CMIRIS2 SDK User Manual

Version 2.1.1 for ARM Linux (Linaro Ubuntu 12.04)
for EMA-30 and ECU-30 Products

Sep 2014

CMITech Company, Ltd.
#904, #905, K-Center, 25, Simin-daero 248beon-gil, Dongan-gu, Anyang-si, Gyeonggi-do, 431-815 Rep.of KOREA
Tel: +82.70.8633.8277
Fax: +82.31.624.4490
Contact: sales@cmi-tech.com

CMITech America, Inc.
2033 Gateway Place, Suite 500
San Jose, CA 95110 USA
Tel: (1) 408 573-6930

Table of Contents

1. History of Changes from Prior Versions of SDK's 3

2. Introduction 5

3. Mainboard and OS Description 6

4. CMIRIS SDK Implementation Guide 7

4.1. Create cmirisLib 7

4.2. Get the initially attached devices 8

4.3. Open the device 8

4.4. Image preview, RGB indicator and image captured events 9

4.5. Start capture 10

4.6. "Either Eye" capture mode 10

4.7. Auto capture 11

4.8. Stop Capture 12

4.9. Capture parameters 12

4.10. Face color image conversion with libcmicolorLib_arm_0.9.0.so 12

5. Reference Manual of Values, Definitions and API Functions of libcmirisLib2_arm_2.1.1.so 12

5.1. Defined Return Values 12

5.2. Type Definitions 13

5.3. API Functions 16

6. MIRLIN Functions in cmirisLib2 30

7. Testing the external I/O interface of EMA-30 or ECU-30 32

1. History of Changes from Prior Versions of SDK's

- Version 2.0.3 was the initial public release, although units shipped under CMITech's EMA and ECU-30 beta program had earlier version of the operating libraries. All EMA-30 and ECU-30 units shipped before July 16, 2014 are supplied with the older version than v2.1.0.

Changes in V2.0.3

- There was a bug in ARM Linux cmirisLib2 library that made EMA / EMB units freeze during capture process. This bug has been temporarily fixed and it will require a firmware upgrade for fundamental improvement. All new production units shipped since May 1, 2014 have the current version of firmware for the EMA and EMB, which is 1.2.16.
- In the released SDK, all folders have been moved under /usr/local/. Please copy all the folders into /usr/local/ and set file's attribute appropriately. cmirisLib_arm_1.1.3.so without MIRLIN algorithm is also released which is used by EMXDemo.
- Known bug: In EMXDemo, USB removal is handled correctly but it is not in CMIMIRDemo. This bug is under investigation.
- Note: "dmesg" will show the kernel message "usb 1-2.1.2: usbfs: usb_submit_urb returned -121" after the capture process. This message is not error but USB debug message and will be off in the release version of Linux.
- Note: "libcmirisLib2_arm_2.0.3.so" was linked with /lib/libusb-1.0.so.2 (libusb-1.0.so.2.0.0) which must reside in the folder /lib. There is another libusb-1.0.so in the folder /lib/arm-linux-gnueabihf/libusb-1.0.so.0 (libusb-1.0.so.0.1.0) which is the older version.

Changes in V2.1.0

- Most bugs were fixed including lockup problem during operation. The new cmirisLib2 library will survive with the frequent USB removal and arrival actions. But in artificial testing, this may cause the USB driver to crash in ECU-30 which cannot be recovered without system reboot. In order to recover the system automatically, "emaUsbMonitor" for EMA-30 and "ecuUsbMonitor" for ECU-30 will be also provided, which will monitor USB status. If something goes wrong in USB driver, it will reboot the EMA-30 or ECU-30. The "emaUsbMonitor"should be found in the folder /usr/local/bin and "/usr/local/bin/emaUsbMonitor >> usbMonitorLog start" should be added at the last line of /etc/rc.local to make it run as a daemon. It should be noted that "emaUsbMonitor" is also necessary for EMA-30, even though there is no chance for USB cable disconnections.
- Face tracking performance on EMA-30 and EMB-30 has been improved.
- Capture process has been optimized for faster performance.
- libcmirisLib_arm_1.x.xx.so without MIRLIN Algorithm will not be released any more, since libcmirisLib2_arm_2.x.xx.so is libcmirisLib_arm_1.x.xx.so plus the MIRLIN Algorithm.
- Move up/down and press/release events are added in Device Manager Event.

- Auto adjustment of target intensity was added to avoid sclera saturation in small percentage of subjects, thereby improving FTC rate.
- The current indicator color is now available in the field of leftPosition or rightPosition in “CMI_IMAGE_INFO” structure for additional feedback to user.
- Rare case of software failure when motor is opened was fixed.
- Rare case of software failure (lockup) for “cmi_closeDevice()” problem was fixed.
- Other minor bugs were fixed.
- Positioning color indication (blue, green, red) was significantly improved to be more responsive and eliminate occasional error in indicator when subject was too close.
- Iris image margins were set as specified in ISO/IEC 19794-6:2011(E), so that the capture area (X and Y dimensions, not Z) was increased.
- Two voice cues were implemented: "Please come closer" and "Please move back". To support voice cues, the new event “CMI_EVENT_INDICATOR” was added, which notifies the indicator color change (blue or red) and the moment when the voice cue should be played. Voice cue sound files are .wav files that can be customized by customer for language or other preferences.
- Either Eye mode was added. In this mode, device tries to capture both eyes first and if the number of frames elapsed after one eye is captured exceeds “maxFrameDiffinEitherEyeMode”, then device just returns image from only one eye. “maxFrameDiffinEitherEyeMode” can have the values from 3 to 9, referring to the number of frames.Rare case of failure in motor open was fixed.
- The new option whether to return to the initial motor position after successful capture was added. It can be enabled or disabled using cmi_setMotorReturnToInitEnabled() before cmi_startCapture() is called.
- CMIMIRDemo will now allow either eye enrollment and recognition.

Changes in V2.1.1

EMX Family:

- The depth of field is now adjustable with cmi_setDepthOfField().
- The center of depth of field is 330mm from the front surface of EMX.
- The minimum and default of each depth of field is 15mm, so total depth of field is 30mm. This is the depth of field for enrollment.
- The maximum is 35mm for inside and 25mm for outside, total maximum depth of field is 60mm, which is for recognition for physical access control application.

2. Introduction

This CMIRIS SDK for Linux user manual describes the SDK for CMITECH's EMA-30 and ECU-30 products, which are intended for physical access control applications. These products share the same embedded mainboard that runs the Linaro Ubuntu 12.04 operating system and utilize CMITech's Linux operating libraries.

The core of CMIRIS SDK is the Iris image capture library, "libcmirisLib2_arm_2.1.1.so". This main operating library was compiled from the Windows version of the same library, and is intended to have the exact same core functionality in image capture and control functions as the Windows version.

The EMA-30 and ECU-30 also have the option for embedded iris image encoding and matching functions based on the Smart Sensors, Ltd. MIRLIN algorithm. This functionality is embedded into the libcmirisLib2_arm_2.1.1.so library, but it is enabled by a switch in the firmware of the EMA-30 or EMB-30 (which would be attached to the ECU-30) cameras. This switch is set at product setup time in the CMITech production operation.

There are only 2 other operating libraries supplied with the EMA-30 and ECU-30 and one of them is "libemaLib_arm_1.0.1.so". This library enables you to control all the I/O functions that EMA-30 or ECU-30 has, such as Wiegand I/O, GPIO, Relay and RS485. The test application "emaTest" and its source code, provided in the folder /usr/local/bin and /usr/local/src/CMITECH respectively, show how to use this library. The detailed description on "emaTest" application can be found in "/usr/local/doc/EMA_EMB_ECU_Quick_User_Guide_V2.1.1.pdf".

