TeliCamAPI

Library manual

Version 3.0.0(2020/01/08)

Toshiba Teli Corporation

Information contained in this document is subject to change without prior notice.

DAA00562A

 www.toshiba-teli.co.jp

Contents

1. Introduction

TeliCamSDK is a software development kit to control Toshiba Teli USB3 and GigE Vision digital camera series from PCs.

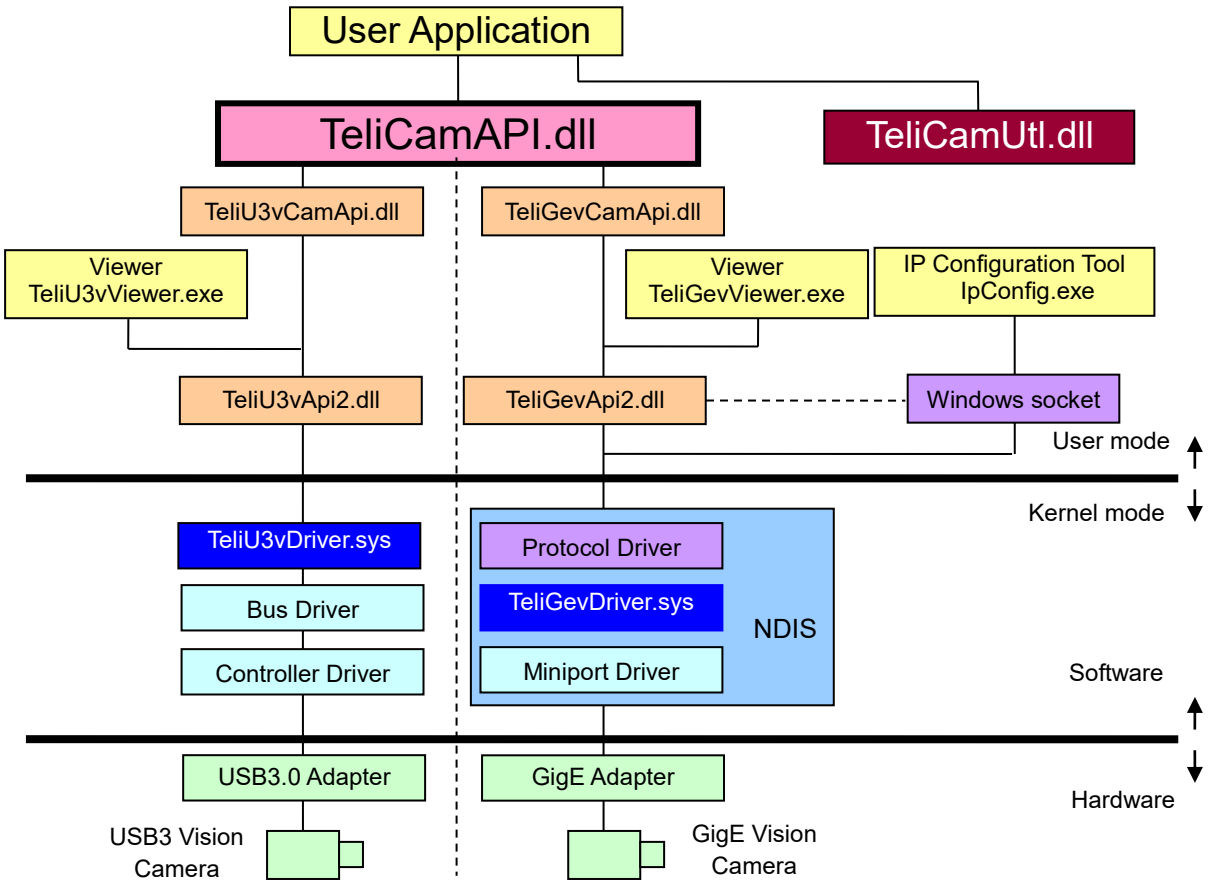
This document describes how to use TeliCamAPI, which is a programming interface for C++ in the TeliCamSDK package. TeliCamAPI allows users to create applications easily without paying attention to the camera interface type.

This document is intended for engineers who build a system using a camera.

2. Configuration

2.1. Configuration under Windows

The following figure indicates software configuration of TeliCamSDK.



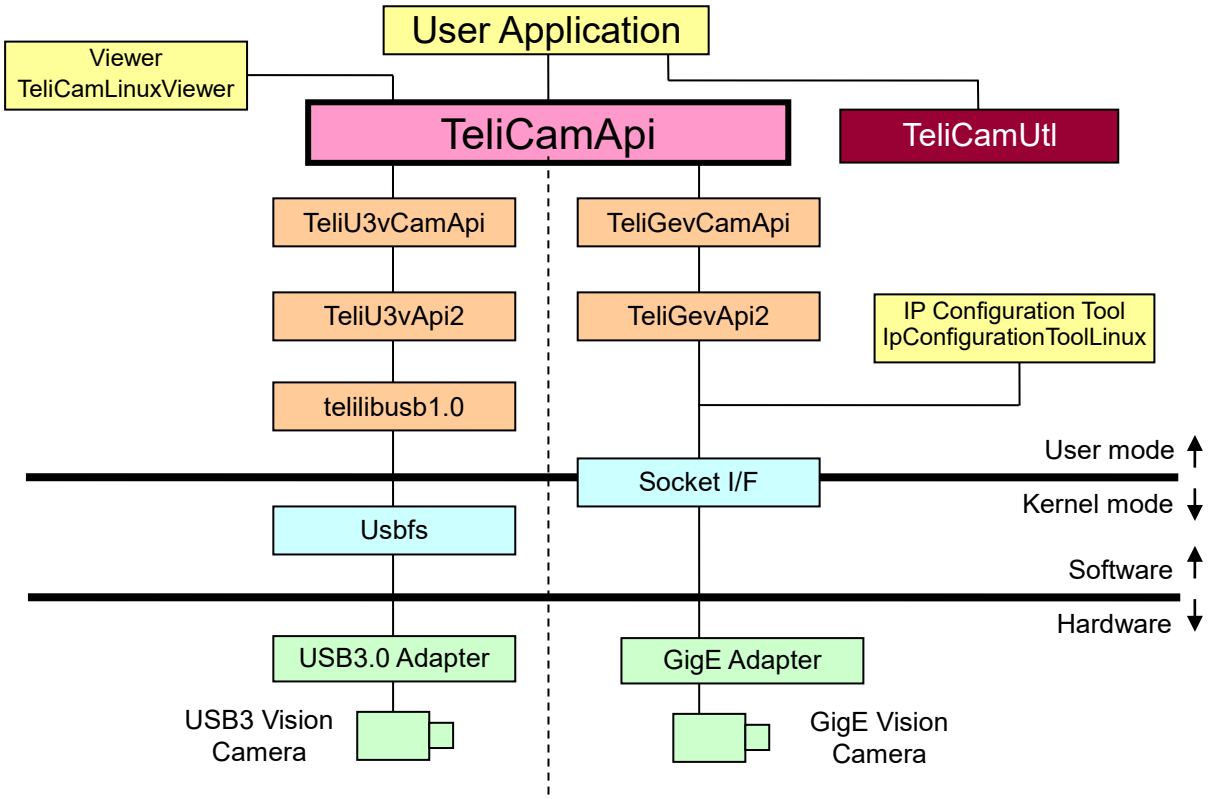
for 32bit OS	for 64bit OS	Description
TeliCamApi.dll	TeliCamApi64.dll	Function library for designing native applications.
TeliU3vCamApi.dll	TeliU3vCamApi_64.dll	Extended function library for USB3 Vision cameras.
TeliU3vApi2.dll	TeliU3vApi2_64.dll	Function library for USB3 Vision cameras.
TeliU3vDriver.sys	TeliU3vDriver64.sys	Device driver for USB3 Vision cameras.
TeliGevCamApi.dll	TeliGevCamApi64.dll	Extended function library for GigE Vision cameras.
TeliGevApi2.dll	TeliGevApi2_64.dll	Function library for GigE Vision cameras.
TeliGevDriver.sys	TeliGevDriver64.sys	Device driver for GigE Vision cameras.
TeliCamUtl.dll	TeliCamUtl64.dll	Utility function library for handling images.
TeliU3vViewer.exe	TeliU3vViewer64.exe	Viewer for checking features and images. (USB3 Vision)
TeliGevViewer.exe	TeliGevViewer64.exe	Viewer for checking features and images.(GigE Vision)
IpCnfg.exe	IpCnfg.exe	Utility application for setting IP addresses of GigE Vision cameras.

Use TeliCamApi.dll and TeliCamUtl.dll for designing user applications.

TeliCamAPI can connect the camera up to 64 units.
Further network adapter is up to 16 units, and GigE Vision camera is up to 16 units per adapter.

2.2. Configuration under Linux

The following figure indicates software configuration of TeliCamSDK for Linux.



	Description
TeliCamApi (libTeliCamApi.so)	Function library for designing native applications.
TeliU3vCamApi (libTeliU3vCamApi.so)	Extended function library for USB3 Vision cameras.
TeliU3vApi2 (libTeliU3vApi2.so)	Function library for USB3 Vision cameras.
TeliGevCamApi (libTeliGevCamApi.so)	Extended function library for GigE Vision cameras.
TeliGevApi2 (libTeliGevApi2.so)	Function library for GigE Vision cameras.
TeliCamUtl (libTeliCamUtl.so)	Utility function library for handling images.
teliusb1.0 (libteliusb1.0.so)	Function library that provides generic access to USB devices
TeliCamLinuxViewer	Viewer for checking features and images.
IpConfigurationToolLinux	Utility application for setting IP addresses of GigE Vision cameras.

Use TeliCamApi and TeliCamUtl for designing user applications.

TeliCamSDK can connect the camera up to 64 units.
Further Ethernet adapter is up to 16 units, and GigE Vision camera is up to 16 units per adapter.

Note that TeliCamAPI does not support multi-cast feature of GigE Vision camera.

3. System Requirements

3.1. System Requirements under Windows

The following table shows environments necessary for using TeliCamAPI.
This environment, however, does not always guarantee operation of all application programs.
A higher performance host PC may be required depending on use conditions.

Supported OS	● Windows 7 32/64-bit ● Windows 8.1 32/64-bit ● Windows 10 32/64-bit *5
Recommended PC Specifications	● CPU : Intel Core2 2.40GHz or above recommended ● Memory : 4Gbytes or above ● Graphics : VRAM with 256 Mbytes or above mounted
Recommended USB3.0 Adapter	● USB3.0 adapter with USB3.0 host controller from Renesas Electronics *1
Recommended Network Adapter	● Gigabit Ethernet adapter supporting Jumbo Frame (Jumbo Packet) *2 (9014 bytes or above) (Intel PRO/1000 series, etc.)
Required Software	● Microsoft Visual C++ 2013 Redistributable Package ● Microsoft Visual C++ 2010 SP1 Redistributable Package ● GenICam GenApi reference implementation v.3.0.1 *3,*4 ● Microsoft Direct X End-User Runtime (DirectX 9.0c or later) *4
Supported Camera	● Toshiba Teli USB3 Vision digital camera ● Toshiba Teli Gig-E Vision digital camera

Notes

*1: Necessary in application that uses USB3 Vision cameras.
*2: Necessary in application that uses GigE Vision cameras.
*3: Necessary in development environment even if GenApi module is disabled in user application.
*4: Necessary in TeliU3vViewer and TeliGevViewer.
*5: It is recommended to use 64 bit OS when using 5M-pixel camera or more or when using multiple cameras.

3.2. System Requirements under Linux

TeliCamApi for Linux runs on Intel / AMD x86 and ARM architecture.
Operation confirmed OS is as follows.

• Intel/AMD

Ubuntu 14.04 LTS amd64 ※1, ※2	For 64-bit Intel/AMD (x86_64)
Ubuntu 16.04 LTS amd64 ※3	For 64-bit Intel/AMD (x86_64)
Ubuntu 18.04 LTS amd64 ※4	For 64-bit Intel/AMD (x86_64)
Debian 8.1.0 amd64 (with the GNOME desktop environment)	For 64-bit Intel/AMD (x86_64)
CentOS 7.3 amd64	For 64-bit Intel/AMD (x86_64)
Fedora 27 amd64	For 64-bit Intel/AMD (x86_64)

• ARM

Jetson TK1 (Jetpack 3.0 Ubuntu 14.04)	For 32-bit ARM (armv7l)
Jetson TX2 (Jetpack 3.2 Ubuntu 16.04)	For 64-bit ARM (aarch64)
Jetson nano (Jetpack 4.2.1 Ubuntu 18.04)	For 64-bit ARM (aarch64)
Odroid XU4 ※5 (Ubuntu mate 16.04)	For 32-bit ARM (armv7l)
Raspberry pi 3 ※5, ※6 Raspbian GNU/Linux 9 (stretch)	For 32-bit ARM (armv7l)

※1 Ubuntu 14.04 has a problem regarding XHCI driver :
If stream start / stop are invoked repeatedly, the stream interface can stop.
If the stream interface stops, change the Strm_Stop function to the Strm_Abort function.
We recommend that you use Ubuntu 14.04.1 or later, or update the kernel version.

※2 Ubuntu 14.04 has a problem regarding suspension and hibernation :
USB ports do not work after suspensions and hibernations.
We recommend that you do not use the suspension and hibernation functions**.**

※3 Supported Ubuntu from 16.04 to 16.04.5.

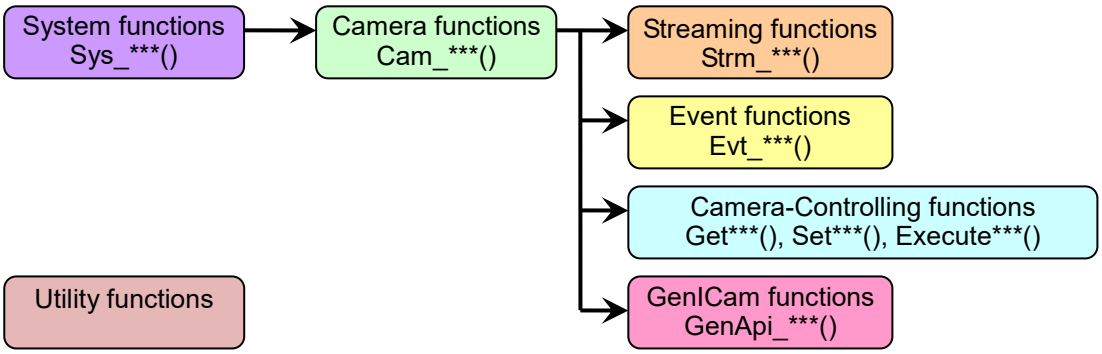
※4 Supported Ubuntu from 18.04 to 18.04.1.

※5 GigE Vision digital camera may not be able to acquire images at maximum frame rate.

※6 USB3 Vision digital camera can not be used.

4. Library

The following diagram shows functions contained in TeliCamAPI Library



Function Group	Description
System functions	Functions for controlling TeliCamAPI system itself.
Camera functions	Functions for controlling a camera.
Streaming functions	Functions for controlling an image stream interface. TeliCamAPI provides 2 types of streaming functions. High-Level functions: Allows simple coding. Low-Level functions: Allows flexible streaming control.
Event functions	Functions for notifying events of the camera. TeliCamAPI provides 2 types of event functions. High-Level functions: Allows simple coding. Low-Level functions: Allows flexible event control.
Camera-Controlling functions	Functions for controlling individual features of the camera.
GenICam functions	Functions for controlling individual features of the camera using GenApi module. User application can control features or get information that Camera-Control functions does not support.
Utility functions	Functions for handling images.

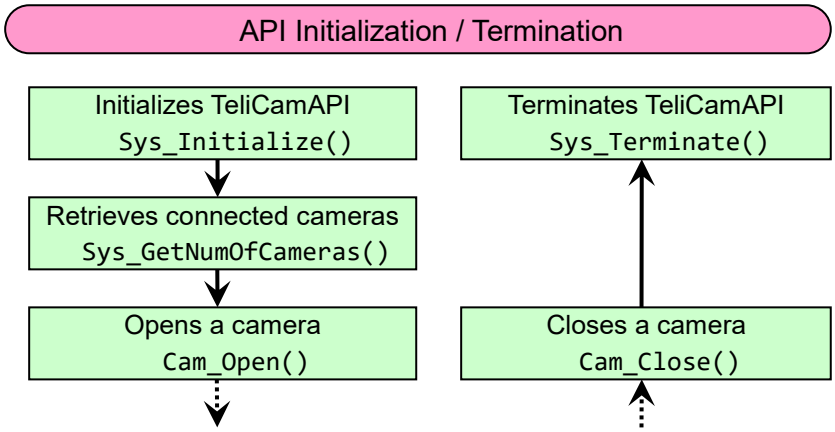
4.1. Usage

This section describes a brief sequence of TeliCamAPI initialization and termination, camera control, stream data acquisition, and camera event (message) data acquisition.

4.1.1. Initialize and terminate API

User applications should call "`Sys_Initialize()`" only once to initialize TeliCamAPI system before calling TeliCamAPI functions. User applications also should call "`Sys_Terminate()`" to release resources used in TeliCamAPI system, after finished using TeliCamAPI.

The following diagram describes steps to start using a camera and finish using a camera:



4.1.2. Enumerate cameras

User applications should call "`Sys_GetNumOfCameras()`" to create a list of available cameras inside TeliCamAPI, before opening any camera. TeliCamAPI gets information from list of available cameras on opening a camera.

"`Cam_GetInformation()`" is available for acquiring camera information before opening it.

4.1.3. Open and close camera

Call "`Cam_Open()`" to open a control interface for the camera. User application can control camera using the Camera Handle returned from "`Cam_Open()`".

If using a specific camera is required, call "`Cam_OpenFromInfo()`" specifying serial number of the camera or

`UserDefinedName (DeviceUserID)`.

User application can open a camera that the other applications are using. User application can get information of a camera that the other application is using, but it will fail in opening camera stream or camera event if the other application is using it.

Call “`Cam_Close`()” to release used resources, after finished using the camera.

4.1.4. Control camera and acquire information on camera

There are three ways to control a camera and acquire information on the camera.

- **Using Camera_Control_Functions (refer to section 5.5)**

  This is the easiest way to access registers on a camera without paying attention to type of camera interface, model of camera and register address. Some of these functions internally use functions of GenICam GenApi and some of them access camera registers directly.

  Calling a function may result in failure if the feature is not implemented in the camera. Please refer to instruction manual of the camera about the implemented features.

- **Using GenICam_Functions (refer to section 5.6)**

  User application can access registers on a camera using “Feature name” defined in GenICam Standard Feature Name Convention or Toshiba Teli specific feature names without paying attention to camera interface and type of the camera.

  User application can obtain various information that Camera Control Functions do not provide, through GenICam_Functions.

  These functions are wrapper functions for APIs of GenICam GenApi, which provide the way to access a register specified by “Feature Name”, and read or write value using numeric value or “strings assigned to valid values”, referring data in a camera description file (XML file) standardized by the GenICam standard. XML file contains information of register address corresponding to “Feature Name”, register numeric value corresponding to “strings assigned to valid values”, and so on.
  Each BU and BG series camera has an XML file inside it.

  For detailed information on GenICam, please refer to http://www.genicam.org .

- **Register direct access**

  Performs direct register access using “`Cam_ReadReg`()” and “`Cam_WriteReg`()” functions.

  User application will run in high performance because searching register address specified by “Feature Name” and converting a “string assigned to valid value” to numeric value are not necessary.

  Information of register address and its valid values or valid value range is necessary for using these functions.

  For detailed information on IIDC2 register map, please refer to http://jiia.org .

4.1.5. Streaming control

TeliCamAPI provides two ways to acquire the stream data, using High-Level API and using Low Level API.

In High-Level API case, TeliCamAPI executes almost all image-receiving process in background process, which will make source code simple and slim. Users can design image-receiving code quickly.

In Low-Level API case, TeliCamAPI executes minimum part of image-receiving process in background process, which allows software designers to organize stream data receiving processing in the user's arbitrary sequence.

4.1.5.1. Acquiring image data using High-Level API functions

"High-Level" means that these functions belong to layer near application layer. High-Level API functions allow software designers to make sourced code for acquiring stream data (image data) simple and slim.

High-Level API will create an ImageRingBuffer for keeping StreamRequest structures inside TeliCamAPI on opening the stream interface. TeliCamAPI will silently fill StreamRequests in the ImageRingBuffer with received stream data (image data) during image acquisition is active.

TeliCamAPI provides two types of "Image Acquired" notification to user application.

- **Using event(signal) object specified on opening the stream interface.**

  Under Windows
  TeliCamAPI will notify reception of image by setting signal state to event(signal) object specified on opening the stream interface.

  In usual case, user application waits image reception using `WaitForSingleObject`() or `WaitForMultipleObjects`(), and gets the latest image data using `Strm_ReadCurrentImage`() and so on, after reception of "ImageAcquired" notification.

  Under Linux
  TeliCamAPI will notify reception of image by setting signal state to signal object specified on opening stream interface.

  In usual case, user application waits image reception using `Sys_WaitForSignal()`, and gets the latest image data using `Strm_ReadCurrentImage`() and so on, after reception of "ImageAcquired" notification.

- **Using callback function**.

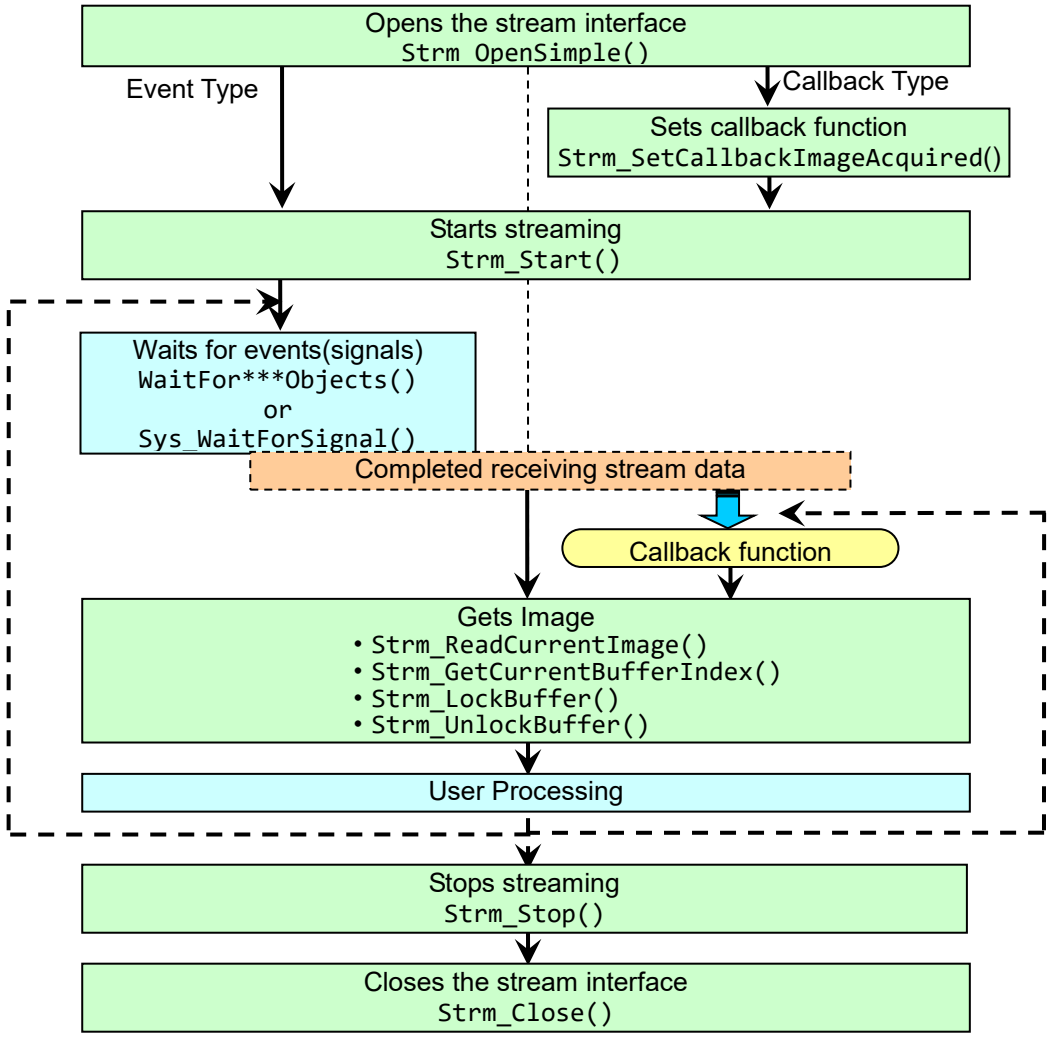
  TeliCamAPI will notify reception of image by calling callback function registered with `Strm_SetCallbackImageAcquired`().

  User application will get the latest image data as argument of the callback function.

`Strm_LockBuffer`() is also available for getting image from ImageRingBuffer, including the past images stored in it. `Strm_LockBuffer`() will be useful in getting past images in Bulk trigger mode or sequential shutter mode.

Note that callback function may block the stream receiving operation inside TeliCamAPI. Perform processing operation, which will take a long time, outside the callback function.

The following diagram shows a general sequence.



4.1.5.1.1. Open the stream interface

Call "`Strm_OpenSimple()`" to open the stream interface. "`Strm_OpenSimple()`" will return a Stream-Handle for the created the stream interface. "`Strm_OpenSimple()`" will create StreamRequest structures for holding received stream data, and create an ImageRingBuffer for keeping StreamRequests.
Note that the ImageRingBuffer inside the TeliCamAPI is different from image buffer in the camera.

User application can specify the number of StreamRequests in the ImageRingBuffer. The default value is eight. Acquiring large size images continuously in high frame rate may cause buffer busy error when a buffer of ImageRingBuffer that the latest image data is going to be saved is locked by user application. Increasing size of ImageRingBuffer or reducing image-processing load may be useful for avoiding the error.

4.1.5.1.2. Register callback function for receiving stream data (image data)

TeliCamAPI can call callback function on receiving a stream data (image data).
Call "`Strm_SetCallbackImageAcquired`()" to register callback function to the stream.

TeliCamAPI will lock the target StreamRequest of a callback function during executing it. On the other hand, TeliCamAPI will continue stream receiving operation in a thread other than the thread of callback function during executing callback function.

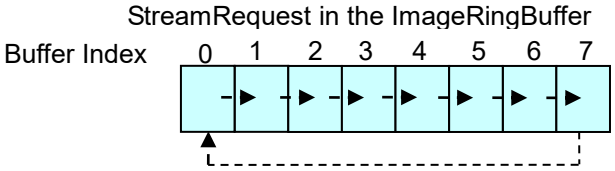
TeliCamAPI will cause a buffer busy error and discard the new stream data, if the callback function keeps using the StreamRequest for writing just received image. Making processing time of callback function as short as possible is recommended.

Calling "`Strm_SetCallbackBufferBusy`()" allows user application to register callback function for detecting buffer busy.

4.1.5.1.3. Start streaming

Call "`Strm_Start`()" for instructing a camera to start acquiring images and start streaming.

On reception of new stream data, TeliCamAPI will replace buffer in ImageRingBuffer with a StreamRequest that contains the received stream data in turns from of index 0 of ImageRingBuffer. The



next index after the final index will be zero. This procedure is done in background process.

TeliCamAPI will notify reception of an image to user application through the event(signal) object specified on calling "`Strm_OpenSimple`()".

4.1.5.1.4. Acquire image data

TeliCamAPI provides three ways for getting image from ImageRingBuffer.

- **Calling "`Strm_ReadCurrentImage()`"**
  "`Strm_ReadCurrentImage`()" will copy the latest image in the ImageRingBuffer to the specified buffer.

- **Calling "`Strm_LockBuffer()`"**
  User application should lock a buffer in ImageRingBuffer before accessing to image in the buffer to avoid the image data being replaced to new image by TeliCamAPI.

  "`Strm_LockBuffer`()" will lock the specified StreamRequest and return pointer to image data in the StreamRequest. "`Strm_GetCurrentBufferIndex`() is available for getting the index of StreamRequest which contains the latest image.

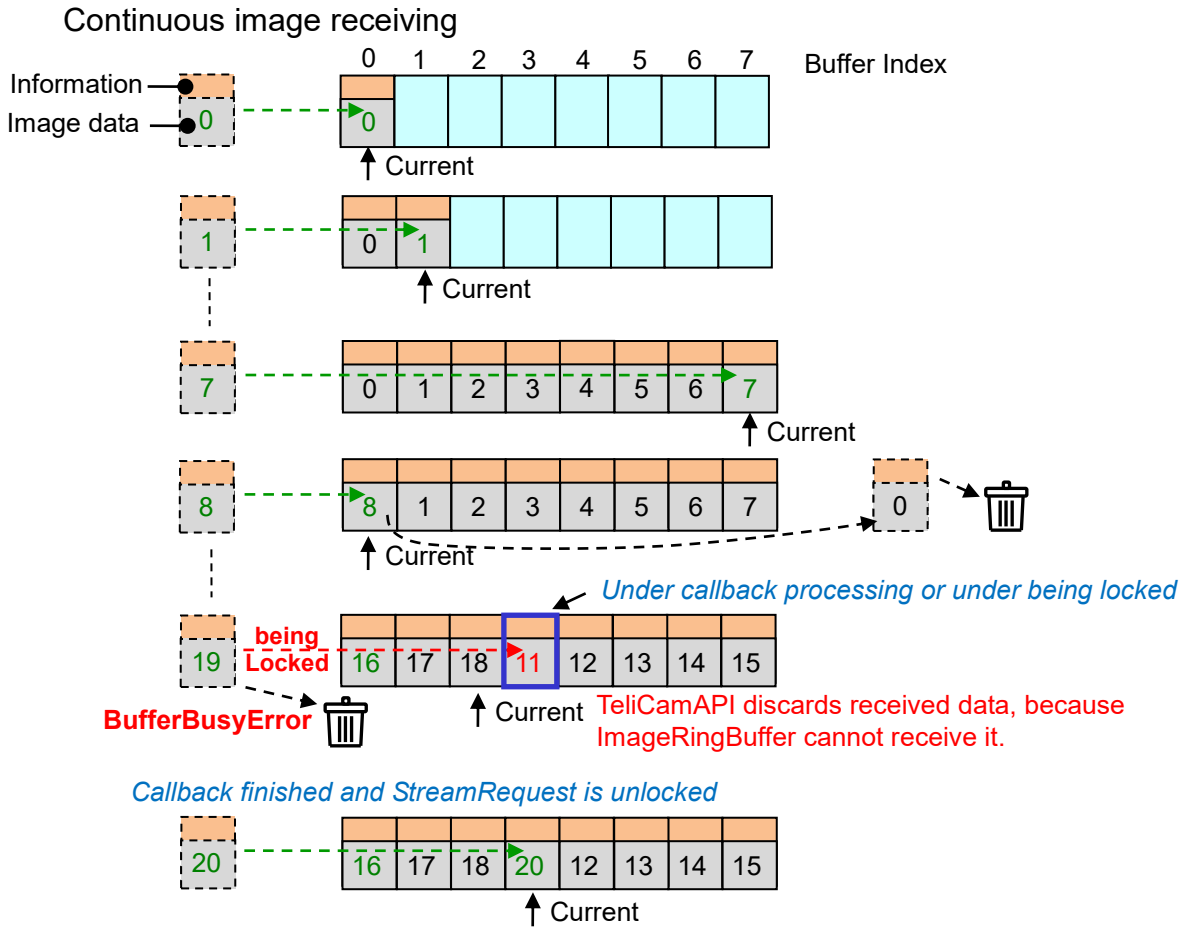
  Unlock the locked buffer using "`Strm_UnlockBuffer`()" as soon as finished accessing to (copying) the image data. Otherwise, buffer busy error may occur.

- **Getting image data in callback function.**
  TeliCamAPI will run callback function on receiving a stream data (image data), with pointer to the

latest image data as its argument.

User application can access to the latest image using the pointer during callback function is running.

If camera image buffer mode is selected, a frame count of images specified by AcqusitionFrameCount register will be transferred to the ImageRingBuffer on each calling of `ExecuteCamImageBufferRead()`. If not enough images were stored in the camera image buffer when `ExecuteCamImageBufferRead()` was called, the rest images will be transferred to the ImageRingBuffer as soon as they are saved to the camera image buffer.



4.1.5.1.5. Stop streaming

Call "`Strm_Stop()`" to instruct a camera to stop acquiring image and stop streaming.

4.1.5.1.6. Close the stream interface

Call "`Strm_Close()`" to terminate use of the stream interface.

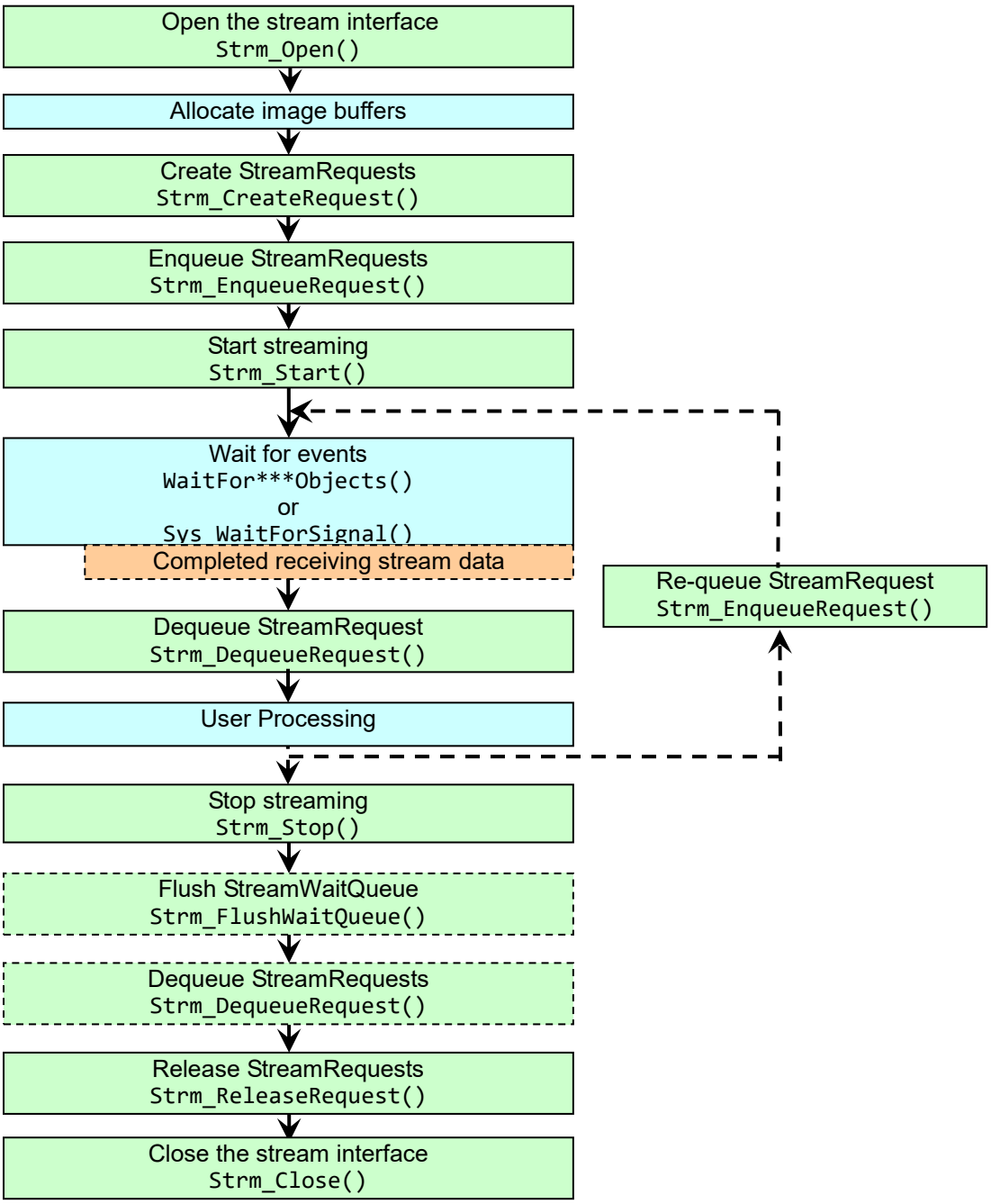
4.1.5.2. Acquiring image data using Low-Level API functions

“Low-Level” means that these functions belong to layer near transport layer.
In Low-Level API case, TeliCamAPI executes minimum part of image-receiving process in background process, which allows software designers to organize stream data receiving processing in the user’s arbitrary sequence.

