC_MODE</td>
<td>R/W</td>
<td>Recording mode (0: normal, 1:circular)</td>
</tr>
<tr>
<td>MVP_SYNCIN_CFG</td>
<td>R/W</td>
<td>Sync In configuration (internal, ext pulse/edge - high/low)</td>
</tr>
<tr>
<td>MVP_ROI_Y_MIN</td>
<td>R/W</td>
<td>Y minimum ROI coordinate (simple ROI mode)</td>
</tr>
<tr>
<td>MVP_ROI_Y_MAX</td>
<td>R/W</td>
<td>Y maximum ROI coordinate (simple ROI mode)</td>
</tr>
<tr>
<td>MVP_ROI_MODE</td>
<td>R/W</td>
<td>ROI mode (simple or multiple)</td>
</tr>
<tr>
<td>MVP_ROI_TABLE</td>
<td>R/W</td>
<td>ROI table (multiple ROI mode)</td>
</tr>
<tr>
<td>MVP_EXP_MODE</td>
<td>R/W</td>
<td>Exposure mode (single or double)</td>
</tr>
<tr>
<td>MVP_PIXEL_GAIN</td>
<td>R/W</td>
<td>Pixel Gain (upper, middle, lower)</td>
</tr>
<tr>
<td>MVP_SHUTTER_LINE</td>
<td>R</td>
<td>The position of the shutter line (if the image does not have it, the MVGetParameter routine returns MV_E_NO_LINE)</td>
</tr>
<tr>
<td>MVP_BKG_INDEX</td>
<td>R</td>
<td>The index of the current background buffer for image correction</td>
</tr>
<tr>
<td>MVP_PSC_INDEX</td>
<td>R</td>
<td>The index of the current pixel sensitivity (PSC) buffer for image correction</td>
</tr>
<tr>
<td>MVP_TRIGIN_CFG</td>
<td>R/W</td>
<td>The trigger in configuration parameter</td>
</tr>
<tr>
<td>MVP_POST_TRIG_N</td>
<td>R/W</td>
<td>The number of frame to stream after the trigger</td>
</tr>
<tr>
<td>MVP_SYS_BAUDRATE</td>
<td>R/W</td>
<td>The serial port baud rate (system parameter)</td>
</tr>
</tbody>
</table>
5.4. Appendix D – Camera Commands

The following table shows the values and a brief description of the command that can be sent to the camera. The numeric values of the parameters can be found in the MVX_API.h header file in the Include subdirectory. “Lo Param” and “Hi Param” are ignored. They have been introduced for future use.

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Value</th>
<th>Lo Param</th>
<th>Hi Param</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>MVCMD_RECORD</td>
<td>0</td>
<td>Not used</td>
<td>Not used</td>
<td>Start recording/streaming</td>
</tr>
<tr>
<td>MVCMD_STOP</td>
<td>1</td>
<td>Not used</td>
<td>Not used</td>
<td>Stop recording/streaming</td>
</tr>
<tr>
<td>MVCMD_TRIGGER</td>
<td>2</td>
<td>Not used</td>
<td>Not used</td>
<td>Send a software trigger</td>
</tr>
<tr>
<td>MVCMD_RESET</td>
<td>3</td>
<td>Not used</td>
<td>Not used</td>
<td>Reset the camera</td>
</tr>
</tbody>
</table>
5.5. Appendix E – Data types

This appendix describes the data types defined in the MVX_API.h header file.

5.5.1. MV_BAUD_RATE

The MV_BAUD_RATE enumerates the serial port baud rates

- MV_BAUDRATE_9600: 9600 pbs.
- MV_BAUDRATE_19200: 19200 pbs.
- MV_BAUDRATE_38400: 38400 pbs.
- MV_BAUDRATE_57600: 57600 pbs.
- MV_BAUDRATE_115200: 115200 pbs.
- MV_BAUDRATE_230400: 230400 pbs.
- MV_BAUDRATE_460800: 460800 pbs.
- MV_BAUDRATE_921600: 921600 pbs.

5.5.2. MV_CAM_MODEL

The MV_CAM_MODEL type enumerates the camera models:

- MV_CM_UNKNOWN: Unknown camera model
- MV_CM_MS_M3: MotionScope M3.
- MV_CM_MS_M5: MotionScope M5.

5.5.3. MV_SNS_MODEL

The MV_CAM_MODEL type enumerates the camera models:

- MV_SM_UNKNOWN: Unknown sensor model
- MV_SM_SIRIUS: Sirius sensor (M3).
- MV_SM_ORION: Orion sensor (M5).
- MV_SM_ORION_II: Orion II sensor (M5).

5.5.4. MV_CFA_PATTERN

The MV_CFA_PATTERN type enumerates the color filter array pattern in color sensors:

- MV_CFA_GRBG: GRBG pattern.
- MV_CFA_BGGR: BGGR pattern.
- MV_CFA_RGGB: RGGB pattern.
5.5.5. MV_FG_TYPE

The MV_FG_TYPE type enumerates the frame grabber models:

- **MV_FG_COR_X64CL**: Dalsa-Coreco X64 Xcelera-CL PX4.
- **MV_FG_NIPCIE1429**: National Instruments PCIe-1429.
- **MV_FG_MATROX_S_H**: Matrox Solios/Helios.
- **MV_FG_EPIXCI_E4**: Epix PIXCI E4.
- **MV_FG_BF_KARBON**: Bitflow Karbon-CL.
- **MV_FG_SSOFT_ME4**: Silicon Software ME4.
- **MV_FG_EDT_PCIE8**: EDT PCI-E8 DV C-Link.