The other library is "libcmicolorLib_arm_0.9.0.so" and it manages the color face imaging function, which converts Bayer raw color images to RGB images. However, the color face image is not supported in Linux OS at this moment and this library is included for compatibility with Windows version.

Another demo application loaded onto the EMA-30 and ECU-30 is "CMIMIRDemo" which shows each functions of libcmirisLib2_arm_2.1.1.so library. The simple description on this demo application can be found in the quick user guide document addressed above and the source code of this demo can be found in the folder /usr/local/src/CMITECH.

Also please note that the BMT-20 product in principal compatible with the Linux OS, but this release (v2.1.1) does not cover the BMT-20 product. Please contact CMITech for more information.

CMIRIS2 SDK distribution contains the following directories and files:

/usr/local

/doc - contains all the documents for the CMIRIS2 Linux SDK.

/bin - contains executable files for demo application as “CMIMIRDemo” and “emaTest”. In addition, “emaUsbMonitor” for EMA-30 and “ecuUsbMonitor” for ECU-30 are also provided, which will monitor USB status. If something goes wrong in USB driver, it will make the EMA-30 or ECU-30 to reboot. In order to make it run as a daemon “/usr/local/bin/emaUsbMonitor >> usbMonitorLog start” should be added at the last line of /etc/rc.local. It should be noted that “emaUsbMonitor” is also necessary for EMA-30, even though there is no chance for USB cable disconnections.

/include – contains header files including cmiglobal.h, cmirislib2.h, emaglobal.h, emalib.h and cmicolorlib.h. The suffix “2” denotes the functionality for the embedded MIRLIN encoding and matching.

/lib – contains the 3 operating libraries, libcmirisLib2_arm_2.1.1.so, libcmicolorLib_arm_0.9.0.so, and libemaLib_arm_1.0.1.so.

/share - the folder “CMITECH” under /share contains the haarcascades folder, the .wav files for vocalizations. The CMIMIRDemo application data base “Enroll.db” will be created if it does not exist. To reset the database, the existing “Enroll.db” can be just deleted.

/src – the “CMITECH” folder contains the source code for the Linux demo application to serve as sample code.

3. Mainboard and OS Description

The CMITech operating libraries as described in this SDK have been implemented on a custom designed mainboard featuring the Samsung Exynos 4412 quad-core CPU / SOC with 2 GB of RAM and 16 GB of flash memory, and the Linaro Ubuntu 12.04 OS. This mainboard also includes all physical connectors necessary for a fully functional physical access control system, including dual Wiegand, RS 232 and 485 serial ports, GPIO and high current relay connectors.

For implementation of this SDK on a different hardware platform, please contact CMITech directly.

It is the company’s policy at this time to support only the Linaro Ubuntu 12.04 OS distribution.

4. CMIRIS SDK Implementation Guide

The CMIMIRDemo application shows most of the features that the CMIRIS SDK has. Please refer to the source code in /usr/local/src/CMITECH folder for details.

In general, because the operating library for the Linaro Ubuntu SDK is virtually identical in functionality to the equivalent in the CMITech Windows SDK, the Implementation Process, Defined Returned Values, and Type Definitions descriptions are the same.

However, there are additional functions and calls for the added functionality of the embedded MIRLIN encoding and matching functions. Please refer to section 6. MIRLIN Functions in cmirisLib2 in this document.

Recommended SDK Implementation Process Steps

4.1. Create cmirisLib

```
#include "cmirislib2.h"

if (cmi_createCMIris(CMI_DMX_EMX_MODEL, &m_cmiHandle) == CMI_SUCCESS) {
     m_dmEventThread = new DMEventThread(NULL, m_cmiHandle);
     if (m_dmEventThread) {
        connect(m_dmEventThread,  SIGNAL(deviceRemoved(CMI_DEVICE_INFO  *)),
              this, SLOT(doDeviceRemoved(CMI_DEVICE_INFO *)));
         m_dmEventThread->start();
         …
     }

     m_eventThread = new EventThread(NULL, m_cmiHandle);
     if (m_eventThread) {
         connect(m_eventThread, SIGNAL(preview(CMI_IMAGE_INFO *)),
                   this, SLOT(doPreview(CMI_IMAGE_INFO *)));
         …
     }
}
```

First of all, `cmi_createCMIris(CMI_DMX_EMX_MODEL, &m_cmiHandle)` must be called to create the cmirisLib instance and create the handle for it. For DMX-30 and EMX-30 devices, the first argument should be `CMI_DMX_EMX_MODEL`, and for the BMT-20 it should be `CMI_BMT_MODEL`. Once cmirisLib instance is created, the device manager will run inside to detect device removal, arrival, power suspend and resume. The devices which have already been connected with the call, `cmi_createCMIris`(), will be available as the device arrival events happen. These events can be read by `cmi_readDMEvent(CMI_HANDLE handle, CMI_DM_EVENT *event, DWORD timeout)` in the separate thread loop just after the function call `cmi_createCMIris`(). Please refer to dmeventthread.cpp for reading device

manager event, `CMI_DM_EVENT`, as defined in cmiglobal.h.

4.2. Get the initially attached devices

After the successful creation of cmirisLib, it is necessary to read the device manager event to find the currently attached devices. In the EMXDemo application, this initial set up is handled in `doDeviceArrived(CMI_DEVICE_INFO *deviceInfo)`, which is the "slot" and will be called whenever `cmi_readDMEvent()` receives the device arrival event. "Signal" and "Slot" are used for communication between objects in Qt and the detailed information can be found in the following link; http://doc.qt.nokia.com/4.7-snapshot/signalsandslots.html.

```
m_deviceInfoList << deviceInfo;
ui->comboBox_SerialNumbers->clear();
int index = 0;
for (i = 0; i < m_deviceInfoList.size(); i++) {
    ui->comboBox_SerialNumbers->addItem(QString(m_deviceInfoList.at(i)-
>serialNumber));
    if (QString(m_deviceInfoList.at(i)->serialNumber) == comboSN) {
        index = i;
    }
}
ui->comboBox_SerialNumbers->setCurrentIndex(index);
```

In the above sample code, each arrived deviceinfo is stored in m_deviceInfoList and the serial number of device will be shown in the combo box menu for user's selection. The device arrival and removal event during operation will be handled in `doDeviceArrived()` and `doDeviceRemoved()`.

4.3. Open the device

```
int ret = cmi_openDevice(m_cmiHandle, dvInfo);
```

`cmi_openDevice()` requires the `CMI_DEVICE_INFO` pointer in the second argument found in the device arrival event. If multiple devices are detected, one of devices should be selected by application. If one device is opened already, it will return `CMI_ERROR_DEVICE_OPENED` error when another device is attempted to be open. `CMI_DEVICE_INFO` has a following structure.

```
typedef struct _CMI_DEVICE_INFO {
     int cbSize;                                  // size of this struct
     char modelName[CMI_MODEL_NAME_MAX_SIZE+1];  // BMT-20
     char serialNumber[CMI_SERIAL_NUMBER_MAX_SIZE+1]; //AA1105AA000128
     char hardwareRev[CMI_HARDWARE_REV_MAX_SIZE+1];   // 1.0.1
     char firmwareRev[CMI_FIRMWARE_REV_MAX_SIZE+1];   // 1.0.86
} CMI_DEVICE_INFO;
```

4.4. Image preview, RGB indicator and image captured events

```
m_eventThread->start();
```

After the successful opening of the device and before starting capture, the thread loop reading image preview events and image captured events should be started. The function `cmi_readEvent(m_cmiHandle, &event, 100)` inside the thread loop in eventthread.cpp will read these events and send them to main GUI through "emit". Each preview event will contain the image and image information, and the image uses the memory already allocated by application. Please see the section 3.5 below how to allocate the memory buffer for the images in these events.