In Low-Level API case, user application can get image data by putting StreamRequests into StreamWaitQueue inside TeliCamAPI and retrieving StreamRequests from StreamCompleteQueue inside TeliCamAPI. StreamRequest is a structure that contains image data buffer and member variables for image information.

TeliCamAPI will write received image data and its information to a StreamRequest in StreamWaitQueue and move it to StreamCompleteQueue silently.

The diagram below shows a general sequence performed by the application.



4.1.5.2.1. Open the stream interface

Call "`Strm_Open`()" to open the stream interface and to register event(signal) object for image acquired event.
Stream-Handle of the opened the stream interface will be returned from "`Strm_Open`()".

4.1.5.2.2. Allocate image buffer

TeliCamAPI provides function for creating StreamRequest structure. However, user application must allocate memory to image buffer for storing received image data in user code, which will become member of StreamRequest structures.

Prepare enough amounts of StreamRequests (and image buffers inside them) for avoiding Frame-Lost during the term that CPU load of user application processing is the heaviest.

User application should not release image buffers while the parent StreamRequest is alive.

The following is steps for releasing image buffers.

1. Call "`Strm_FlushWaitQueue`()" to move StreamRequests in StreamWaitQueue to StreamCompleteQueue.
2. Call "`Strm_DequeueRequest`()" until StreamCompleteQueue becomes empty.
3. Call "`Strm_ReleaseRequest`()" to release the parent StreamRequest.
4. Release the image buffer.

4.1.5.2.3. Create StreamRequest

Call "`Strm_CreateRequest`()" to create a StreamRequest for acquiring stream data (image data). User application has to prepare memory allocated image buffer as a member of the created StreamRequest before calling "`Strm_CreateRequest`()".

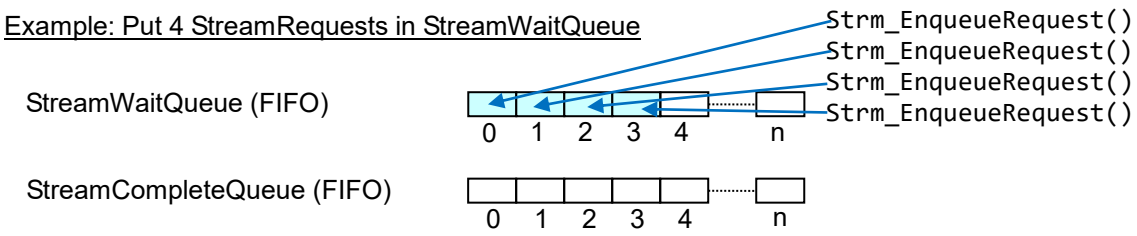
User application cannot change size of image buffer contained in the StreamRequest.

Release StreamRequests and create new StreamRequests including image buffer of new image size, when Image size is changed.

User application should release all StreamRequests and their children image buffers before terminating the TeliCamAPI.

4.1.5.2.4. Put StreamRequest in StreamWaitQueue

Call "`Strm_EnqueueRequest`()" to put StreamRequest in StreamWaitQueue for receiving stream data (image data).



4.1.5.2.5. Start streaming

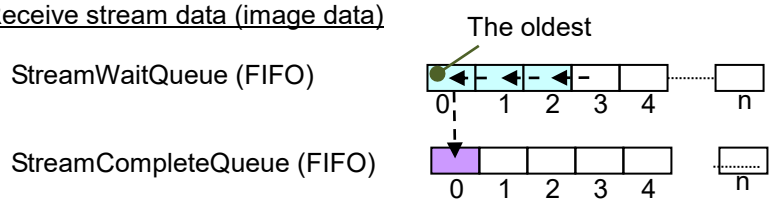
Call "`Strm_Start`()" to instruct a camera to start acquiring images and start streaming.

4.1.5.2.6. Receive stream data (image data)

TeliCamAPI will save the incoming stream data to the oldest StreamRequest in StreamWaitQueue. When reception of a stream data (image data) is completed, TeliCamAPI will move the oldest StreamRequest in StreamWaitQueue to StreamCompleteQueue, and set the event(signal) object for notifying “Image acquired” signaled.

If CameraImageBuffer mode is active, camera will transfer specified number of images on each calling of `ExecuteCamImageBufferRead()`. If there are no images stored in the camera image buffer at the timing `ExecuteCamImageBufferRead()` is called, camera will transfer images as soon as they are saved to the camera image buffer.

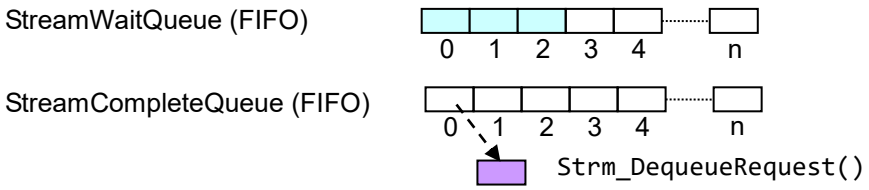
Receive stream data (image data)



4.1.5.2.7. Extract StreamRequest from StreamCompleteQueue

Call “`Strm_DequeueRequest()`” to extract the oldest StreamRequest from StreamCompleteQueue, after acquiring a stream (image). The extracted StreamRequest will contain received image,

Extract StreamRequest from StreamCompleteQueue



4.1.5.2.8. Stop streaming of camera

Call “`Strm_Stop()`” to instruct a camera to stop acquiring image and stop streaming.

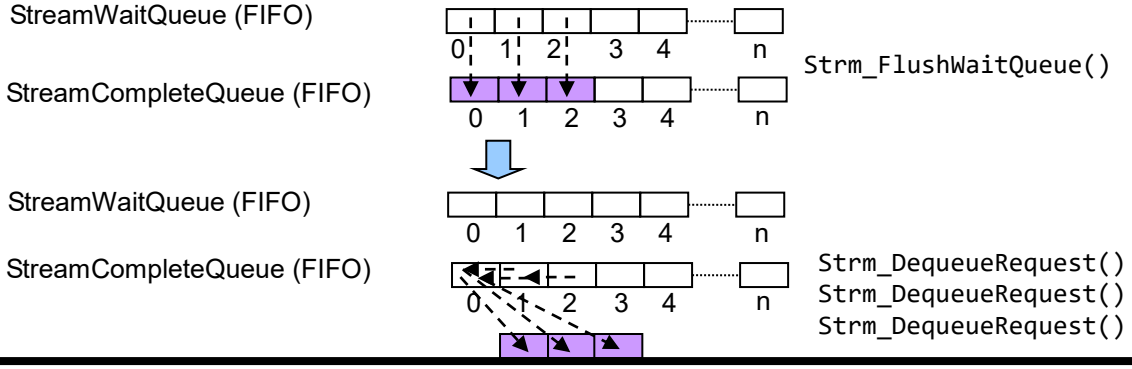
4.1.5.2.9. Finish streaming of TeliCamAPI (PC)

User application should clear StreamWaitQueue and StreamCompleteQueue before closing stream.

Call “`Strm_FlushWaitQueue()`” to move StreamRequests in StreamWaitQueue to StreamCompleteQueue, which will stop the stream receiving operation of TeliCamAPI.

Call “`Strm_DequeueRequest()`” until StreamCompleteQueue before closing the stream interface.

Stop streaming of TeliCamAPI (PC)



4.1.5.2.10. Release StreamRequest

Call "`Strm_ReleaseRequest`()" for each StreamRequest to release it.
User application can utilize image buffers used in the released StreamRequests until user application releases them.

4.1.5.2.11. Close the stream interface

Call "`Strm_Close`()" to terminate the stream interface.

4.1.6. CameraEvent control

CameraEvent means event or message sent from camera for notifying event occurred in the camera. This document uses naming “CameraEvent” to distinguish from event object of Windows and event as general meaning. “Event object” means an event object of Windows.

TeliCamAPI provides two ways to utilize CameraEvent, using High-Level API and using Low Level API. Using High-Level API will make source code simple and slim. Low-Level API allows software designers to organize event processing in the user’s arbitrary sequence.

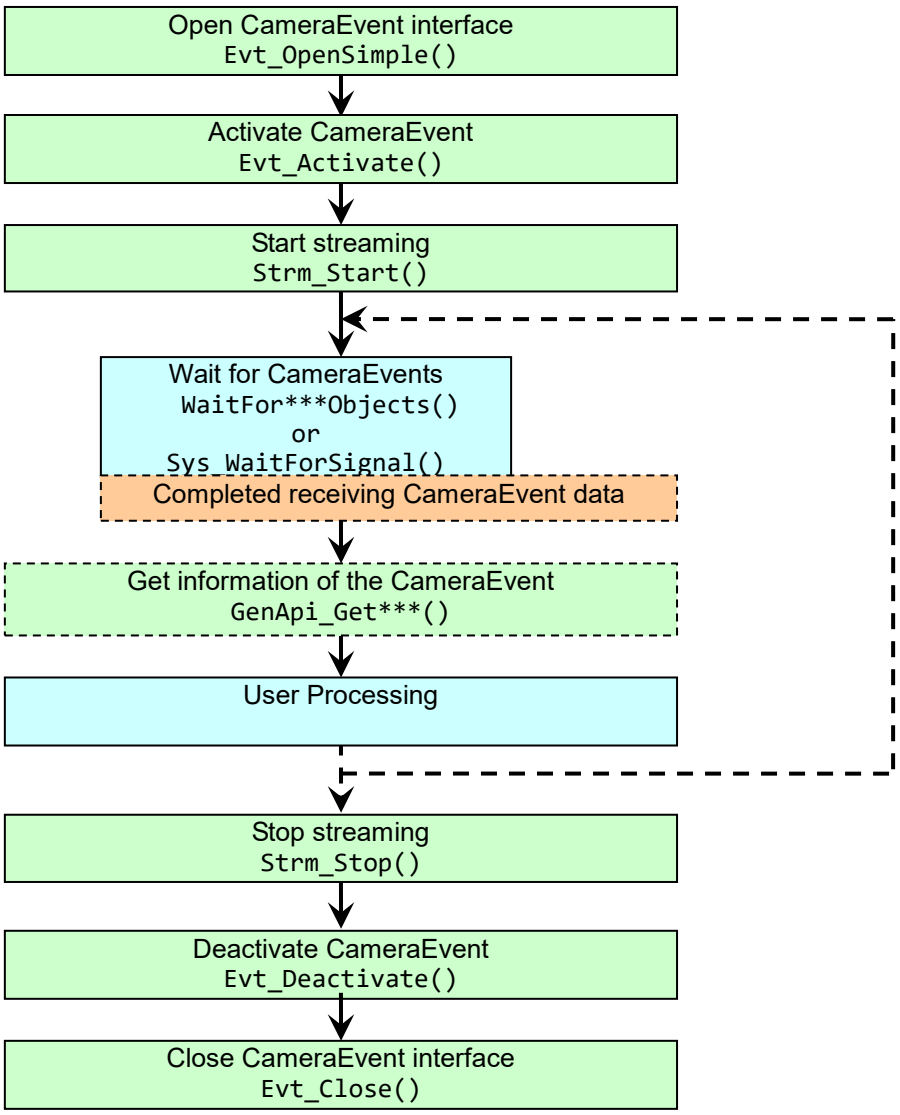
4.1.6.1. CameraEvent notification using High-Level API functions

“High-Level” means that these functions belong to layer near application layer. High-Level API functions allow software designers to make sourced code for processing CameraEvent simple and slim.

TeliCamSDK notifies reception of CameraEvent to user application through event(signal) objects.

High-Level API functions use GenICam GenApi inside them. If user application opened a camera with GenICam access disabled, High-Level API functions will not work.

The diagram below shows a general sequence.



4.1.6.1.1. Open CameraEvent interface

Call "`Evt_OpenSimple()`" to open a CameraEvent interface. "`Evt_OpenSimple()`" will return Event-Handle for the opened interface. "`Evt_OpenSimple()`" will also create EventRequest structures for holding received CameraEvent data, and create an EventRingBuffer for keeping EventRequests.

4.1.6.1.2. Activate CameraEvent

Call "`Evt_Activate()`" to activate a CameraEvents and register an event(signal) object which will be set signaled on reception of the CameraEvent. User application should create the event(signal) object beforehand.

4.1.6.1.3. Start streaming

Call "`Strm_Start()`" to instruct a camera to start acquiring images and start streaming.

On reception of new CameraEvent data, TeliCamAPI will replace buffer in EventRingBuffer with an EventRequest that contains the received CameraEvent data in background process.

TeliCamAPI will notify reception of the CameraEvent to user application through the event(signal) object specified on calling ""`Evt_Activate()`"

4.1.6.1.4. Acquire payload data of the CameraEvent

User application can get information in the received CameraEvent data using GenICam functions ("GenApi_GetIntValue()", and so on).

At the time of the release of this document, there are no accompanying information except the time stamp in USB3 Vision camera CameraEvents.

4.1.6.1.5. Stop streaming

Call "`Strm_Stop()`" to instruct stop acquiring image and stop streaming to the camera.

4.1.6.1.6. Deactivate CameraEvent

Call "`Evt_Deactivate()`" to instruct stop acquiring image and stop streaming to the camera.

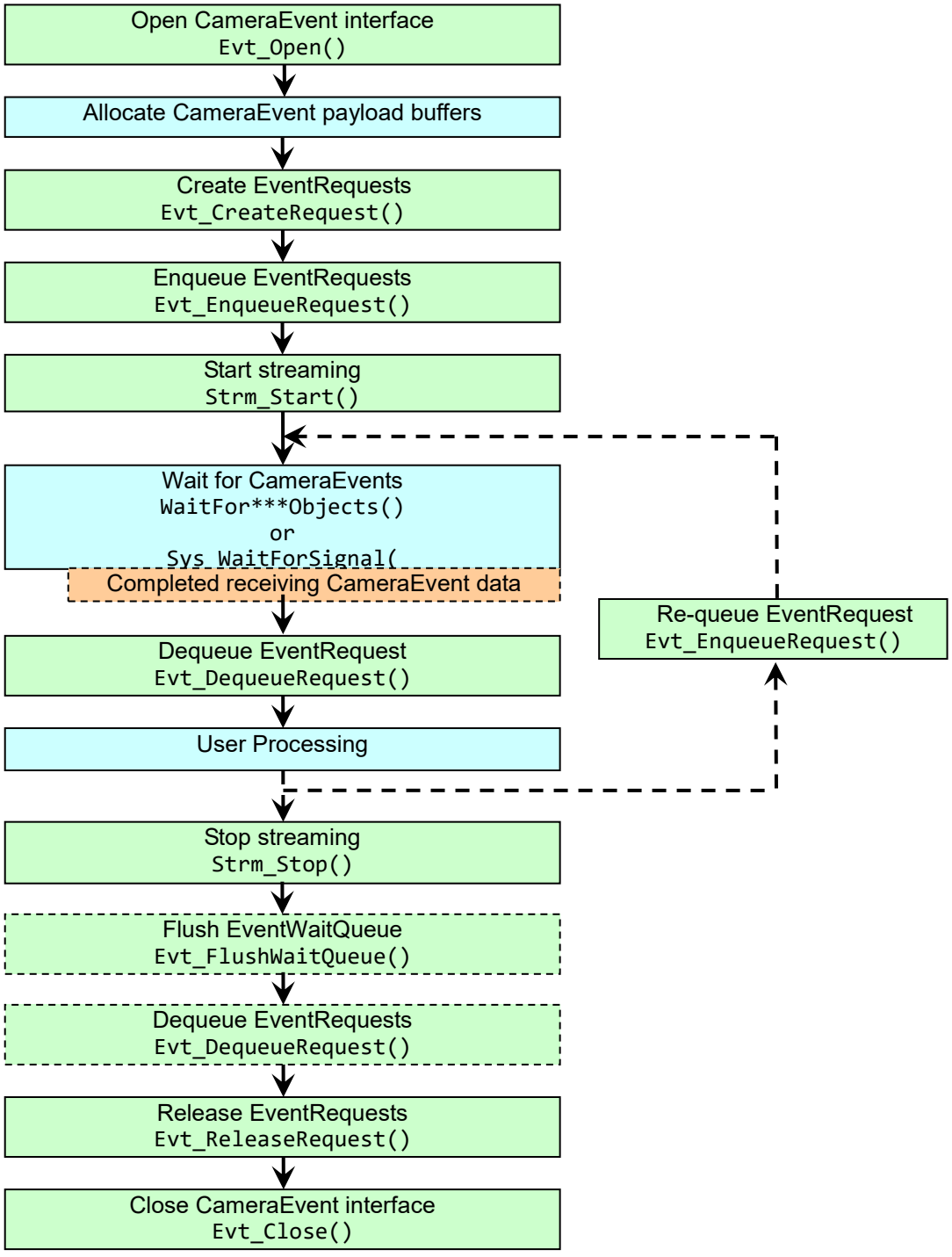
4.1.6.1.7. Close CameraEvent interface

Call "`Evt_Close()`" to terminate use of the CameraEvent interface.

4.1.6.2. Event notification using Low-Level API functions

"Low-Level" means that these functions belong to layer near transport layer. Low-Level API Functions allow software designers to organize a CameraEvent data receiving processing in the user's arbitrary sequence. This means that software designer must design various sequences used for receiving CameraEvent as user application code, for example, creating EventRequests, managing re-queue sequence of EventRequest, and so on. Advanced knowledge about controlling camera will be required for utilizing Low-Level API.

In the current version TeliCamAPI, Low-Level API supports USB3 Vision camera only.
The diagram below shows a general sequence.



4.1.6.2.1. Open CameraEvent interface

Call "`Evt_Open`()" to open a CameraEvent interface and to register event(signal) object for CameraEvent.
Event-Handle of the opened the event interface will be returned from "`Evt_Open`()"

4.1.6.2.2. Allocate CameraEvent payload buffer

User application must allocate payload buffer for storing received CameraEvent data, which will become member of EventRequest structures.

Preparing at least two EventRequests (and payload buffers inside them) is recommended to allow saving newly received CameraEvent during user processing for a received CameraEvent data.

User application should not release the payload buffers while the parent EventRequest is alive.

The following is steps for releasing payload buffers.

1. Call "`Evt_FlushWaitQueue`()" to move EventRequests in EventWaitQueue to EventCompleteQueue.
2. Call "`Evt_DequeueRequest`()" until EventCompleteQueue becomes empty.
3. Call "`Evt_ReleaseRequest`()" to release the parent EventRequest.
4. Release the payload buffer.

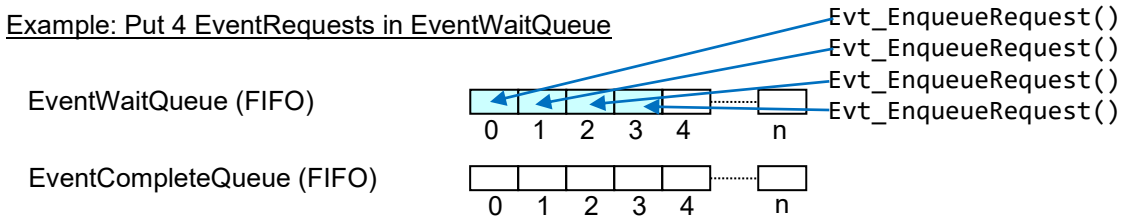
4.1.6.2.3. Create event request

Call "`Evt_CreateRequest`()" to create an EventRequest for receiving a CameraEvent data with the allocated payload buffer as a member of the created EventRequest.

User application should release all EventRequests and their children payload buffers before terminating the TeliCamAPI.

4.1.6.2.4. Put EventRequest in EventWaitQueue

Call "`Evt_EnqueueRequest`()" to put EventRequest in EventWaitQueue for receiving CameraEvent.

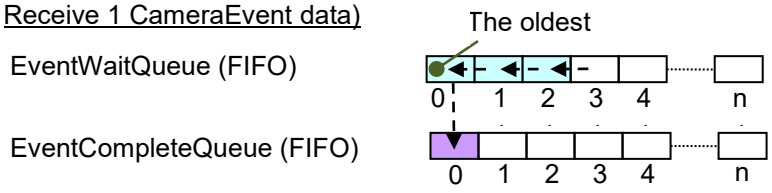


4.1.6.2.5. Start streaming

Call "`Strm_Start`()" to instruct a camera to start acquiring images and start streaming.

4.1.6.2.6. Receive CameraEvent data

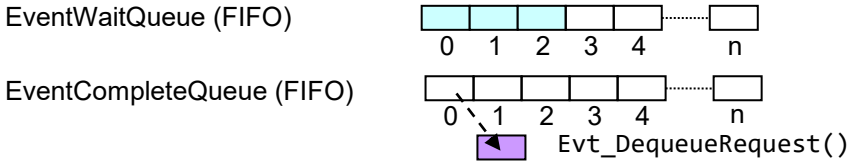
TeliCamAPI will save the incoming CameraEvent data to the oldest EventRequest in EventWaitQueue. When reception of a CameraEvent data is completed, TeliCamAPI will move the oldest EventRequest in EventWaitQueue to EventCompleteQueue, and set the event(signal) object for notifying reception of the CameraEvent signaled.



4.1.6.2.7. Extract EventRequest from EventCompleteQueue

Call "`Evt_DequeueRequest()`" to extract the oldest EventRequest from EventCompleteQueue, after receiving the CameraEvent. The extracted EventRequest will contain CameraEvent data,

Extract EventRequest from EventCompleteQueue



4.1.6.2.8. Stop streaming

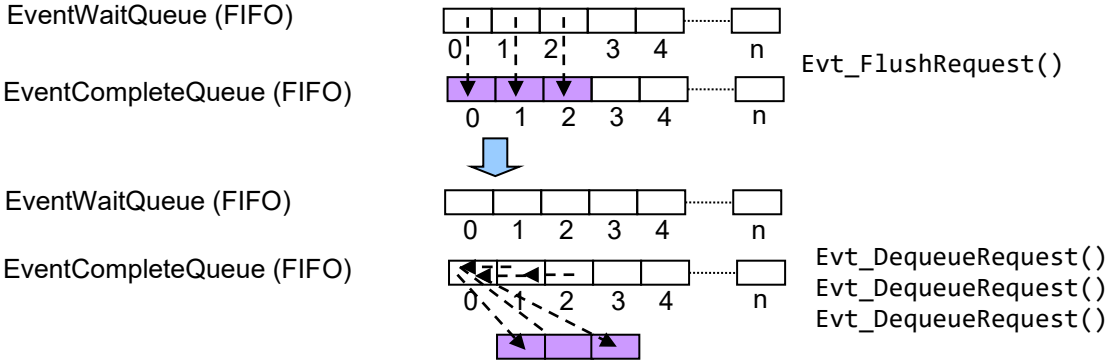
Call "`Strm_Stop()`" to instruct a camera to stop acquiring image and stop streaming.

4.1.6.2.9. Stop receiving CameraEvent data operation of TeliCamAPI (PC)

Call "`Evt_FlushWaitQueue()`" to move EventRequests in EventWaitQueue to EventCompleteQueue, which will stop the CameraEvent data receiving operation of TeliCamAPI.

Call "`Evt_DequeueRequest()`" until EventCompleteQueue becomes empty to prepare for closing CameraEvent interface.

Stop receiving CameraEvent data operation of TeliCamAPI (PC)



4.1.6.2.10. Release event request

Call "`Evt_ReleaseRequest()`" for each created EventRequest to release it.
User application can utilize Image buffers used in the released StreamRequests until they are released.

4.1.6.2.11. Close CameraEvent interface

Call "`Evt_Close()`" to terminate the CameraEvent interface.

4.1.7. Camera Access mode (Control Channel Privilege)

There are three modes of control channel privilege for Gig-E Vision cameras:

- Exclusive access mode (exclusive privilege)
  An application opened a camera in exclusive access mode can fully control the camera exclusively.
  The other applications cannot open the camera.

- Control access mode (control privilege)
  An application opened a camera in control access mode can fully control the camera.

The other applications can open the camera only in the open access mode.

- Open access mode (open privilege)
  The application opened a camera in open access mode can read registers on the camera, but cannot write data to them.

USB3 Vision cameras do not implement access mode feature.

4.1.8. Heartbeat process

Periodic access to the opened camera from the owner application, so called “Heartbeat sequence”, is necessary to maintain the obtained privilege in GigE Vision camera.

When user application opens a Gig-E Vision camera, TeliCamAPI will enable heartbeat process with 15 seconds heartbeat timeout. TeliCamAPI will check the interval of communication with the camera and perform dummy communication if necessary in background process.

Normally, User application does not need to care about heartbeat process.

However, when you interrupt processing for a long time due to debugs, etc. unless you change the heartbeat setting, a heartbeat timeout error will occur, and the obtained access privilege will be canceled which means that user application cannot access the camera when the interrupted processing is resumed. Disabling the heartbeat or lengthen the period of the heartbeat timeout using “`Cam_SetHeartbeat()`” will be useful for such a case.

USB3 Vision cameras do not implement heartbeat feature.

4.2. SDK Installation

Refer to the following document for the installation method of TeliCamSDK.
These documents are located in the “Documents” folder of the TeliCamSDK installation folder.

Windows : TeliCamSDK Start-up Guide Jpn.pdf
Linux : TeliCamSDK for Linux Getting Started Guide Jpn.pdf

4.3. SDK Uninstallation

Refer to the document described in section 4.2 for the uninstallation method of TeliCamSDK.

4.4. Set up development environment

4.4.1. Set up development environment under Windows

The following settings are necessary for developing an application using Visual Studio.
Settings may slightly differ depending on the version of Visual Studio.

- Add the following directory to compiler additional include directory.
  ([Configuration Property] – [C/C++] – [General] – [Additional Include Directory])

  “$(TeliCamSDK)TeliCamAPI\include”

- Add the following directory to linker additional library directory.
  ([Configuration Property] – [Linker] – [General] – [Additional Library Directory])

  32bit OS: “$(TeliCamSDK)TeliCamAPI\lib\x86”
  64bit OS: “$(TeliCamSDK)TeliCamAPI\lib\x64”

- Add the following file names to linker additional dependencies/.
  ([Configuration Property] – [Linker] – [General] – [Input] – [Additional Dependencies]

  32bit OS: TeliCamAPI.lib;TeliCamUtl.lib
  64bit OS: TeliCamAPI64.lib;TeliCamUtl64.lib

4.4.2. Set up development environment under Linux

To compile and run applications using TeliCamSDK, you must set the environment variables.

TELICAMSDK=/opt/TeliCamSDK
export TELICAMSDK

export
LD_LIBRARY_PATH=$TELICAMSDK/lib:$TELICAMSDK/genicam/bin/Linux64_x64:$LD_LIBRARY_PATH

This can be set by running a shell script.

Source /opt/TeliCamSDK/set_env.sh

Library files are located in the following directory.
/opt/TeliCamSDK/lib
Include files are located in the following directory.
/opt/TeliCamSDK/include
Using Makefile to compile applications, you must libraries and include settings.
For example:
g++ -c sample.cpp –I/opt/TeliCamSDK/include –L/opt/TeliCamSDK/lib –lTeliCamApi

Please refer to sample projects that are installed.

4.5. Performance tuning under Linux

TeliCamSDK for Linux raises the priority of the packet reception threads to minimize the variation of image grab time required.
However, non- superuser on Linux cannot raise the priority of a thread.

If you want to run applications that require high performance, do one of the following methods to raise the priority of a thread:

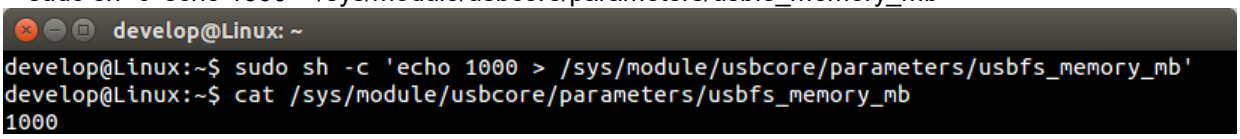
- Run applications as the superuser.
- Modify the system configuration to allow users to change the priority of real-time processes :
  You can do this using the pam_limits module of the Pluggable Authentication Modules (PAM). By default values are specified in /etc/security/limits.conf.
  For example, the following line allows all users to change the priority of real-time processes.
  * - rtprio 99
  You must restart the system after modifying the configuration file.

- If you are using many USB3 Vision cameras or if CAM_API_STS_IO_DEVICE_ERROR occurs, change usbfs memory limit.

The setting method is as follows.

1. Open a terminal window (gnome-terminal).
2. Confirm the current value.

3. Change the memory limit.
   For example: To change the memory limit to 1000 [MB]
   sudo sh -c 'echo 1000 > /sys/module/usbcore/parameters/usbfs_memory_mb'



or

Please edit “/sys/module/usbcore/parameters/usbfs_memory_mb” in the editor.

*After rebooting, the set value is 16 [MB]. Please set again.

•If the CAM_API_STS_TOO_MANY_PACKET_MISSING error (error code: 0x100C) occurs while using the GigE Vision camera, make the following settings.

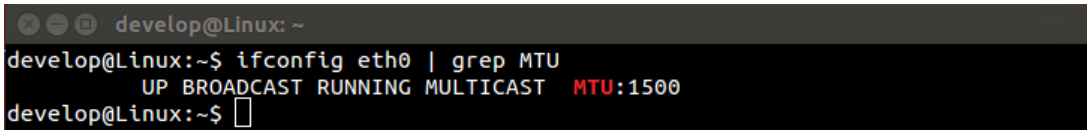
- Jumbo frame setting value
- Receive buffer size
- Packet setting of network interface card (NIC)
- HUB (QoS setting)

Please try the following method.

• Jumbo frame

For ifconfig command:

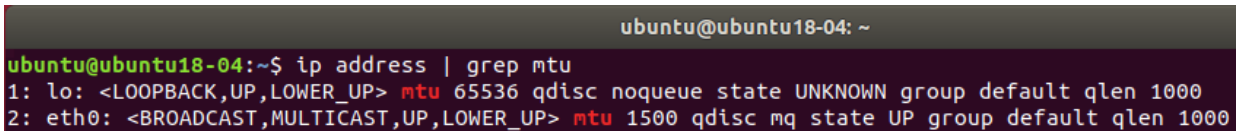
1. Open a terminal window (gnome-terminal).
2. Confirm the current value.
   ifconfig [interface] | grep MTU



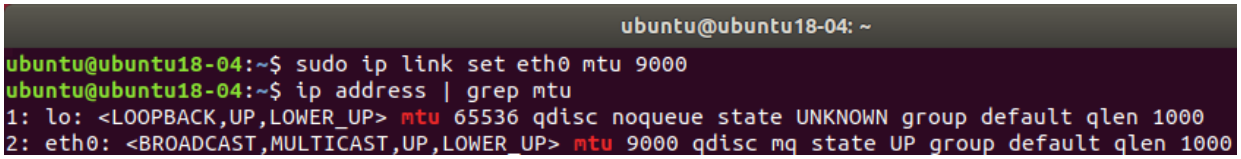
3. Change jumbo frame setting.
   For example:
   sudo ifconfig [interface] mtu 9000

For ip command:

1. Open a terminal window (gnome-terminal).
2. Confirm the current value.
   ip address | grep mtu



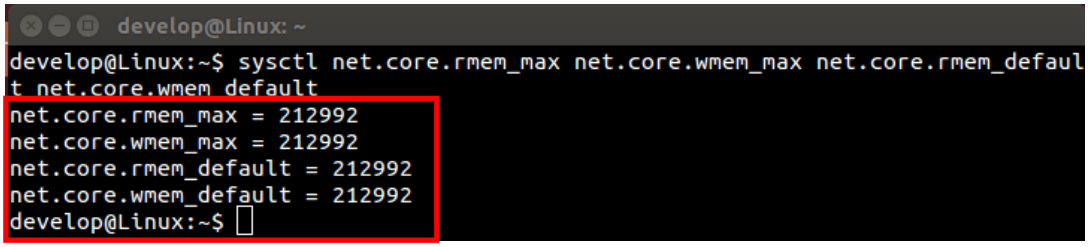
3. Change jumbo frame setting.
   For example:
   sudo ip link set [interface] mtu 9000



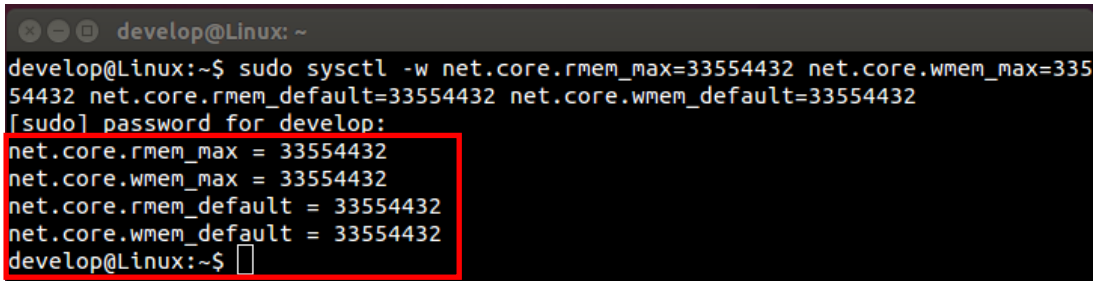
•

Receiving buffer size

1. Open a terminal window (gnome-terminal).
2. Confirm the current value.
   sysctl net.core.rmem_max net.core.wmem_max net.core.rmem_default net.core.wmem_default

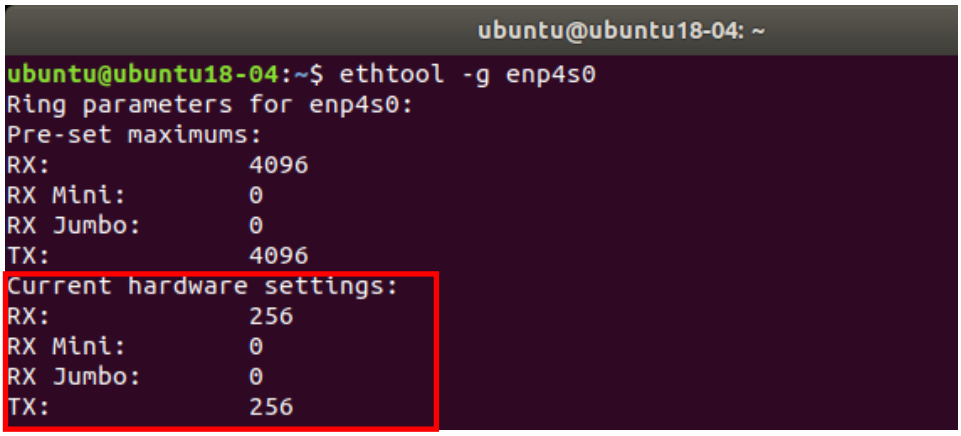


3. Change Receiving buffer size setting.
   For example:
   sysctl -w net.core.rmem_max=33554432 net.core.wmem_max=33554432
   net.core.rmem_default=33554432 net.core.wmem_default=33554432

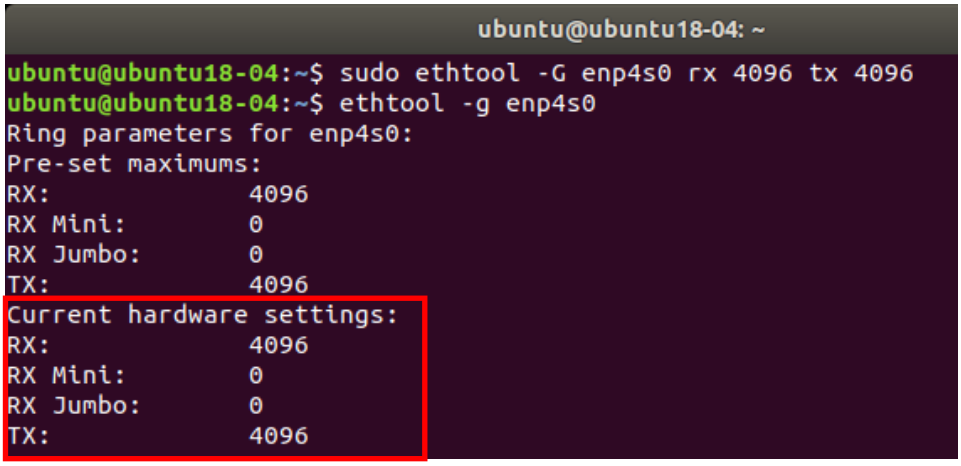


• Receiving buffer size

1. Open a terminal window (gnome-terminal).
2. Confirm the current value.
   ethtool -g [interface]



3. Change Network Interface Cards (NICs) packet setting.
   For example:
   ethtool -G [interface] rx 4096 tx 4096



• HUB(QoS setting)

Please set the priority of your HUB port to the highest priority.
(For the setting method, please refer to the manual of the HUB you are using.
QoS settings may not be available in the HUB you are using.)