5.5.6. MV_REC_MODE

The MV_REC_MODE enumerates the camera record modes:

- **MV_RM_NORMAL**: normal acquisition mode.
- **MV_RM_CIRCULAR**: circular acquisition mode.

5.5.7. MV_GAIN

The MV_GAIN enumerates the camera sensor gain:

- **MV_GAIN_0**: no gain (-3 dB).
- **MV_GAIN_1**: gain 1 (0 dB).
- **MV_GAIN_2**: gain 2 (+3 dB).
- **MV_GAIN_3**: gain 3 (+6 dB).

5.5.8. MV_SYNCIN_CFG

The MV_SYNCIN_CFG enumerates the configuration of the Sync In:

- **MV_SIC_INTERNAL**: internal frame rate acquisition.
- **MV_SIC_EXT_EDGE_HI**: external, exposure starts on edge, active high.
- **MV_SIC_EXT_EDGE_LO**: external, exposure starts on edge, active low.
- **MV_SIC_EXT_PULSE_HI**: external, exposure integrated over pulse, active high.
- **MV_SIC_EXT_PULSE_LO**: external, exposure integrated over pulse, active low.
5.5.9. MV_TRIGIN_CFG

The MV_TRIGIN_CFG enumerates the configuration of the Trigger In:

- **MV_TIC_EDGE_HI**: the trigger starts on edge, active high.
- **MV_TIC_EDGE_LO**: the trigger starts on edge, active low.
- **MV_TIC_SWC**: the trigger starts on Switch Closure.
- **MV_TIC_DISABLE**: the trigger is disabled.

5.5.10. MV_ROI_MODE

The MV_ROI_MODE enumerates the ROI modes:

- **MV_ROI_SIMPLE**: simple ROI mode.
- **MV_ROI_MULTI**: multiple ROI mode.

5.5.11. MV_EXP_MODE

The MV_EXP_MODE enumerates the exposure modes:

- **MV_EM_SINGLE**: single exposure.
- **MV_EM_DOUBLE**: double exposure.

5.5.12. MV_PIXEL_GAIN

The MV_PIXEL_GAIN enumerates the camera pixel gain:

- **MV_PG_UPPER**: upper bits.
- **MV_PG_MIDDLE**: middle bits.
- **MV_PG_LOWER**: lower bits.
5.5.13. **MV_ERROR**

The MV_ERROR enumerates the return codes. See Appendix A.

5.5.14. **MV_INFO**

The MV_INFO enumerates the camera information index. See Appendix B.

5.5.15. **MV_PARAM**

The MV_PARAM enumerates the camera parameters. See Appendix C.

5.5.16. **MV_COMMAND**

The MV_COMMAND enumerates the camera commands. See Appendix D.
5.6. Appendix F – Structures

This appendix describes the structures defined in the MVX_API.h header file.

5.6.1. MV_ENUMITEM

The MV_ENUMITEM structure contains information about a camera. It must be used in the camera enumeration procedure with the MVEnumCameras routine.

```c
typedef struct
{
    unsigned long cbSize;
    unsigned long bIsOpen;
    unsigned long nCamID;
    unsigned long nCamModel;
    unsigned long nSnsModel;
    unsigned long nColor;
    unsigned long nSerial;
    unsigned long nRevision;
    unsigned long nFGType;
    unsigned long nSnsWid;
    unsigned long nSnsHgt;
    unsigned long nFwVersion;
    unsigned long nCPAPattern;
    char szPortID[L_PORT_ID];
} MV_ENUMITEM, *PMV_ENUMITEM;
```

**Members**

*cbSize*

It specifies the size of the MV_ENUMITEM structure.

*bIsOpen*

It specifies whether the camera is currently open or not.

*nCamID*

It specifies the ID which identifies the camera. The user must use this camera ID to open the camera with MVOpenCamera.

*nCamModel*

It specifies the camera model (M3 or M5).

*nSnsModel*

It specifies the sensor model (Sirius, Orion, and Orion II).

*nColor*

It specifies whether the camera is color or not.

*nSerial*
It specifies the camera serial number (10 decimal digits value).

nRevision
It specifies the camera revision number.

nFGType
It specifies the Camera Link Frame Grabber model.

nSnsWid
It specifies the sensor width in pixels.

nSnsHgt
It specifies the sensor height in pixels.

nFwVersion
It specifies the camera firmware version.

nCFAPattern
It specifies the color filter array pattern (color sensors only).

szPortID
It specifies the serial port identifier string.
5.7. Appendix G – Camera Link Directory

The camera link directory is the folder that contains the camera link serial dynamic linking libraries.

The path to the directory is specified in the following registry key:

**HKEY_LOCAL_MACHINE\Software\CameraLink**

The path is specified as a value named “**CLSerialPath**” with type string REG_SZ.

Each frame grabber manufacturer should install its own camera link serial DLL in the path specified by the value above. If the above key does not exist, they should create it and add the directory value.