After the application finishes, in order to use the image that comes with the preview event (after it displays the preview image on the monitor), the memory should be released by the function `cmi_releasePreviewBuffer(m_cmiHandle, image)`. Otherwise, the successive preview events cannot use this particular buffer area for the next preview events. If the buffer is not available on time, the preview event will contain the image information only, but not image itself. It should be noted that each preview event contains only one image. Please refer to the function `displayLiveImage (CMI_IMAGE_INFO *imageInfo)` in mainwindow.cpp.

For the events of `CMI_EVENT_IRIS_IMAGE_UNSELECTED` and `CMI_EVENT_IRIS_IMAGES_SELECTED`, the image buffer doesn't need to be released, since these buffer areas are dedicated to the captured images and not reused.

With Linux SDK Version 2.1.1, there is a new event type `CMI_EVENT_INDICATOR`, which is introduced to implement vocal positioning cues of "move closer" and "move back" to the subject. This event notifies when the RGB indicator color was changed and when to play vocal cues. When this event occurs, `leftPosition` and `rightPosition` in `imageInfo` structure contains the RGB color information (`CMI_INDICATOR_COLOR_OFF`, `CMI_INDICATOR_COLOR_RED`, …) and which vocal cue (`CMI_VOICE_COME_CLOSER`, `CMI_VOICE_MOVE_BACK`) should be played. No image will be returned in this event, so `leftImage` and `rightImage` in `imageInfo` structure should be NULL and `cmi_releasePreviewBuffer(CMI_HANDLE handle, unsigned char *buffer)` should not be called. The event values are defined in `cmiglobal.h`.

In order to utilize this `CMI_EVENT_INDICATOR`, you need to enable this event using the function `cmi_setIndicatorEventEnabled(CMI_HANDLE handle, int enabled)`, since the default is disabled for backward compatibility. The time interval of vocal cues can be adjusted with the function `cmi_setVoiceCueTimeInterval(CMI_HANDLE handle, int firstTimeInterval_frames, int nextTimeInterval_frames)`, where the time interval is in the unit of frame number, i.e. 33ms. The default value of `firstTimeInterval_frames` is 75 and 90 for `nextTimeInterval_frames`.

`firstTimeInterval_frames` is the first vocal cue time interval and `nextTimeInterval_frames` will determine the time interval thereafter.

4.5. Start capture

```
if (!m_applBuffer) {
     cmi_getBufferSize(m_cmiHandle, 10, MAX_UNSELECTED_RECT,
     &m_applBufferSize);
      m_applBuffer = new unsigned char [m_applBufferSize];
}
int ret = cmi_startCapture(m_cmiHandle, m_applBufferSize, m_applBuffer);
```

Before starting the capture process, the application should allocate buffer space to store the preview and captured images. To find the buffer size for these images, the function `cmi_getBufferSize()` should be called before the function `cmi_startCapture()`. The function `cmi_getBufferSize()` calculates the buffer size using the number of preview images and the number of unselected images, and will set the necessary internal parameters. After the function `cmi_openDevice()` is called, the function `cmi_getBufferSize()` must be called at least once. As for BMT-20, `cmi_setBMTMode`() should be called before `cmi_getBufferSize()`, since CMI_BMT_USER_MODE providing the full resolution preview images requires more buffer size.

The “unselected” images are the captured images within the operating range that were not selected as two (left and right irises) final captured images by the CMIRIS SDK capture algorithm. These unselected images are provided to be used by the application in the case that the CMIRIS SDK capture algorithm has not picked the best images. These unselected images are only available in DMX-10 and EMX-10, but not in BMT-20. In the EMXDemo application, these unselected images will be also saved when the save button is clicked. For faster capture speed, please set the argument “`maxNumUnselectedImages`” to zero.

4.6. “Either Eye” capture mode

In Linux SDK Version 2.1.1, “Either Eye” capture mode is introduced to allow subjects with only one eye to be imaged. This is in addition to “Both”, “Left”, and “Right” eye capture modes. In the “Either Eye” mode, the device tries to capture both eyes for a certain period of time. But if it is able to capture only one eye during this period, it just returns the single captured eye with a face image.

The waiting time interval to capture the other eye once the first eye is captured is determined with the value of `maxFrameDiffinEitherEyeMode` in the function `cmi_setMaxFrameDiff InEitherEyeMode(CMI_HANDLE handle, int maxFrameDiffinEitherEyeMode)`. The minimum value of this parameter is 3 which mean that the captured left and right image frame difference allowed for both eye capture is 3, and if not, the only one eye captured is returned. The maximum value is set to 9, which is

the slowest option.

4.7. Auto capture

After starting the capture process, a preview event will not be issued until a subject is detected by the device. The preview event is the type of `CMI_EVENT` with the event value `CMI_EVENT_PREVIEW_FRAME_INFO`.

```
typedef struct _CMI_EVENT {
     int cbSize;
     int event;
     CMI_IMAGE_INFO imageInfo;
} CMI_EVENT;
```

`CMI_IMAGE_INFO` contains the left iris, right iris, and face image with respective image information. Please refer to `cmiglobal.h` for the definition of `CMI_IMAGE_INFO`. As for the iris images, only one of the left and right iris image is alternatively available for each preview event. If there is no preview image buffer available which the application allocated, the image pointer will be `NULL` and the image width and height are 0's. Please make sure the used image buffer is released by the function call `cmi_releasePreviewBuffer()`. If the image or its information is not available, the value of each element will be one of `NULL`, `0, -1, CMI_UNKNOWN, or CMI_EYE_UNKNOWN`.

If the CMIRIS SDK capture algorithm finds both iris images which satisfy the given requirements, such as maximum XY and Z movement restrictions, it will send the `CMI_EVENT_CAPTURE_DONE` event, so that the application can provide an audible or visible feedback to the subject. In this event, no image and image information will be provided in `imageInfo`. XY plane is defined as the plane parallel to the front surface of the device and Z direction is perpendicular to the front surface.

Right after the `CMI_EVENT_CAPTURE_DONE` event is sent to the application, the `CMI_EVENT_IRIS_IMAGE_UNSELECTED` and `CMI_EVENT_IRIS_IMAGES_SELECTED` will be followed successively. If `maxNumUnselectedImages` in `cmi_getBufferSize()` is set to 0, no `CMI_EVENT_IRIS_IMAGE_UNSELECTED` event will occur.

The `CMI_EVENT_IRIS_IMAGES_SELECTED` event contains the best left and right iris images selected by the CMIRIS SDK capture algorithm with image information. In addition to the iris images, the face image (360 x 480) captured just after 200 ~ 250 ms later will be stored in the "`faceImage`" pointer of `CMI_IMAGE_INFO`. The distance, XY-movement, Z-movement, inter-pupillary distance(IPD), iris center and radius in pixels will be provided. The unit of all lengths in `CMI_IMAGE_INFO` is μ(micron) to avoid the floating value. The `rollAngle` and `rollAngleUncertainty` have not been implemented in this version of SDK.

4.8. Stop Capture

```
ret = cmi_stopCapture(m_cmiHandle);
```

This function will wait until all the images from the device are received and then clean the entire remaining preview and image captured events before it returns.

4.9. Capture parameters

Please refer to 4.3. API Functions for optional setting parameters.