5. Library functions

5.1. System functions

5.1.1. Sys_Initialize

This function initializes TeliCamAPI system.

[Syntax]

```
CAM_API_STATUS  Sys_Initialize (
  CAM_TYPE      eCamType = CAM_TYPE_ALL
);
```

[Parameters]

Parameter		Description
eCamType	[in]	Interface type of camera to use in user application. This argument is optional. Specify "CAM_TYPE_ALL" or specify nothing to use all types that TeliCamAPI supports.

[*CAM_TYPE* Enumeration]

Member	Description
CAM_TYPE_UNKNOWN	Unknown type. Don't use this as parameter of Sys_Initialize().
CAM_TYPE_U3V	USB3 Vision Camera.
CAM_TYPE_GEV	GigE Vision Camera.
CAM_TYPE_ALL	All types.

[Return value]

Returns result status. Refer to 5.8 Status code.

[Remarks]

User application must call this function once, before calling any other functions in TeliCamAPI.
"|" (OR) is available for specifying plural types of camera to *eCamType*. The following is example.

```
Sys_Initialize(CAM_TYPE_U3V | CAM_TYPE_GEV);
```

When "Sys_Initialize()" failed to load dll (TeliU3vApi2.dll or TeliGevApi2.dll) which controls the specified type of camera, this function will return CAM_API_STS_UNSUCCESSFUL.

When no dlls are loaded successfully even though CAM_TYPE_ALL was specified to *eCamType*, this function will return CAM_API_STS_UNSUCCESSFUL.

[Example]

Refer to example of 5.1.3 Sys_GetInformation.

5.1.2. Sys_Terminate

This function finalizes DLLs and drivers.

[Syntax]

```
CAM_API_STATUS  Sys_Terminate (void);
```

[Return value]

Returns result status. Refer to 5.8 Status code.

[Remarks]

User application must call this function once, before closing itself.

[Example]

Refer to example of 5.1.3 Sys_GetInformation.

5.1.3. Sys_GetInformation

This function gets the information of TeliCamAPI system.

[Syntax]

```
CAM_API_STATUS  Sys_GetInformation (
  CAM_SYSTEM_INFO      *psSysInfo
);
```

[Parameters]

Parameter		Description
psSysInfo	[in]	A pointer to a variable that receives the acquired system information.

[CAM_SYSTEM_INFO structure]

```
typedef struct _CAM_SYSTEM_INFO {
    U3V_SYSTEM_INFO      sU3vInfo;
    GEV_SYSTEM_INFO      sGevInfo;
    char                 szDllVersion[MAX_INFO_STR];
} CAM_SYSTEM_INFO, *PCAM_SYSTEM_INFO;
```

Member		Description
sU3vInfo	[out]	Structure containing system information of U3vAPI.
sGevInfo	[out]	Structure containing system information of GevAPI.。
szDllVersion	[out]	Version string of TeliCamAPI.

[U3V_SYSTEM_INFO structure]

```
typedef struct _U3V_SYSTEM_INFO {
    char      szDriverVersion[MAX_INFO_STR];
    char      szDllVersion[MAX_INFO_STR];
    char      szDllExVersion[MAX_INFO_STR];
} U3V_SYSTEM_INFO, *PU3V_SYSTEM_INFO;
```

Member		Description
szDriverVersion	[out]	Version string of USB3 Vision driver (TeliU3vDriver.sys). If no USB3 Vision cameras are connected, this will be blank.
szDllVersion	[out]	Version string of USB3 Vision API (TeliU3vApi.dll).
szDllExVersion	[out]	Version string of USB3 Vision API extention(TeliU3vCamApi.dll).

[GEV_SYSTEM_INFO structure]

```
typedef struct _GEV_SYSTEM_INFO {
      char    szDriverVersion[MAX_INFO_STR];
      char    szDllVersion[MAX_INFO_STR];
      char    szDllExVersion[MAX_INFO_STR];
} GEV_SYSTEM_INFO, *PGEV_SYSTEM_INFO;
```

Member		Description
szDriverVersion	[out]	Version string of GigE Vision driver (TeliGevDriver.sys). Even if no cameras are connected, version will be indicated.
szDllVersion	[out]	Version string of GigE Vision API (TeliGevApi.dll).
szDllExVersion	[out]	Version string of GigE Vision API extention(TeliGevCamApi.dll).

[Return value]

Returns result status. Refer to 5.8 Status code.

[Remarks]

Including TeliCamAPI.h is required.

[Example]

C++

```
CAM_API_STATUS      uiStatus;
CAM_SYSTEM_INFO     sSysInfo;
uint32_t            uiNum;

// Initialize system.
uiStatus = Sys_Initialize();
if (uiStatus != CAM_API_STS_SUCCESS)
    return -1;

// Get SDK system information.
uiStatus = Sys_GetInformation(&sSysInfo);
if (uiStatus != CAM_API_STS_SUCCESS)
    return -1;

printf(" <System information>\n");
printf("  TeliU3vDriver.sys  version : %s\n", sSysInfo.sU3vInfo.szDriverVersion);
printf("  TeliU3vApi2.dll    version : %s\n", sSysInfo.sU3vInfo.szDllVersion);
printf("  TeliU3vCamApi.dll  version : %s\n", sSysInfo.sU3vInfo.szDllExVersion);
printf("  TeliGevDriver.sys  version : %s\n", sSysInfo.sGevInfo.szDriverVersion);
printf("  TeliGevApi2.dll    version : %s\n", sSysInfo.sGevInfo.szDllVersion);
printf("  TeliGevCamApi.dll  version : %s\n", sSysInfo.sGevInfo.szDllExVersion);
printf("  TeliCamAPI.dll     version : %s\n", sSysInfo.szDllVersion);

// Get number of cameras.
uiStatus = Sys_GetNumOfCameras(&uiNum);
if (uiStatus != CAM_API_STS_SUCCESS)
    return -1;

printf("%d camera(s) found.\n", uiNum);

// Terminate system.
Sys_Terminate();
```


5.1.4. Sys_GetNumOfCameras

This function retrieves connected cameras and makes list of the available cameras inside TeliCamAPI, and reports number of the available cameras.

[Syntax]

```
CAM_API_STATUS  Sys_GetNumOfCameras (
  uint32_t      *puiNum
);
```

[Parameters]

Parameter		Description
puiNum	[out]	A pointer to a variable that receives number of detected cameras.

[Return value]

Returns result status. Refer to 5.8 Status code.

[Remarks]

Camera index, whose value is up to (**puiNum* − 1) from zero, is assigned to each detected camera for identifying it.

User application cannot open camera before calling this function, because camera list inside TeliCamAPI is not ready.

Including TeliCamAPI.h is required.

[Example]

Refer to example of 5.1.3 Sys_GetInformation.

5.1.5. Sys_CreateSignal

This function creates a signal object and returns a signal object handle.

[Syntax]

For Windows

```
CAM_API_STATUS  Sys_CreateSignal (
  HANDLE    *phHandle
);
```

For Linux

```
CAM_API_STATUS  Sys_CreateSignal (
  SIGNAL_HANDLE     *phHandle
);
```

[Parameters]

Parameters		Description
phHandle	[out]	Pointer to a signal object handle.

[Return value]

Returns result status. Refer to 5.8 Status code.

[Remarks]

A signal object is used to wait for signal.
A signal object created by this function must be closed by Sys_CloseSignal function.

This function is similar to CreateEvent() function of WINAPI.
In the Windows version, you can also use CreatEvent() to create the handle of the signal object.

Including TeliCamAPI.h is required.

5.1.6. Sys_CloseSignal

This function closes a signal object.

[Syntax]

For Windows

```
CAM_API_STATUS  Sys_CloseSignal (
  HANDLE            hHandle
);
```

For Linux

```
CAM_API_STATUS  Sys_CloseSignal (
  SIGNAL_HANDLE     hHandle
);
```

[Parameters]

Parameters		Description
hHandle	[in]	A signal object handle.

[Return value]

Returns result status. Refer to 5.8 Status code.

[Remarks]

This function is similar to CloseHandle() function of WINAPI.
In the Windows version, you can also use CloseHandle() to close the the signal object.

Including TeliCamAPI.h is required.

5.1.7. Sys_WaitForSignal

This function checks the current state of the specified signal object.
If the signal object is signaled, it returns immediately. At this time, the signal object will return to the nonsignaled state.
If the signal object is nonsignaled, the calling thread enters the wait state until the signal object is signaled or the time-out interval elapses.

[Syntax]

For Windows

```
CAM_API_STATUS  Sys_WaitForSignal (
  HANDLE           hHandle,
  uint32_t         uiMilliseconds
);
```

For Linux

```
CAM_API_STATUS  Sys_WaitForSignal (
  SIGNAL_HANDLE    hHandle,
  uint32_t         uiMilliseconds
);
```

[Parameters]

Parameters		Description
hHandle	[in]	A signal object handle.
uiMilliseconds	[in]	The time-out interval, in milliseconds. If a nonzero value is specified, this function waits until the specified signal object is signaled or the interval elapses. If uiMilliseconds is zero, the function does not enter a wait state if the specified signal object is not signaled.(It always returns immediately.) If uiMilliseconds is CAM_SIGNAL_TIMEOUT_INFINITE (0xFFFFFFFF) , this function will return only when the specified object is signaled.

[Return value]

If this function succeeds, the return value is as follows:

CAM_API_STS_SUCCESS	The state of the specified signal object is signaled.
CAM_API_STS_TIMEOUT	The time-out interval elapsed. The state of the specified signal object is nonsignaled.
CAM_API_STS_UNSUCCESSFUL	The function has failed.

[Remarks]

This function is similar to WaitForSingleObject() function of WINAPI.
In the Windows version, you can also use WaitForSingleObject() to check the current status of the signal object.

Including TeliCamAPI.h is required.

5.1.8. Sys_ResetSignal

This function sets the specified signal object to the nonsignaled state.

[Syntax]

For Windows

```
CAM_API_STATUS  Sys_ResetSignal (
  HANDLE            hHandle
);
```

For Linux

```
CAM_API_STATUS  Sys_ResetSignal (
  SIGNAL_HANDLE     hHandle
);
```

[Parameters]

Parameters		Description
hHandle	[in]	A signal object handle.

[Return value]

Returns result status. Refer to 5.8 Status code.

[Remarks]

This function is similar to ResetEvent() function of WINAPI.
In the Windows version, you can also use ResetEvent() to reset the signal object.

Including TeliCamAPI.h is required.

5.2. Camera functions

5.2.1. Cam_GetInformation

This function reports information of specified camera.

[Syntax]

```
CAM_API_STATUS  Cam_GetInformation (
  CAM_HANDLE        hCam,
  uint32_t          uiCamIdx,
  CAM_INFO          *psCamInfo
);
```

[Parameters]

Parameter		Description
hCam	[in]	Handle of the already opened camera. If the camera is not opened yet, specify index of camera to *uiCamIndex* and specify NULL to *hCam*.
uiCamIdx	[in]	Index of camera. Index value should be up to number of detected cameras minus 1, from zero . This parameter will be ignored when *hCam* parameter is not NULL.
psCamInfo	[out]	A pointer to a variable that receives information of the camera.

[CAM_INFO structure]

```
typedef struct _CAM_INFO {
      CAM_TYPE            eCamType;
      char                szManufacturer[MAX_INFO_STR];
      char                szModelName[MAX_INFO_STR];
      char                szSerialNumber[MAX_INFO_STR];
      char                szUserDefinedName[MAX_INFO_STR];
      U3V_CAM_INFO        sU3vCamInfo;
      GEV_CAM_INFO        sGevCamInfo;
} CAM_INFO, *PCAM_INFO;
```

Member		Description
eCamType	[out]	Interface type of camera.
szManufacturer	[out]	Name of manufacturer.
szModelName	[out]	Camera model name.
szSerialNumber	[out]	Serial number of the camera.
szUserDefinedName	[out]	Name that user defined.
sU3vCamInfo	[out]	Structure which contains information of USB 3 vision camera.
sGevCamInfo	[out]	Structure which contains information of GigE Vision camera.

[U3V_CAM_INFO structure]

```
typedef struct _U3V_CAM_INFO {
      char        szFamilyName[MAX_INFO_STR];
      char        szDeviceVersion[MAX_INFO_STR];
      char        szManufacturerInfo[MAX_INFO_STR];
      uint32_t    uiAdapterVendorId;
      uint32_t    uiAdapterDeviceId;
      uint32_t    uiAdapterDfltMaxPacketSize;
} U3V_CAM_INFO, *PU3V_CAM_INFO;
```

Member		Description
szFamilyName	[out]	Name of camera family.
szDeviceVersion	[out]	Version of the camera hardware.
szManufacturerInfo	[out]	String that shows Information of manufacturer.
uiAdapterVendorId	[out]	Vendor ID of USB controller chip, which is used in USB adapter that the USB3 Vision cameras are connected.
uiAdapterDeviceId	[out]	Device ID of USB controller chip, which is used in USB adapter that the USB3 Vision cameras are connected.
uiAdapterDfltMaxPacketSize	[out]	Recommended "MaxPacketSize" parameter value for USB adapter that USB3 Vision cameras are connected. This value is read out from data table inside TeliCamAPI whose search kety is Vender ID of the USB chip. If 0 is specified to argument "uiMaxPacketSize" on calling "Strm_OpenSimple()" or "Strm_Open()", this value will be used as "MaxPacketSize" parameter value.

[GEV_CAM_INFO structure]

```
typedef struct _GEV_CAM_INFO {
      char        szDisplayName[512];
      uint8_t     aucMACAddress[6];
      int8_t      cSupportIP_LLA;
      int8_t      cSupportIP_DHCP;
      int8_t      cSupportIP_Persistent;
      int8_t      cCurrentIP_LLA;
      int8_t      cCurrentIP_DHCP;
      int8_t      cCurrentIP_Persistent;
      uint8_t     aucIPAddress[4];
      uint8_t     aucSubnet[4];
      uint8_t     aucGateway[4];
      uint8_t     aucAdapterMACAddress[6];
      uint8_t     aucAdapterIPAddress[4];
      uint8_t     aucAdapterSubnet[4];
      uint8_t     aucAdapterGateway[4];
      char        szAdapterDisplayName[1024];
} GEV_CAM_INFO, *PGEV_CAM_INFO;
```

Member		Description
szDisplayName	[out]	Name of camera for displaying purpose.
aucMACAddress	[out]	MAC Address of the camera.
cSupportIP_LLA	[out]	Shows if the camera supports Link Local Adderss IP configuration sheme. 0: Not supported, Other than 0: Supported.
cSupportIP_DHCP	[out]	Shows if the camera supports DHCP IP configuration sheme. 0: Not supported, Other than 0: Supported.
cSupportIP_Persistent	[out]	Shows if the camera supports Persistent-IP IP configuration sheme. 0: Not supported, Other than 0: Supported.
cCurrentIP_LLA	[out]	Shows if Link Local Adderss is active or not. 0: Inactive, Other than 0: Active.
cCurrentIP_DHCP	[out]	Shows if DHCP is active or not. 0: Inactive, Other than 0: Active.
cCurrentIP_Persistent	[out]	Shows if Persistent-IP is active or not. 0: Inactive, Other than 0: Active.
aucIPAddress	[out]	Current IP address setting of the camera.
aucSubnet	[out]	Current subnet mask setting of the camera.
aucGateway	[out]	Current default gateway setting of the camera.
aucAdapterMACAddress	[out]	MAC address of the network adapter that GigE Vision cameras are connected.
aucAdapterIPAddress	[out]	Current IP address of the network adapter that GigE Vision cameras are connected.
aucAdapterSubnet	[out]	Current subnet mask of the network adapter that GigE Vision cameras are connected.
aucAdapterGateway	[out]	Current default gateway of the network adapter that GigE Vision cameras are connected.
szAdapterDisplayName	[out]	Name of network adapter for displaying purpose.

[Return value]

Returns result status. Refer to 5.8 Status code.

[Remarks]

If you have updated the camera list by calling the "Sys_GetNumOfCameras()", specify the NULL to hCam and specify the index of the camera to uiCamIdx.
Including TeliCamAPI.h is required.

[Example]

C++

```
CAM_API_STATUS   uiStatus;
CAM_INFO         sCamInfo;
U3V_CAM_INFO     *psU3vCamInfo;
GEV_CAM_INFO     *psGevCamInfo;
uint32_t         uiNum;

// Initialize system.
uiStatus = Sys_Initialize();
if (uiStatus != CAM_API_STS_SUCCESS)
    return -1;

// Get number of cameras.
uiStatus = Sys_GetNumOfCameras(&uiNum);
if (uiStatus != CAM_API_STS_SUCCESS)
    return -1;

// Get information of a camera.
for (uint32_t i = 0; i < uiNum; i++) {
    memset((void*)&sCamInfo, 0, sizeof(CAM_INFO));

    uiStatus = Cam_GetInformation((CAM_HANDLE)NULL, i, &sCamInfo);
    if (uiStatus != CAM_API_STS_SUCCESS)
        return -1;

    printf("\n<Camera%d information>\n", i);
    if (sCamInfo.eCamType == CAM_TYPE_U3V)
        printf(" Type                                : USB3 Vision camera\n");
    else if (sCamInfo.eCamType == CAM_TYPE_GEV)
        printf(" Type                                : GigE Vision camera\n");

    printf(" Manufacturer                        : %s\n", sCamInfo.szManufacturer);
    printf(" Model name                          : %s\n", sCamInfo.szModelName);
    printf(" Serial number                       : %s\n", sCamInfo.szSerialNumber);
    printf(" User defined name                   : %s\n", sCamInfo.szUserDefinedName);

    if (sCamInfo.eCamType == CAM_TYPE_U3V) {
        psU3vCamInfo = &sCamInfo.sU3vCamInfo;

        printf(" U3v family name                     : %s\n",
            psU3vCamInfo->szFamilyName);
        printf(" U3v device version                  : %s\n",
            psU3vCamInfo->szDeviceVersion);
        printf(" U3v manufacturer information        : %s\n",
            psU3vCamInfo->szManufacturerInfo);
        printf(" U3v adapter vendor ID               : 0x%04X\n",
            psU3vCamInfo->uiAdapterVendorId);
        printf(" U3v adapter device ID               : 0x%04X\n",
            psU3vCamInfo->uiAdapterDeviceId);
        printf(" U3v Adapter default MaxPacketSize   : %d\n",
            psU3vCamInfo->uiAdapterDfltMaxPacketSize);
    } else if (sCamInfo.eCamType == CAM_TYPE_GEV) {
        psGevCamInfo = &sCamInfo.sGevCamInfo;

        printf(" Gev display name                    : %s\n",
            psGevCamInfo->szDisplayName);
        printf(" Gev MAC address                     : %02X-%02X-%02X-%02X-%02X-%02X\n",
            psGevCamInfo->aucMACAddress[0],
            psGevCamInfo->aucMACAddress[1],
            psGevCamInfo->aucMACAddress[2],
            psGevCamInfo->aucMACAddress[3],
            psGevCamInfo->aucMACAddress[4],
            psGevCamInfo->aucMACAddress[5]);
        printf(" Gev support IP LLA                  : %d\n",
            psGevCamInfo->cSupportIP_LLA);
        printf(" Gev support IP DHCP                 : %d\n",
            psGevCamInfo->cSupportIP_DHCP);
```

```
            printf(" Gev Support IP Persistent-IP       : %d\n",
                psGevCamInfo->cSupportIP_Persistent);
            printf(" Gev current IP LLA                 : %d\n",
                psGevCamInfo->cCurrentIP_LLA);
            printf(" Gev current IP DHCP                : %d\n",
                psGevCamInfo->cCurrentIP_DHCP);
            printf(" Gev current IP Persistent-IP       : %d\n",
                psGevCamInfo->cCurrentIP_Persistent);
            printf(" Gev IP Address                     : %d.%d.%d.%d\n",
                psGevCamInfo->aucIPAddress[0],
                psGevCamInfo->aucIPAddress[1],
                psGevCamInfo->aucIPAddress[2],
                psGevCamInfo->aucIPAddress[3]);
            printf(" Gev subnet mask                    : %d.%d.%d.%d\n",
                psGevCamInfo->aucSubnet[0],
                psGevCamInfo->aucSubnet[1],
                psGevCamInfo->aucSubnet[2],
                psGevCamInfo->aucSubnet[3]);
            printf(" Gev default gateway                : %d.%d.%d.%d\n",
                psGevCamInfo->aucGateway[0],
                psGevCamInfo->aucGateway[1],
                psGevCamInfo->aucGateway[2],
                psGevCamInfo->aucGateway[3]);
            printf(" Gev adapter MAC address            : %02X-%02X-%02X-%02X-%02X-%02X\n",
                psGevCamInfo->aucAdapterMACAddress[0],
                psGevCamInfo->aucAdapterMACAddress[1],
                psGevCamInfo->aucAdapterMACAddress[2],
                psGevCamInfo->aucAdapterMACAddress[3],
                psGevCamInfo->aucAdapterMACAddress[4],
                psGevCamInfo->aucAdapterMACAddress[5]);
            printf(" Gev adapter IP address             : %d.%d.%d.%d\n",
                psGevCamInfo->aucAdapterIPAddress[0],
                psGevCamInfo->aucAdapterIPAddress[1],
                psGevCamInfo->aucAdapterIPAddress[2],
                psGevCamInfo->aucAdapterIPAddress[3]);
            printf(" Gev adapter subnet mask            : %d.%d.%d.%d\n",
                psGevCamInfo->aucAdapterSubnet[0],
                psGevCamInfo->aucAdapterSubnet[1],
                psGevCamInfo->aucAdapterSubnet[2],
                psGevCamInfo->aucAdapterSubnet[3]);
            printf(" Gev adapter default gateway        : %d.%d.%d.%d\n",
                psGevCamInfo->aucAdapterGateway[0],
                psGevCamInfo->aucAdapterGateway[1],
                psGevCamInfo->aucAdapterGateway[2],
                psGevCamInfo->aucAdapterGateway[3]);
            printf(" Gev adapter display name           : %s\n",
                psGevCamInfo->szAdapterDisplayName);
        }
    }

    // Terminate system.
    Sys_Terminate();
```

5.2.2. Cam_Open

This function opens a camera specified by index of the camera (*uiCamIdx*).
Only in the Windows version, user application can open cameras that the other applications are using.
User application will fail in opening camera stream or camera event if the other application is using it.

[Syntax]

For Windows

```
CAM_API_STATUS  Cam_Open (
  uint32_t          uiCamIdx,
  CAM_HANDLE        *phCam,
  HANDLE            hRmv = NULL,
  bool8_t           bUseGenICam = true,
  void              *pvXml = NULL,
  CAM_ACCESS_MODE   eAccessMode = CAM_ACCESS_MODE_CONTROL
);
```

For Linux

```
CAM_API_STATUS  Cam_Open (
  uint32_t          uiCamIdx,
  CAM_HANDLE        *phCam,
  SIGNAL_HANDLE     hRmv = NULL,
  bool8_t           bUseGenICam = true,
  void              *pvXml = NULL,
  CAM_ACCESS_MODE   eAccessMode = CAM_ACCESS_MODE_CONTROL
);
```

[Parameters]

Parameter		Description
uiCamIdx	[in]	Index of the camera. Index value should be up to number of detected cameras minus 1, from zero. Number of detected camra can be retrieved using "Sys_GetNumOfCameras()".
phCam	[out]	A pointer to a variable that receives Camera-Handle assigned to the opening camera.
hRmv	[in]	Handle of event(signal) object for notifying disconnection of the camera. This event(signal) object will also be set signaled, on Heartbeat timeout in GigE Vision camera. Under Windows, event(signal) object is created by Sys_CreateSignal() or CreateEvent() of Win32 API. Under Linux, event(signal) object is created by Sys_CreateSignal(). Specify NULL or do not specify argument, if notification of disconnection event is not necessary.
bUseGenICam	[in]	Flag for enabling GenICam function. If true, TeliCamAPI will enable GenICam function after loading camera description file (Xml file). If false, TeliCamAPI does not load camera description file (Xml file) and

Parameter	Description
	GenICam function will be disabled. All APIs in section 5.4.1, 5.6 and most APIs in section 5.5 use GenICam function inside. We recommend users to specify true to this parameter, except there is solid reason to disable GenICam function. If this parameter is not specified, true will be used as parameter value.
pvXml [in]	A pointer to camera description data (Xml data). Camera description data will be loaded from data buffer that the argument points. If NULL is specified or this argument is not specified, data will be loaded from camera. This parameter will be ignored when *bUseGenICam* is false,.
eAccessMode [in]	Access mode(Control Channel Privilege) of the GigE Vision camera. TeliCamAPI will try to acquire the specified privilrge. If no value is specified, CAM_ACCESS_MODE_CONTROL will be used as argument value. This parameter will be ignored when type of camera is not GigE Vision.

[*CAM_ACCESS_MODE* Enumeration]

Member	Description
CAM_ACCESS_MODE_EXCLUSIVE	User application can fully control the camera. The other application cannot open the camera.
CAM_ACCESS_MODE_CONTROL	User application can fully control the camera. The other application can read registers of the camera, cannot write data.
CAM_ACCESS_MODE_OPEN	User application can read registers of thecamer, cannot write data.

[Return value]

Returns result status. Refer to 5.8 Status code.

[Remarks]

In USB3 Vision camera case, this function clears "StreamEnable" node (SIControl register) and "EventEnable" node (EIControl register) to 0 (Disable) to stop image streaming and CameraEvent notification, if no other applications are using the camera.

In GigE Vision camera case, TeliCamAPI will enable Heartbeat process on opening camera, with Heartbeat timeout value 15sec. The camera and user application will periodically check the existence of communication in their own Heartbeat process. User application may send dummy command to check communication state if necessary. When no communication state continued, camera or TeliCamAPI will judge that Heartbeat timeout error occurred. The camera will cancel the privilege given to the host application, on Heartbeat timeout. User application will make hRmv event(signal) object signaled on Heartbeat timeout.

If user application is interrupted for a long time, for example, debugging case and so on, the camera

will judge that Heartbeat timeout occurred and will cancel control channel privilege, which will cause the situation that the application cannot access the camera after the application resumed.

By disabling Heartbeat process or setting log timeout value using "Cam_SetHeartbeat()" API, camera will not judge that Heartbeat timeout occurred.

Refer to 4.1.7 Camera Access mode (Control Channel Privilege) about Control Channel Privilege.
Refer to 4.1.8 Heartbeat process about Heartbeat process.

When true is specified to *bUseGenICam* parameter, TeliCamAPI will load camera description file data (XML file) and prepare internal data for enabling GenICam function. If it is the first time that the target camera is used in the user application, it may take time for loading and preparing data.

When false is specified to *bUseGenICam* parameter, processing time of opening camera will be shortened, but 5.4.1 High-level API functions, 5.6 GenICam functions and several APIs in section 5.5 Controlling camera feature functions will become unavailable.

If user application called a function that is not available because false is specified to *bUseGenICam*, the function will return CAM_API_STS_NOT_AVAILABLE.

Including TeliCamAPI.h is required.

[Example]

C++

```
CAM_API_STATUS   uiStatus;
uint32_t         uiNum;
CAM_HANDLE       hCam;

// Initialize system.
uiStatus = Sys_Initialize();
if (uiStatus != CAM_API_STS_SUCCESS)
    return -1;

// Get number of cameras.
uiStatus = Sys_GetNumOfCameras(&uiNum);
if ((uiStatus != CAM_API_STS_SUCCESS) || (uiNum == 0))
    return -1;

// Open camera that is detected first, in this sample code.
uiStatus = Cam_Open(0, &hCam);
if (uiStatus != CAM_API_STS_SUCCESS)
    return -1;

printf("Cam_Open completed successfully.\n");

// TODO: add your handling code here.

// Close camera.
uiStatus = Cam_Close(hCam);
if (uiStatus != CAM_API_STS_SUCCESS)
    return -1;

// Terminate TeliCamAPI system.
Sys_Terminate();
```

5.2.3. Cam_OpenFromInfo

This function opens the camera specified by camera information.
Only in the Windows version, user application can open cameras that the other applications are using.
User application will fail in opening camera stream or camera event if the other application is using it.

[Syntax]

For Windows

```
CAM_API_STATUS  Cam_OpenFromInfo (
  const char        *pszSerialNo,
  const char        *pszModelName,
  const char        *pszUserDefinedName,
  CAM_HANDLE        *phCam,
  HANDLE            hRmv = NULL,
  bool8_t           bUseGenICam = true,
  void              *pvXml = NULL,
  CAM_ACCESS_MODE   eAccessMode = CAM_ACCESS_MODE_CONTROL
);
```

For Linux

```
CAM_API_STATUS  Cam_OpenFromInfo (
  const char        *pszSerialNo,
  const char        *pszModelName,
  const char        *pszUserDefinedName,
  CAM_HANDLE        *phCam,
  SIGNAL_HANDLE     hRmv = NULL,
  bool8_t           bUseGenICam = true,
  void              *pvXml = NULL,
  CAM_ACCESS_MODE   eAccessMode = CAM_ACCESS_MODE_CONTROL
);
```

[Parameters]

Parameter		Description
pszSerialNo	[in]	A pointer to a character array that contains serial number of the opening camera as NULL terminated string data. Specify NULL when serial number is not used as search key.
pszModelName	[in]	A pointer to a character array that contains model name of the opening camera as NULL terminated string data. Specify NULL when model name is not used as search key.
pszUserDefinedName	[in]	A pointer to a character array that contains user defined name of the camera as NULL terminated string data. Specify NULL when user defined name is not used as search key.
phCam	[out]	A pointer to a variable that receives Camera-Handle assigned to the opening camera.
hRmv	[in]	Handle of an event(signal) object for notifying disconnection of the camera. This event(signal) object will also be set signaled, on Heartbeat

Parameter		Description
		timeout in GigE Vision camera. Under Windows, event(signal) object is created by Sys_CreateSignal() or CreateEvent() of Win32 API. Under Linux, event(signal) object is created by Sys_CreateSignal(). Specify NULL or do not specify this arugument, if notification of disconnection event is not necessary.
bUseGenICam	[in]	Flag for enabling GenICam function. If true, TeliCamAPI will enable GenICam function after loading camera description file (Xml file). If false, TeliCamAPI does nto load camera description file (Xml file) and GenICam function will be disabled. All APIs in section 5.4.1, 5.6 and most APIs in section 5.5 use GenICam function inside them. We recommend users to specify true to this parameter, except there is solid reason to disable GenICam function. If this parameter is not specified, true will be used as parameter value.
pvXml	[in]	A pointer to a camera description data (Xml data). If NULL is specified or nothing are specified, camera description data will be loaded from the camera. If value other than NULL is specified, camera description data will be loaded from data buffer that the argument points. This parameter will be ignored when *bUseGenICam* is false.
eAccessMode	[in]	Access mode(Control Channel Privilege) of the GigE Vision camera. TeliCamAPI will try to acquire the specified privilrge. If no value is specified, CAM_ACCESS_MODE_CONTROL will be used as argument value. This parameter will be ignored when type of camera is not GigE Vision.

[Return value]

Returns result status. Refer to 5.8 Status code.

[Remarks]

In USB3 Vision camera case, this function clears "StreamEnable" node (SIControl register) and "EventEnable" node (EIControl register) to 0 (Disable) to stop image streaming and CameraEvent notification, if no other applications are using the camera.

In GigE Vision camera case, TeliCamAPI will enable Heartbeat process on opening camera, with Heartbeat timeout value 15sec. The camera and user application will periodically check the existence of communication in their own Heartbeat process. User application may send dummy command to check communication state if necessary. When no communication state continued, camera or TeliCamAPI will judge that Heartbeat timeout error occurred. The camera will cancel the privilege given to the host application, on Heartbeat timeout. User application will make *hRmv* event(signal)

object signaled on Heartbeat timeout.

If user application is interrupted for a long time, for example, debugging case and so on, the camera will judge that Heartbeat timeout occurred and will cancel control channel privilege, which will cause the situation that the application cannot access the camera after the application resumed.

By disabling Heartbeat process or setting log timeout value using "`Cam_SetHeartbeat()`" API, camera will not judge that Heartbeat timeout occurred.

Refer to 4.1.7 Camera Access mode (Control Channel Privilege) about Control Channel Privilege.
Refer to 4.1.8 Heartbeat process about Heartbeat process.

When true is specified to *`bUseGenICam`* parameter TeliCamAPI will load camera description file data (Xml file) and prepare internal data for enabling GenICam function. If it is the first time that the target camera is used in the user application, it may take time for loading and preparing data.

When false is specified to *`bUseGenICam`* parameter processing time of opening camera will be shortened, but 5.4.1 High-level API functions, 5.6 GenICam functions, and several APIs in section 5.5 Controlling camera feature functions will become unavailable.

If user application called a function that is not available because false is specified to *`bUseGenICam`*, the function will return `CAM_API_STS_NOT_AVAILABLE`.

Including TeliCamAPI.h is required.

[Example]

C++

```
CAM_API_STATUS   uiStatus;
uint32_t         uiNum;
CAM_HANDLE       hCam;

// Initialize system.
uiStatus = Sys_Initialize();
if (uiStatus != CAM_API_STS_SUCCESS)
    return -1;

// Get number of cameras.
uiStatus = Sys_GetNumOfCameras(&uiNum);
if ((uiStatus != CAM_API_STS_SUCCESS) || (uiNum == 0))
    return -1;

// Open camera from the serial number and the model name.
uiStatus = Cam_OpenFromInfo("0000001", "BU406M", NULL, &hCam);
if (uiStatus != CAM_API_STS_SUCCESS)
    return -1;

printf("Cam_OpenFromInfo completed successfully.\n");

// TODO: add your handling code here.

// Close camera.
uiStatus = Cam_Close(hCam);
if (uiStatus != CAM_API_STS_SUCCESS)
    return -1;

// Terminate system.
Sys_Terminate();
```


5.2.4. Cam_Close

This function closes camera specified by Camera-Handle.

[Syntax]

```
CAM_API_STATUS  Cam_Close (
  CAM_HANDLE        hCam
);
```

[Parameters]

Parameter		Description
hCam	[in]	Camera-Handle of the opened camera.