4.10. Face color image conversion with libcmicolorLib_arm_0.9.0.so

At this time, color face is only supported in Windows OS, not in Linux OS. That is, it is not supported in either the EMA-30 or the EMB-30/ECU-30 combination. In the future, the face color image will be supported by the libcmicolorLib_arm_0.9.0.so library in Linux OS.

5. Reference Manual of Values, Definitions and API Functions of libcmirisLib2_arm_2.1.1.so

Please refer to "`cmiglobal.h`" and "`cmirislib2.h`" for the detailed information.

5.1. Defined Return Values

```
#define CMI_SUCCESS                              0
#define CMI_ERROR_WAIT_TIMEOUT                  -1
#define CMI_ERROR_READ_EVENT_CANCELLED          -2
#define CMI_ERROR_INVALID_HANDLE                -3
#define CMI_ERROR_INVALID_MODEL                 -4
#define CMI_ERROR_FAIL_TO_SEND_COMMAND          -5
#define CMI_ERROR_INVALID_BUFFER                -6
#define CMI_ERROR_CALL_GET_BUFFER_SIZE_FIRST    -7
#define CMI_ERROR_BUFFER_SIZE_TOO_SMALL         -8
#define CMI_ERROR_CANNOT_FIND_DEVICE            -9
#define CMI_ERROR_DEVICE_OPENED                 -10
#define CMI_ERROR_DEVICE_CLOSED                 -11
#define CMI_ERROR_DEVICE_STARTED                -12
```

```
#define CMI_ERROR_DEVICE_STOPPED                -13
#define CMI_ERROR_DEVICE_UPSIDE_DOWN            -14
#define CMI_ERROR_IN_ARGUMENTS                  -15
#define CMI_ERROR_FAIL_TO_OPEN_IMAGER_DEVICE    -16
#define CMI_ERROR_EEPROM_READ_TIMEOUT           -17
#define CMI_ERROR_FAIL_TO_OPEN_MOTOR_DEVICE     -18
#define CMI_ERROR_EEPROM_VERSION_INVALID        -19
#define CMI_ERROR_CANNOT_FIND_HAARCASCADE_FILES -20
#define CMI_ERROR_CANNOT_ALLOC_MEMORY           -21
#define CMI_ERROR_INVALID_MODE                  -22
#define CMI_ERROR_UDEV_FAILED                   -23
#define CMI_ERROR_BEGINTHREAD_FAILED            -24
```

5.2. Type Definitions

CMI_DEVICE_INFO

Definition

```
typedef struct _CMI_DEVICE_INFO {
 int cbSize;
 char modelName[CMI_MODEL_NAME_MAX_SIZE+1];
 char serialNumber[CMI_SERIAL_NUMBER_MAX_SIZE+1];
 char hardwareRev[CMI_HARDWARE_REV_MAX_SIZE+1];
 char firmwareRev[CMI_FIRMWARE_REV_MAX_SIZE+1];
} CMI_DEVICE_INFO;
```

Members

`cbSize`
: The size of the CMI_DEVICE_INFO structure

`modelName[CMI_MODEL_NAME_MAX_SIZE+1]`
: String of Model Name

`serialNumber[CMI_SERIAL_NUMBER_MAX_SIZE+1]`
: String of Serial Number

`hardwareRev[CMI_HARDWARE_REV_MAX_SIZE+1]`
: String of Hardware Revision

`firmwareRev[CMI_FIRMWARE_REV_MAX_SIZE+1]`
: String of Firmware Revision

CMI_IMAGE_INFO

Definition

```
typedef struct _CMI_IMAGE_INFO {
 int cbSize;
 unsigned char *leftImage, *rightImage, *faceImage;
```

```
 int leftWidth, rightWidth, faceWidth;
 int leftHeight, rightHeight, faceHeight;
 int leftResolution, rightResolution, faceResolution;
 int leftIrisCenterX, rightIrisCenterX;
 int leftIrisCenterY, rightIrisCenterY;
 int leftIrisRadius, rightIrisRadius;
 int leftPupilRadius, rightPupilRadius;
 int leftExposedIrisArea, rightExposedIrisArea;
 unsigned char leftIrisMargin;
 unsigned char rightIrisMargin;
 int leftLiveness, rightLiveness;
 int leftDistance;
 int rightDistance;
 int leftPosition;
 int rightPosition;
 int leftXYMovement;
 int rightXYMovement;
 int leftZMovement;
 int rightZMovement;
 int ipd;
 int rollAngle;
 int rollAngleUncertainty;
 int doesLeftLookFront;
 int doesRightLookFront;
 int isLeftImageQualityOK;
 int isRightImageQualityOK;
 int isDeviceUpSideDown;
 int isDeviceStablized;
 int avgProcessingTime;
} CMI_IMAGE_INFO;
```

Members

`cbSize`

The size of the CMI_DEVICE_INFO structure.

`*leftImage`

`*rightImage`

`*faceImage`

`leftWidth`

`rightWidth`

`faceWidth`

`leftHeight`

`rightHeight`

`faceHeight`

`leftResolution, rightResolution, faceResolution`

Resolution is sub sampling factor

full resolution: 1, half resolution: 2, quad resolution: 4, ...

`leftIrisCenterX`

`rightIrisCenterX`

`leftIrisCenterY`

`rightIrisCenterY`

`leftIrisRadius, rightIrisRadius`
Pixels
`leftPupilRadius, rightPupilRadius`
Pixels - TBI(To Be Implemented)
`leftExposedIrisArea, rightExposedIrisArea`
percentage - TBI
`leftIrisMargin`
enough iris margins? see enum IrisMargin
`rightIrisMargin`
enough iris margins? see enum IrisMargin
`leftLiveness, rightLiveness`
Live eye detection result - TBI
`leftDistance`
Distance(unit: micron) of the left eye from the front of device.
`rightDistance`
Distance(unit: micron) of the right eye from the front of device.
`leftPosition`
Estimated left-eye position in BMT. Indicator LED color in EMX Family.
`rightPosition`
Estimated right eye position in BMT. Indicator LED color in EMX Family.
`leftXYMovement, rightXYMovement`
XY Movement of Eye in micron during dt seconds
XY-plane is perpendicular to the optical axis of camera
`leftZMovement, rightZMovement`
Z Movement of Eye in micron during dt seconds. .
+Z-direction is from the device to the eye.
`ipd`
Inter-Pupillary Distance in microns.
`rollAngle`
Roll angle of a line drawn between the centers of the left and right irises. Unit is 1/10 degree and it is positive if the left iris is higher than right iris. (counter-clockwise) TBI
`rollAngleUncertainty`
Roll angle uncertainty in 1/10 degrees. TBI
`doesLeftLookFront`
Does left eye look front?
`doesRightLookFront`
Does right eye look front?
`isLeftImageQualityOK, isRightImageQualityOK`
Is left/right iris image quality OK?
`isDeviceUpSideDown`
Is device upside down? BMT only
`isDeviceStablized`
Is device stabilized - TBI
`avgProcessingTime`
average image processing time in ms for each captured frame.

CMI_EVENT

```
typedef struct _CMI_EVENT {
 int cbSize;
 int event;
```

```
 CMI_IMAGE_INFO imageInfo;
} CMI_EVENT;
```

Members

`cbSize`

The size of the CMI_EVENT structure

`event`

`imageInfo`

CMI_DM_EVENT

```
typedef struct _CMI_DM_EVENT{
 int cbSize;
 int event;
 CMI_IMAGE_INFO deviceInfo;
} CMI_DM_EVENT;
```

Members

`cbSize`

The size of the CMI_DM_EVENT structure event

`deviceInfo`

5.3. API Functions

1. cmi_getCMIrisVersion

Syntax

```
void cmi_getCMIrisVersion(int *major, int *minor, int *revision);
```

Parameters

major[out]
minor[out]
revision[out]

Return Value

None

Remarks

Get cmirisLib version and revision number for compatibility.
Application should check if major version number is consistent.