[Return value]

Returns result status. Refer to 5.8 Status code.

[Remarks]

Calling this function during the other thread is using the camera may cause error in the other thread. Call this function after all threads finished using the camera.

Including TeliCamAPI.h is required.

[Example]

Refer to example of 5.2.2 Cam_Open

5.2.5. Cam_ReadReg

This function reports data in a series of registers in the specified camera.

[Syntax]

```
CAM_API_STATUS  Cam_ReadReg (
  CAM_HANDLE        hCam,
  uint64_t          ullAdrs,
  uint32_t          uiSizeQuadlet,
  void              *pvData
);
```

[Parameters]

Parameter		Description
hCam	[in]	Camera-Handle of target camera.
ullAdrs	[in]	Start addres of registers to read.
uiSizeQuadlet	[in]	Size of data to read in Quadlet. Quadlet is 4 bytes in size. Specifying 1 means reading 4 bytes.
pvData	[out]	A pointer to an array that receives the data.

[Return value]

Returns result status. Refer to 5.8 Status code.

[Remarks]

Registers in the camera are aligned to 4-bytes boundary. Register data is read per Quadlet (4 bytes).

GigE Vision camera handles Big-endian data. TeliCamAPI handles Big-endian byte order register data received from GigE Vision camera as 32bit integer array data. TeliCamAPI will change byte order of received data within each 4 bytes (Quadlet) range to convert Big-endian integer data to Littel-endian data, and write them to '*pvData*' parameter of this function. When received data contains data whose data type is not 4 bytes length data type, user application should change byte order of '*pvData*' by itself to make the data valid.

When reading register in USB3 Vision camera, TeliCamAPI will wait for 100 milliseconds until the response to "read" command returns, by default. If response does not return within specified period, TeliCamAPI will close "`Cam_ReadReg()`" with returning result status `CAM_API_STS_TIMEOUT`.

When reading register in GigE Vision camera, TeliCamAPI will wait for 200 milliseconds until the response to "read" command returns, by default. If response does not return within specified period, TeliCamAPI will retry waiting response once. If response does not return within specified period again, TeliCamAPI will close "`Cam_ReadReg()`" with returning result status `CAM_API_STS_TIMEOUT`.

Including TeliCamAPI.h is required.

[Example]

C++

```
CAM_API_STATUS   uiStatus;
uint32_t         uiNum, uiWriteData, uiReadData1, uiReadData2;
uint64_t         ullAddress;
bool8_t          bSupportIIDC2;
CAM_HANDLE       hCam;

// Initialize system.
uiStatus = Sys_Initialize();
```

```
if (uiStatus != CAM_API_STS_SUCCESS)
    return -1;

// Get number of cameras.
uiStatus = Sys_GetNumOfCameras(&uiNum);
if ((uiStatus != CAM_API_STS_SUCCESS) || (uiNum == 0))
    return -1;

// Open camera that is detected first, in this sample code.
uiStatus = Cam_Open(0, &hCam);
if (uiStatus != CAM_API_STS_SUCCESS)
    return -1;

// Check whether the camera support IIDC2 standard.
uiStatus = GetCamSupportIIDC2(hCam, &bSupportIIDC2);
if (uiStatus != CAM_API_STS_SUCCESS)
    return -1;

// Set Width address.
if (bSupportIIDC2) {
    // BU series or BG series without CPU
    ullAddress = 0x00202098;  // 0x100000(IIDC Address)
                              //  + 0x102000(OffsetCategoryBlocck2)
                              //  + 0x98(Width Offset)
} else {
    // BG series with CPU
    ullAddress = 0x0000A804;
  }

// Read register.
uiStatus = Cam_ReadReg(hCam, ullAddress, 1, (void*)&uiReadData1);
if (uiStatus != CAM_API_STS_SUCCESS)
    return -1;

// Write register.
uiWriteData = 320;
uiStatus = Cam_WriteReg(hCam, ullAddress, 1, (void*)&uiWriteData);
if (uiStatus != CAM_API_STS_SUCCESS)
    return -1;

// Readback register.
uiStatus = Cam_ReadReg(hCam, ullAddress, 1, (void*)&uiReadData2);
if (uiStatus != CAM_API_STS_SUCCESS)
    return -1;

printf("Width Address : 0x%08X , Before data : %d , After data : %d\n",
    (uint32_t)ullAddress, uiReadData1, uiReadData2);

// Write a original data.
uiStatus = Cam_WriteReg(hCam, ullAddress, 1, (void*)&uiReadData1);
if (uiStatus != CAM_API_STS_SUCCESS)
    return -1;

// Close camera.
uiStatus = Cam_Close(hCam);
if (uiStatus != CAM_API_STS_SUCCESS)
    return -1;

// Terminate system.
Sys_Terminate();
```


5.2.6. Cam_WriteReg

This function writes data to a series of registers in the specified camera.

[Syntax]

```
CAM_API_STATUS  Cam_WriteReg (
  CAM_HANDLE        hCam,
  uint64_t          ullAdrs,
  uint32_t          uiSizeQuadlet,
  void              *pvData
);
```

[Parameters]

Parameter		Description
hCam	[in]	Camera-Handle of target camera.
ullAdrs	[in]	Start addres of registers to write data.
uiSizeQuadlet	[in]	Size of data to write, in Quadlet. Quadlet is 4 bytes in size. Specifying 1 means writing 4 bytes.
pvData	[in]	A pointer to an array that contains new register value.

[Return value]

Returns result status. Refer to 5.8 Status code.

[Remarks]

Registers in the camera are aligned to 4-bytes boundary. Register data is written per Quadlet (4 bytes).

GigE Vision camera handles Big-endian data. TeliCamAPI handles '*pvData*' parameter of this function as Little-endian byte order 32bit integer array data. TeliCamAPI will change byte order of '*pvData*' parameter within each 4 bytes (Quadlet) range to convert Little-endian integer data to Bittle-endian data, and send them to the camera. When '*pvData*' contains data whose data type is not 4 bytes length data type, user application should change byte order of the data beforehand to send valid data to the camera.

When writing data to registers in USB3 Vision camera, TeliCamAPI will wait for 100 milliseconds until the response to "write" command returns, by default. If response does not return within specified period, TeliCamAPI will close "`Cam_WriteReg`()" with returning result status `CAM_API_STS_TIMEOUT`.

When writing data to registers in GigE Vision camera, TeliCamAPI will wait for 200 milliseconds until the response to "write" command returns, by default. If response does not return within specified period, TeliCamAPI will retry waiting response once. If response does not return within specified period again, TeliCamAPI will close "`Cam_WriteReg`()" with returning result status `CAM_API_STS_TIMEOUT`.

Note that the returned status `CAM_API_STS_SUCCESS` means that TeliCamAPI performed the process with no error including that the camera received and response to "write" command with no error. It does not mean that the camera accepted the sent data.

Confirm the register data by reading the register after writing it, if necessary.

Including TeliCamAPI.h is required.

[Example]

Refer to example of 5.2.5 Cam_ReadReg.

5.2.7. Cam_ResetPort

This function will reset the port of the host adapter or host controller in the USB adapter board or motherboard. The argument Camera-Handle is used for specifying USB3 port.

[Syntax]

```
CAM_API_STATUS  Cam_ResetPort (
  CAM_HANDLE        hCam
);
```

[Parameters]

Parameter		Description
hCam	[in]	Camera-Handle of target camera.

[Return value]

Returns result status. Refer to 5.8 Status code.

[Remarks]

This function is available only for USB3 Vision camera.

When this function is called, port of the USB host adapter or host controller that specifid camera is coonnected is reset. *hRmv* event(signal) object specified with "Cam_Open()" or "Cam_OpenFromInfo()" will be signaled, after the port is reset.

Cameras connected to the port will be closed inside this function. Calling "Cam_Close()" is not required.

Release all resources that created or allocated for stream function or event function, and create or allocate them again, on opening the camera again after calling this function.

Plug and play service will search USB controller and cameras and attach device driver again, after this function is executed. It will take for a while for TeliCamAPI to recognize all camers connected to the reset port. Calling "Sys_GetNumOfCameras()" to confirm number of recognized cameras before opening camera is recommended.

Most problems will be recovered by calling this function when problem occurred. But there may be problems which will not be recovered by calling this function.

Including TeliCamAPI.h is required.

[Example]

C++

```
CAM_API_STATUS   uiStatus;
uint32_t         uiNum, uiNumOld, i;
CAM_HANDLE       hCam = (CAM_HANDLE)NULL;
SIGNAL_HANDLE    hRemoveEvt = (SIGNAL_HANDLE)NULL;

// Initialize system.
uiStatus = Sys_Initialize();
if (uiStatus != CAM_API_STS_SUCCESS)
    return -1;

// Get number of cameras.
uiStatus = Sys_GetNumOfCameras(&uiNum);
if ((uiStatus != CAM_API_STS_SUCCESS) || (uiNum == 0))
    return -1;
```

```
uiNumOld = uiNum;

// Create event for detecting camera removing.
uiStatus = Sys_CreateSignal(&hRemoveEvt);
if (uiStatus != CAM_API_STS_SUCCESS)
    return -1;

// Open camera that is detected first, in this sample code.
uiStatus = Cam_Open(0, &hCam, hRemoveEvt);
if (uiStatus != CAM_API_STS_SUCCESS)
    return -1;

printf("Cam_Open was successful.(1st)\n");

do
{
    // Reset port.
    uiStatus = Cam_ResetPort(hCam);
    if (uiStatus != CAM_API_STS_SUCCESS)
        break;

    printf("Wait remove-event that is signaled ...\n");

    // Wait remove-event that is signaled.
    uiStatus = Sys_WaitForSignal(hRemoveEvt, 2000);
    if (uiStatus!= CAM_API_STS_SUCCESS)
        break;

    // Re-open

    // Re-recognition processing of the camera
    for (i = 0; i < 100; i++) {
        // Get number of cameras.
        uiStatus = Sys_GetNumOfCameras(&uiNum);
        if (uiStatus != CAM_API_STS_SUCCESS)
            continue;

        if (uiNum == uiNumOld)
            break;

#if defined (_WIN32)
        Sleep(100);      // For sample
#else
        usleep(100000);  // For sample
#endif
    }

    if (i == 100)
        break;       // Timeout error for sample.(10sec)

    // Open camera that is detected first, in this sample code.
    uiStatus = Cam_Open(0, &hCam, hRemoveEvt);
    if (uiStatus != CAM_API_STS_SUCCESS)
        break;

    printf("Cam_Open was successful.(2nd)\n");
} while(false);

// Close camera.
if (hCam != (CAM_HANDLE)NULL)
{
    uiStatus = Cam_Close(hCam);
    if (uiStatus != CAM_API_STS_SUCCESS)
        printf("Cam_Close error! (0x%x)", uiStatus);
}

// Close event.
if (hRemoveEvt != (SIGNAL_HANDLE)NULL)
    Sys_CloseSignal(hRemoveEvt);
```

```
// Terminate system.
Sys_Terminate();

printf("Completion.\n");
```


5.2.8. Cam_GetHeartbeat

This function reports current Heartbeat settings of the camera.

[Syntax]

```
CAM_API_STATUS  Cam_GetHeartbeat (
  CAM_HANDLE        hCam,
  bool8_t           *pbEnable,
  uint32_t          *puiHbTimeout
);
```

[Parameters]

Parameter		Description
hCam	[in]	Camera-Handle of target camera.
pbEnable	[out]	A pointer to a variable that receives current Heartbeat enable state. If true, Heartbeat process is enabled, otherwise disabled.
puiHbTimeout	[out]	A pointer to a variable that receives current Heartbeat timeout value of the camera, in milliseconds.

[Return value]

Returns result status. Refer to 5.8 Status code.

[Remarks]

This function is available only for GigE Vision camera.

Refer to 4.1.8 Heartbeat process about Heartbeat process.

Including TeliCamAPI.h is required.

5.2.9. Cam_SetHeartbeat

This function writes Heartbeat process settings of the camera.

[Syntax]

```
CAM_API_STATUS  Cam_SetHeartbeat (
  CAM_HANDLE        hCam,
  bool8_t           bEnable,
  uint32_t          uiHbTimeout
);
```

[Parameters]

Parameter		Description
hCam	[in]	Camera-Handle of target camera.
bEnable	[in]	Flag for enabling Heartbeat process. If true, Heartbeat process is enabled. There are cameras which will not accept false value.
uiHbTimeout	[in]	Heartbeat timeout period in milliseconds. Value must be equal to or greater than 500 milliseconds.

[Return value]

Returns result status. Refer to 5.8 Status code.

[Remarks]

This function isavailable only for GigE Vision camera.

There are cameras whose Heartbeat function cannot be disabled.
User application cannot disable Heartbeat function of GiantDragon series camera.

TeliCamAPI will enable Heartbeat function of camera and set timeout period 15 sec, on opening camera if the cameera is GigE Vision type.

It will not be necessary to change Heartbeat settings, usually.

Change Heartbeat settings using this function when user application may be interrupted more than a few seconds, for example debugging user application.

Never change `GevHeartbeatTimeout` register or `GevGCCPHeartbeatDisable` register directly using "`Cam_WriteReg`()" or GenICam functions, because TeliCamAPI controls Heartbeat process.

Refer to 4.1.8 Heartbeat process about Heartbeat process.

Including TeliCamAPI.h is required.

[Example]

Refer to example of 5.3.1.1 `Strm_OpenSimple`.

5.2.10.Cam_GetMulticast

This function reports current multcast settings.

[Syntax]

```
CAM_API_STATUS  Cam_GetMulticast (
  CAM_HANDLE        hCam,
  bool8_t           *pbEnable,
  uint32_t          *puiMulticastIP,
  uint16_t          ushSCP,
  uint16_t          ushMCP

);
```

[Parameters]

Parameter		Description
hCam	[in]	Camera-Handle of target camera.
pbEnable	[out]	A pointer to a variable that receives the value of multicast mode. If true, multicast mode is enabled, otherwise disabled.
puiMulticastIP	[out]	A pointer to a variable that receives the multicast IP address. For example, the value 0x0A0101EF would be the multicast IP address "239.1.1.10".
ushSCP	[out]	A pointer to a variable that receives the port number of the stream channel (the stream interface) used for multicasting. Specify NULL or do not specify argument, if you do not need to acquire it.
ushMCP	[out]	A pointer to a variable that receives the port number of the message channel (the event interface) used for multicasting. Specify NULL or do not specify argument, if you do not need to acquire it.

[Return value]

Returns result status. Refer to 5.8 Status code.

[Remarks]

Only Windows version is officially supported.
This function is available only for GigE Vision camera.

Including TeliCamAPI.h is required.

5.2.11.Cam_SetMulticast

This function writes multicast settings.

[Syntax]

```
CAM_API_STATUS  Cam_SetMulticast (
  CAM_HANDLE        hCam,
  bool8_t           bEnable,
  uint32_t          uiMulticastIP,
  uint16_t          ushSCP,
  uint16_t          ushMCP
);
```

[Parameters]

Parameter		Description
hCam	[in]	Camera-Handle of target camera.
bEnable	[in]	The value of multicast mode. If true, multicast mode is enabled. The default value is false. When true is specified, join the multicast group specified by *uiMulticastIP*.
uiMulticastIP	[in]	The integer value of multicast IP address. The multicast IP address must be specified Class D address (from 224.0.0.0 to 239.255.255.255). If the camera is opened with open access mode (open privilege), you can specify 0. When 0 is specified, API reads the SCDA (Stream Channel Destination Address) register of the camera and sets it as the multicast IP address. When 0x0A0101EF is set, the multicast IP address is "239.1.1.10".
ushSCP	[in]	The port number of the stream channel (the stream interface) used for multicasting. If specify 0, the API sets the appropriate value. If there is no special reason, specify 0 or do not specify this argument.
ushMCP	[in]	The port number of the message channel (the event interface) used for multicasting. If specify 0, the API sets the appropriate value. If there is no special reason or when not using camera event notification, specify 0 or do not specify this argument.

[Return value]

Returns result status. Refer to 5.8 Status code.

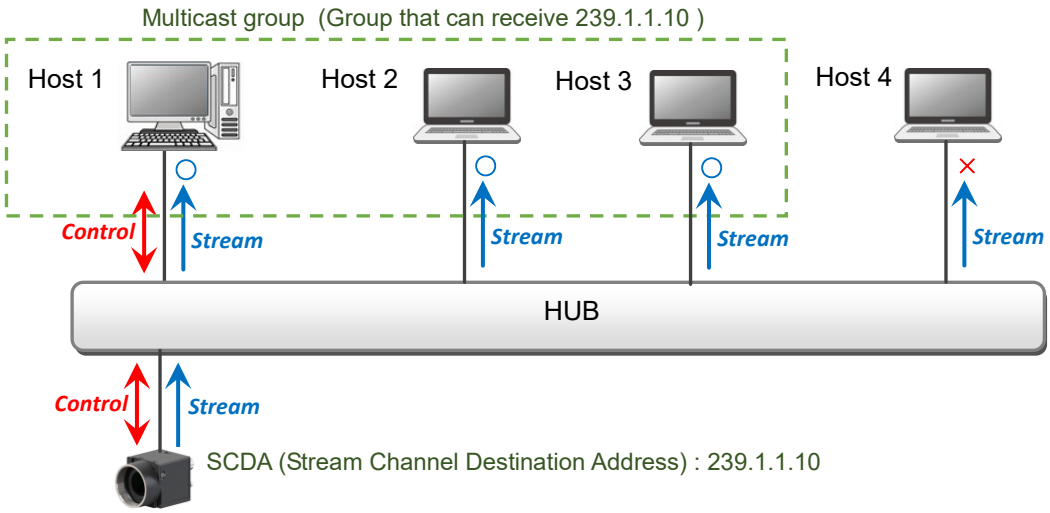
[Remarks]

Only Windows version is officially supported.
This function isavailable only for GigE Vision camera.

This function must be called before opening the stream interface.

Including TeliCamAPI.h is required.

In order to grab images of one camera on two or more hosts (PC), it is necessary to use multicast.
The host (PC) that controls the camera should be one, and it is necessary to open the camera with control access mode (control privilege).
The hosts (PC) used as the monitor must open the camera in open access mode (open privilege).



When the host (PC) used as the monitor opens the stream interface before the host (PC) controlling the camera opens the stream interface, each application needs to specify valid values for *uiMulticastIP*, *ushSCP* and *ushMCP*.

5.3. Camera streaming functions

TeliCamAPI provides two types of camera streaming functions for receiving camera image.
Using High-Level API functions will make source code simple and slim.
Low-Level API allows software designers to organize stream data receiving processing in the user's arbitrary sequence.

Only in the Windows version, multiple applications can open the same camera simultaneously. But user application cannot open a stream interface that in use by the other application.

5.3.1. High-Level API functions

High-Level API stream functions use ImageRingBuffer inside TeliCamAPI for receiving image stream from the camera to make user application code simple.
Refer to 4.1.5.1 Acquiring image data using High-Level API functions.

5.3.1.1. Strm_OpenSimple

This function creates ImageRingBuffer inside TeliCamAPI and opens the stream interface for receiving images from camera. ImageRingBuffer consists of structure `CAM_STRM_REQUEST_INFO`.

[Syntax]

For Windows

```
CAM_API_STATUS  Strm_OpenSimple (
  CAM_HANDLE        hCam,
  CAM_STRM_HANDLE   *phStrm,
  uint32_t          *puiMaxPayloadSize,
  HANDLE            hCmpEvt = NULL,
  uint32_t          uiApiBufferCount = DEFAULT_API_BUFFER_CNT,
  uint32_t          uiMaxPacketSize = 0
);
```

For Linux

```
CAM_API_STATUS  Strm_OpenSimple (
  CAM_HANDLE        hCam,
  CAM_STRM_HANDLE   *phStrm,
  uint32_t          *puiMaxPayloadSize,
  SIGNAL_HANDLE     hCmpEvt = NULL,
  uint32_t          uiApiBufferCount = DEFAULT_API_BUFFER_CNT,
  uint32_t          uiMaxPacketSize = 0
);
```

[Parameters]

Parameter		Description
hCam	[in]	Camera-Handle of target camera.
phStrm	[out]	A pointer to a variable that receives Stream-Handle of the new stream.
puiMaxPayloadSize	[out]	A pointer to a variable that contains payload size (image size) of a stream in bytes. Image size settings and or PixelFormat will affect this value.

Parameter		Description
hCmpEvt	[in]	Handle of an event(signal) object for notifying that at least one image has been received from the camera and stored in ImageRingBuffer. This argument is optional. Under Windows, event(signal) object is created by Sys_CreateSignal() or CreateEvent() of Win32 API. Under Linux, event(signal) object is created by Sys_CreateSignal(). Specify NULL or do not specify argument, if notification of imsge aquired event is not necessary.
uiApiBufferCount	[in]	Number of image buffer (StreamRequests) to create in ImageRingBuffer. This argument is optional. Valid value range is up to 128 from 1. If 0 is specified or this argument is not specified, default number DEFAULT_API_BUFFER_CNT (8) is used as this parameter.
uiMaxPacketSize	[in]	The maximum packet size that camera driver can receive, in bytes. This argument is optional. If 0 is specified, or value is not specified, the following default value will be used. USB3 Vision camera : 65536 bytes GigE Vision camera : 1500 bytes When decreasing overhead of streaming is required, enable Jumbo-Frame of network card that GigE Vision camera is connected, and specify the Jumbo-Frame size as *uiMaxPacketSize*,. Note that *uiMaxPacketSize* value is packet size excluding Ethernet header (14 bytes) to this argument. If the network card uses packet size value including Ethernet header as Jumbo-Frame size, specify (Jumbo-Frame size – 14). For example, when Jumbo-Frame size is 9014, which usually includes Ethernet header size, specify 9000 instead of 9014. Specify 0 or do not specify value for USB3 Vision camera.

[Return value]

Returns result status. Refer to 5.8 Status code.

[Remarks]

If user application calls this function for the stream interface that the other application is using, this function will return CAM_API_STS_ALREADY_OPENED.

This function creates ImageRingBuffer inside TeliCamAPI and open the stream interface for receiving images from camera. ImageRingBuffer consists of structure CAM_STRM_REQUEST_INFO. Users can design image-acquiring code without complex code such as image receiving thread process,

Use “Strm_Open()” instead of this function, when the maximum performance of the camera or special sequence of processing is required for receiving image.

Register callback function using “Strm_SetCallbackImageAcquired()” to execute it when a frame of

image has been transeferred and the contents of ImageRingBuffer has been updated, if necessary. User application can receive pointer to the received image.

High-Level API provides three ways to get received images.

1. Using argument of callback function registered with `Strm_SetCallbackImageAcquired`().
   psImageInfo argument of Image-Acquired callback function points to `CAM_IMAGE_INFO` structure that contains the latest image data and its information. User application can use *psImageInfo* argument data during the callback function is running.

2. Using "`Strm_ReadCurrentImage`()".
   "`Strm_ReadCurrentImage`()" copies the latest image to the specified buffer and informs pointer to information of the latest image.

3. Using "`Strm_UnlockBuffer`()".
   User application can get image in any index of the ImageRingBuffer.
   The followings shows steps for getting an image data.

   A. Call "`Strm_GetCurrentBufferIndex`()" to get index of the StreamRequest that contains current image in the `ImageRingBuffer`.
   B. Calculate index of target StreamRequest using index of current Streamrequest.
   C. Call "`Strm_LockBuffer` ()" to lock the target StreamRequest .
   D. Get image data and its information using data returned to "`Strm_LockBuffer`()" arguments.
   E. Call "`Strm_LockBuffer`()" to unlock target Streamrequest.

TeliCamAPI will set the registered `hCmpEvt` event(signal) object signaled and call the callback funtion, when TeliCamAPI replaced a StreamRequest in the ImageRingBuffer with a newly received image data.

If the higher priority threads are running, plural frames of newly received image may have been saved to ImageRingBuffer before a processig starts which is activated by `hCmpEvt` event(signal) object.

Note that `hCmpEvt` event(signal) object will not be signaled again correspoinding to the already saved image.

When plural frames of newly saved image may have been saved to ImageRingBuffer before the previous callback is completed, single calback will be called for the plural images saved to ImageRingBuffer during the previous calback.

Use "`Strm_GetCurrentBufferIndex`()" for checking received frame count, and use "`Strm_LockBuffer`()" and "`Strm_UnlockBuffer`()" for processing iamges not processed yet, if processing all frames is required.

Note that ImageRingBuffer in TeliCamAPI is not image buffer in the camera.

Including TeliCamAPI.h is required.

[Example]

C++

```
CAM_API_STATUS       uiStatus;
uint32_t             uiNum, uiPyldSize, uiAve, i;
CAM_HANDLE           hCam = (CAM_HANDLE)NULL;
CAM_STRM_HANDLE      hStrm = (CAM_STRM_HANDLE)NULL;
SIGNAL_HANDLE        hStrmCmpEvt = (SIGNAL_HANDLE)NULL;
void*                pvPayloadBuf = NULL;
CAM_IMAGE_INFO       sImageInfo;


// Initialize system.
uiStatus = Sys_Initialize();
if (uiStatus != CAM_API_STS_SUCCESS)
    return -1;

// Get number of cameras.
uiStatus = Sys_GetNumOfCameras(&uiNum);
if ((uiStatus != CAM_API_STS_SUCCESS) || (uiNum == 0))
    return -1;

// Open camera that is detected first, in this sample code.
uiStatus = Cam_Open(0, &hCam);
if (uiStatus != CAM_API_STS_SUCCESS)
    return -1;

do
{
#ifdef _DEBUG
    // For GigE Vision camera debug.
    Cam_SetHeartbeat(hCam, false, 0);
#endif

    // Create completion event object for stream.
    uiStatus = Sys_CreateSignal(&hStrmCmpEvt);
    if (uiStatus != CAM_API_STS_SUCCESS)
        break;

    // Open stream interface.
    uiStatus = Strm_OpenSimple(hCam, &hStrm, &uiPyldSize, hStrmCmpEvt);
    if (uiStatus != CAM_API_STS_SUCCESS)
        break;

    // Allocate buffer for receiving image data.
    pvPayloadBuf = (void *)malloc(uiPyldSize);
    if (pvPayloadBuf == NULL)
        break;

    // Set TriggerMode true, in this sample code.
    uiStatus = SetCamTriggerMode(hCam, true);
    if (uiStatus != CAM_API_STS_SUCCESS)
        break;

    // Set TriggerSource software.
    uiStatus = SetCamTriggerSource(hCam, CAM_TRIGGER_SOFTWARE);
    if (uiStatus != CAM_API_STS_SUCCESS)
        break;

    // Start stream.
    uiStatus = Strm_Start(hStrm);
    if (uiStatus != CAM_API_STS_SUCCESS)
        break;

    // Send Software Trigger command.
    uiStatus = ExecuteCamSoftwareTrigger(hCam);
    if (uiStatus != CAM_API_STS_SUCCESS)
        break;

    // Wait for receiving image completion event.
```

```
    uiStatus = Sys_WaitForSignal(hStrmCmpEvt, 2000);
    if (uiStatus!= CAM_API_STS_SUCCESS)
        break;

    // Get current image.
    uiStatus = Strm_ReadCurrentImage(hStrm, pvPayloadBuf, &uiPyldSize, &sImageInfo);
    if (uiStatus != CAM_API_STS_SUCCESS)
        break;

    // Calculate average of pixel value. (for monochrome camera)
    uiAve = 0;
    for(i = 0; i < uiPyldSize; i++)
        uiAve += ((uint8_t *)pvPayloadBuf)[i];

    uiAve /= uiPyldSize;
    printf("Average picture level = %d.\n", uiAve);
} while(false);

if (hStrm != (CAM_STRM_HANDLE)NULL) {
    // Stop stream.
    uiStatus = Strm_Stop(hStrm);
    if (uiStatus != CAM_API_STS_SUCCESS)
        printf("Strm_Stop error! (0x%x)", uiStatus);

    // Close stream interface.
    uiStatus = Strm_Close(hStrm);
    if (uiStatus != CAM_API_STS_SUCCESS)
        printf("Strm_Close error! (0x%x)", uiStatus);
}

// Close camera.
if (hCam != (CAM_HANDLE)NULL) {
    uiStatus = Cam_Close(hCam);
    if (uiStatus != CAM_API_STS_SUCCESS)
        printf("Cam_Close error! (0x%x)", uiStatus);
}

// Release buffer for receiving image data.
if (pvPayloadBuf != NULL)
    free(pvPayloadBuf);

// Close completion event object for stream.
if (hStrmCmpEvt != (SIGNAL_HANDLE)NULL)
    Sys_CloseSignal(hStrmCmpEvt);

// Terminate system.
Sys_Terminate();
```

5.3.1.2. Strm_ReadCurrentImage

This function reports information of the latest image in the ImageRingBuffer, and copy the latest image data to the specified memory block.

[Syntax]

```
CAM_API_STATUS  Strm_ReadCurrentImage (
  CAM_STRM_HANDLE   hStrm,
  void              *pvBuf,
  uint32_t          *puiSize,
  CAM_IMAGE_INFO    *psImageInfo
);
```

[Parameters]

Parameter		Description
hStrm	[in]	Stream-Handle of the target stream interface.
pvBuf	[out]	A pointer to a memory block that receives current image.
puiSize	[in, out]	A pointer to a variable that contains size of memory block pointed by *pvBuf*. If size of memory block is less than size of the received image, this function returns error status. This function writes size of copied image in bytes to this variable after copying image data.
psImageInfo	[out]	A pointer to a variable that receives additional information of current image. Specify NULL, if additional information is not required.

[CAM_IMAGE_INFO structure]

```
typedef struct _CAM_IMAGE_INFO {
    uint64_t            ullTimestamp;
    TC_PXL_FMT          uiPixelFormat;
    uint32_t            uiSizeX;
    uint32_t            uiSizeY;
    uint32_t            uiOffsetX;
    uint32_t            uiOffsetY;
    uint32_t            uiPaddingX;
    uint64_t            ullBlockId;
    void                *pvBuf;
    uint32_t            uiSize;
    uint64_t            ullImageId;
    CAM_API_STATUS      uiStatus;
} CAM_IMAGE_INFO, *PCAM_IMAGE_INFO;
```

Member		Description
ullTimestamp	[out]	Timestamp of the camera when image data was sent. Unit of this value is nano second in USB3 Vision Camera, Unit of this value is 16 nano second in BG series camera. User application can get uint of timestamp value using "GevTimestampTickFrequency" node in GigE Vision camera.
uiPixelFormat	[out]	PixelFormat of the image in pvBuf.
uiSizeX	[out]	Image width in pixels.
uiSizeY	[out]	Image height in pixels.
uiOffsetX	[out]	Start position in horizontal direction of image in pvBuf in pixel.
uiOffsetY	[out]	Start position in vertical direction of image in pvBuf in pixel.
uiPaddingX	[out]	Number of padding data at the end of row, in bytes.
ullBlockId	[out]	Frame index of image in pvBuf issued by the camera. Camera will clear frame index on stopping image acquisition. Note that there may be cameras that do not provide block ID data or which may not clear index. Use ullImageId if value of ullBlockId is not available.
pvBuf	[out]	A pointer to image data.
uiSize	[out]	Size of copied image in bytes.
ullImageId	[out]	ID number of image in pvBuf issued by TeliCamAPI. User application can use this value instead of ullBlockId. TeliCamAPI will clear Image ID on stopping image acquisition.
uiStatus	[out]	Image status at the timing that TeliCamAPI received it.

[Return value]

Returns result status. If error occurred during acquiring images, this function returns error code.
Refer to 5.8 Status code.

[Remarks]

This function is available when "Strm_OpenSimple()" was used for opening the image stream.

ID number of image (ullImageId) will be incremented every time reception of a frame is finished regardless of succeeded in reception or not.

Including TeliCamAPI.h is required.

[Example]

Refer to example of 5.3.1.1 Strm_OpenSimple().

5.3.1.3. Strm_GetCurrentBufferIndex

This function reports index of StreamRequest that contains the latest image, in ImageRingBuffer.

[Syntax]

```
CAM_API_STATUS  Strm_GetCurrentBufferIndex (
  CAM_STRM_HANDLE   hStrm,
  uint32_t          *puiBufferIndex
);
```

[Parameters]

Parameter		Description
hStrm	[in]	Stream-Handle of the target stream interface.
puiBufferIndex	[out]	A pointer to a variable that receives index of current StreamRequest. The index will be from zero up to the number specified as argument uiApiBufferCount of “Strm_OpenSimple()” minus 1. If no images have been stored to ImageRingBuffer, this function will write 0xFFFFFFFF as buffer index.

[Return value]

Returns result status. Refer to 5.8 Status code.

[Remarks]

This function is available when “Strm_OpenSimple()” was used for opening the image stream.

Including TeliCamAPI.h is required.

[Example]

C++

```
CAM_API_STATUS       uiStatus;
uint32_t             uiNum, uiPyldSize, uiBufferIndex, uiAve, i;
CAM_HANDLE           hCam = (CAM_HANDLE)NULL;
CAM_STRM_HANDLE      hStrm = (CAM_STRM_HANDLE)NULL;
SIGNAL_HANDLE        hStrmCmpEvt = (SIGNAL_HANDLE)NULL;
void*                pvPayloadBuf = NULL;
CAM_IMAGE_INFO       sImageInfo;


// Initialize system.
uiStatus = Sys_Initialize();
if (uiStatus != CAM_API_STS_SUCCESS)
    return -1;

// Get number of cameras.
uiStatus = Sys_GetNumOfCameras(&uiNum);
if ((uiStatus != CAM_API_STS_SUCCESS) || (uiNum == 0))
    return -1;

// Open camera that is detected first, in this sample code.
uiStatus = Cam_Open(0, &hCam);
if (uiStatus != CAM_API_STS_SUCCESS)
    return -1;

do
{
    // Create completion event object for stream.
```



```
    uiStatus = Sys_CreateSignal(&hStrmCmpEvt);
    if (uiStatus != CAM_API_STS_SUCCESS)
        break;

    // Open stream interface.
    uiStatus = Strm_OpenSimple(hCam, &hStrm, &uiPyldSize, hStrmCmpEvt);
    if (uiStatus != CAM_API_STS_SUCCESS)
        break;

    // Set TriggerMode true, in this sample code.
    uiStatus = SetCamTriggerMode(hCam, true);
    if (uiStatus != CAM_API_STS_SUCCESS)
        break;

    // Set TriggerSource software.
    uiStatus = SetCamTriggerSource(hCam, CAM_TRIGGER_SOFTWARE);
    if (uiStatus != CAM_API_STS_SUCCESS)
        break;

    // Start stream.
    uiStatus = Strm_Start(hStrm);
    if (uiStatus != CAM_API_STS_SUCCESS)
        break;

    // Send Software Trigger command.
    uiStatus = ExecuteCamSoftwareTrigger(hCam);
    if (uiStatus != CAM_API_STS_SUCCESS)
        break;

    // Wait for receiving image completion event.
    uiStatus = Sys_WaitForSignal(hStrmCmpEvt, 2000);
    if (uiStatus!= CAM_API_STS_SUCCESS)
        break;

    // Get current ring buffer index.
    uiStatus = Strm_GetCurrentBufferIndex(hStrm, &uiBufferIndex);
    if (uiStatus != CAM_API_STS_SUCCESS)
        break;

    // Lock the ring buffer pointer.
    uiStatus = Strm_LockBuffer(hStrm, uiBufferIndex, &sImageInfo);
    if (uiStatus != CAM_API_STS_SUCCESS)
        break;

    pvPayloadBuf = sImageInfo.pvBuf;

    // Calculate average of pixel value. (for monochrome camera)
    uiAve = 0;
    for (i = 0; i < sImageInfo.uiSize; i++)
        uiAve += ((uint8_t *)pvPayloadBuf)[i];

    uiAve /= sImageInfo.uiSize;
    printf("Average picture level = %d.\n", uiAve);

    // Unlock the ring buffer pointer.
    uiStatus = Strm_UnlockBuffer(hStrm, uiBufferIndex);
    if (uiStatus != CAM_API_STS_SUCCESS)
        break;
} while(false);

if (hStrm != (CAM_STRM_HANDLE)NULL) {
    // Stop stream.
    uiStatus = Strm_Stop(hStrm);
    if (uiStatus != CAM_API_STS_SUCCESS)
        printf("Strm_Stop error! (0x%x)", uiStatus);

    // Close stream interface.
    uiStatus = Strm_Close(hStrm);
    if (uiStatus != CAM_API_STS_SUCCESS)
        printf("Strm_Close error! (0x%x)", uiStatus);
```

```
}

// Close camera.
if (hCam!= (CAM_HANDLE)NULL) {
    uiStatus = Cam_Close(hCam);
    if (uiStatus != CAM_API_STS_SUCCESS)
        printf("Cam_Close error! (0x%x)", uiStatus);
}

// Close completion event object for stream.
if (hStrmCmpEvt != (SIGNAL_HANDLE)NULL)
    Sys_CloseSignal(hStrmCmpEvt);

// Terminate system.
Sys_Terminate();
```

5.3.1.4. Strm_LockBuffer

This function locks specified StreamRequest in the ImageRingBuffer inside TeliCamAPI and read out image information.