2. cmi_createCMIris

Syntax

```
int cmi_createCMIris(int model, CMI_HANDLE *phandle);
```

Parameters

model[in] - CMI_DMX_EMX_MODEL or CMI_BMT_MODEL

phandle[out] - CMI_HANDLE value if successful. Otherwise, NULL.

Return Value
CMI_ERROR_CANNOT_ALLOC_MEMORY
CMI_ERROR_INVALID_MODEL - if model is invalid. Only CMI_DMX_EMX_MODEL and CMI_BMT_MODEL are allowed at this moment
CMI_SUCCESS

Remarks
Create CMIris. As soon as it is created, it can receive device manager event
Please refer to cmi_readDMEvent()

3. cmi_destroyCMIris

Syntax
```
int cmi_destroyCMIris(CMI_HANDLE handle);
```

Parameters
handle[in] - CMI_HANDLE value

Return Value
CMI_ERROR_INVALID_HANDLE
CMI_SUCCESS

Remarks
Destroy CMIris.

4. cmi_readDMEvent

Syntax
```
int cmi_readDMEvent(   CMI_HANDLE handle,
CMI_DM_EVENT *event,
DWORD timeout);
```

Parameters
handle[in] - CMI_HANDLE value
event[out] - event occurred
timeout[in] - time out in milliseconds

Return Value
CMI_ERROR_INVALID_HANDLE
CMI_ERROR_READ_EVENT_CANCELLED: canceled pending readDMEvent
CMI_ERROR_WAIT_TIMEOUT: timeout
CMI_SUCCESS: event occurred

Remarks
Read Device Manager event. DEVICE_ARRIVAL, DEVICE_REMOVAL, POWER_SUSPEND, POWER_RESUMED
Device Manager event will be available right after cmi_createCMIris() is called until the destroy call cmi_destroyCMIris().

5. cmi_clearDMEventQueue

Syntax
```
int cmi_clearDMEventQueue(CMI_HANDLE handle);
```

Parameters
handle[in] - CMI_HANDLE value

Return Value
CMI_ERROR_INVALID_HANDLE
CMI_SUCCESS

Remarks
Clear DMEvent queue

6. cmi_cancelPendingReadDMEvent

Syntax
```
int cmi_cancelPendingReadDMEvent(CMI_HANDLE handle);
```

Parameters
handle[in] - CMI_HANDLE value

Return Value
CMI_ERROR_INVALID_HANDLE
CMI_SUCCESS

Remarks
Cancel pending cmi_readDMEvent()

7. cmi_readEvent

Syntax
```
int cmi_readEvent(CMI_HANDLE handle, CMI_EVENT *event, DWORD timeout);
```

Parameters
handle[in] - CMI_HANDLE value
event[out] - event occurred
timeout[in] - time out in milliseconds

Return Value
CMI_ERROR_INVALID_HANDLE
CMI_ERROR_READ_EVENT_CANCELLED: canceled pending readEvent
CMI_ERROR_WAIT_TIMEOUT: timeout
CMI_SUCCESS: event occurred

Remarks

Read event. CMI_EVENT_...

8. cmi_clearEventQueue

Syntax
```
int cmi_clearEventQueue(CMI_HANDLE handle);
```

Parameters
handle[in] - CMI_HANDLE value

Return Value
CMI_ERROR_INVALID_HANDLE
CMI_SUCCESS

Remarks
Clear DMEvent queue

9. cmi_cancelPendingReadEvent

Syntax
```
int cmi_cancelPendingReadEvent(CMI_HANDLE handle);
```

Parameters
handle[in] - CMI_HANDLE value

Return Value
CMI_ERROR_INVALID_HANDLE
CMI_SUCCESS

Remarks
Cancel pending cmi_readDMEvent()

10. cmi_openDevice

Syntax
```
cmi_openDevice(CMI_HANDLE handle, CMI_DEVICE_INFO *pdeviceInfo);
```

Parameters
Open the specified device by pdeviceInfo.
Only one device should be open at any time for guaranteed performance

Return Value
CMI_ERROR_INVALID_HANDLE
CMI_ERROR_IN_ARGUMENTS
CMI_ERROR_DEVICE_OPENED
CMI_ERROR_CANNOT_FIND_DEVICE
CMI_ERROR_FAIL_TO_OPEN_IMAGER_DEVICE
CMI_ERROR_CANNOT_ALLOC_MEMORY
CMI_ERROR_EEPROM_READ_TIMEOUT
CMI_ERROR_EEPROM_VERSION_INVALID

CMI_ERROR_CANNOT_FIND_HAARCASCADE_FILES
CMI_ERROR_FAIL_TO_OPEN_MOTOR_DEVICE
CMI_SUCCESS

Remarks
Open the specified device by pdeviceInfo.
Only one device should be open at any time for guaranteed performance

11. cmi_closeDevice

Syntax
```
int cmi_closeDevice(CMI_HANDLE handle);
```

Parameters
handle[in] - CMI_HANDLE value

Return Value
CMI_ERROR_INVALID_HANDLE
CMI_SUCCESS

Remarks
Close the currently opened device.

12. cmi_getBufferSize

Syntax
```
int cmi_getBufferSize( CMI_HANDLE handle,
int numPreviewFrames,
int maxNumUnselectedImages,
int *bufferSize);
```

Parameters
handle[in] - CMI_HANDLE value
numPreviewFrames[in] - maximum number of preview frames the queue can hold.
maxNumUnselectedImages[in] - maximum number of unselected captured images.
bufferSize[out] - necessary buffer size

Return Value
CMI_ERROR_INVALID_HANDLE
CMI_ERROR_IN_ARGUMENTS: input value is out of range numPreviewFrames <= 128 && maxNumUnselectedImages <= 64
CMI_SUCCESS

Remarks
Get buffer size for the specified number of preview frames (1 frame contains one eye image) and iris images captured. "maxNumUnselectedImages" indicates maximum number of captured but not selected images the application wants to keep.
If numPreviewFrames == 0, the event CMI_EVENT_PREVIEW_FRAME will contain the image information only without the image itself.
This function should be called after cmi_openDevice() and before cmi_startCapture().

Otherwise, CMI_ERROR_CALL_GET_BUFFER_SIZE_FIRST will be returned in cmi_startCapture().
After cmi_stopCapture() or cmi_forceCapture(), this function does not need to be called again, if the allocated buffer is not freed.

13. cmi_startCapture

Syntax

```
int cmi_startCapture(  CMI_HANDLE handle,
int bufferSize,
unsigned char *buffer);
```

Parameters
handle[in] - CMI_HANDLE value
bufferSize[in] - working buffer size allocated by application
buffer[in] - working buffer allocated by application

Return Value
CMI_ERROR_INVALID_HANDLE
CMI_ERROR_CALL_GET_BUFFER_SIZE_FIRST
CMI_ERROR_IN_ARGUMENTS
CMI_ERROR_BUFFER_SIZE_TOO_SMALL
CMI_ERROR_DEVICE_CLOSED
CMI_ERROR_DEVICE_STARTED
CMI_ERROR_EEPROM_VERSION_INVALID
CMI_ERROR_FAIL_TO_SEND_COMMAND
CMI_SUCCESS

Remarks
Start capture.

14. cmi_releasePreviewBuffer

Syntax

```
int cmi_releasePreviewBuffer(CMI_HANDLE handle, unsigned char *buffer);
```

Parameters
handle[in] - CMI_HANDLE value

Return Value
CMI_ERROR_INVALID_HANDLE
CMI_ERROR_INVALID_BUFFER - buffer is invalid
CMI_SUCCESS

Remarks
Release preview buffer to re-use in the next CMI_EVENT_PREVIEW_FRAME_INFO event.
For each image pointer, leftImage, rightImage and faceImage, if it is not NULL this function should be called separately.

15. cmi_stopCapture

Syntax
```
int cmi_stopCapture(CMI_HANDLE handle);
```

Parameters
handle[in] - CMI_HANDLE value

Return Value
CMI_ERROR_INVALID_HANDLE
CMI_ERROR_DEVICE_CLOSED
CMI_ERROR_DEVICE_STOPPED
CMI_ERROR_FAIL_TO_SEND_COMMAND
CMI_SUCCESS

Remarks
Stop capturing. It will clean the event queue and return after all images go through.
cmi_cancelPendingReadEvent() can be called if necessary.

16. cmi_forceCapture

Syntax
```
int cmi_forceCapture(CMI_HANDLE handle);
```

Parameters
handle[in] - CMI_HANDLE value

Return Value
CMI_ERROR_INVALID_HANDLE
CMI_ERROR_DEVICE_CLOSED
CMI_ERROR_DEVICE_STOPPED
CMI_ERROR_DEVICE_UPSIDE_DOWN
CMI_ERROR_FAIL_TO_SEND_COMMAND
CMI_ERROR_WAIT_TIMEOUT - wait 5 secs and failed to capture. It should not happen
CMI_SUCCESS

Remarks
Force to capture. In Auto capture mode, this function forces to turn on white LED if it is not on yet and it will wait until the white LED is on for at least 0.5 secs before taking images. In manual mode, white LED is on just after cmi_startCapture() is called and it also wait until the white LED is on at least 0.5 secs before capture.

17. cmi_getWhichEye, cmi_setWhichEye

Syntax
```
int cmi_getWhichEye(CMI_HANDLE handle, int *whichEye);
int cmi_setWhichEye(CMI_HANDLE handle, int whichEye);
```

Parameters
handle[in] - CMI_HANDLE value
whichEye[in/out] - CMI_LEFT_EYE, CMI_RIGHT_EYE, CMI_BOTH_EYES(default)

Return Value
Additional Return value - CMI_ERROR_IN_ARGUMENTS: invalid whichEye

Remarks
Get/Set whichEye option..

18. cmi_getIrisMargins, cmi_setIrisMargins

Syntax
```
int cmi_getIrisMargins(       CMI_HANDLE handle,
int *left,
int *right,
int *top,
int *bottom);
int cmi_setIrisMargins(       CMI_HANDLE handle,
int left,
int right,
int top,
int bottom);
```

Parameters
handle[in] - CMI_HANDLE value
left, right, top, bottom[in/out] - percentage margin to the iris radius

Return Value

Remarks
Get/Set IrisMargins option

19. cmi_getGazeDetectionEnabled, cmi_setGazeDetectionEnabled

Syntax
```
int cmi_getGazeDetectionEnabled(CMI_HANDLE handle, int *enable);
int cmi_setGazeDetectionEnabled(CMI_HANDLE handle, int enable);
```

Parameters
handle[in] - CMI_HANDLE value
enable[in/out] - CMI_TRUE(default), CMI_FALSE

Return Value
Additional Return value - CMI_ERROR_IN_ARGUMENTS: invalid enable

Remarks
Get/Set Gaze Detection Enable option

20. cmi_getLivenessDetectionEnabled, cmi_setLivenessDetectionEnabled

Syntax

```
int cmi_getLivenessDetectionEnabled(CMI_HANDLE handle, int *enable);
int cmi_setLivenessDetectionEnabled(CMI_HANDLE handle, int enable);
```

Parameters

handle[in] - CMI_HANDLE value
enable[in/out] - CMI_TRUE, CMI_FALSE(default)

Return Value

Additional Return value - CMI_ERROR_IN_ARGUMENTS: invalid enable

Remarks

Get/Set Liveness Detection Enable option. Not implemented yet

21. cmi_getTargetIntensity, cmi_setTargetIntensity

Syntax

```
int cmi_getTargetIntensity(CMI_HANDLE handle, unsigned char *intensity);
int cmi_setTargetIntensity(CMI_HANDLE handle, unsigned char intensity);
```

Parameters

handle[in] - CMI_HANDLE value
intensity[in/out] - default is 150.

Return Value

Remarks

Get/Set target intensity of captured iris image (360x360 pixels).

22. cmi_getMaxXYMovement, cmi_setMaxXYMovement

Syntax

```
int cmi_getMaxXYMovement(CMI_HANDLE handle, int *maxXYMovement);
int cmi_setMaxXYMovement(CMI_HANDLE handle, int maxXYMovement);
```

Parameters

handle[in] - CMI_HANDLE value
maxXYMovement[in/out] - defaults are 1,000 µ

Return Value

Remarks

Get/Set maximum movement allowed in XY plane during dt secs. Unit is micron (1/1000 mm)

23. cmi_getMaxZMovement, cmi_setMaxZMovement

Syntax

```
int cmi_getMaxZMovement(CMI_HANDLE handle, int *maxZMovement);
int cmi_setMaxZMovement(CMI_HANDLE handle, int maxZMovement);
```

Parameters

handle[in] - CMI_HANDLE value
maxZMovement[in/out] - defaults are 12,000 μ .

Return Value

Remarks

Get/Set maximum movement allowed in Z direction during dt secs. Unit is micron (1/1000 mm)

24. cmi_getBMTMode, cmi_setBMTMode

Syntax

```
int cmi_getBMTMode(CMI_HANDLE handle, int *mode);
int cmi_setBMTMode(CMI_HANDLE handle, int mode);
```

Parameters

handle[in] - CMI_HANDLE value
mode[in/out] - CMI_BMT_AUTO_MODE(default), CMI_BMT_MANUAL_MODE, CMI_BMT_FAST_AUTO_MODE, CMI_BMT_USER_MODE

Return Value

Additional Return value - CMI_ERROR_IN_ARGUMENTS: invalid mode

Remarks

Get/Set BMT capture mode.

25. cmi_getLeftOffset, cmi_setLeftOffset

Syntax

```
int cmi_getLeftOffset( CMI_HANDLE handle,
int *leftXOffset,
int *leftYOffset);
int cmi_setLeftOffset( CMI_HANDLE handle,
int leftXOffset,
int leftYOffset);
```

Parameters

handle[in] - CMI_HANDLE value
leftXOffset, leftYOffset[in/out] - x, y offset of 640x480 image in 880x520 image

Return Value

Additional Return value - CMI_ERROR_IN_ARGUMENTS: invalid range

Remarks

Get/Set the offset of left iris image in BMT manual mode

26. cmi_getRightOffset, cmi_setRightOffset

Syntax

```
int cmi_getRightOffset(        CMI_HANDLE handle,
int *rightXOffset,
int *rightYOffset);
int cmi_setRightOffset(        CMI_HANDLE handle,
int rightXOffset,
int rightYOffset);
```

Parameters

handle[in] - CMI_HANDLE value
rightXOffset, rightYOffset[in/out] - x, y offset of 640x480 image in 880x520 image

Return Value

Additional Return value - CMI_ERROR_IN_ARGUMENTS: invalid range

Remarks

Get/Set the offset of right iris image in BMT manual mode

27. cmi_getIndicatorBrightness, cmi_setIndicatorBrightness

Syntax

```
int cmi_getIndicatorBrightness(       CMI_HANDLE handle,
unsigned char *red,
unsigned char *green,
                                      unsigned char *brightGreen,
unsigned char *blue);
int cmi_setIndicatorBrightness(       CMI_HANDLE handle,
unsigned char red,
unsigned char green,
unsigned char brightGreen,
unsigned char blue);
```

Parameters

handle[in] - CMI_HANDLE value
red, green, brightGreen, blue[in/out] - RGB brightness. 0 ~ 255

Return Value

Remarks

Get/Set the RGB indicator brightness.