[Syntax]

```
CAM_API_STATUS  Strm_LockBuffer (
  CAM_STRM_HANDLE   hStrm,
  uint32_t          uiBufferIndex,
  CAM_IMAGE_INFO    *psImageInfo
);
```

[Parameters]

Parameter		Description
hStrm	[in]	Stream-Handle of the target stream interface.
uiBufferIndex	[in]	Index of the StreamRequest to lock. The available argument value is from zero up to the number specified as argument uiApiBufferCount of Strm_OpenSimple() minus 1.
psImageInfo	[out]	A pointer to a variable that receives information of image in the target StreamRequest. Specify NULL, if image information is not necessary.

[Return value]

Returns result status. Refer to 5.8 Status code.

[Remarks]

This function is available when "Strm_OpenSimple()" was used for opening the image stream.

The StreamRequest locked using this function should be unlocked using "Strm_UnlockBuffer()" as soon as possible.

Pointers of StreamRequests (structure which contains image data) that newly received image are written will be saved to ImageRingBuffer in the fixed order. If a buffer that pointer to a StreamRequest that contains the latest image should be written is locked by user application, the latest image will be discarded and callback function for buffer busy registered using "Strm_SetCallbackBufferBusy()" will be called,

To avoid buffer busy error, shorten processing contents performed during a StreamRequest is locked, or set the larger value to uiApiBufferCount on calling "Strm_OpenSimple()".

Including TeliCamAPI.h is required.

[Example]

Refer to example of 5.3.1.3 Strm_GetCurrentBufferIndex.

5.3.1.5. Strm_UnlockBuffer

This function unlocks specified StreamRequest in the ImageRingBuffer inside TeliCamAPI.

[Syntax]

```
CAM_API_STATUS  Strm_UnlockBuffer (
  CAM_STRM_HANDLE   hStrm,
  uint32_t          uiBufferIndex
);
```

[Parameters]

Parameter		Description
hStrm	[in]	Stream-Handle of the target stream interface.
uiBufferIndex	[in]	Index of the StreamRequest to unlock. The available argument value is from zero up to the number specified as argument `uiApiBufferCount` of “Strm_OpenSimple()” minus 1.

[Return value]

Returns result status. Refer to 5.8 Status code.

[Remarks]

This function is available when “`Strm_OpenSimple`()” was used for opening the image stream.

The StreamRequest locked using “`Strm_LockBuffer`()” should be unlocked using this function as soon as possible.

Including TeliCamAPI.h is required.

[Example]

Refer to example of 5.3.1.3. Strm_GetCurrentBufferIndex.

5.3.1.6. Strm_SetCallbackImageAcquired

This function registers callback function to TeliCamAPI that will be called when a successfully received stream data (a frame of image) is stored to StreamRequest in the ImageRingBuffer inside TeliCamAPI.

[Syntax]

```
CAM_API_STATUS  Strm_SetCallbackImageAcquired (
  CAM_STRM_HANDLE   hStrm,
  void              *pvContext,
  void  (CALLBACK *func)(
            CAM_HANDLE          hRcvCam,
            CAM_STRM_HANDLE     hRcvStrm,
            CAM_IMAGE_INFO      *psImageInfo,
            uint32_t            uiBufferIndex,
            void                *pvRcvContext
        )
);
```

[Parameters]

Parameter		Description
hStrm	[in]	Stream-Handle of the target stream interface.
pvContext	[in]	A pointer to any object that will be used as an arugument.on calling the callback function If user application is designed using C++ language, pointer to the parent object of callback function may be included in this object.
func	[in]	A pointer to a callback function.

[Parameters of Callback function]

Parameter		Description
hRcvCam	[out]	Camera-Handle of the camera.
hRcvStrm	[out]	Stream-Handle of the stream.
psImageInfo	[out]	A pointer to image information structure, which contains image data and its additional information. Refer to 5.3.1.2 Strm_ReadCurrentImage about CAM_IMAGE_INFO.
uiBufferIndex	[out]	Index of StreamRequest in the ImageRingBuffer.
pvRcvContext	[out]	Pointer specified as *pvContext* argument of "Strm_SetCallbackImageAcquired()".

[Return value]

Returns result status. Refer to 5.8 Status code.

[Remarks]

This function is available when "Strm_OpenSimple()" was used for opening the image stream.

We recommend users to make processing cost of callback function minimum.

If TeliCamAPI received another image during performing callback for an image, TeliCamAPI will postpone calling the next callback until current callback is finished.

If TeliCamAPI received further more images during performing callback for an image, TeliCamAPI will cancel callbacks in standby state except the latest one, and will call callback for the latest image after

current callback is finished.

Pointers of StreamRequests (structure which contains image data) that newly received image are written will be saved to ImageRingBuffer in the fixed order. If a buffer that pointer to a StreamRequest that contains the latest image should be written is locked by user application, the latest image will be discarded and callback function for buffer busy registered using "`Strm_SetCallbackBufferBusy()`" will be called,

Including TeliCamAPI.h is required.

[Example]

C++

```
CAM_HANDLE       s_hCam;
CAM_STRM_HANDLE  s_hStrm;

void CALLBACK CallbackImageAcquired(
    CAM_HANDLE       hRcvCam,
    CAM_STRM_HANDLE  hRcvStrm,
    CAM_IMAGE_INFO   *psImageInfo,
    uint32_t         uiBufferIndex,
    void             *pvContext)
{
    void             *pImageBuf = NULL;
    uint32_t         uiSize, uiAve, i;

    pImageBuf = psImageInfo->pvBuf;
    uiSize = psImageInfo->uiSize;

    // Calculate average of pixel value. (for monochrome camera)
    uiAve = 0;
    for(i = 0; i < uiSize; i++)
        uiAve += ((uint8_t *)pImageBuf)[i];

    uiAve /= uiSize;
    printf("BlockId = %d , BufNo = %d : Ave. picture level = %d.\n",
        (uint32_t)psImageInfo->ullBlockId, uiBufferIndex, uiAve);
}

void CALLBACK CallbackImageError(
    CAM_HANDLE       hRcvCam,
    CAM_STRM_HANDLE  hRcvStrm,
    CAM_API_STATUS   iErrorStatus,
    uint32_t         uiBufferIndex,
    void             *pvContext)
{
    printf("Image Error! (%d)\n", iErrorStatus);
}

void CALLBACK CallbackBufferBusy(
    CAM_HANDLE       hRcvCam,
    CAM_STRM_HANDLE  hRcvStrm,
    uint32_t         uiBufferIndex,
    void             *pvContext)
{
    printf("Buffer Busy!\n");
}

uint32_t OpenStream()
{
    CAM_API_STATUS   uiStatus;
    uint32_t         uiPyldSize;

    // Open stream interface.
    uiStatus = Strm_OpenSimple(s_hCam, &s_hStrm, &uiPyldSize);
    if (uiStatus != CAM_API_STS_SUCCESS)
        return -1;
```

```
    // Set callback functions.
    uiStatus = Strm_SetCallbackImageAcquired(s_hStrm, (void*)0x1234,
  CallbackImageAcquired);
    if (uiStatus != CAM_API_STS_SUCCESS)
        return -1;

    uiStatus = Strm_SetCallbackImageError(s_hStrm, NULL, CallbackImageError);
    if (uiStatus != CAM_API_STS_SUCCESS)
        return -1;

    uiStatus = Strm_SetCallbackBufferBusy(s_hStrm, NULL, CallbackBufferBusy);
    if (uiStatus != CAM_API_STS_SUCCESS)
        return -1;

    // Set TriggerMode true, in this sample code.
    uiStatus = SetCamTriggerMode(s_hCam, true);
    if (uiStatus != CAM_API_STS_SUCCESS)
        return -1;

    // Set TriggerSource software.
    uiStatus = SetCamTriggerSource(s_hCam, CAM_TRIGGER_SOFTWARE);
    if (uiStatus != CAM_API_STS_SUCCESS)
        return -1;

    // Start stream.
    uiStatus = Strm_Start(s_hStrm);
    if (uiStatus != CAM_API_STS_SUCCESS)
        return -1;

    for (uint32_t i = 0; i < 8; i++) {
        // Send Software Trigger command.
        uiStatus = ExecuteCamSoftwareTrigger(s_hCam);
        if (uiStatus != CAM_API_STS_SUCCESS)
            return -1;

#if defined (_WIN32)
        Sleep(50);       // For sample
#else
        usleep(50000);   // For sample
#endif
    }

    return 0;
}
```

5.3.1.7. Strm_SetCallbackImageError

This function registers callback function to TeliCamAPI that will be called when TeliCamAPI failed to update a StreamRequest in the ImageRingBuffer due to streaming error.

[Syntax]

```
CAM_API_STATUS  Strm_SetCallbackImageError (
  CAM_STRM_HANDLE   hStrm,
  void              *pvContext,
  void  (CALLBACK *func)(
            CAM_HANDLE          hRcvCam,
            CAM_STRM_HANDLE     hRcvStrm,
            CAM_API_STATUS      uiErrorStatus,
            uint32_t            uiBufferIndex,
            void                *pvRcvContext
        )
);
```

[Parameters]

Parameter		Description
hStrm	[in]	Stream-Handle of the target stream interface.
pvContext	[in]	A pointer to any object that will be used as an argument on calling the callback function. If user application is designed using C++ language, pointer to the parent object of callback function may be included in this object.
func	[in]	A pointer to a callback function.

[Parameters of Callback function]

Parameter		Description
hRcvCam	[out]	Camera-Handle of the camera.
hRcvStrm	[out]	Stream-Handle of the stream.
uiErrorStatus	[out]	Error code of the error in receiving stream.
uiBufferIndex	[out]	Index of StreamRequest in the ImageRingBuffer.
pvRcvContext	[out]	Pointer specified as *pvContext* argument on calling "Strm_SetCallbackImageError()".

[Return value]

Returns result status. Refer to 5.8 Status code.

[Remarks]

This function is available when "Strm_OpenSimple()" was used for opening the image stream.

The processing time of callback function should be as short as possible.

Including TeliCamAPI.h is required.

[Example]

Refer to example of 5.3.1.6 Strm_SetCallbackImageAcquired.

5.3.1.8. Strm_SetCallbackBufferBusy

This function registers callback function to TeliCamAPI that will be called when TeliCamAPI discarded a stream data due to lock state of target StreamRequest in the ImageRingBuffer.

[Syntax]

```
CAM_API_STATUS  Strm_SetCallbackBufferBusy (
  CAM_STRM_HANDLE   hStrm,
  void              *pvContext,
  void  (CALLBACK *func)(
            CAM_HANDLE          hRcvCam,
            CAM_STRM_HANDLE     hRcvStrm,
            uint32_t            uiBufferIndex,
            void                *pvRcvContext
        )
);
```

[Parameters]

Parameter		Description
hStrm	[in]	Stream-Handle of the target stream interface.
pvContext	[in]	A pointer to any object that will be used as an argument on calling the callback function. If user application is designed using C++ language, pointer to the parent object of callback function may be included in this object.
func	[in]	A pointer to a callback function.

[Parameters of Callback function]

Parameter		Description
hRcvCam	[out]	Camera-Handle of camera.
hRcvStrm	[out]	Stream-Handle of the stream.
uiBufferIndex	[out]	Index of StreamRequest that the discarded image was to be stored.
pvRcvContext	[out]	Pointer specified as *pvContext* argument of "Strm_SetCallbackBufferBusy()".

[Return value]

Returns result status. Refer to 5.8 Status code.

[Remarks]

This function is available when "Strm_OpenSimple()" was used for opening the image stream.

Buffer busy error occurs when a buffer of ImageRingBuffer, that a StreamRequest that contains the latest image data is to be written, is locked by user application. (ImageAcquired callback or calling "Strm_LockBuffer()".) We recommend users to make processing cost of callback function minimum.

Including TeliCamAPI.h is required.

[Example]

Refer to example of 5.3.1.6 Strm_SetCallbackImageAcquired.

5.3.2. Low-level API functions

Use functions in this section when the optimum image acquisition performance is required or special image acquiring sequence is required. Users have to describe code for managing StreamRequest. Refer to 4.1.5.2 Acquiring image data using Low-Level API functions.

5.3.2.1. Strm_Open

This function opens the stream interface for receiving images from camera.

[Syntax]

For Windows

```
CAM_API_STATUS  Strm_Open (
  CAM_HANDLE        hCam,
  HANDLE            hCmpEvt,
  uint32_t          *puiMaxPayloadSize,
  CAM_STRM_HANDLE   *phStrm,
  uint32_t          uiMaxPacketSize = 0
);
```

For Linux

```
CAM_API_STATUS  Strm_Open (
  CAM_HANDLE        hCam,
  SIGNAL_HANDLE     hCmpEvt,
  uint32_t          *puiMaxPayloadSize,
  CAM_STRM_HANDLE   *phStrm,
  uint32_t          uiMaxPacketSize = 0
);
```

[Parameters]

Parameter		Description
hCam	[in]	Camera-Handle of target camera.
hCmpEvt	[in]	Handle of an event(signal) object for notifying that at least one image has been received from the camera and stored in CompleteQueue. This argument is optional. Under Windows, event(signal) object is created by Sys_CreateSignal() or CreateEvent() of Win32 API. Under Linux, event(signal) object is created by Sys_CreateSignal(). Specify NULL or do not specify this argument if notification of image aquired event is not necessary.

Parameter	Description
puiMaxPayloadSize [in, out]	A pointer to a variable that contains the maximum payload size (image size in bytes) of a block, which is written to image buffer of a StreamRequest. The image size and or PixelFormat may affect payload size. "`GetCamStreamPayloadSize`()" will reports the default value for this argument. If zero is specified, the default value will be used inside this function.
phStrm [out]	A pointer to a variable that receives Stream-Handle assigned to the opening stream interface.
uiMaxPacketSize [in]	The maximum packet size in bytes that the camera driver can receive. This argument is optional. If 0 is specified, or value is not specified, the following default value will be used. USB3 Vision camera : 65536 bytes GigE Vision camera : 1500 bytes Enable Jumbo-Frame of network card that GigE Vision camera is connected and use the Jumbo-Frame size as this argument, when decreasing overhead of streaming is required. Specify packet size excluding Ethernet header (14 bytes) to this argument. If the network card uses packet size value including Ethernet header as Jumbo-Frame size, specify (Jumbo-Frame size – 14). For example, when Jumbo-Frame size is 9014, which usually includes Ethernet header size, specify 9000 instead of 9014. Specify 0 or do not specify value for USB3 Vision camera.

[Return value]

Returns result status. Refer to 5.8 Status code.

[Remarks]

Use this functions when the maximum image acquisition performance is required or special image acquiring sequence is required. Users have to describe code for managing StreamRrequest, Refer to 4.1.5.2 Acquiring image data using Low-Level API functions.

Use "`Strm_OpenSimple`()" instead of this function, if simple coding is required.

If user application calls this function for the stream interface that the other application is using, this function will return CAM_API_STS_ALREADY_OPENED.

TeliCamAPI will make event(signal) object `hCmpEvt` signaled on reception of each image (on every moving StreamRequest structure from StreamWaitQueue to StreamCompleteQueue).

TeliCamAPI does not make `hCmpEvt` signaled when "`Strm_FlushWaitQueue`()" is used for moving StreamRequests to StreamCompleteQueue.

Including TeliCamAPI.h is required.

[Example]

C++

```
const int STRM_REQUEST_NUM = 5;

CAM_API_STATUS            uiStatus;
uint32_t                  uiNum, uiPyldSize, uiRcvSize, i;
CAM_HANDLE                hCam = (CAM_HANDLE)NULL;
CAM_STRM_HANDLE           hStrm = (CAM_STRM_HANDLE)NULL;
SIGNAL_HANDLE             hStrmCmpEvt = (SIGNAL_HANDLE)NULL;
void*                     pvPayloadBuf = NULL;
CAM_STRM_REQUEST_HANDLE   hStrmReq[STRM_REQUEST_NUM];
void*                     pvRcvPayloadBuf = NULL;
CAM_STRM_REQUEST_HANDLE   hRcvStrmReq = (CAM_STRM_REQUEST_HANDLE)NULL;


// Initialize parameter.
for (i=0; i<STRM_REQUEST_NUM; i++) {
    hStrmReq[i] = (CAM_STRM_REQUEST_HANDLE)NULL;
}

// Initialize system.
uiStatus = Sys_Initialize();
if (uiStatus != CAM_API_STS_SUCCESS)
    return -1;

// Get number of cameras.
uiStatus = Sys_GetNumOfCameras(&uiNum);
if ((uiStatus != CAM_API_STS_SUCCESS) || (uiNum == 0))
    return -1;

// Open camera that is detected first, in this sample code.
uiStatus = Cam_Open(0, &hCam, NULL);
if (uiStatus != CAM_API_STS_SUCCESS)
    return -1;

do
{
    // Create completion event object for stream.
    uiStatus = Sys_CreateSignal(&hStrmCmpEvt);
    if (uiStatus != CAM_API_STS_SUCCESS)
        break;

    // Open stream channel.
    // Value of uiPyld is set to 0, for using default payload size
    // which can be get by GetCamStrmPayloadSize().
    uiPyldSize = 0;
    uiStatus = Strm_Open(hCam, hStrmCmpEvt, &uiPyldSize, &hStrm);
    if (uiStatus != CAM_API_STS_SUCCESS)
        break;

    // Allocate buffer for receiving image data.
    pvPayloadBuf = (void *)malloc(uiPyldSize * STRM_REQUEST_NUM);
    if (pvPayloadBuf == NULL)
        break;

    for (i=0; i<STRM_REQUEST_NUM; i++) {
        // Create a StreamRequest inside TeliCamAPI and register image buffer to it.
        uiStatus = Strm_CreateRequest(
            hStrm,
            (void*)((uint8_t*)pvPayloadBuf + (uiPyldSize * i)),
            uiPyldSize,
            &hStrmReq[i]);

        if (uiStatus != CAM_API_STS_SUCCESS)
            break;
```

```
        // Enqueue a StreamRequest to stream wait queue.
        uiStatus = Strm_EnqueueRequest(hStrm, hStrmReq[i]);
        if (uiStatus != CAM_API_STS_SUCCESS)
            break;
    }

    // Set TriggerMode true, in this sample code.
    uiStatus = SetCamTriggerMode(hCam, true);
    if (uiStatus != CAM_API_STS_SUCCESS)
        break;

    // Set TriggerSource software.
    uiStatus = SetCamTriggerSource(hCam, CAM_TRIGGER_SOFTWARE);
    if (uiStatus != CAM_API_STS_SUCCESS)
        break;

    // Start stream.
    uiStatus = Strm_Start(hStrm);
    if (uiStatus != CAM_API_STS_SUCCESS)
        break;

    for (i=0; i<10; i++) {
        // Send Software Trigger command.
        uiStatus = ExecuteCamSoftwareTrigger(hCam);
        if (uiStatus != CAM_API_STS_SUCCESS)
            break;

        // Wait for receiving image completion event.
        uiStatus = Sys_WaitForSignal(hStrmCmpEvt, 2000);
        if (uiStatus!= CAM_API_STS_SUCCESS)
            break;

        // Retrieve a StreamRequest from stream complete queue.
        uiStatus = Strm_DequeueRequest(
            hStrm,
            &hRcvStrmReq,
            &pvRcvPayloadBuf,
            &uiRcvSize);
        if (uiStatus != CAM_API_STS_SUCCESS)
            break;

        printf("Receive stream.(%d) : 0x%08X\n", i, (uint32_t)hRcvStrmReq);

        // TODO: add your handling code here.

        // Re-enqueue a StreamRequest to stream wait queue.
        uiStatus = Strm_EnqueueRequest(hStrm, hRcvStrmReq);
        if (uiStatus != CAM_API_STS_SUCCESS)
            break;
    }
} while(false);

if (hStrm != (CAM_STRM_HANDLE)NULL) {
    // Stop stream.
    uiStatus = Strm_Stop(hStrm);
    if (uiStatus != CAM_API_STS_SUCCESS)
        printf("Strm_Stop error! (0x%x)", uiStatus);

    // Move StreamRequests in stream wait queue to complete queue.
    uiStatus = Strm_FlushWaitQueue(hStrm);
    if (uiStatus != CAM_API_STS_SUCCESS)
        printf("Strm_FlushWaitQueue error! (0x%x)", uiStatus);

    for (i=0; i<STRM_REQUEST_NUM; i++) {
        // Retrieve a StreamRequest from stream complete queue.
        uiStatus = Strm_DequeueRequest(
            hStrm,
            &hRcvStrmReq,
            &pvRcvPayloadBuf,
```

```
            &uiRcvSize);

        printf("Strm_ReleaseRequest return value(%d) : 0x%08X\n", i, uiStatus);

        if (uiStatus == CAM_API_STS_EMPTY_COMPLETE_QUEUE)
            break;
    }

    // Release StreamRequests inside TeliCamAPI.
    for (i=0; i<STRM_REQUEST_NUM; i++) {
        if (hStrmReq[i] != (CAM_STRM_REQUEST_HANDLE)NULL) {
            uiStatus = Strm_ReleaseRequest(hStrm, hStrmReq[i]);
            if (uiStatus != CAM_API_STS_SUCCESS)
                printf("Strm_ReleaseRequest error! (0x%x)", uiStatus);
        }
    }

    // Close stream interface.
    uiStatus = Strm_Close(hStrm);
    if (uiStatus != CAM_API_STS_SUCCESS)
        printf("Strm_Close error! (0x%x)", uiStatus);
}

// Close camera.
if (hCam != (CAM_HANDLE)NULL) {
    uiStatus = Cam_Close(hCam);
    if (uiStatus != CAM_API_STS_SUCCESS)
        printf("Cam_Close error! (0x%x)", uiStatus);
}

// Release buffer for receiving image data.
if (pvPayloadBuf != NULL)
    free(pvPayloadBuf);

// Close completion event object for stream.
if (hStrmCmpEvt != (SIGNAL_HANDLE)NULL)
    Sys_CloseSignal(hStrmCmpEvt);

// Terminate system.
Sys_Terminate();
```


5.3.2.2. Strm_CreateRequest

This function creates a StreamRequest structure for acquiring stream data (image data).

This function creates a StreamRequest structure, but it does not allocate memory to image data buffer. User application must prepare image data buffer beforehand.

TeliCamAPI will write received image data to a StreamRequest structure in the StreamWaitQueue inside TeliCamAPI, then, move the StreamRequest structure to the StreamCompleteQueue in TeliCamAPI.

User application can receive image by taking out the StreamRequest structure from the StreamCompleteQueue.

[Syntax]

```
CAM_API_STATUS  Strm_CreateRequest (
  CAM_STRM_HANDLE          hStrm,
  void                     *pvPayloadBuf,
  uint32_t                 uiPayloaSize,
  CAM_STRM_REQUEST_HANDLE  *phStrmRequest
);
```

[Parameters]

Parameter		Description
hStrm	[in]	Stream-Handle of the target stream interface.
pvPayloadBuf	[in]	A pointer to an image data buffer that receives the image data. User application must prepare image data buffer, whose size is at least the value specified in `uiPayloadSize` argument.
uiPayloadSize	[in]	Payload size (image buffer size) of an image in bytes. Specify value returned from "`Strm_Open`()" as *puiMaxPayloadSize*, in usual case.
phStrmRequest	[out]	A pointer to a varibale that receives handle of new StreamRequest.

[Return value]

Returns result status. Refer to 5.8 Status code.

[Remarks]

This function is available when "`Strm_Open`()" was used for opening the image stream.

If stream is opened using "`Strm_OpenSimple`()", this function does not work correctly.

Use "`Strm_EnqueueRequest`()" to put created StreamRequest structure to the StreamWaitQueue inside TeliCamAPI. TeliCamAPI will write received image data to the oldest StreamRequest in the StreamWaitQueue. TeliCamAPI will move the StreamRequest containing the latest image data to StreamCompleteQueue in TeliCamAPI on finished writing image data.

User application can take out StreamRequest from the StreamCompleteQueue to get the received image using "`Strm_DequeueRequest`()".

Image buffer size of the created StreamRequest structure cannot be changed.

When image size or PixelFormat is changed, it may be necessary to release current StreamRequests using "`Strm_ReleaseRequest`()" and create them again.

Releasing image data buffer being used in StreamRequest without releasing the stream request may

cause unexpected errors. Please keep the following sequence.

1. Stop streaming using “`Strm_Stop`()”
2. Move StreamRequests in StreamWaitQueue to StreamComnpleteQueu using “`Strm_FlushWaitQueue`()”.
3. Take out all StreamRequests from the StreamCompleteQueue using “`Strm_DequeueRequest`()”.
4. Release StreamRequests using “`Strm_ReleaseRequest`()”.
5. Release image data buffer specified by `pvPayloadBuf` argument on calling this function.

Including TeliCamAPI.h is required.

[Example]

Refer to example of 5.3.2.1 Strm_Open.

5.3.2.3. Strm_ReleaseRequest

This function releases StreamRequest structure.

[Syntax]

```
CAM_API_STATUS  Strm_ReleaseRequest (
  CAM_STRM_HANDLE              hStrm,
  CAM_STRM_REQUEST_HANDLE      hStrmRequest
);
```

[Parameters]

Parameter		Description
hStrm	[in]	Stream-Handle of the target stream interface.
hStrmRequest	[in]	Handle of StreamRequest structure.

[Return value]

Returns result status. Refer to 5.8 Status code.

[Remarks]

This function is available when "Strm_Open()" was used for opening the image stream.

If stream is opened using "Strm_OpenSimple()", this function does not work correctly.

StreamRequest structures created with "Strm_CreateRequest()" must be released using this function before closing user application to avoid memory leakage.

Including TeliCamAPI.h is required.

[Example]

Refer to example of 5.3.2.1 Strm_Open.

5.3.2.4. Strm_EnqueueRequest

This function puts specified StreamRequest into the StreamWaitQueue inside TeliCamAPI for receiving image from the camera.

[Syntax]

```
CAM_API_STATUS  Strm_EnqueueRequest (
  CAM_STRM_HANDLE              hStrm,
  CAM_STRM_REQUEST_HANDLE      hStrmRequest
);
```

[Parameters]

Parameter		Description
hStrm	[in]	Stream-Handle of the target stream interface.
hStrmRequest	[in]	Handle of StreamRequest structure, which is going to be put into the StreamWaitQueue inside TeliCamAPI.

[Return value]

Returns result status. Refer to 5.8 Status code.

[Remarks]

This function is available when "Strm_Open()" was used for opening the image stream.

If stream is opened using "Strm_OpenSimple()", this function does not work correctly.

On receiving stream data (image data), TeliCamAPI will take out a StreamRequest structure from the StreamWaitQueue and start writing received data to image data block in the StreamRequest structure.

On completion of receiving a frame, TeliCamAPI will put the StreamRequest into the StreamCompleteQueue inside TeliCamAPI.

It is necessary to keep plural StreamRequest structures in the StreamWaitQueue by appending StreamRequests, whose image data will not be used any more, to the StreamWaitQueue periodically, to receive all images without dropping out.

Including TeliCamAPI.h is required.

[Example]

Refer to example of 5.3.2.1 Strm_Open.

5.3.2.5. Strm_DequeueRequest

This function takes out a StreamRequest structure from the CompleteQueue.

[Syntax]

```
CAM_API_STATUS  Strm_DequeueRequest (
  CAM_STRM_HANDLE              hStrm,
  CAM_STRM_REQUEST_HANDLE      *phStrmRequest,
  void                         **ppvPayloadBuf,
  uint32_t                     *puiPayloadSize
);
```

[Parameters]

Parameter		Description
hStrm	[in]	Stream-Handle of the target stream interface.
phStrmRequest	[out]	A pointer to a variable that receives handle of the retrieved StreamRequest structure.
ppvPayloadBuf	[out]	A pointer to a variable that receives pointer to image data in the retrieved StreamRequest.
puiPayloadSize	[out]	A pointer to a variable that reeives payload size(image size) of the retrieved StreamRequest.

[Return value]

Returns result status. Refer to 5.8 Status code.

[Remarks]

This function is available when "`Strm_Open`()" was used for opening the image stream.

If stream is opened using "`Strm_OpenSimple`()", this function does not work correctly.

This function takes out the oldest StreamRequest in the StreamCompleteQueue.

When this function is called during the StreamCompleteQueue is empty, TeliCamAPI will return `CAM_API_STS_EMPTY_COMPLETE_QUEUE` as satus code.

Including TeliCamAPI.h is required.

[Example]

Refer to example of 5.3.2.1 Strm_Open.

5.3.2.6. Strm_FlushWaitQueue

This function moves all StreamRequest structures in the StreamWaitQueue to the StreamCompleteQueue, aborting stream reception to StreamRequest structure.

[Syntax]

```
CAM_API_STATUS  Strm_FlushWaitQueue (
  CAM_STRM_HANDLE   hStrm
);
```

[Parameters]

Parameter		Description
hStrm	[in]	Stream-Handle of the target stream interface.

[Return value]

Returns result status. Refer to 5.8 Status code.

[Remarks]

This function is available when "Strm_Open()" was used for opening the image stream.

If stream is opened using "Strm_OpenSimple()", this function does not work correctly.

Before closing the stream interface using "Strm_Close()", the StreamWaitQueue and the StreamCompleteQueue must be empty.
This method moves all StreamRequest structures in the StreamWaitQueue to StreamCompleteQueue and empties the StreamWaitQueue.
To empty the StreamCompleteQueue, execute "Strm_DequeueRequest()" repeatedly until CAM_API_STS_EMPTY_COMPLETE_QUEUE returns.

When "Strm_DequeueRequest()" takes out a StreamRequest moved by executing this function, CAM_API_STS_FLUSH_REQUESTED is returned to indicate that valid image data is not saved.

Including TeliCamAPI.h is required.

[Example]

Refer to example of 5.3.2.1 Strm_Open.

5.3.2.7. Strm_GetStrmReqInfo

This function reports information about StreamRequest retrieved from the StreamCompleteQueue.

[Syntax]

```
CAM_API_STATUS  Strm_GetStrmReqInfo (
  CAM_STRM_HANDLE              hStrm,
  CAM_STRM_REQUEST_HANDLE      hStrmRequest,
  CAM_STRM_REQUEST_INFO        *psStrmReqInfo
);
```

[Parameters]

Parameter		Description
hStrm	[in]	Stream-Handle of the target stream interface.
hStrmRequest	[in]	Handle of StreamRequest for retrieving its information.
psStrmReqInfo	[out]	A pointer to a variable that receives information of the StreamRequest.

[CAM_STRM_REQUEST_INFO structure]

```
typedef struct _CAM_STRM_REQUEST_INFO {
      U3V_STRM_REQUEST_INFO  sU3vInfo;
      GEV_STRM_REQUEST_INFO  sGevInfo;
} CAM_STRM_REQUEST_INFO, *PCAM_STRM_REQUEST_INFO;
```

Member		Description
sU3vInfo	[out]	StreamRequest structure for USB3 Vision camera. When the target camera is USB3 Vision type camera, information will be copied to this structure.
sGevInfo	[out]	StreamRequest structure for GigE Vision camera. When the target camera is GigE Vision type camera, information will be copied to this structure.

[U3V_STRM_REQUEST_INFO structure]

```
typedef struct _U3V_STRM_REQUEST_INFO {
    void      *pvLeader;
    void      *pvPayload;
    void      *pvTrailer;
    uint32_t  uiPayloadSize;
} U3V_STRM_REQUEST_INFO, *PU3V_STRM_REQUEST_INFO;
```

Member		Description
pvLeader	[out]	A pointer to the Leader data of the stream.
pvPayload	[out]	A pointer to the Payload data of the stream.
pvTrailer	[out]	A pointer to the Trailer data of the stream.
uiPayloadSize	[out]	Size of actually received payload (image) in bytes.

[GEV_STRM_REQUEST_INFO structure]

```
typedef struct _GEV_STRM_REQUEST_INFO {
    void          *pvLeader;
    void          *pvPayload;
    void          *pvTrailer;
    uint32_t      uiNumOfPayloadPacket;
    uint32_t      uiPayloadSize;
    uint32_t      uiNumOfResendPacket;
} GEV_STRM_REQUEST_INFO, *PGEV_STRM_REQUEST_INFO;
```

Member		Description
pvLeader	[out]	A pointer to the Leader data of the stream.
pvPayload	[out]	A pointer to the Payload data of the stream.
pvTrailer	[out]	A pointer to the Trailer data of the stream.
uiNumOfPayloadPacket	[out]	Number of actually received payload packet.
uiPayloadSize	[out]	Size of actually received payload (image) in bytes.
uiNumOfResendPacket	[out]	Number of resend request packet sent to the camera.

[Return value]

Returns result status. Refer to 5.8 Status code.

[Remarks]

This function is available when “Strm_Open()” was used for opening the image stream.

If stream is opened using “Strm_OpenSimple()”, this function does not work correctly,

Knowledge about USB3 Vision standard and GigE Vision standard are necessary to design code using this function.

Declare structure for Leader data or Trailer data of the stream and cast Leader data the Leader structure type or cast Trailer data to the Trailer structure type to get the information in them, if necessary.

It is not necessary to use this function in usual case. Refrain from using this function except there is solid reason to use this function.

Including TeliCamAPI.h is required.

5.3.3. Common functions

Functions in this section are available with both High-level API streaming functions and low-level API streaming functions.

5.3.3.1. Strm_Close

This function closes the stream interface for receiving images from camera.

[Syntax]

```
CAM_API_STATUS  Strm_Close (
  CAM_STRM_HANDLE   hStrm
);
```

[Parameters]

Parameter		Description
hStrm	[in]	Stream-Handle of the target stream interface.

[Return value]

Returns result status. Refer to 5.8 Status code.

[Remarks]

When streaming interfaces are opend using “Strm_OpenSimple()” or “Strm_Open()”, never forget to close them using this function after finished using them.

Calling this function during the other thread is using streaming function with the Stream-Handle going to be closed may cause error in the other thread.

Call this function after all threads finished using the streaming function with the Stream-Handle

Including TeliCamAPI.h is required.

[Example]

Refer to example of 5.3.1.1 Strm_OpenSimple.

5.3.3.2. Strm_Start

This function sends streaming start commands to the camera for acquiring images.

[Syntax]

```
CAM_API_STATUS  Strm_Start (
  CAM_STRM_HANDLE       hStrm
  CAM_ACQ_MODE_TYPE     eAcqMode = CAM_ACQ_MODE_CONTINUOUS
);
```

[Parameters]

Parameter		Description
hStrm	[in]	Stream-Handle of the target stream interface.
eAcqMode	[in]	Image acquisition mode. If this parameter is not speciffied, continuous acquisition mode (CAM_ACQ_MODE_CONTINUOUS) will be used.

[CAM_ACQ_MODE_TYPE Enumeration]

Member	Description
CAM_ACQ_MODE_CONTINUOUS	The camera repeats image acquisition and transfer until "Strm_Stop()" is called. "Continuous" is set in the AcquisitionMode register of the camera.
CAM_ACQ_MODE_SINGLE_FRAME	The camera acquires a single image and transfers it. When this function is called, the AcquisitionStart command is executed and a image is acquired. To acquire a new image, execute this function or "ExecuteCamAcquisitionStart()". When this function is called, the value of AcquisitionFrameCount register is rewritten to 1.
CAM_ACQ_MODE_MULTI_FRAME	The camera repeats image acquisition and transfer until the number of images specified by "SetCamAcquisitionFrameCount()" is acquired. When this function is called, the AcquisitionStart command is executed and images are acquired. To acquire new images, execute this function or "ExecuteCamAcquisitionStart()".
CAM_ACQ_MODE_IMAGE_BUFFER_READ	The camera repeats acquiring images and saving them to buffers in the camera until "Strm_Stop()" is called. User application can instruct camera to send image in the camera buffer using "ExecuteCamImageBufferRead()". Refer to 5.5.9 ImageBuffer.

[Return value]

Returns result status. Refer to 5.8 Status code.

[Remarks]

Refer to instruction manual of the camera about the available image acquisition mode.

Set ImageBuffer mode Off in usual case.
Set ImageBuffer mode ON using "SetImageBufferMode()" before calling "Strm_Start()" with CAM_ACQ_MODE_IMAGE_BUFFER_READ mode, whenthe user application uses ImageBuffer mode. .

TeliCamAPI will write 1 to TLParamsLocked register when this function is called. TLParamsLocked is a variable in GenApi.

Including TeliCamAPI.h is required.

[Example]

Refer to example of 5.3.1.1 Strm_OpenSimple.

5.3.3.3. Strm_Stop

This function sends streaming stop commands to the camera for stopping streaming.

This function stops the acquisition of the camera at the end of the current Frame.

[Syntax]

```
CAM_API_STATUS  Strm_Stop (
  CAM_STRM_HANDLE   hStrm
);
```

[Parameters]

Parameter		Description
hStrm	[in]	Stream-Handle of the target stream interface.

[Return value]

Returns result status. Refer to 5.8 Status code.

[Remarks]

TeliCamAPI will write 0 to TLParamsLocked register when this function is called. TLParamsLocked is a variable in GenApi.

On Linux, when the stream start / stop is repeated continuously, the stream interface may stop working. If this problem causes the stream interface to stop, change the Strm_Stop function to the Strm_Abort function.

Including TeliCamAPI.h is required.

[Example]

Refer to example of 5.3.1.1 Strm_OpenSimple.

5.3.3.4. Strm_Abort

This function sends streaming abort commands to the camera for aborting streaming.

This function abort the acquisition immediately.
This will end the capture without completing the current Frame.

[Syntax]

```
CAM_API_STATUS  Strm_Abort (
  CAM_STRM_HANDLE   hStrm
);
```

[Parameters]

Parameter		Description
hStrm	[in]	Stream-Handle of the target stream interface.

[Return value]

Returns result status. Refer to 5.8 Status code.

[Remarks]

The aborted frame is given an error status.

If the frame is aborted without completing, the callback function registered by Strm_SetCallbackImageError() is called.

TeliCamAPI will write 0 to TLParamsLocked register when this function is called. TLParamsLocked is a variable in GenApi.

Including TeliCamAPI.h is required.

5.4. Camera Event notification functions

TeliCamAPI provides two types of Camera Event notification functions, High-level API and low-level API functions. Refer to 4.1.6 CameraEvent control.

It is recommended to use low-level functions if you need to receive Camera Event notifications in the order received, or if you need to continuously acquire stream data (image data) in free-run or bulk trigger mode.

Only in the Windows version, multiple applications can open the same camera simultaneously. But user application cannot open a event interface that in useby the other application.

5.4.1. High-level API functions

Camera Event functions of High-level API will hide handling of EventRequest,that will make code of user application simple.

However, if user application receives the same Camera Event before resetting the signal of the event(signal) object (*hCmpEvt*) using a function such as WaitForSingleObject() / Sys_WaitForSignal() or ResetEvent() / Sys_ResetSignal(), please note that the event(signal) object (*hCmpEvt*) does not become the signal state again after it resetted.

If GenICam function was disabled on opening camera, user application cannot use High-level API functions, because these functions use GenApi.

5.4.1.1. Evt_OpenSimple

This function opens the event interface for receiving Camera Event notifications (message), creates EventRequest structures, and creates the EventRingBuffer for Camera Event notifications in which the created EventRequest structures are kept.

[Syntax]

```
CAM_API_STATUS  Evt_OpenSimple (
  CAM_HANDLE        hCam,
  CAM_EVT_HANDLE    *phEvt,
  uint32_t          uiApiBufferCount = DEFAULT_API_BUFFER_CNT
);
```

[Parameters]

Parameter		Description
hCam	[in]	Camera-Handle of target camera.
phEvt	[out]	A pointer to a variable that receives Event-Handle assigned to the opening event interface.
uiApiBufferCount	[in]	Number of EventRequest for the opening event interface. This argument is optional. Valid value range is up to 30 from 3. If zero is specified or argument is not specified, the default value DEFAULT_API_BUFFER_CNT (8) is used as EventRingBuffer size.

[Return value]

Returns result status. Refer to 5.8 Status code.

[Remarks]

If user application calls this function for a Camera Event interface that the other application is using, this function will return CAM_API_STS_ALREADY_OPENED.

Use GenApi functions to get detail information of a Camera Event, when the event(signal) object is signaled. Refer to the following sample code.

When multiple Camera Event notifications are activated for one camera, multiple Camera Event notifications occur in a short period of time. Therefore, processing may not be completed according to the usage environment and application. At this time, one or more received Camera Event notifications are discarded without any action.

It is recommended to use low-level functions if you need to receive Camera Event notifications in the order received, or if you need to continuously acquire stream data (image data) in free-run or bulk trigger mode.

Including TeliCamAPI.h is required.

[Example]

C++

```
CAM_API_STATUS   uiStatus;
uint32_t         uiNum, uiPyldSize;
int64_t          llVal;
CAM_HANDLE       hCam = (CAM_HANDLE)NULL;
CAM_STRM_HANDLE  hStrm = (CAM_STRM_HANDLE)NULL;
CAM_EVT_HANDLE   hEvent = (CAM_EVT_HANDLE)NULL;
SIGNAL_HANDLE    hStrmCmpEvt = (SIGNAL_HANDLE)NULL;
SIGNAL_HANDLE    hFrmTrgEvt = (SIGNAL_HANDLE)NULL;
void*            pvPayloadBuf = NULL;


// Initialize system.
uiStatus = Sys_Initialize();
if (uiStatus != CAM_API_STS_SUCCESS)
    return -1;

// Get number of cameras.
uiStatus = Sys_GetNumOfCameras(&uiNum);
if ((uiStatus != CAM_API_STS_SUCCESS) || (uiNum == 0))
    return -1;

// Open camera that is detected first, in this sample code.
uiStatus = Cam_Open(0, &hCam, NULL);
if (uiStatus != CAM_API_STS_SUCCESS)
    return -1;

do
{
#ifdef _DEBUG
    // For GigE Vision camera debug.
    Cam_SetHeartbeat(hCam, false, 0);
#endif

    // Set TriggerMode true, in this sample code.
    uiStatus = SetCamTriggerMode(hCam, true);
    if (uiStatus != CAM_API_STS_SUCCESS)
```

```
        break;

    // Set TriggerSource software.
    uiStatus = SetCamTriggerSource(hCam, CAM_TRIGGER_SOFTWARE);
    if (uiStatus != CAM_API_STS_SUCCESS)
        break;

    // Open event interface.
    uiStatus = Evt_OpenSimple(hCam, &hEvent);
    if (uiStatus != CAM_API_STS_SUCCESS)
        break;

    // Create FrameTrigger event handle.
    uiStatus = Sys_CreateSignal(&hFrmTrgEvt);
    if (uiStatus != CAM_API_STS_SUCCESS)
        break;

    // Activate FrameTrigger event.
    uiStatus = Evt_Activate(hEvent, "FrameTrigger", hFrmTrgEvt);
    if (uiStatus != CAM_API_STS_SUCCESS)
        break;

    // Create completion event object for stream.
    uiStatus = Sys_CreateSignal(&hStrmCmpEvt);
    if (uiStatus != CAM_API_STS_SUCCESS)
        break;

    // Open stream interface.
    uiStatus = Strm_OpenSimple(hCam, &hStrm, &uiPyldSize, hStrmCmpEvt);
    if (uiStatus != CAM_API_STS_SUCCESS)
        break;

    // Allocate buffer for receiving image data.
    pvPayloadBuf = (void *)malloc(uiPyldSize);
    if (pvPayloadBuf == NULL)
        break;

    // Start stream.
    uiStatus = Strm_Start(hStrm);
    if (uiStatus != CAM_API_STS_SUCCESS)
        break;

    for (uint32_t i = 0; i < 5; i++) {
        // Send Software Trigger command.
        uiStatus = ExecuteCamSoftwareTrigger(hCam);
        if (uiStatus != CAM_API_STS_SUCCESS)
            break;

        // Wait FrameTrigger event signaled.
        uiStatus = Sys_WaitForSignal(hFrmTrgEvt, 2000);
        if (uiStatus!= CAM_API_STS_SUCCESS)
            break;

        printf("Receive hEventCmpEvt %d : ", i);

        // Get Timestamp.
        uiStatus = GenApi_GetIntValue(hCam, "EventFrameTriggerTimestamp", &llVal);
        if (uiStatus != CAM_API_STS_SUCCESS)
            break;
        printf("Timestamp%d = %I64u\n", i, (uint64_t)llVal);

        // Wait for receiving image completion event.
        uiStatus = Sys_WaitForSignal(hStrmCmpEvt, 2000);
        if (uiStatus!= CAM_API_STS_SUCCESS)
            break;
    }
```

```
    return 0;
  } while(false);

  if (hStrm != (CAM_STRM_HANDLE)NULL) {
    // Stop stream.
    uiStatus = Strm_Stop(hStrm);
    if (uiStatus != CAM_API_STS_SUCCESS)
        printf("Strm_Stop error! (0x%x)", uiStatus);
}

if (hEvent != (CAM_EVT_HANDLE)NULL) {
    // Deactivate FrameTrigger event.
    uiStatus = Evt_Deactivate(hEvent, "FrameTrigger");
    if (uiStatus != CAM_API_STS_SUCCESS)
        printf("Evt_Deactivate error! (0x%x)\n", uiStatus);

    // Close  event interface.
    uiStatus = Evt_Close(hEvent);
    if (uiStatus != CAM_API_STS_SUCCESS)
        printf("Evt_Close error! (0x%x)\n", uiStatus);
}

// Close  stream interface.
  if (hStrm != (CAM_STRM_HANDLE)NULL) {
    uiStatus = Strm_Close(hStrm);
    if (uiStatus != CAM_API_STS_SUCCESS)
        printf("Strm_Close error! (0x%x)\n", uiStatus);
}

// Close camera.
if (hCam != (CAM_HANDLE)NULL) {
    uiStatus = Cam_Close(hCam);
    if (uiStatus != CAM_API_STS_SUCCESS)
        printf("Cam_Close error! (0x%x)\n", uiStatus);
}

// Close completion event object for stream.
if (hStrmCmpEvt != (SIGNAL_HANDLE)NULL)
    Sys_CloseSignal(hStrmCmpEvt);

// Close FrameTrigger event handle.
if (hFrmTrgEvt!= (SIGNAL_HANDLE)NULL)
    Sys_CloseSignal(hFrmTrgEvt);

// Terminate system.
Sys_Terminate();
```


5.4.2. Low-Level API functions

Low-Level API event functions allows user application to design a special sequence control of camera. On the other hand. user application have to manage EventRequest structures and event queues.

5.4.2.1. Evt_Open

This function opens the event interface for receiving Camera Event notifications (message).

[Syntax]

For Windows

```
CAM_API_STATUS  Evt_Open (
  CAM_HANDLE        hCam,
  CAM_EVT_HANDLE    *phEvt,
  uint32_t          *puiMaxPayloadSize,
  HANDLE            hCmpEvt = NULL
);
```

For Linux

```
CAM_API_STATUS  Evt_Open (
  CAM_HANDLE        hCam,
  CAM_EVT_HANDLE    *phEvt,
  uint32_t          *puiMaxPayloadSize,
  SIGNAL_HANDLE     hCmpEvt = NULL
);
```

[Parameters]

Parameter		Description
hCam	[in]	Camera-Handle of target camera.
phEvt	[out]	A pointer to a variable that receives Event-Handle assigned to the opening event interface.
puiMaxPayloadSize	[in, out]	A pointer to a variable that contains the maximum payload size (event information size in bytes) of one block. If `0` is specified, the default value will be used inside this function. Specify zero, in usual case.
hCmpEvt	[in]	Handle of an event(signal) object for notifying reception of Camera Event notification from the camera. Under Windows, event(signal) object is created by Sys_CreateSignal() or `CreateEvent()` of Win32 API. Under Linux, event(signal) object is created by Sys_CreateSignal(). Specify NULL, if notification of the Camera Event notification is not necessary.

[Return value]

Returns result status. Refer to 5.8 Status code.

[Remarks]

If user application calls this function for a CameraEvent interface that the other application is using, this function will return CAM_API_STS_ALREADY_OPENED.

Note that if multiple Camera Event notifications are activated, the event(signal) object `hCmpEvt` will be signaled several times in a very short period on every image reception. If it is the case, fast processing for these events are requested.

Including TeliCamAPI.h is required.

[Example]

C++

```
CAM_API_STATUS            uiStatus;
uint32_t                  uiNum, uiPyldSize, i;
CAM_HANDLE                hCam = (CAM_HANDLE)NULL;
CAM_STRM_HANDLE           hStrm = (CAM_STRM_HANDLE)NULL;
CAM_EVT_HANDLE            hEvent = (CAM_EVT_HANDLE)NULL;
SIGNAL_HANDLE             hStrmCmpEvt = (SIGNAL_HANDLE)NULL;
SIGNAL_HANDLE             hEventCmpEvt = (SIGNAL_HANDLE)NULL;
void*                     pvPayloadBuf = NULL;
void*                     pvEvtPayloadBuf = NULL;
uint32_t                  uiEvtMaxPayloadSize = 0;
CAM_EVT_REQUEST_HANDLE    hEvtRequest = (CAM_EVT_REQUEST_HANDLE)NULL;
CAM_EVT_REQUEST_HANDLE    hRcvEvtRequest = (CAM_EVT_REQUEST_HANDLE)NULL;
EVT_REQUEST_INFO          sEventReqInfo;
CAM_INFO                  sCamInfo;


// Initialize system.
uiStatus = Sys_Initialize();
if (uiStatus != CAM_API_STS_SUCCESS)
    return -1;

// Get number of cameras.
uiStatus = Sys_GetNumOfCameras(&uiNum);
if ((uiStatus != CAM_API_STS_SUCCESS) || (uiNum == 0))
    return -1;

// Open camera that is detected first, in this sample code.
uiStatus = Cam_Open(0, &hCam, NULL);
if (uiStatus != CAM_API_STS_SUCCESS)
    return -1;

do
{
    // Get camera information.
    uiStatus = Cam_GetInformation(hCam, 0, &sCamInfo);
    if (uiStatus != CAM_API_STS_SUCCESS)
        break;

#ifdef _DEBUG
    // For GigE Vision camera debug.
    Cam_SetHeartbeat(hCam, false, 0);
#endif

    // Set TriggerMode true, in this sample code.
    uiStatus = SetCamTriggerMode(hCam, true);
    if (uiStatus != CAM_API_STS_SUCCESS)
        break;

    // Set TriggerSource software.
    uiStatus = SetCamTriggerSource(hCam, CAM_TRIGGER_SOFTWARE);
    if (uiStatus != CAM_API_STS_SUCCESS)
        break;
```

```
    // Start & open event.
    {
        // Create completion event for event.
        uiStatus = Sys_CreateSignal(&hEventCmpEvt);
        if (uiStatus != CAM_API_STS_SUCCESS)
            break;

        // Open event interface.
        uiStatus = Evt_Open(hCam, &hEvent, &uiEvtMaxPayloadSize, hEventCmpEvt);
        if (uiStatus != CAM_API_STS_SUCCESS)
            break;

        printf("Evt_Open Success.(%d)\n", uiEvtMaxPayloadSize);

        // Activate ExxposureStart event.
        uiStatus = Evt_Activate(hEvent, "ExposureStart", NULL);
        if (uiStatus != CAM_API_STS_SUCCESS)
            break;

        printf("Evt_Activate Success.\n");

        // Allocate buffer for receiving event data.
        pvEvtPayloadBuf = (void *)malloc(uiEvtMaxPayloadSize);
        if (pvEvtPayloadBuf == NULL)
            return -1;

        // Create a EventRequest inside TeliCamAPI and register event buffer to it.
        uiStatus = Evt_CreateRequest(
            hEvent,
            pvEvtPayloadBuf,
            uiEvtMaxPayloadSize,
            &hEvtRequest);
        if (uiStatus != CAM_API_STS_SUCCESS)
            return -1;
        printf("Evt_CreateRequest Success.\n");
    }

    // Create completion event object for stream.
    uiStatus = Sys_CreateSignal(&hStrmCmpEvt);
    if (uiStatus != CAM_API_STS_SUCCESS)
        break;

    // Open stream interface.
    uiStatus = Strm_OpenSimple(hCam, &hStrm, &uiPyldSize, hStrmCmpEvt);
    if (uiStatus != CAM_API_STS_SUCCESS)
        break;

    // Allocate buffer for receiving image data.
    pvPayloadBuf = (void *)malloc(uiPyldSize);
    if (pvPayloadBuf == NULL)
        break;

    // Start stream.
    uiStatus = Strm_Start(hStrm);
    if (uiStatus != CAM_API_STS_SUCCESS)
        break;

    for (i = 0; i < 5; i++) {
        // Enqueue a EventmRequest to event wait queue.
        uiStatus = Evt_EnqueueRequest(hEvent, hEvtRequest);
        if (uiStatus != CAM_API_STS_SUCCESS)
            return -1;

        // Send Software Trigger command.
        uiStatus = ExecuteCamSoftwareTrigger(hCam);
        if (uiStatus != CAM_API_STS_SUCCESS)
            return -1;

        // Wait for receiving completion event.
```

```
        uiStatus = Sys_WaitForSignal(hEventCmpEvt, 2000);
        if (uiStatus!= CAM_API_STS_SUCCESS)
            break;

        printf("Receive hEventCmpEvt %d\n", i);

        // Retrieve a EventRequest from event complete queue.
        uiStatus = Evt_DequeueRequest(hEvent, &hRcvEvtRequest, &sEventReqInfo);
        if (uiStatus != CAM_API_STS_SUCCESS)
            return -1;

        if (sCamInfo.eCamType == CAM_TYPE_U3V) {
            printf("  request_id = %d, event_id = 0x%04x\n",
                sEventReqInfo.sU3vInfo.ushRequestId,
                sEventReqInfo.sU3vInfo.ushEventId);
            printf("  Timestamp = %llu, pPayload = %p\n",
                (long long unsigned int)sEventReqInfo.sU3vInfo.ullTimestamp,
                sEventReqInfo.sU3vInfo.pvPayload);
        } else if (sCamInfo.eCamType == CAM_TYPE_GEV) {
            printf("  request_id = %d, event_id = 0x%04x\n",
                sEventReqInfo.sGevInfo.ushRequestId,
                sEventReqInfo.sGevInfo.ushEventId);
            printf("  Timestamp = %llu, pPayload = %p\n",
                (long long unsigned int)sEventReqInfo.sGevInfo.ullTimestamp,
                sEventReqInfo.sGevInfo.pvPayload);
        }

        // Wait for receiving image completion event.
        uiStatus = Sys_WaitForSignal(hStrmCmpEvt, 2000);
        if (uiStatus!= CAM_API_STS_SUCCESS)
            break;
    }

    // Stop stream.
    uiStatus = Strm_Stop(hStrm);
    if (uiStatus != CAM_API_STS_SUCCESS)
        break;
} while(false) ;

if (hEvent != (CAM_EVT_HANDLE)NULL) {
    // Move EventRequests in event wait queue to event complete queue.
    uiStatus = Evt_FlushWaitQueue(hEvent);
    if (uiStatus != CAM_API_STS_SUCCESS)
        printf("Evt_FlushWaitQueue error! (0x%x)", uiStatus);

    // Retrieve a EventRequest from event complete queue.
    Evt_DequeueRequest(hEvent, &hRcvEvtRequest, &sEventReqInfo);

    // Release a EventRequest inside TeliCamAPI and register event buffer to it.
    uiStatus = Evt_ReleaseRequest(hEvent, hEvtRequest);
    if (uiStatus != CAM_API_STS_SUCCESS)
        printf("Evt_ReleaseRequest error! (0x%x)", uiStatus);

    uiStatus = Evt_Deactivate(hEvent, "ExposureStart");
    if (uiStatus != CAM_API_STS_SUCCESS)
        printf("Evt_Deactivate error! (0x%x)\n", uiStatus);

    uiStatus = Evt_Close(hEvent);
    if (uiStatus != CAM_API_STS_SUCCESS)
        printf("Evt_Close error! (0x%x)", uiStatus);
}

// Close stream interface.
if (hStrm != (CAM_STRM_HANDLE)NULL) {
    uiStatus = Strm_Close(hStrm);
    if (uiStatus != CAM_API_STS_SUCCESS)
        printf("Strm_Close error! (0x%x)", uiStatus);
}

// Close camera.
```

```
if (hCam != (CAM_HANDLE)NULL) {
    uiStatus = Cam_Close(hCam);
    if (uiStatus != CAM_API_STS_SUCCESS)
        printf("Cam_Close error! (0x%x)", uiStatus);
}

// Close completion event object for stream.
if (hStrmCmpEvt != (SIGNAL_HANDLE)NULL)
    Sys_CloseSignal(hStrmCmpEvt);

// Close completion event object for event.
if (hEventCmpEvt!= (SIGNAL_HANDLE)NULL)
    Sys_CloseSignal(hEventCmpEvt);

// Release buffer for receiving event data.
if (pvEvtPayloadBuf != NULL)
    free(pvEvtPayloadBuf);

// Release buffer for receiving image data.
if (pvPayloadBuf != NULL)
    free(pvPayloadBuf);

// Terminate system.
Sys_Terminate();

printf("Completion.\n");
```

5.4.2.2. Evt_CreateRequest

This function creates EventRequest structure for receiving Camera Event notification (message) data. Put the created EventRequest structure in EventWaitQueue for receiving Camera Event notification (message) data.

[Syntax]

```
CAM_API_STATUS  Evt_CreateRequest (
  CAM_EVT_HANDLE           hEvt,
  void                     *pvPayloadBuf,
  uint32_t                 uiPayloaSize,
  CAM_EVT_REQUEST_HANDLE   *phEvtRequest
);
```

[Parameters]

Parameter		Description
hEvt	[in]	Event-Handle of the target event interface..
pvPayloadBuf	[in]	A pointer to a data buffer that receives the Camera Event notification data. User application must prepare a buffer of Camera Event notification, whose size is at least value specified in *uiPayloadSize* argument.
uiPayloadSize	[in]	Payload size (event data size) of one Camera Event notification data in bytes. Specify puiMaxPayloadSize argument value returned from Evt_Open(), in usual case.
phEvtRequest	[out]	A pointer to a variable that receives handle of the new EventRequest.

[Return value]

Returns result status. Refer to 5.8 Status code.

[Remarks]

This function is available when the event interface was opened using "Evt_Open()".

If the event interface was opened using "Evt_OpenSimple()", this function does not work correctly.

Use "Evt_EnqueueRequest()" to put created EventRequest to the EventWaitQueue inside TeliCamAPI. On receiving Camera Event notification, TeliCamAPI will write received Camera Event notification data to the oldest EventRequest in the EventWaitQueue, move the EventRequest filled with Camera Event notification data to EventCompleteQueue in TeliCamAPI, and set *hCmpEvt* event to signal state. User application can take out EventRequest from the EventCompleteQueue using "Evt_DequeueRequest()" to get the received Camera Event notification data.

Releasing the buffer specified by *pvPayloadBuf* without releasing the EventRequest may cause unexpected errors. Please keep the following sequence.

1. Stop streaming using "Strm_Stop()".
2. Call "Evt_FlushWaitQueue()" to move all EventRequests in the EventWaitQueue to the EventComnpleteQueue,
3. Take out all EventRequests from the EventCompleteQueue using "Evt_DequeueRequest()".
4. Release EventRequests using "Evt_ReleaseRequest()".
5. Release *pvPayloadBuf*.

Including TeliCamAPI.h is required.

[Example]

Refer to example of 5.4.2.1 Evt_Open.

5.4.2.3. Evt_ReleaseRequest

This function releases EventRequest structure.

[Syntax]

```
CAM_API_STATUS  Evt_ReleaseRequest (
  CAM_EVT_HANDLE           hEvt,
  CAM_EVT_REQUEST_HANDLE   hEvtRequest
);
```

[Parameters]

Parameter		Description
hEvt	[in]	Event-Handle of the target event interface.
hEvtRequest	[in]	Handle of EventRequest structure.

[Return value]

Returns result status. Refer to 5.8 Status code.

[Remarks]

This function is available when the event interface was opened using "Evt_Open()".

If the event interface was opened using "Evt_OpenSimple()", this function does not work correctly.

EventRequest structures created with "Evt_CreateRequest()" must be released using this function before closing user application to avoid memory leakage.

Including TeliCamAPI.h is required.

[Example]

Refer to example of 5.4.2.1 Evt_Open.

5.4.2.4. Evt_EnqueueRequest

This function puts specified EventRequest into the EventWaitQueue inside TeliCamAPI for receiving Camera Event notification data from the camera.

[Syntax]

```
CAM_API_STATUS  Evt_EnqueueRequest (
  CAM_EVT_HANDLE           hEvt,
  CAM_EVT_REQUEST_HANDLE   hEvtRequest
);
```

[Parameters]

Parameter		Description
hEvt	[in]	Event-Handle of the target event interface..
hEvtRequest	[in]	Handle of an EventRequest for putting into the EventWaitQueue inside TeliCamAPI.

[Return value]

Returns result status. Refer to 5.8 Status code.

[Remarks]

This function is available when the event interface was opened using "Evt_Open()".

If the event interface was opened using "Evt_OpenSimple()", this function does not work correctly.

Please refer to the Remarks of "Evt_CreateRequest()".

Including TeliCamAPI.h is required.

[Example]

Refer to example of 5.4.2.1 Evt_Open.

5.4.2.5. Evt_DequeueRequest

This function retrieves an EventRequest structure from the EventCompleteQueue.

[Syntax]

```
CAM_API_STATUS  Evt_DequeueRequest (
  CAM_EVT_HANDLE           hEvt,
  CAM_EVT_REQUEST_HANDLE   *phEvtRequest,
  EVT_REQUEST_INFO         *psEvtReqInfo
);
```

[Parameters]

Parameter		Description
hEvt	[in]	Event-Handle of the target event interface..
phEvtRequest	[out]	A pointer to a variable that receives handle of the retrieved EventRequest.
psEvtReqInfo	[out]	A pointer to a variable that receives event data in the retrieved EventRequest.

[*EVT_REQUEST_INFO* structure]

```
typedef struct _EVT_REQ_BUF_INFO
{
    U3V_EVT_REQUEST_INFO sU3vInfo;
    GEV_EVT_REQUEST_INFO sGevInfo;
} EVT_REQUEST_INFO, *PEVT_REQ_BUF_INFO;
```

Member		Description
sU3vInfo	[out]	Event request information for USB3 Vision camera.
sGevInfo	[out]	Event request information for GigE Vision camera.

[*U3V_EVT_REQUEST_INFO* structure]

```
typedef struct _U3V_EVT_REQUEST_INFO
{
    uint16_t      ushRequestId;  // request id
    uint16_t      ushEventId;    // event id
    uint64_t      ullTimestamp;  // timestamp
    void*         pvPayload;     // payload buffer pointer
} U3V_EVT_REQUEST_INFO, *PU3V_EVT_REQUEST_INFO;
```

Member		Description
ushRequestId	[out]	request id
ushEventId	[out]	event id
ullTimestamp	[out]	timestamp
pvPayload	[out]	payload buffer pointer

[*GEV_EVT_REQUEST_INFO* structure]

```
typedef struct _GEV_EVT_REQUEST_INFO
{
    uint16_t      ushRequestId;  // request id
    uint16_t      ushEventId;    // event id
    uint64_t      ullTimestamp;  // timestamp
    void*         pvPayload;     // payload buffer pointer
    uint32_t      uiReserved1;
    uint32_t      uiReserved2;
} GEV_EVT_REQUEST_INFO, *PGEV_EVT_REQUEST_INFO;
```

Member		Description
ushRequestId	[out]	request id
ushEventId	[out]	event id
ullTimestamp	[out]	timestamp
pvPayload	[out]	payload buffer pointer
uiReserved1	[out]	Reserved1(unused)
uiReserved2	[out]	Reserved2(unused)

[Return value]

Returns result status. Refer to 5.8 Status code.

[Remarks]

This function is available when the event interface was opened using "Evt_Open()".

If the event interface was opened using "Evt_OpenSimple()", this function does not work correctly.

This function retrieves the oldest EventRequest in the EventCompleteQueue.

When this function is called during EventCompleteQueue is empty, TeliCamAPI will return CAM_API_STS_EMPTY_COMPLETE_QUEUE as satus code.

Including TeliCamAPI.h is required.

[Example]

Refer to example of 5.4.2.1 Evt_Open.

5.4.2.6. Evt_FlushWaitQueue

This function stops all CameraEvent receiving operations, and moves all EventRequest structures in the EventWaitQueue to the EventCompleteQueue,

[Syntax]

```
CAM_API_STATUS  Evt_FlushWaitQueue (
  CAM_EVT_HANDLE           hEvt
);
```

[Parameters]

Parameter		Description
hEvt	[in]	Event-Handle of the target event interface..

[Return value]

Returns result status. Refer to 5.8 Status code.

[Remarks]

This function is available when the event interface was opened using "Evt_Open()".

If the event interface was opened using "Evt_OpenSimple()", this function does not work correctly.

Before closing the event interface using "Evt_Close()", call this function to make the EventWaitQueue empty, then repeat calling "Evt_EnqueueRequest()" until EventCompleteQueue becomes empty,.

If EventRequest retrieved using "Evt_DequeueRequest()", was moved to the EventCompleteQueue using this function, TeliCamAPI will return CAM_API_STS_FLUSH_REQUESTE as result status of "Evt_DequeueRequest()".

Including TeliCamAPI.h is required.

[Example]

Refer to example of 5.4.2.1 Evt_Open.

5.4.3. Common functions

Functions in this section are available with both High-level API event functions and low-level API event functions.

5.4.3.1. Evt_Close

This function closes event (message) interface.

[Syntax]

```
CAM_API_STATUS  Evt_Close (
  CAM_EVT_HANDLE           hEvt
);
```

[Parameters]

Parameter		Description
hEvt	[in]	Event-Handle of the target event interface..

[Return value]

Returns result status. Refer to 5.8 Status code.

[Remarks]

When the event interfaces are opend using "Evt_OpenSimple()" or "Evt_Open()", never forget to close them using this function after finished using them.

Never close the event interface during another thread is using the event interface.

Including TeliCamAPI.h is required.

[Example]

Refer to example of 5.4.1.1 Evt_OpenSimple.

5.4.3.2. Evt_Activate

This function activates specified Camera Event notification.

[Syntax]

For Windows

```
CAM_API_STATUS  Evt_Activate (
  CAM_EVT_HANDLE    hEvt,
  const char        *pszEvtName,
  HANDLE            hCmpEvt
);
```

For Linux

```
CAM_API_STATUS  Evt_Activate (
  CAM_EVT_HANDLE    hEvt,
  const char        *pszEvtName,
  SIGNAL_HANDLE     hCmpEvt
);
```

[Parameters]

Parameter		Description
hEvt	[in]	Handle of the target event interface..
pszEvtName	[in]	A pointer to a character array that contains node name (register name) of the activating Camera Event notification.
hCmpEvt	[in]	Handle of an event(signal) object for notifying reception of Camera Event notification from the camera. Under Windows, event(signal) object is created by Sys_CreateSignal() or CreateEvent() of Win32 API. Under Linux, event(signal) object is created by Sys_CreateSignal(). Specify NULL, if notification of the event is not necessary. This argument is used when the event interface is opened using "Evt_OpenSimple()". This argument is ignored when the event interface is opened using "Evt_Open()".

[Return value]

Returns result status. Refer to 5.8 Status code.

[Remarks]

Case opened using "Evt_OpenSimple()"

When the event(signal) object hCmpEvt is signaled, user application can get information of the Camera Event notification that was activated using this function.

The event(signal) object hCmpEvt will be signaled on receiving Camera Event notification from the camera. However, if user application receives the same Camera Event notification before resetting the signal of the event(signal) object hCmpEvt using a function such as WaitForSingleObject() / Sys_WaitForSignal() or ResetEvent() / Sys_ResetSignal(), please note that the event(signal) object hCmpEvt does not become the signal state again after resetting.

The value specified to *pszEvtName* will be set to EventSelector node to activate the Camera Event notification. The node value (string) available for EventSelector node may vary corresponding to model of the camera. Refer to instruction manual of the camera to check the available value.

The following table shows example of values available to *pszEvtName* and information node name for the Camera Event notification.

EVENT_NAME	Description	Node name of event information
FrameTrigger	Accepted trigger signal for acquiring an image.	EventFrameTriggerTimestamp
FrameTriggerError	Received trigger signal for acquiring image at invalid timing.	EventFrameTriggerErrorTimestamp
FrameTriggerWait	Trigger signal for image acquisition becomes acceptable.	EventFrameTriggerWaitTimestamp
FrameTransferStart	Started transferring image stream of a frame.	EventFrameTransferStartTimestamp
FrameTransferEnd	Completed transferring image stream of a frame.	EventFrameTransferEndTimestamp
ExposureStart	Started exposure.	EventExposureStartTimestamp
ExposureEnd	Finished exposure.	EventExposureEndTimestamp
Timer0Start	Timer0 started counting.	EventTimer0StartTimestamp
Timer0End	Timer0 finished counting.	EventTimer0EndTimestamp
ALCLatestInformation	Updated Automatic Luminance Control data.	EventALCLatestInformationTimestamp
		EventALCLatestInformationTotalLuminance
		EventALCLatestInformationAverageLuminance
		EventALCLatestInformationExposureTime
		EventALCLatestInformationGain
ALCConverged	Converged Automatic Luminance Control operation.	EventALCConvergedTimestamp
		EventALCConvergedLuminanceTotal
		EventALCConvergedLuminanceAverage
		EventALCConvergedExposureTime
		EventALCConvergedGain

Refrain from accessing to node whose node name is the name specified to *pstEveName* parameter with prefix “Event” using GenApi functions.

Accessing to node whose node name is the name specified to *pstEveName* parameter with prefix “Event” will make the *hCmpEvt* event(signal) object signaled.

User application has to reset *hCmpEvt* event(signal) object using “`ResetEvent`()”, otherwise TeliCamAPI cannot manage CameraEvent in High-Level API mode.

For example, accessing to “EventFrameTrigger” node will make the *hCmpEvt* event(signal) object signaled if “FrameTrigger” event has been activated.