28. cmi_getWhiteBrightness, cmi_setWhiteBrightness

Syntax

```
int cmi_getWhiteBrightness(CMI_HANDLE handle, unsigned char *white);
int cmi_setWhiteBrightness(CMI_HANDLE handle, unsigned char white);
```

Parameters

handle[in] - CMI_HANDLE value
red, green, brightGreen, blue[in/out] - white brightness. 0 ~ 255

Return Value

Remarks
Get/Set white LED brightness in BMT

29. cmi_getOperatingRangeCenter

Syntax
```
int cmi_getOperatingRangeCenter(      CMI_HANDLE handle,
int *operatingRangeCenter);
```

Parameters
handle[in] - CMI_HANDLE value
operatingRangeCenter[out] - current value is 330mm

Return Value

Remarks
Get the center of the operating range in mm in DMX/EMX/EMB

30. cmi_getDepthOfField

Syntax
```
int cmi_getDepthOfField(CMI_HANDLE handle, int *depthOfField);
```

Parameters
handle[in] - CMI_HANDLE value
depthOfField[out] - current value is 30mm

Return Value

Remarks
Get the depth of field in mm in DMX/EMX

31. cmi_getIndicatorEventEnabled, cmi_setIndicatorEventEnabled

Syntax
```
int cmi_getIndicatorEventEnabled(CMI_HANDLE handle, int *enabled);
int cmi_setIndicatorEventEnabled(CMI_HANDLE handle, int enabled);
```

Parameters
handle[in] - CMI_HANDLE value
enabled[In/out] - CMI_TRUE, CMI_FALSE(default)

Return Value
Additional Return value - CMI_ERROR_IN_ARGUMENTS: invalid range

Remarks
Get or set the color indicator event enabled or disabled

32. cmi_getVoiceCueTimeInterval, cmi_setVoiceCueTimeInterval

Syntax
```
int cmi_getVoiceCueTimeInterval (CMI_HANDLE handle, int *firstTimeInterval
_frames, int *nextTimeInterval_frames);
int cmi_setVoiceCueTimeInterval (CMI_HANDLE handle, int firstTimeInterval
_frames, int nextTimeInterval_frames);
```

Parameters
handle[in] - CMI_HANDLE value
firstTimeInterval _frames[In/out] – first time interval in frames before the first voice message is played. The default is 75 (2.5secs) and cannot be smaller than 15 (0.5 secs).
nextTimeInterval _frames[In/out] – the time interval in frames before the next voice message is played. The default is 90 (3secs) and cannot be smaller than 15 (0.5 secs).

Return Value
Additional Return value - CMI_ERROR_IN_ARGUMENTS: invalid range

Remarks
Get or set the firstTimeInterval _frames and nextTimeInterval _frames.

33. cmi_getMaxFrameDiffInEitherEyeMode, cmi_setMaxFrameDiffInEitherEyeMode

Syntax
```
int cmi_getMaxFrameDiffInEitherEyeMode(CMI_HANDLE handle, int *maxFrameDiff
inEitherEyeMode);
int cmi_setMaxFrameDiffInEitherEyeMode(CMI_HANDLE handle, int maxFrameDiff
inEitherEyeMode);
```

Parameters
handle[in] - CMI_HANDLE value
maxFrameDiffinEitherEyeMode [In/out] – Maximum allowed difference in frames between left and right captured eye in order to capture both eyes in "Either Eye" mode. The range of this parameter should be between 3 and 9.

Return Value
Additional Return value - CMI_ERROR_IN_ARGUMENTS: invalid range

Remarks
Get or set the parameter maxFrameDiffinEitherEyeMode.

34. cmi_get FaceCaptureMode, cmi_set FaceCaptureMode

Syntax
```
int cmi_getFaceCaptureMode(CMI_HANDLE handle, int *faceCaptureMode);
```

```
int cmi_setFaceCaptureMode(CMI_HANDLE handle, int faceCaptureMode);
```

Parameters
handle[in] - CMI_HANDLE value
faceCaptureMode[in/out] - CMI_FACE_FULL_RESOLUTION or CMI_FACE_HALF_RESOLUTION. default is CMI_FACE_HALF_RESOLUTION

Return Value
Additional Return value - CMI_ERROR_IN_ARGUMENTS: invalid range

Remarks
Get or set the face image capture mode. This function should be called before cmi_openDevice(). CMI_FACE_FULL_RESOLUTION is only supported in Windows and it should be set for color face image in EMX-30 or DMX-30. Once the face full resolution Bayer image is captured, it can be converted to color image with the library cmicolorLib.dll. The full resolution is 720x960 and the half resolution is 360x480.

35. cmi_getMotorReturnToInitEnabled, cmi_setMotorReturnToInitEnabled

Syntax
```
int cmi_getMotorReturnToInitEnabled (CMI_HANDLE handle, int *enabled);
int cmi_setMotorReturnToInitEnabled(CMI_HANDLE handle, int enabled);
```

Parameters
handle[in] - CMI_HANDLE value
enabled[In/out] - CMI_TRUE, CMI_FALSE(default)

Return Value
Additional Return value - CMI_ERROR_IN_ARGUMENTS: invalid range

Remarks
Get or set whether to return to initial motor position after the images are captured in EMX/EMB/EMA. The default is enabled. If this is disabled, it will stay at the last position after the images are captured and this may be useful to verify a user after successful enrollment. cmi_setMotorReturnToInitEnabled() should be called before cmi_startCapture().

36. cmi_getInOutDepthOfField, cmi_setInOutDepthOfField

Syntax
```
DWORD cmi_ getInOutDepthOfField (CMI_HANDLE handle, int *inDepthOfField, int
*outDepthOfField);
DWORD cmi_ setInOutDepthOfField (CMI_HANDLE handle, int inDepthOfField, int
outDepthOfField);
```

Parameters
handle[in] - CMI_HANDLE value
insideDepthOfField[in/out] -If it not in the range of 15 to 35, CMI_ERROR_IN_ARGUMENTS will be returned. Default is 15(mm).
outsideDepthOfField[in/Out] - If it not in the range of 15 to 25, CMI_ERROR_IN_ARGUMENTS will be returned. Default is 15(mm).

Return Value
Additional Return value - CMI_ERROR_DEVICE _STARTED: when the set function is called after start.

Remarks
Get or set the inside and outside depth of field. The center of depth of field is 330mm from the front surface of EMX. The minimum and default of each depth of field is 15mm, so total depth of field is 30mm. This is the recommended depth of field for enrollment. The maximum is 35mm for inside and 25mm for outside, total maximum depth of field is 60mm, which is for recognition for physical access control application.

6. MIRLIN Functions in cmirisLib2

There are 5 MIRLIN encoding and matching functions that are embedded into the "cmirisLib2" main library. These are detailed in the `cmirilib2.h` header file in the /usr/local/include directory. Please refer to /usr/local/src/CMITECH/CMIMIRDemo_src for the sample code.