Case opened using “`Evt_Open()`”

Only activate the specified Camera Event notification.
The value specified to *pszEvtName* may vary corresponding to model of the camera.
Refer to instruction manual of the camera to check the available value.

EVENT_NAME	Description
FrameTrigger	Accepted trigger signal for acquiring an image.
FrameTriggerError	Received trigger signal for acquiring image at invalid timing.
FrameTriggerWait	Trigger signal for image acquisition becomes acceptable.
FrameTransferStart	Started transferring image stream of a frame.
FrameTransferEnd	Completed transferring image stream of a frame.
ExposureStart	Started exposure.
ExposureEnd	Finished exposure.
Timer0Start	Timer0 started counting.
Timer0End	Timer0 finished counting.
ALCLatestInformation	Updated Automatic Luminance Control data.
ALCConverged	Converged Automatic Luminance Control operation.

There are various restrictions in using Camera Event notification. For example, setting parameters in a wrong order may cause or FrameTrigger cannot be activated when TriggerMode is off, and so on. Refer to instruction manual of the camera.

Including TeliCamAPI.h is required.

The following figures indicate typical timing of Camera Event notifications.

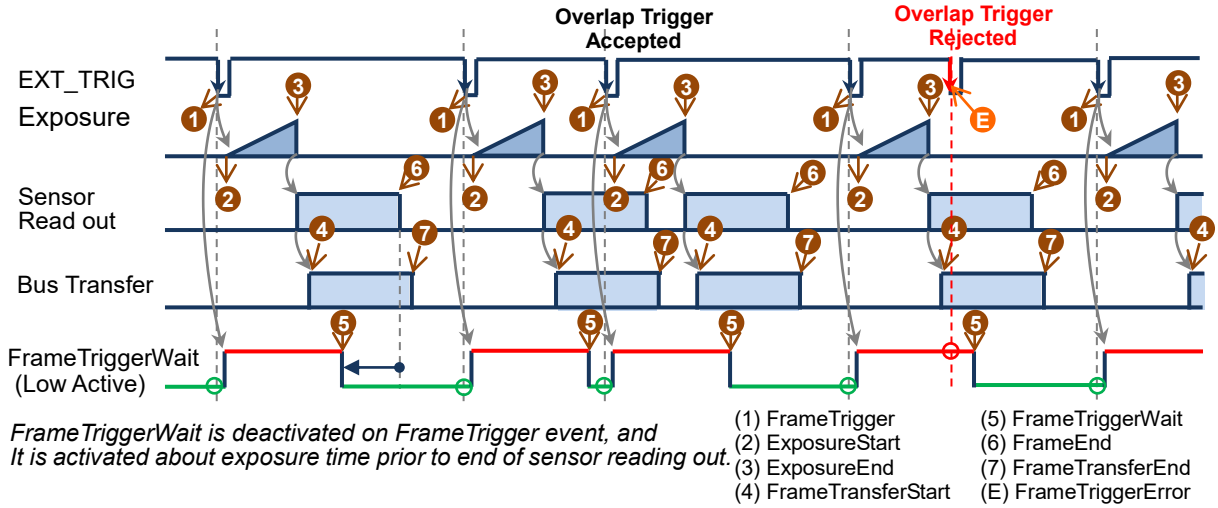
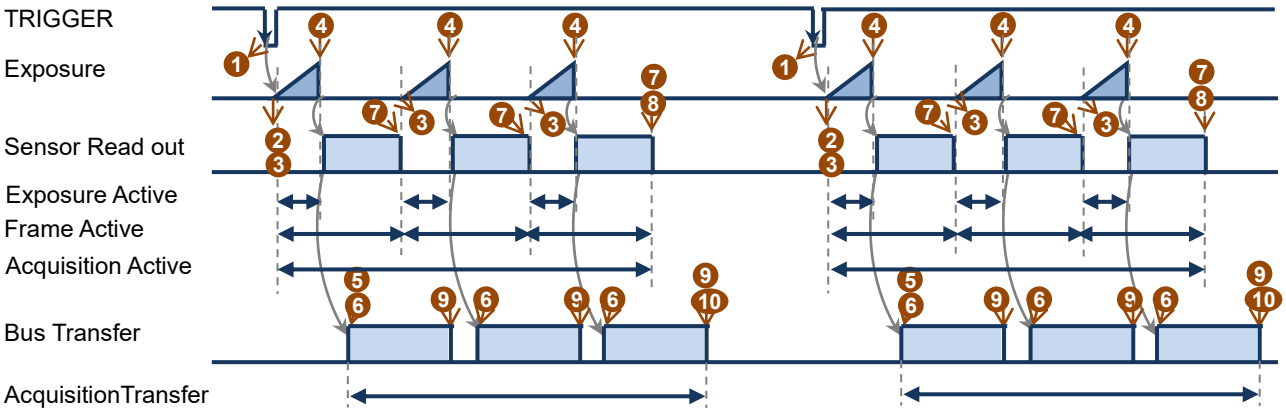
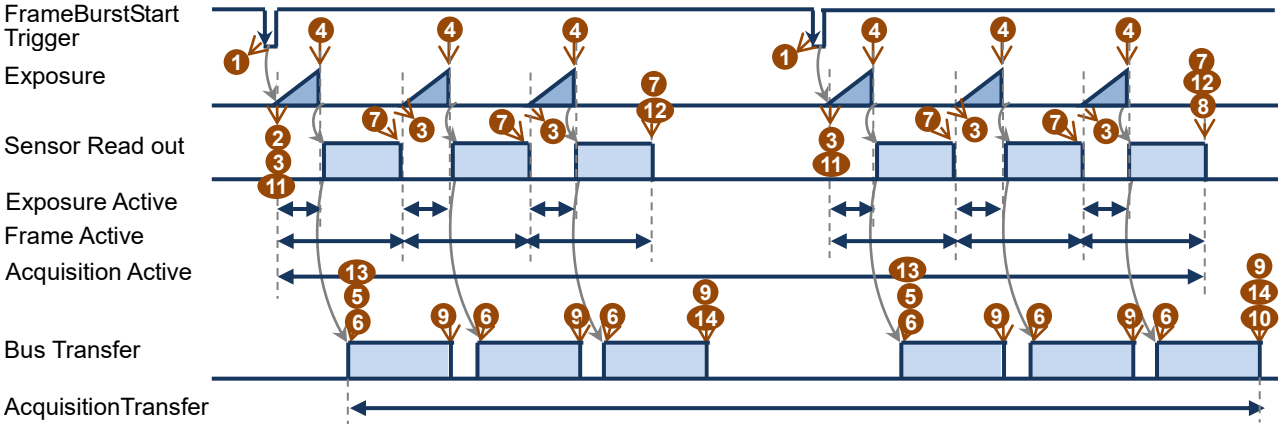


Figure 1 External Trigger mode



(1) AcquisitionTriggert
(2) AcquisitionStart
(3) ExposuretStart
(4) ExposureEnd
(5) AcquisitionTransferStart
(6) FrameTransferStart
(7) FrameEnd
(8) AcquisitionEnd
(9) FrameTransferEnd
(10) AcquisitionTransferEnd

Figure 2 Bulk Trigger mode, 3 frames per a trigger



(1) AcquisitionTriggert
(2) AcquisitionStart
(3) ExposuretStart
(4) ExposureEnd
(5) AcquisitionTransferStart
(6) FrameTransferStart
(7) FrameEnd
(8)AcquisitionEnd
(9) FrameTransferEnd
(10)AcquisitionTransferEnd
(11)FrameBurstStart
(12)FrameBurstEnd
(13)FrameBurstTransferStart
(14)FrameBurstTransferEnd

Figure 3 Burst Signals

[Example]

Refer to example of 5.4.1.1 Evt_OpenSimple.

5.4.3.3. Evt_Deactivate

This function deactivates specified Camera Event notification.

[Syntax]

```
CAM_API_STATUS  Evt_Deactivate (
  CAM_EVT_HANDLE        hEvt,
  const char            *pszEvtName
);
```

[Parameters]

Parameter		Description
hEvt	[in]	Handle of the target event interface..
pszEvtName	[in]	A pointer to a character array that contains node name (register name) of the deactivating Camera Event notification.

[Return value]

Returns result status. Refer to 5.8 Status code.

[Remarks]

The Camera Event notification that activated using "`Evt_Activate()`" must be released using this function before closing the event interface.

Including TeliCamAPI.h is required.

[Example]

Refer to example of 5.4.1.1 Evt_OpenSimple.

5.5. Controlling camera feature functions

The functions in this section perform control of camera features. These functions allow user application to control a camera without paying attention to interface type, model of the camera, and address of registers.

For cameras that comply with IIDC2 standard, GenICam GenApi libraries are not used. It is processed at high speed by register access.(Except for some functions)

For cameras that do not comply with IIDC 2, the GenICam GenApi library is used. All functions are not available when GenApi module was disabled on opening the camera.

There are functions that are available in all cameras. There also are functions that are not available in some model of camera because the corresponding features are not supported. Please check instruction manual of the camera to confirm the availability of the feature.

Note that there are functions whose functin name are different from register name of the camera or node nameof the camera description file (XML file) of the camera.

Note that Enum values used in functions of this section function are original values. Most enum values are same as node value of the camera description file (XML file) and same as camera register values, but some of them are different.

For example, integer value of USB3 Vision camera Trigger Source node is different from that of GigE Vision camera. Integer value of ‘SoftwareTrigger’ is 64 in USB3 Vision camera, 0 in GigE Vision camera.

Functions in this section will convert enum value to corresponding register value or node value, if necessary.

5.5.1. ImageFormatControl

This feature group performs control of image format of the camera. (Format0 ～ Format2)
The feature of image format control depends on the camera or camera's firmware version.

For details about ImageFormatControl features, refer to instruction manual of the camera.

5.5.1.1. GetCamImageFormatSelector

Gets the image format of the camera.

[Syntax]

```
CAM_API_STATUS  GetCamImageFormatSelector (
  CAM_HANDLE                    hCam,
  CAM_IMAGE_FORMAT_SELECTOR_TYPE   *peFormat
);
```

[Parameters]

Parameter		Description
hCam	[in]	Camera-Handle of target camera.
peFormat	[out]	A pointer to a variable that receives the type of image format.

[*CAM_IMAGE_FORMAT_SELECTOR_TYPE* Enumeration]

Member	Description
IMAGE_FORMAT0	Format0
IMAGE_FORMAT1	Format1
IMAGE_FORMAT2	Format2

[Return value]

Returns result status. Refer to 5.8 Status code.

[Remarks]

This function will return error status if ImageFormatSelector register (or node) is not implemented in the camera.

The feature may be different depending on the camera.
For details about ImageFormatControl features, refer to instruction manual of the camera.

Refer to TeliCamAPI.h about CAM_IMAGE_FORMAT_SELECTOR_TYPE.

Including TeliCamAPI.h is required.

5.5.1.2. SetCamImageFormatSelector

Sets the image format of the camera.

[Syntax]

```
CAM_API_STATUS  SetCamImageFormatSelector (
  CAM_HANDLE                        hCam,
  CAM_IMAGE_FORMAT_SELECTOR_TYPE    eFormat
);
```

[Parameters]

Parameter		Description
hCam	[in]	Camera-Handle of target camera.
eFormat	[in]	The type of image format.

[Return value]

Returns result status. Refer to 5.8 Status code.

[Remarks]

This function will return error status if ImageFormatSelector register (or node) is not implemented in the camera.

Settings can not be changed during image stream data output.

Feature may vary depending on the camera.
For details about ImageFormatControl features, refer to instruction manual of the camera.

Refer to TeliCamAPI.h about `CAM_IMAGE_FORMAT_SELECTOR_TYPE`.

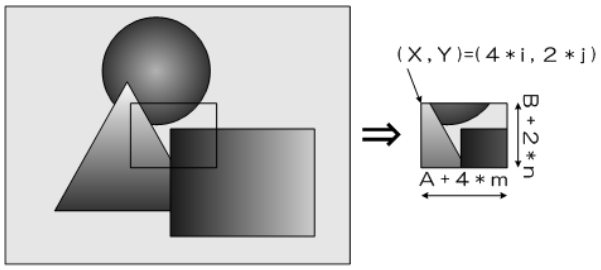
Including TeliCamAPI.h is required.

5.5.2. Scalable

This feature group performs control of scalable mode of the camera.

Scalable function reads out the region of interest (ROI) of the sensor.
If height size is set small, it is possible to increase the frame rate.
Only single rectangle is selectable. Concave or convex shape is not selectable.

- Window size: {A + 4 × m (H)} × {B + 2 × n (V)}
  A, B = minimum unit size
  m, n = integer
  The window size is equal or less than maximum image size.
- Start address: {4 x i (H)} x {2 x j (V)}
  i, j = integer
  The window size is equal or less than maximum image size.



For details about Scalable features, refer to instruction manual of the camera.

5.5.2.1. GetCamSensorWidth

Gets the effective width of the camera in pixels.

[Syntax]

```
CAM_API_STATUS  GetCamSensorWidth (
  CAM_HANDLE        hCam,
  uint32_t          *puiSensorWidth
);
```

[Parameters]

Parameter		Description
hCam	[in]	Camera-Handle of target camera.
puiSensorWidth	[out]	A pointer to a variable that receives the effective width of the sensor, in pixels.

[Return value]

Returns result status. Refer to 5.8 Status code.

[Remarks]

Including TeliCamAPI.h is required.

5.5.2.2. GetCamSensorHeight

Gets the effective height of the camera in pixels.

[Syntax]

```
CAM_API_STATUS  GetCamSensorHeight
  CAM_HANDLE        hCam,
  uint32_t          *puiSensorHeight
);
```

[Parameters]

Parameter		Description
hCam	[in]	Camera-Handle of target camera.
puiSensorHeight	[out]	A pointer to a variable that receives the effective height of the sensor, in pixels.

[Return value]

Returns result status. Refer to 5.8 Status code.

[Remarks]

Including TeliCamAPI.h is required.

5.5.2.3. GetCamRoi

Gets an image Region Of Interest (ROI) of the camera.

[Syntax]

```
CAM_API_STATUS  GetCamRoi (
  CAM_HANDLE        hCam,
  uint32_t          *puiWidth,
  uint32_t          *puiHeight,
  uint32_t          *puiOffsetX,
  uint32_t          *puiOffsetY
);
```

[Parameters]

Parameter		Description
hCam	[in]	Camera-Handle of target camera.
puiWidth	[out]	A pointer to a variable that receives the current image width, in pixels.
puiHeight	[out]	A pointer to a variable that receives the current image height, in pixels.
puiOffsetX	[out]	A pointer to a variable that receives the current horizontal offset from the origin to the region of interest, in pixels.
puiOffsetY	[out]	A pointer to a variable that receives the current vertical offset from the origin to the region of interest, in pixels .

[Return value]

Returns result status. Refer to 5.8 Status code.

[Remarks]

Including TeliCamAPI.h is required.

5.5.2.4. SetCamRoi

Sets an image Region Of Interest (ROI) of the camera.

[Syntax]

```
CAM_API_STATUS  SetCamRoi (
  CAM_HANDLE        hCam,
  uint32_t          uiWidth,
  uint32_t          uiHeight,
  uint32_t          uiOffsetX,
  uint32_t          uiOffsetY
);
```

[Parameters]

Parameter		Description
hCam	[in]	Camera-Handle of target camera.
uiWidth	[in]	The image width, in pixels.
uiHeight	[in]	The image height, in pixels.
uiOffsetX	[in]	The horizontal offset from the origin to the region of interest, in pixels.
uiOffsetY	[in]	The vertical offset from the origin to the region of interest, in pixels.

[Return value]

Returns result status. Refer to 5.8 Status code.

[Remarks]

Settings cannot be changed during image stream data output.
However, some cameras can only change OffsetX and OffsetY even during image stream data output.
For details about Scalable features, refer to instruction manual of the camera.

Including TeliCamAPI.h is required.

5.5.2.5. GetCamWidthMinMax

Gets the minimum and maximum available image width and increment value of the image width.

[Syntax]

```
CAM_API_STATUS  GetCamWidthMinMax (
  CAM_HANDLE        hCam,
  uint32_t          *puiWidthMin,
  uint32_t          *puiWidthMax,
  uint32_t          *puiWidthInc
);
```

[Parameters]

Parameter		Description
hCam	[in]	Camera-Handle of target camera.
puiWidthMin	[out]	A pointer to a variable that receives the minimum image width, in pixels.
puiWidthMax	[out]	A pointer to a variable that receives the maximum image width, in pixels.
puiWidthInc	[out]	A pointer to a variable that receives the increment value of the image width, in pixels.

[Return value]

Returns result status. Refer to 5.8 Status code.

[Remarks]

Including TeliCamAPI.h is required.

5.5.2.6. GetCamWidth

Gets the current image width in pixels.

[Syntax]

```
CAM_API_STATUS  GetCamWidth (
  CAM_HANDLE        hCam,
  uint32_t          *puiWidth
);
```

[Parameters]

Parameter		Description
hCam	[in]	Camera-Handle of target camera.
puiWidth	[out]	A pointer to a variable that receives the current image width, in pixels.

[Return value]

Returns result status. Refer to 5.8 Status code.

[Remarks]

Including TeliCamAPI.h is required.

5.5.2.7. SetCamWidth

Sets the current image width in pixels.

[Syntax]

```
CAM_API_STATUS  SetCamWidth (
  CAM_HANDLE        hCam,
  uint32_t          uiWidth
);
```

[Parameters]

Parameter		Description
hCam	[in]	Camera-Handle of target camera.
uiWidth	[in]	The image width, in pixels

[Return value]

Returns result status. Refer to 5.8 Status code.

[Remarks]

Settings cannot be changed during image stream data output.

Including TeliCamAPI.h is required.

5.5.2.8. GetCamHeightMinMax

Gets the minimum and maximum available image height and increment value of the image height.

[Syntax]

```
CAM_API_STATUS  GetCamHeightMinMax (
  CAM_HANDLE        hCam,
  uint32_t          *puiHeightMin,
  uint32_t          *puiHeightMax,
  uint32_t          *puiHeightInc
);
```

[Parameters]

Parameter		Description
hCam	[in]	Camera-Handle of target camera.
puiHeightMin	[out]	A pointer to a variable that receives the minimum image height, in pixels.
puiHeightMax	[out]	A pointer to a variable that receives the maximum image height, in pixels.
puiHeightInc	[out]	A pointer to a variable that receives the increment value of the image height, in pixels.

[Return value]

Returns result status. Refer to 5.8 Status code.

[Remarks]

Including TeliCamAPI.h is required.

5.5.2.9. GetCamHeight

Sets the current image height in pixels.

[Syntax]

```
CAM_API_STATUS  GetCamHeight (
  CAM_HANDLE        hCam,
  uint32_t          *puiHeight
);
```

[Parameters]

Parameter		Description
hCam	[in]	Camera-Handle of target camera.
puiHeight	[out]	A pointer to a variable that receives the current image height, in pixels.

[Return value]

Returns result status. Refer to 5.8 Status code.

[Remarks]

Including TeliCamAPI.h is required.

5.5.2.10. SetCamHeight

Sets the current image height in pixels.

[Syntax]

```
CAM_API_STATUS  SetCamHeight (
  CAM_HANDLE        hCam,
  uint32_t          uiHeight
);
```

[Parameters]

Parameter		Description
hCam	[in]	Camera-Handle of target camera.
uiHeight	[in]	The image height, in pixels.

[Return value]

Returns result status. Refer to 5.8 Status code.

[Remarks]

Settings cannot be changed during image stream data output.

Including TeliCamAPI.h is required.

5.5.2.11. GetCamOffsetXMinMax

Gets the minimum and maximum available horizontal offset from the origin to the region of interest and the increment value.

[Syntax]

```
CAM_API_STATUS  GetCamOffsetXMinMax (
  CAM_HANDLE        hCam,
  uint32_t          *puiOffsetXMin,
  uint32_t          *puiOffsetXMax,
  uint32_t          *puiOffsetXInc
);
```

[Parameters]

Parameter		Description
hCam	[in]	Camera-Handle of target camera.
puiOffsetXMin	[out]	A pointer to a variable that receives the minimum horizontal image offset, in pixels.
puiOffsetXMax	[out]	A pointer to a variable that receives the maximum horizontal image offset, in pixels.
puiOffsetXInc	[out]	A pointer to a variable that receives the increment value of horizontal image offset, in pixels.

[Return value]

Returns result status. Refer to 5.8 Status code.

[Remarks]

Including TeliCamAPI.h is required.

5.5.2.12. GetCamOffsetX

Gets the horizontal offset from the origin to the region of interest in pixels.

[Syntax]

```
CAM_API_STATUS  GetCamOffsetX (
  CAM_HANDLE        hCam,
  uint32_t          *puiOffsetX
);
```

[Parameters]

Parameter		Description
hCam	[in]	Camera-Handle of target camera.
puiOffsetX	[out]	A pointer to a variable that receives the current horizontal image offset, in pixels.

[Return value]

Returns result status. Refer to 5.8 Status code.

[Remarks]

Including TeliCamAPI.h is required.

5.5.2.13. SetCamOffsetX

Sets the horizontal offset from the origin to the region of interest in pixels.

[Syntax]

```
CAM_API_STATUS  SetCamOffsetX (
  CAM_HANDLE        hCam,
  uint32_t          uiOffsetX
);
```

[Parameters]

Parameter		Description
hCam	[in]	Camera-Handle of target camera.
uiOffsetX	[in]	The horizontal image offset, in pixels.

[Return value]

Returns result status. Refer to 5.8 Status code.

[Remarks]

Whether it can be set during image stream data output varies depending on the camera.
For details about Scalable features, refer to instruction manual of the camera.

Including TeliCamAPI.h is required.

5.5.2.14. GetCamOffsetYMinMax

Gets the minimum and maximum available vertical offset from the origin to the region of interest and the increment value.

[Syntax]

```
CAM_API_STATUS  GetCamOffsetYMinMax (
  CAM_HANDLE        hCam,
  uint32_t          *puiOffsetYMin,
  uint32_t          *puiOffsetYMax,
  uint32_t          *puiOffsetYInc
);
```

[Parameters]

Parameter		Description
hCam	[in]	Camera-Handle of target camera.
puiOffsetYMin	[out]	A pointer to a variable that receives the minimum vertical image offset, in pixels.
puiOffsetYMax	[out]	A pointer to a variable that receives the maximum vertical image offset, in pixels.
puiOffsetYInc	[out]	A pointer to a variable that receives the pitch of vertical image offset, in pixels.

[Return value]

Returns result status. Refer to 5.8 Status code.

[Remarks]

Including TeliCamAPI.h is required.

5.5.2.15. GetCamOffsetY

Gets the vertical offset from the origin to the region of interest in pixels.

[Syntax]

```
CAM_API_STATUS  GetCamOffsetY (
  CAM_HANDLE        hCam,
  uint32_t          *puiOffsetY
);
```

[Parameters]

Parameter		Description
hCam	[in]	Camera-Handle of target camera.
puiOffsetY	[out]	A pointer to a variable that receives the current vertical image offset, in pixels.

[Return value]

Returns result status. Refer to 5.8 Status code.

[Remarks]

Including TeliCamAPI.h is required.

5.5.2.16. SetCamOffsetY

Sets the vertical offset from the origin to the region of interest in pixels.

[Syntax]

```
CAM_API_STATUS  SetCamOffsetY (
  CAM_HANDLE        hCam,
  uint32_t          uiOffsetY
);
```

[Parameters]

Parameter		Description
hCam	[in]	Camera-Handle of target camera.
uiOffsetY	[in]	The vertical image offset, in pixels.

[Return value]

Returns result status. Refer to 5.8 Status code.

[Remarks]

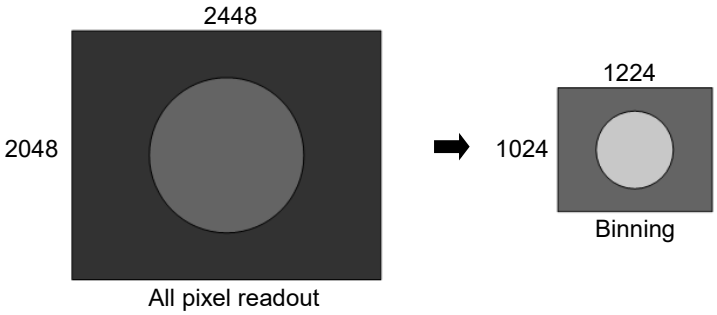
Whether it can be set during image stream data output varies depending on the camera.
For details about Scalable features, refer to instruction manual of the camera.

Including TeliCamAPI.h is required.

5.5.3. Binning

This feature group performs control of binning features of the camera.

Binning features add the neighboring pixels together.
Binning features can increase the sensitivity of the image, make frame rate faster, and decrease interface bandwidth occupation.



Binning operation（e.g. 2x2 binning）

For details about Binning features, refer to instruction manual of the camera.

5.5.3.1. GetCamBinningHorizontalMinMax

Gets the minimum and maximum available horizontal binning mode value of the camera.

[Syntax]

```
CAM_API_STATUS  GetCamBinningHorizontalMinMax (
  CAM_HANDLE        hCam,
  uint32_t          *puiMin
  uint32_t          *puiMax
);
```

[Parameters]

Parameter		Description
hCam	[in]	Camera-Handle of target camera.
puiMin	[out]	A pointer to a variable that receives the minimum value of horizontal binning mode.
puiMax	[out]	A pointer to a variable that receives the maximum value of horizontal binning mode.

[Return value]

Returns result status. Refer to 5.8 Status code.

[Remarks]

This function will return error status if BinningHorizontal register (or node) is not implemented in the camera.

Conditions that can get and set the binning parameters depend on the camera.
For details about Binning features, refer to instruction manual of the camera.

Including TeliCamAPI.h is required.

5.5.3.2. GetCamBinningHorizontal

Gets the horizontal binning mode value of the camera.

[Syntax]

```
CAM_API_STATUS  GetCamBinningHorizontal (
  CAM_HANDLE        hCam,
  uint32_t          *puiValue
);
```

[Parameters]

Parameter		Description
hCam	[in]	Camera-Handle of target camera.
puiValue	[out]	A pointer to a variable that receives the value of horizontal binning mode.

[Return value]

Returns result status. Refer to 5.8 Status code.

[Remarks]

This function will return error status if BinningHorizontal register (or node) is not implemented in the camera.

Conditions that can get and set the binning parameters depend on the camera.
For details about Binning features, refer to instruction manual of the camera.

Including TeliCamAPI.h is required.

5.5.3.3. SetCamBinningHorizontal

Sets the horizontal binning mode value of the camera.

[Syntax]

```
CAM_API_STATUS  SetCamBinningHorizontal (
  CAM_HANDLE        hCam,
  uint32_t          uiValue
);
```

[Parameters]

Parameter		Description
hCam	[in]	Camera-Handle of target camera.
uiValue	[in]	The value of horizontal binning mode.

[Return value]

Returns result status. Refer to 5.8 Status code.

[Remarks]

This function will return error status if BinningHorizontal register (or node) is not implemented in the camera.
Settings cannot be changed during image stream data output.

Conditions that can get and set the binning parameters depend on the camera.
For details about Binning features, refer to instruction manual of the camera.

Including TeliCamAPI.h is required.

5.5.3.4. GetCamBinningVerticalMinMax

Gets the minimum and maximum available vertical binning mode value of the camera.

[Syntax]

```
CAM_API_STATUS  GetCamBinningVerticalMinMax (
  CAM_HANDLE        hCam,
  uint32_t          *puiMin
  uint32_t          *puiMax
);
```

[Parameters]

Parameter		Description
hCam	[in]	Camera-Handle of target camera.
puiMin	[out]	A pointer to a variable that receives the minimum value of vertical binning mode.
puiMax	[out]	A pointer to a variable that receives the maximum value of vertical binning mode.

[Return value]

Returns result status. Refer to 5.8 Status code.

[Remarks]

This function will return error status if BinningVertical register (or node) is not implemented in the camera.

Conditions that can get and set the binning parameters depend on the camera.
For details about Binning features, refer to instruction manual of the camera.

Including TeliCamAPI.h is required.

5.5.3.5. GetCamBinningVertical

Gets the vertical binning mode value of the camera.

[Syntax]

```
CAM_API_STATUS  GetCamBinningVertical (
  CAM_HANDLE        hCam,
  uint32_t          *puiValue
);
```

[Parameters]

Parameter		Description
hCam	[in]	Camera-Handle of target camera.
puiValue	[out]	A pointer to a variable that receives the value of vertical binning mode.

[Return value]

Returns result status. Refer to 5.8 Status code.

[Remarks]

This function will return error status if BinningVertical register (or node) is not implemented in the camera.

Conditions that can get and set the binning parameters depend on the camera.
For details about Binning features, refer to instruction manual of the camera.

Including TeliCamAPI.h is required.

5.5.3.6. SetCamBinningVertical

Sets the vertical binning mode value of the camera.

[Syntax]

```
CAM_API_STATUS  SetCamBinningVertical (
  CAM_HANDLE        hCam,
  uint32_t          uiValue
);
```

[Parameters]

Parameter		Description
hCam	[in]	Camera-Handle of target camera.
uiValue	[in]	The value of vertical binning mode.

[Return value]

Returns result status. Refer to 5.8 Status code.

[Remarks]

This function will return error status if BinningVertical register (or node) is not implemented in the camera.

Settings cannot be changed during image stream data output.

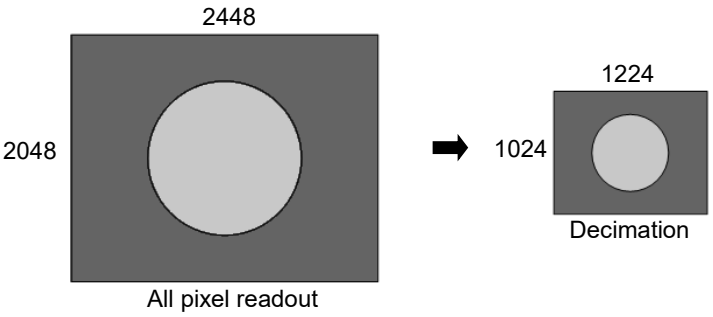
Conditions that can get and set the binning parameters depend on the camera.
For details about Binning features, refer to instruction manual of the camera.

Including TeliCamAPI.h is required.

5.5.4. Decimation

This feature group performs control of decimation feature of the camera.

Decimation features reads out all effective areas at high speed by skipping pixels and lines.
Decimation features can make frame rate faster, and decrease interface bandwidth occupation.



Decimation operation（e.g. 2x2 decimation）

For details about Decimation features, refer to instruction manual of the camera.

5.5.4.1. GetCamDecimationHorizontalMinMax

Gets the minimum and maximum available horizontal decimation mode value of the camera.

[Syntax]

```
CAM_API_STATUS  GetCamDecimationHorizontalMinMax (
  CAM_HANDLE        hCam,
  uint32_t          *puiMin
  uint32_t          *puiMax
);
```

[Parameters]

Parameter		Description
hCam	[in]	Camera-Handle of target camera.
puiMin	[out]	A pointer to a variable that receives the minimum value of horizontal decimation mode.
puiMax	[out]	A pointer to a variable that receives the maximum value of horizontal decimation mode.

[Return value]

Returns result status. Refer to 5.8 Status code.

[Remarks]

This function will return error status if DecimationHorizontal register (or node) is not implemented in the camera.

Conditions that can get and set the decimation parameters depend on the camera.
For details about Decimation features, refer to instruction manual of the camera.

Including TeliCamAPI.h is required.

5.5.4.2. GetCamDecimationHorizontal

Gets the horizontal decimation mode value of the camera.

[Syntax]

```
CAM_API_STATUS  GetCamDecimationHorizontal (
  CAM_HANDLE        hCam,
  uint32_t          *puiValue
);
```

[Parameters]

Parameter		Description
hCam	[in]	Camera-Handle of target camera.
puiValue	[out]	A pointer to a variable that receives the value of horizontal decimation mode.

[Return value]

Returns result status. Refer to 5.8 Status code.

[Remarks]

This function will return error status if DecimationHorizontal register (or node) is not implemented in the camera.

Conditions that can get and set the decimation parameters depend on the camera.
For details about Decimation features, refer to instruction manual of the camera.

Including TeliCamAPI.h is required.

5.5.4.3. SetCamDecimationHorizontal

Sets the horizontal decimation mode value of the camera.

[Syntax]

```
CAM_API_STATUS  SetCamDecimationHorizontal (
  CAM_HANDLE        hCam,
  uint32_t          uiValue
);
```

[Parameters]

Parameter		Description
hCam	[in]	Camera-Handle of target camera.
uiValue	[in]	The value of horizontal decimation mode.

[Return value]

Returns result status. Refer to 5.8 Status code.

[Remarks]

This function will return error status if DecimationHorizontal register (or node) is not implemented in the camera.

Settings cannot be changed during image stream data output.

Conditions that can get and set the decimation parameters depend on the camera.
For details about Decimation features, refer to instruction manual of the camera.

Including TeliCamAPI.h is required.

5.5.4.4. GetCamDecimationVerticalMinMax

Gets the minimum and maximum available vertical decimation mode value of the camera.

[Syntax]

```
CAM_API_STATUS  GetCamDecimationVerticalMinMax (
  CAM_HANDLE        hCam,
  uint32_t          *puiMin
  uint32_t          *puiMax
);
```

[Parameters]

Parameter		Description
hCam	[in]	Camera-Handle of target camera.
puiMin	[out]	A pointer to a variable that receives the minimum value of vertical decimation mode.
puiMax	[out]	A pointer to a variable that receives the maximum value of vertical decimation mode.

[Return value]

Returns result status. Refer to 5.8 Status code.

[Remarks]

This function will return error status if DecimationVertical register (or node) is not implemented in the camera.

Conditions that can get and set the decimation parameters depend on the camera.
For details about Decimation features, refer to instruction manual of the camera.

Including TeliCamAPI.h is required.

5.5.4.5. GetCamDecimationVertical

Gets the vertical decimation mode value of the camera.

[Syntax]

```
CAM_API_STATUS  GetCamDecimationVertical (
  CAM_HANDLE        hCam,
  uint32_t          *puiValue
);
```

[Parameters]

Parameter		Description
hCam	[in]	Camera-Handle of target camera.
puiValue	[out]	A pointer to a variable that receives the value of vertical decimation mode.

[Return value]

Returns result status. Refer to 5.8 Status code.

[Remarks]

This function will return error status if DecimationVertical register (or node) is not implemented in the camera.

Conditions that can get and set the decimation parameters depend on the camera.
For details about Decimation features, refer to instruction manual of the camera.

Including TeliCamAPI.h is required.

5.5.4.6. SetCamDecimationVertical

Sets the vertical decimation mode value of the camera.

[Syntax]

```
CAM_API_STATUS  SetCamDecimationVertical (
  CAM_HANDLE        hCam,
  uint32_t          uiValue
);
```

[Parameters]

Parameter		Description
hCam	[in]	Camera-Handle of target camera.
uiValue	[in]	The value of vertical decimation mode.

[Return value]

Returns result status. Refer to 5.8 Status code.

[Remarks]

This function will return error status if DecimationVertical register (or node) is not implemented in the camera.

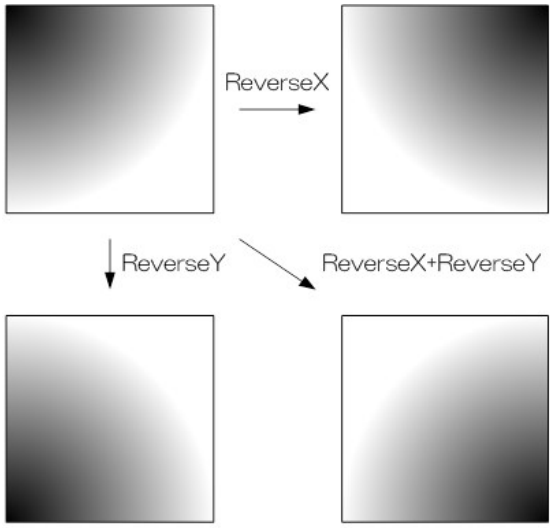
Settings cannot be changed during image stream data output.