In order to use these MIRLIN functions, MIRLIN License should be purchased with EMA-30, EMB-30, and EMX-30, so that MIRLIN License level in the device should be set to 1 in the factory. If the MIRLIN License level is 0, then the template generation and comparing functions will return error `CMI_MIR_ERROR_INVALIDE_LICENSE`.

1. cmi_mir_getVersion

Syntax
```
int cmi_mir_getVersion(CMI_HANDLE handle, DWORD *major, DWORD *minor, DWORD *revision);
```

Parameters
Handle[in] – CMI_HANDLE value
major[out]
minor[out]
revision[out]

Return Value

Remarks
Get version of MIRLIN Algorithm. The current version number is 2.32.0.

2. cmi_mir_getLicenseLevel

Syntax
```
int cmi_mir_getLicenseLevel(CMI_HANDLE handle, int *level);
```

Parameters
handle[in] - CMI_HANDLE value
level[out] - 0 (MIRLIN disabled), 1 (MIRLIN enabled)

Return Value

Remarks
Get License Level of MIRLIN Algorithm. Currently there is only one level, which is 1.

3. cmi_mir_getEnrolTemplates

Syntax
```
int cmi_mir_getEnrolTemplates(CMI_HANDLE handle, unsigned char
*leftEnrolTemplate, unsigned char *rightEnrolTemplate, int templateSize,
CMI_IMAGE_INFO *imageInfo, int showSegmentation);
```

Parameters
handle[in] - CMI_HANDLE value
leftEnrolTemplate[out] - 579 bytes template
rightEnrolTemplate[out] - 579 bytes template
templateSize[in] - should be CMI_MIR_ENROL_TEMPLATE_SIZE = 579
imageInfo[in] - CMI_IMAGE_INFO structure returned from the event
CMI_EVENT_IRIS_IMAGES_SELECTED
showSegmentation[in] - whether to show the iris and pupil segmentation on the images in imageInfo
CMI_MIR_DO_NOT_SHOW_SEGMENTATION or CMI_MIR_SHOW_SEGMENTATION

Return Value
Additional Return value - CMI_MIR_ERROR_UNKNOWN_LICENSE: cannot read the license
CMI_MIR_ERROR_INVALID_LICENSE: no MIRLIN license in the device
CMI_MIR_ERROR_WRONG_TEMPLATE_SIZE: wrong template size
CMI_MIR_ERROR_NO_INPUT_IMAGES: both iris images are NULL
CMI_MIR_ERROR_FAIL_TO_GENERATE_TEMPLATES: fail to generate both templates
CMI_MIR_LEFT_TEMPLATE_GENERATED_ONLY: only left iris template was generated
CMI_MIR_RIGHT_TEMPLATE_GENERATED_ONLY: only right iris template was generated
CMI_SUCCESS: both iris templates were generated

Remarks
Get enroll templates generated by MIRLIN Algorithm.

4. cmi_mir_getMatchTemplates

Syntax
```
int cmi_mir_getMatchTemplates(CMI_HANDLE handle, unsigned char
*leftMatchTemplate, unsigned char *rightMatchTemplate, int templateSize,
CMI_IMAGE_INFO *imageInfo, int showSegmentation);
```

Parameters
handle[in] - CMI_HANDLE value
leftMatchTemplate[out] - 579 bytes template
rightMatchTemplate[out] - 579 bytes template
templateSize[in] - should be one of the following sizes. The default is 7479 for 13 rotations

CMI_MIR_MATCH_ROT_0_TEMPLATE_SIZE	*579*	*-> 0 rotations*
CMI_MIR_MATCH_ROT_5_TEMPLATE_SIZE	*2879*	*-> 5 rotations*
CMI_MIR_MATCH_ROT_7_TEMPLATE_SIZE	*4029*	*-> 7 rotations*
CMI_MIR_MATCH_ROT_9_TEMPLATE_SIZE	*5179*	*-> 9 rotations*
CMI_MIR_MATCH_ROT_13_TEMPLATE_SIZE	*7479*	*->13 rotations - default*
CMI_MIR_MATCH_ROT_25_TEMPLATE_SIZE	*14379*	*->25 rotations*

imageInfo[in] - CMI_IMAGE_INFO structure returned from the event

CMI_EVENT_IRIS_IMAGES_SELECTED
showSegmentation[in] - whether to show the iris and pupil segmentation on the images in imageInfo
CMI_MIR_DO_NOT_SHOW_SEGMENTATION or CMI_MIR_SHOW_SEGMENTATION

Return Value
Additional Return value - CMI_MIR_ERROR_UNKNOWN_LICENSE: cannot read the license
CMI_MIR_ERROR_INVALID_LICENSE: no MIRLIN license in the device
CMI_MIR_ERROR_WRONG_TEMPLATE_SIZE: wrong template size
CMI_MIR_ERROR_NO_INPUT_IMAGES: both iris images are NULL
CMI_MIR_ERROR_FAIL_TO_GENERATE_TEMPLATES: fail to generate both templates
CMI_MIR_LEFT_TEMPLATE_GENERATED_ONLY: only left iris template was generated
CMI_MIR_RIGHT_TEMPLATE_GENERATED_ONLY: only right iris template was generated
CMI_SUCCESS: both iris templates were generated

5. cmi_mir_compareTemplate

Syntax
```
int cmi_mir_compareTemplate(CMI_HANDLE handle, unsigned char *enrolTemplate,
unsigned char *matchTemplate, int matchTemplateSize, float *hammingDistance);
```

Parameters
handle[in] - CMI_HANDLE value
enrolTemplate [out] - 579 bytes template
matchTemplate [out] - matchTemplateSize bytes template
matchTemplateSize [in] - matchTemplate size, which should be one of match template sizes defined above.
hammingDistance[out] - hamming distance between two templates. If both templates are perfectly matched the value is 0 and it is 0.5 if two templates are totally non-matched. The recommended threshold is 0.15.

Return Value
Additional Return value - CMI_MIR_ERROR_UNKNOWN_LICENSE: cannot read the license
CMI_MIR_ERROR_INVALID_LICENSE: no MIRLIN license in the device
CMI_MIR_ERROR_WRONG_TEMPLATE_SIZE: wrong template size
CMI_MIR_ERROR_EMPTY_ENROL_TEMPLATE: enrol template is all zeros. This is useful when one valid iris template is enrolled and the other iris enrol template is stored as zeros.
CMI_MIR_ERROR_FAIL_TO_COMPARE_TEMPLATES: fail to compare two templates
CMI_SUCCESS: success in comparing two templates.

Remarks
Compare enroll template with match template by MIRLIN Algorithm and get the hamming distance.

7. Testing the external I/O interface of EMA-30 or ECU-30

There is a test application, “emaTest” in the /usr/local/bin directory to evaluate the external interfaces and wiring connections of the devices. Please refer to “/usr/local/doc/EMA_EMB_ECU_Quick_User_Guide_V2.1.1.pdf” for detailed description.

Wiegand:

- ECU-30 has 2 Wiegand Out channels, but EMA has one.
- Before writing Wiegand Out, need to set number of bits, pulse width, and pulse interval with “Set Config”. This configuration setup will affect on both channels in ECU-30.
- “Auto Detect” will detect the Wiegand In format.

GPIO: GPI will be read when emaLib is open and when the value is changed. The default value of GPIO is high.

Relay: Normal is the default state of “Normally Open (NO) and Abnormal is alternative state of “Normally Closed” (NC).

USB Port Reset: USB OFF cuts off USB power when port is not working properly. Application needs to send USB ON to complete reset.

Please refer to emaglobal.h and emalib.h for detailed functions.