Conditions that can get and set the decimation parameters depend on the camera.
For details about Decimation features, refer to instruction manual of the camera.

Including TeliCamAPI.h is required.

5.5.5. Reverse

This feature group performs control of image flipping feature of the camera.



For details about Reverse features, refer to instruction manual of the camera.

5.5.5.1. GetCamReverseX

Gets the horizontal image flip mode value of the camera.

[Syntax]

```
CAM_API_STATUS  GetCamReverseX (
  CAM_HANDLE       hCam,
  bool8_t          *pbValue
);
```

[Parameters]

Parameter		Description
hCam	[in]	Camera-Handle of target camera.
pbValue	[out]	A pointer to a variable that receives the value of horizontal image flip mode. If the value is false, the image is not flipped horizontally. If the value is true, the image is flipped horizontally.

[Return value]

Returns result status. Refer to 5.8 Status code.

[Remarks]

This function will return error status if ReverseX register (or node) is not implemented in the camera.

Including TeliCamAPI.h is required.

5.5.5.2. SetCamReverseX

Sets the horizontal image flip mode value of the camera.

[Syntax]

```
CAM_API_STATUS  SetCamReverseX (
  CAM_HANDLE        hCam,
  bool8_t           bValue
);
```

[Parameters]

Parameter		Description
hCam	[in]	Camera-Handle of target camera.
bValue	[in]	The value of horizontal image flip mode. If the value is false, the image is not flipped horizontally. If the value is true, the image is flipped horizontally.

[Return value]

Returns result status. Refer to 5.8 Status code.

[Remarks]

This function will return error status if ReverseX register (or node) is not implemented in the camera.

Settings cannot be changed during image stream data output.

Including TeliCamAPI.h is required.

5.5.5.3. GetCamReverseY

Gets the vertical image flip mode value of the camera.

[Syntax]

```
CAM_API_STATUS  GetCamReverseY (
  CAM_HANDLE        hCam,
  bool8_t           *pbValue
);
```

[Parameters]

Parameter		Description
hCam	[in]	Camera-Handle of target camera.
pbValue	[out]	A pointer to a variable that receives the value of vertical image flip mode. If the value is false, the image is not flipped vertically. If the value is true, the image is flipped vertically.

[Return value]

Returns result status. Refer to 5.8 Status code.

[Remarks]

This function will return error status if ReverseY register (or node) is not implemented in the camera.

Including TeliCamAPI.h is required.

5.5.5.4. SetCamReverseY

Sets the vertical image flip mode value of the camera.

[Syntax]

```
CAM_API_STATUS  SetCamReverseY (
  CAM_HANDLE        hCam,
  bool8_t           bValue
);
```

[Parameters]

Parameter		Description
hCam	[in]	Camera-Handle of target camera.
bValue	[in]	The value of vertical image flip mode. If the value is false, the image is not flipped vertically. If the value is true, the image is flipped vertically.

[Return value]

Returns result status. Refer to 5.8 Status code.

[Remarks]

This function will return error status if ReverseY register (or node) is not implemented in the camera.

Settings cannot be changed during image stream data output.

Including TeliCamAPI.h is required.

5.5.6. PixelFormat

This feature group performs control of PixelFormat of image to be acquired.
For details about PixelFormat features, refer to instruction manual of the camera.

5.5.6.1. GetCamPixelFormat

Gets the pixel format of the camera.

[Syntax]

```
CAM_API_STATUS  GetCamPixelFormat (
  CAM_HANDLE           hCam,
  CAM_PIXEL_FORMAT     *puiPixelFormat
);
```

[Parameters]

Parameter		Description
hCam	[in]	Camera-Handle of target camera.
puiPixelFormat	[out]	A pointer to a variable that receives the type of pixel format.

[*CAM_PIXEL_FORMAT* Enumeration]

Member	Description
PXL_FMT_Unknown	Unknown format
PXL_FMT_Mono8	Mono8 format
PXL_FMT_Mono10	Mono10 format
PXL_FMT_Mono12	Mono12 format
PXL_FMT_Mono16	Mono16 format
PXL_FMT_BayerGR8	BayerGR8 format
PXL_FMT_BayerGR10	BayerGR10 format
PXL_FMT_BayerGR12	BayerGR12 format
PXL_FMT_BayerRG8	BayerRG8 format
PXL_FMT_BayerRG10	BayerRG10 format
PXL_FMT_BayerRG12	BayerRG12 format
PXL_FMT_BayerGB8	BayerGB8 format
PXL_FMT_BayerGB10	BayerGB10 format
PXL_FMT_BayerGB12	BayerGB12 format
PXL_FMT_BayerBG8	BayerBG8 format
PXL_FMT_BayerBG10	BayerBG10 format
PXL_FMT_BayerBG12	BayerBG12 format
PXL_FMT_RGB8	RGB8 format
PXL_FMT_BGR8	BGR8 format
PXL_FMT_BGR10	BGR10 format
PXL_FMT_BGR12	BGR12 format
PXL_FMT_YUV411_8	YUV411_8 format
PXL_FMT_YUV422_8	YUV422_8 format
PXL_FMT_YUV8	YUV8 format

[Return value]

Returns result status. Refer to 5.8 Status code.

[Remarks]

This function will return error status if PixelFormat register (or node), or PixelCoding and PixelSize registers (or nodes) are not implemented in the camera.

Refer to TeliCamAPI.h about CAM_PIXEL_FORMAT.

Including TeliCamAPI.h is required.

5.5.6.2. SetCamPixelFormat

Sets the pixel format of the camera.

[Syntax]

```
CAM_API_STATUS  SetCamPixelFormat (
  CAM_HANDLE           hCam,
  CAM_PIXEL_FORMAT     uiPixelFormat
);
```

[Parameters]

Parameter		Description
hCam	[in]	Camera-Handle of target camera.
uiPixelFormat	[in]	The type of pixel format.

[Return value]

Returns result status. Refer to 5.8 Status code.

[Remarks]

This function will return error status if PixelFormat register (or node), or PixelCoding and PixelSize registers (or nodes) are not implemented in the camera.

Settings cannot be changed during image stream data output.

Refer to TeliCamAPI.h about CAM_PIXEL_FORMAT.

Including TeliCamAPI.h is required.

5.5.7. TestPattern

This feature group performs control of test pattern generation function of the camera.

For details about TestPattern features, refer to instruction manual of the camera.

5.5.7.1. GetCamTestPattern

Gets the test pattern type of the camera.

[Syntax]

```
CAM_API_STATUS  GetCamTestPattern (
  CAM_HANDLE                hCam,
  CAM_TEST_PATTERN_TYPE     *peTestPattern
);
```

[Parameters]

Parameter		Description
hCam	[in]	Camera-Handle of target camera.
peTestPattern	[out]	A pointer to a variable that receives the type of test pattern.

[*CAM_TEST_PATTERN_TYPE* Enumeration]

Member	Description
CAM_TEST_PATTERN_OFF	Test pattern disable(Normal data output)
CAM_TEST_PATTERN_BLACK	All pixel = 0 LSB
CAM_TEST_PATTERN_WHITE	All pixel = 255 @Mono8
CAM_TEST_PATTERN_GREY_A	All pixel = 170 @Mono8
CAM_TEST_PATTERN_GREY_B	All pixel = 85 @Mono8
CAM_TEST_PATTERN_GREY_HORIZONTAL_RAMP	Grey Horizontal Ramp
CAM_TEST_PATTERN_GREY_SCALE	Grey scale
CAM_TEST_PATTERN_COLOR_BAR	Color Bar
CAM_TEST_PATTERN_GREY_VERTICAL_RAMP	Grey Vertical Ramp

[Return value]

Returns result status. Refer to 5.8 Status code.

[Remarks]

This function will return error status if TestPattern, or TestImageSelector register (or node) is not implemented in the camera.

Refer to TeliCamAPI.h about `CAM_TEST_PATTERN_TYPE`.

Including TeliCamAPI.h is required.

5.5.7.2. SetCamTestPattern

Sets the test pattern type of the camera.

[Syntax]

```
CAM_API_STATUS  SetCamTestPattern (
  CAM_HANDLE              hCam,
  CAM_TEST_PATTERN_TYPE   eTestPattern
);
```

[Parameters]

Parameter		Description
hCam	[in]	Camera-Handle of target camera.
eTestPattern	[in]	The type of test pattern.

[Return value]

Returns result status. Refer to 5.8 Status code.

[Remarks]

Refer to TeliCamAPI.h about CAM_TEST_PATTERN_TYPE.

This function will return error status if “TestPattern” or “TestImageSelector” register or node is not implemented in the camera.

Including TeliCamAPI.h is required.

5.5.8. AcquisitionControl

This feature group performs control of image acquisition of the camera.

For details about AcquisitionControl features, refer to instruction manual of the camera.

5.5.8.1. GetCamStreamPayloadSize

Gets the current payload size of image stream.
The payload size is the number of bytes transferred for each image or chunk on the stream channel.

[Syntax]

```
CAM_API_STATUS  GetCamStreamPayloadSize (
  CAM_HANDLE        hCam,
  uint32_t          *puiPayloadSize
);
```

[Parameters]

Parameter		Description
hCam	[in]	Camera-Handle of target camera.
puiPayloadSize	[out]	A pointer to a variable that receives the current payload size in bytes.

[Return value]

Returns result status. Refer to 5.8 Status code.

[Remarks]

Including TeliCamAPI.h is required.

5.5.8.2. GetStreamEnable

Gets the enable state of image streaming in the camera.

[Syntax]

```
CAM_API_STATUS  GetStreamEnable (
  CAM_HANDLE        hCam,
  bool8_t           *pbEnable
);
```

[Parameters]

Parameter		Description
hCam	[in]	Camera-Handle of target camera.
pbEnable	[out]	A pointer to a variable that receives the enable state of image streaming. If false, image streaming is disable. If true, image streaming is enable.

[Return value]

Returns result status. Refer to 5.8 Status code.

[Remarks]

This function is available only for USB3 Vision camera.
Including TeliCamAPI.h is required.

5.5.8.3. GetCamAcquisitionFrameCountMinMax

Gets the minimum and maximum available AcquisitionFrameCount value of the camera.

AcquisitionFrameCount is the number of frames to transfer in MultiFrame/ImageBuffer mode.

[Syntax]

```
CAM_API_STATUS  GetCamAcquisitionFrameCountMinMax (
  CAM_HANDLE        hCam,
  uint32_t          *puiAcqFrameCountMin,
  uint32_t          *puiAcqFrameCountMax
);
```

[Parameters]

Parameter		Description
hCam	[in]	Camera-Handle of target camera.
puiAcqFrameCountMin	[out]	A pointer to a variable that receives the minimum value of AcquisitionFrameCount.
puiAcqFrameCountMax	[out]	A pointer to a variable that receives the maximum value of AcquisitionFrameCount.

[Return value]

Returns result status. Refer to 5.8 Status code.

[Remarks]

This function will return error status if AcquisitionFrameCount register (or node) is not implemented in the camera.

Including TeliCamAPI.h is required.

5.5.8.4. GetCamAcquisitionFrameCount

Gets the AcquisitionFrameCount value of the camera.

In MultiFrame acquisition mode, the camera will stop image acquisition when transferred image count reached to the AcquisitionFrameCount value.

In ImageBufferRead mode, the camera will send the number of frames set to the AcquisitionFrameCount from the image buffer when ExecuteCamImageBufferRead() is called.

[Syntax]

```
CAM_API_STATUS  GetCamAcquisitionFrameCount (
  CAM_HANDLE        hCam,
  uint32_t          *puiAcqFrameCount
);
```

[Parameters]

Parameter		Description
hCam	[in]	Camera-Handle of target camera.
puiAcqFrameCount	[out]	A pointer to a variable that receives the value of AcquisitionFrameCount.

[Return value]

Returns result status. Refer to 5.8 Status code.

[Remarks]

This function will return error status if AcquisitionFrameCount register (or node) is not implemented in the camera.

Including TeliCamAPI.h is required.

5.5.8.5. SetCamAcquisitionFrameCount

Sets the AcquisitionFrameCount value of the camera.

In MultiFrame acquisition mode, the camera will stop image acquisition when transferred image count reached to the AcquisitionFrameCount value.

In ImageBufferRead mode, the camera will send the number of frames set to the AcquisitionFrameCount from the image buffer when ExecuteCamImageBufferRead() is called.

[Syntax]

```
CAM_API_STATUS  SetCamAcquisitionFrameCount (
  CAM_HANDLE        hCam,
  uint32_t          uiAcqFrameCount
);
```

[Parameters]

Parameter		Description
hCam	[in]	Camera-Handle of target camera.
uiAcqFrameCount	[in]	The value of AcquisitionFrameCount.

[Return value]

Returns result status. Refer to 5.8 Status code.

[Remarks]

This function will return error status if AcquisitionFrameCount register (or node) is not implemented in the camera.

Settings cannot be changed during image stream data output.

Including TeliCamAPI.h is required.

5.5.8.6. GetCamAcquisitionFrameRateControl

Gets the frame rate setting of the camera.

[Syntax]

```
CAM_API_STATUS  GetCamAcquisitionFrameRateControl (
  CAM_HANDLE                    hCam,
  CAM_ACQ_FRAME_RATE_CTRL_TYPE  *peFrameRateControl
);
```

[Parameters]

Parameter		Description
hCam	[in]	Camera-Handle of target camera.
peFrameRateControl	[out]	A pointer to a variable that receives the type of frame rate setting.

[*CAM_ACQ_FRAME_RATE_CTRL_TYPE* Enumeration]

Member	Description
CAM_ACQ_FRAME_RATE_CTRL_NO_SPECIFY	The camera will decide frame rate using parameters other than AcquisitionFrameRate register and the decided frame rrate value will be read out from AcquisitionFrameRate register.
CAM_ACQ_FRAME_RATE_CTRL_MANUAL	The camera will control frame rate to realize AcquisitionFrameRate register value.

[Return value]

Returns result status. Refer to 5.8 Status code.

[Remarks]

This function will return error status if AcquisitionFrame register (or AcquisitionFrameControl / AcquisitionFrameRateEnable node) is not implemented in the camera.

Refer to TeliCamAPI.h about CAM_ACQ_FRAME_RATE_CTRL_TYPE.

Including TeliCamAPI.h is required.

5.5.8.7. SetCamAcquisitionFrameRateControl

Sets the frame rate setting of the camera.

[Syntax]

```
CAM_API_STATUS  SetCamAcquisitionFrameRateControl (
  CAM_HANDLE                    hCam,
  CAM_ACQ_FRAME_RATE_CTRL_TYPE  eFrameRateControl
);
```

[Parameters]

Parameter		Description
hCam	[in]	Camera-Handle of target camera.
eFrameRateControl	[in]	The type of frame rate setting.

[Return value]

Returns result status. Refer to 5.8 Status code.

[Remarks]

This function will return error status if AcquisitionFrame register (or AcquisitionFrameControl / AcquisitionFrameRateEnable node) is not implemented in the camera.

When CAM_ACQ_FRAME_RATE_CTRL_NO_SPECIFY is set, the AcquisitionFrameRate register (or node) value may change. Please check the operation of the camera to be used before using.

Refer to TeliCamAPI.h about `CAM_ACQ_FRAME_RATE_CTRL_TYPE`.
Including TeliCamAPI.h is required.

5.5.8.8. GetCamAcquisitionFrameRateMinMax

Gets the minimum and maximum available frame rate value of the camera.

[Syntax]

```
CAM_API_STATUS  GetCamAcquisitionFrameRateMinMax (
  CAM_HANDLE      hCam,
  float64_t       *pdAcqFrameRateMin,
  float64_t       *pdAcqFrameRateMax
);
```

[Parameters]

Parameter		Description
hCam	[in]	Camera-Handle of target camera.
pdAcqFrameRateMin	[out]	A pointer to a variable that receives the minimum value of frame rate.
pdAcqFrameRateMax	[out]	A pointer to a variable that receives the maximum value of frame rate.

[Return value]

Returns result status. Refer to 5.8 Status code.

[Remarks]

This function will return error status if AcquisitionFrame register (or AcquisitionFrameControl / AcquisitionFrameRateEnable node) is not implemented in the camera.

Including TeliCamAPI.h is required.

5.5.8.9. GetCamAcquisitionFrameRate

Gets the frame rate value of the camera.

[Syntax]

```
CAM_API_STATUS  GetCamAcquisitionFrameRate (
  CAM_HANDLE        hCam,
  float64_t         *pdAcqFrameRate
);
```

[Parameters]

Parameter		Description
hCam	[in]	Camera-Handle of target camera.
pdAcqFrameRate	[out]	A pointer to a variable that receives the value of frame rate.

[Return value]

Returns result status. Refer to 5.8 Status code.

[Remarks]

This function will return error status if AcquisitionFrame register (or AcquisitionFrameControl / AcquisitionFrameRateEnable node) is not implemented in the camera.

Including TeliCamAPI.h is required.

5.5.8.10. SetCamAcquisitionFrameRate

Sets the frame rate value of the camera.

[Syntax]

```
CAM_API_STATUS  SetCamAcquisitionFrameRate (
  CAM_HANDLE        hCam,
  float64_t         dAcqFrameRate
);
```

[Parameters]

Parameter		Description
hCam	[in]	Camera-Handle of target camera.
dAcqFrameRate	[in]	The value of frame rate.

[Return value]

Returns result status. Refer to 5.8 Status code.

[Remarks]

This function will return error status if AcquisitionFrame register (or AcquisitionFrameControl / AcquisitionFrameRateEnable node) is not implemented in the camera.

If AcquisitionFrameRateControl value is `CAM_ACQ_FRAME_RATE_CTRL_NO_SPECIFY`, this function may fail to write data, or the camera may overwrite AcquisitionFrameRateControl value with other value immediately.

Whether it can be set during image stream data output varies depending on the camera.
For details about AcquisitionControl features, refer to instruction manual of the camera.

Including TeliCamAPI.h is required.

5.5.8.11. ExecuteCamAcquisitionStart

Sends a AcquisitionStart command to the camera.

It is valid only when streaming is running in single-frame mode or multi-frame mode.
Execute when acquiring new images.

[Syntax]

```
CAM_API_STATUS  ExecuteCamAcquisitionStart (
  CAM_HANDLE            hCam
);
```

[Parameters]

Parameter		Description
hCam	[in]	Camera-Handle of target camera.

[Return value]

Returns result status. Refer to 5.8 Status code.

[Remarks]

This function will return error status if AcquisitionCommand resister (or node) is not implemented in the camera.

Do not execute this function except when streaming is running in single-frame mode or multi-frame mode.

Including TeliCamAPI.h is required.

5.5.8.12. GetCamHighFramerateMode

Gets the HighFramerateMode value of the camera.

Some color cameras have HighFramerateMode. You can improve the frame rate by using HighFramerateMode.

[Syntax]

```
CAM_API_STATUS  GetCamHighFramerateMode (
  CAM_HANDLE        hCam,
  bool8_t           *pbValue
);
```

[Parameters]

Parameter		Description
hCam	[in]	Camera-Handle of target camera.
pbValue	[out]	A pointer to a variable that receives the value of HighFramerateMode. If false, mode is OFF. If true, mode is ON.

[Return value]

Returns result status. Refer to 5.8 Status code.

[Remarks]

This function will return error status if HighFramerateMode register (or node) is not implemented in the camera.

For details about HighFramerateMode features, refer to instruction manual of the camera.

Including TeliCamAPI.h is required.

5.5.8.13. SetCamHighFramerateMode

Sets the HighFramerateMode value of the camera.

Some color cameras have HighFramerateMode. You can improve the frame rate by using HighFramerateMode.

[Syntax]

```
CAM_API_STATUS  SetCamHighFramerateMode (
  CAM_HANDLE        hCam,
  bool8_t           bValue
);
```

[Parameters]

Parameter		Description
hCam	[in]	Camera-Handle of target camera.
bValue	[in]	The value of HighFramerateMode. If false, mode is OFF. If true, mode is ON.

[Return value]

Returns result status. Refer to 5.8 Status code.

[Remarks]

This function will return error status if HighFramerateMode register (or node) is not implemented in the camera.

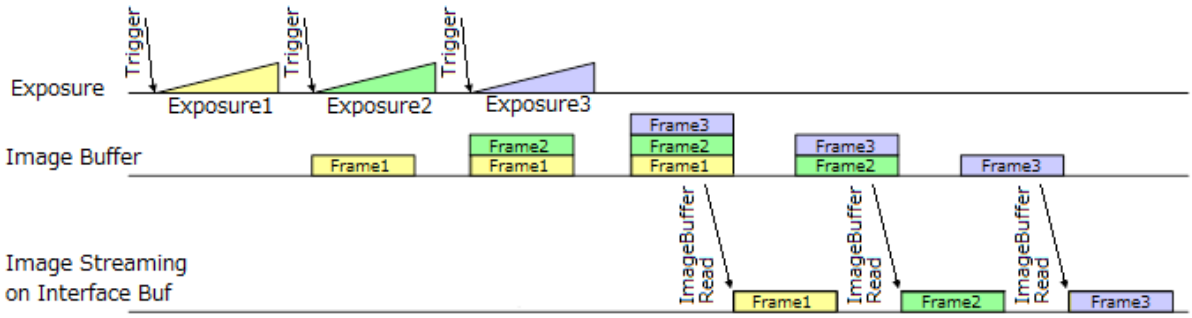
For details about HighFramerateMode features, refer to instruction manual of the camera.

Including TeliCamAPI.h is required.

5.5.9. ImageBuffer

This feature group performs control of image buffer function of the camera.

In ImageBuffer mode, Camera stores images temporarily in image buffer, and read them out in arbitrary timing.
This function is typically used in Random Trigger Shutter mode.



For details about ImageBuffer features, refer to instruction manual of the camera.

5.5.9.1. GetCamImageBufferMode

Gets the image buffer mode type of the camera.

[Syntax]

```
CAM_API_STATUS  GetCamImageBufferMode (
  CAM_HANDLE                  hCam,
  CAM_IMAGE_BUFFER_MODE_TYPE  *peMode
);
```

[Parameters]

Parameter		Description
hCam	[in]	Camera-Handle of target camera.
peMode	[out]	A pointer to a variable that receives the type of image buffer mode.

[*CAM_IMAGE_BUFFER_MODE_TYPE* Enumeration]

Member	Description
CAM_IMAGE_BUFFER_MODE_OFF	Image buffer mode is disabled.
CAM_IMAGE_BUFFER_MODE_ON	Image buffer mode is enabled.

[Return value]

Returns result status. Refer to 5.8 Status code.

[Remarks]

This function will return error status if ImageBufferMode register (or node) is not implemented in the camera.

Refer to TeliCamAPI.h about CAM_IMAGE_BUFFER_MODE_TYPE.

Including TeliCamAPI.h is required.

5.5.9.2. SetCamImageBufferMode

Sets the image buffer mode type of the camera.

[Syntax]

```
CAM_API_STATUS  SetCamImageBufferMode (
  CAM_HANDLE                  hCam,
  CAM_IMAGE_BUFFER_MODE_TYPE  eMode
);
```

[Parameters]

Parameter		Description
hCam	[in]	Camera-Handle of target camera.
eMode	[in]	The type of image buffer mode.

[Return value]

Returns result status. Refer to 5.8 Status code.

[Remarks]

This function will return error status if ImageBufferMode register (or node) is not implemented in the camera.

Settings cannot be changed during image stream data output.

Refer to TeliCamAPI.h about `CAM_IMAGE_BUFFER_MODE_TYPE`.

Including TeliCamAPI.h is required.

5.5.9.3. GetCamImageBufferFrameCount

Gets the number of images stored in the image buffer of the camera.

[Syntax]

```
CAM_API_STATUS  GetCamImageBufferFrameCount (
  CAM_HANDLE      hCam,
  uint32_t        *puiCount
);
```

[Parameters]

Parameter		Description
hCam	[in]	Camera-Handle of target camera.
puiCount	[out]	A pointer to a variable that receives the number of images stored in the image buffer of the camera.

[Return value]

Returns result status. Refer to 5.8 Status code.

[Remarks]

This function will return error status if ImageBufferMode register (or node) is not implemented in the camera.

Including TeliCamAPI.h is required.

5.5.9.4. ExecuteCamImageBufferRead

Reads images from the image buffer of the camera.

[Syntax]

```
CAM_API_STATUS  ExecuteCamImageBufferRead (
  CAM_HANDLE        hCam
);
```

[Parameters]

Parameter		Description
hCam	[in]	Camera-Handle of target camera.

[Return value]

Returns result status. Refer to 5.8 Status code.

[Remarks]

This function will return error status if ImageBufferMode register (or node) is not implemented in the camera.

" ImageBufferRead" is set in the AcquisitionMode register of the camera.

The camera will send specified frame count images for a single `ExecuteCamImageBufferRead`() call. User application can specify frame count by `SetCamAcquisitionFrameCount()`.

User application can get images using 5.3 Camera streaming functions.

For details about ImageBuffer features, refer to instruction manual of the camera.

Including TeliCamAPI.h is required.

5.5.10.TriggerControl

This feature group performs control of trigger function of the camera.

TriggerControl features are related to image acquisition using trigger.
This camera series provides two kinds of exposure synchronization.

1. Normal Shutter mode : Free run operation (internal synchronization)
2. Random Trigger Shutter mode : Synchronized with external trigger input

For details about TriggerControl features, refer to instruction manual of the camera.

5.5.10.1. GetCamTriggerMode

Gets the trigger mode value of the camera.

[Syntax]

```
CAM_API_STATUS  GetCamTriggerMode (
  CAM_HANDLE        hCam,
  bool8_t           *pbValue
);
```

[Parameters]

Parameter		Description
hCam	[in]	Camera-Handle of target camera.
pbValue	[out]	A pointer to a variable that receives the value of trigger mode. If false, trigger mode is OFF. If true, trigger mode is ON.

[Return value]

Returns result status. Refer to 5.8 Status code.

[Remarks]

This function will return error status if TriggerMode register (or node) is not implemented in the camera.

If trigger mode is OFF (Normal Shutter mode), Free-Run mode will be used as synchronization mode, and exposure time will be controlled by ExposureTime register. Value of TriggerSequence and TriggerSource registers will be ignored in this mode.

If trigger mode is ON (Random Trigger Shutter mode), Hardware or Software Trigger mode is used as synchronization mode, and exposure time is controlled in various modes controlled by TriggerSequence and / or some other registers.

For details about TriggerControl features, refer to instruction manual of the camera.

Including TeliCamAPI.h is required.

5.5.10.2. SetCamTriggerMode

Sets the trigger mode value of the camera.

[Syntax]

```
CAM_API_STATUS  SetCamTriggerMode (
  CAM_HANDLE        hCam,
  bool8_t           bValue
);
```

[Parameters]

Parameter		Description
hCam	[in]	Camera-Handle of target camera.
bValue	[in]	The value of trigger mode. If false, trigger mode is OFF. If true, trigger mode is ON.

[Return value]

Returns result status. Refer to 5.8 Status code.

[Remarks]

This function will return error status if TriggerMode register (or node) is not implemented in the camera.

Whether it can be set during image stream data output varies depending on the camera.
For details about TriggerControl features, refer to instruction manual of the camera.

Including TeliCamAPI.h is required.

5.5.10.3. GetCamTriggerSequence

Gets the trigger sequence of random trigger shutter.

[Syntax]

```
CAM_API_STATUS  GetCamTriggerSequence (
  CAM_HANDLE                  hCam,
  CAM_TRIGGER_SEQUENCE_TYPE   *peTriggerSequence
);
```

[Parameters]

Parameter		Description
hCam	[in]	Camera-Handle of target camera.
peTriggerSequence	[out]	A pointer to a variable that receives the type of trigger sequence.

[*CAM_TRIGGER_SEQUENCE_TYPE* Enumeration]

Member	Description
CAM_TRIGGER_SEQUENCE0	Edge mode. The exposure time is determined by Exposure Time setting. • The camera that complie with IIDC2 : TriggerSequence register = TriggerSequence0 • The camera that does not comply with IIDC2 : ExposureMode register = Timed TriggerSelector register = FrameStart
CAM_TRIGGER_SEQUENCE1	Level mode. The exposure time is determined by the pulse width of the trigger signal. • The camera that complie with IIDC2: TriggerSequence register = TriggerSequence1 • The camera that does not comply with IIDC2 : ExposureMode register = TriggerWidth TriggerSelector register = FrameStart
CAM_TRIGGER_SEQUENCE6	Bulk (or FrameBurst) mode. Camera exposes and transfers multiple frames by a single trigger. • The camera that complie with IIDC2: TriggerSequence register = TriggerSequence6 • The camera that does not comply with IIDC2 : ExposureMode register = Timed TriggerSelector register = FrameBurstStart

[Return value]

Returns result status. Refer to 5.8 Status code.

[Remarks]

This function will return error status if TriggerSequence register (or node), or ExposureMode and TriggerSelector registers (or nodes) are not implemented in the camera.

The registers to be accessed are different depending on the camera.
For details about TriggerControl features, refer to instruction manual of the camera.

Refer to TeliCamAPI.h about `CAM_TRIGGER_SEQUENCE_TYPE`.
Including TeliCamAPI.h is required.

5.5.10.4. SetCamTriggerSequence

Sets the trigger sequence of random trigger shutter.

[Syntax]

```
CAM_API_STATUS  SetCamTriggerSequence (
  CAM_HANDLE                  hCam,
  CAM_TRIGGER_SEQUENCE_TYPE   eTriggerSequence
);
```

[Parameters]

Parameter		Description
hCam	[in]	Camera-Handle of target camera.
eTriggerSequence	[in]	The type of trigger sequence.

[Return value]

Returns result status. Refer to 5.8 Status code.

[Remarks]

This function will return error status if TriggerSequence register (or node), or ExposureMode and TriggerSelector registers (or nodes) are not implemented in the camera.

The registers to be accessed are different depending on the camera.
For details about TriggerControl features, refer to instruction manual of the camera.

Including TeliCamAPI.h is required.

5.5.10.5. GetCamTriggerSource

Gets the trigger source of random trigger shutter.

[Syntax]

```
CAM_API_STATUS  GetCamTriggerSource (
  CAM_HANDLE                hCam,
  CAM_TRIGGER_SOURCE_TYPE   *peTriggerSource
);
```

[Parameters]

Parameter		Description
hCam	[in]	Camera-Handle of target camera.
peTriggerSource	[out]	A pointer to a variable that receives the type of trigger source.

[*CAM_TRIGGER_SOURCE_TYPE* Enumeration]

Member	Description
CAM_TRIGGER_LINE0	Hardware Trigger Line0
CAM_TRIGGER_LINE1	Hardware Trigger Line1
CAM_TRIGGER_LINE2	Hardware Trigger Line2
CAM_TRIGGER_SOFTWARE	Software Trigger

[Return value]

Returns result status. Refer to 5.8 Status code.

[Remarks]

This function will return error status if TriggerSource register (or node) is not implemented in the camera.

Including TeliCamAPI.h is required.

5.5.10.6. SetCamTriggerSource

Sets the trigger source of random trigger shutter.

[Syntax]

```
CAM_API_STATUS  SetCamTriggerSource (
  CAM_HANDLE                hCam,
  CAM_TRIGGER_SOURCE_TYPE   eTriggerSource
);
```

[Parameters]

Parameter		Description
hCam	[in]	Camera-Handle of target camera.
eTriggerSource	[in]	The type of trigger source.

[Return value]

Returns result status. Refer to 5.8 Status code.

[Remarks]

This function will return error status if TriggerSource register (or node) is not implemented in the camera.

Including TeliCamAPI.h is required.

5.5.10.7. GetCamTriggerAdditionalParameterMinMax

Gets the minimum and maximum available TriggerAdditionalParameter (or AcquisitionBurstFrameCount) value of the camera.

TriggerAdditionalParameter (or AcquisitionBurstFrameCount) is the number of frames to exposure in Bulk (or FrameBurst) mode.

[Syntax]

```
CAM_API_STATUS  GetCamTriggerAdditionalParameter (
  CAM_HANDLE        hCam,
  uint32_t          *puiAdditionalParameterMin,
  uint32_t          *puiAdditionalParameterMax
);
```

[Parameters]

Parameter		Description
hCam	[in]	Camera-Handle of target camera.
puiAdditionalParameterMin	[out]	A pointer to a variable that receives the minimum value of TriggerAdditionalParameter(or AcquisitionBurstFrameCount).
puiAdditionalParameterMax	[out]	A pointer to a variable that receives the maximum value of TriggerAdditionalParameter(or AcquisitionBurstFrameCount).

[Return value]

Returns result status. Refer to 5.8 Status code.

[Remarks]

This function will return error status if TriggerAdditionalParameter or AcquisitionBurstFrameCount register (or node) is not implemented in the camera.

The register to be accessed and the feature may be different depending on the camera.
For details about TriggerControl features, refer to instruction manual of the camera.

Including TeliCamAPI.h is required.

5.5.10.8. GetCamTriggerAdditionalParameter

Gets the TriggerAdditionalParameter(or AcquisitionBurstFrameCount) value of the camera.

TriggerAdditionalParameter (or AcquisitionBurstFrameCount) is the number of frames to exposure in Bulk (or FrameBurst) mode.

[Syntax]

```
CAM_API_STATUS  GetCamTriggerAdditionalParameter (
  CAM_HANDLE        hCam,
  uint32_t          *puiAdditionalParameter
);
```

[Parameters]

Parameter		Description
hCam	[in]	Camera-Handle of target camera.
puiAdditionalParameter	[out]	A pointer to a variable that receives the value of TriggerAdditionalParameter(or AcquisitionBurstFrameCount).

[Return value]

Returns result status. Refer to 5.8 Status code.

[Remarks]

This function will return error status if TriggerAdditionalParameter or AcquisitionBurstFrameCount register (or node) is not implemented in the camera.

The register to be accessed is different depending on the camera.
For details about TriggerControl features, refer to instruction manual of the camera.

Including TeliCamAPI.h is required.

5.5.10.9. SetCamTriggerAdditionalParameter

Sets the TriggerAdditionalParameter(or AcquisitionBurstFrameCount) value of the camera.

TriggerAdditionalParameter (or AcquisitionBurstFrameCount) is the number of frames to exposure in Bulk (or FrameBurst) mode.

[Syntax]

```
CAM_API_STATUS  SetCamTriggerAdditionalParameter (
  CAM_HANDLE        hCam,
  uint32_t          uiAdditionalParameter
);
```

[Parameters]

Parameter		Description
hCam	[in]	Camera-Handle of target camera.
uiAdditionalParameter	[in]	The value of TriggerAdditionalParameter (or AcquisitionBurstFrameCount).

[Return value]

Returns result status. Refer to 5.8 Status code.

[Remarks]

This function will return error status if TriggerAdditionalParameter or AcquisitionBurstFrameCount register (or node) is not implemented in the camera.

The register to be accessed is different depending on the camera.
For details about TriggerControl features, refer to instruction manual of the camera.

Including TeliCamAPI.h is required.

5.5.10.10. GetCamTriggerDelayMinMax

Gets the minimum and maximum available TriggerDelay value.

TriggerDelay is the delay to apply after the trigger signal reception before effectively activating it.

[Syntax]

```
CAM_API_STATUS  GetCamTriggerDelayMinMax (
  CAM_HANDLE        hCam,
  float64_t         *pdDelayUsMin,
  float64_t         *pdDelayUsMax
);
```

[Parameters]

Parameter		Description
hCam	[in]	Camera-Handle of target camera.
pdDelayUsMin	[out]	A pointer to a variable that receives the minimum value of TriggerDelay in microseconds.
pdDelayUsMax	[out]	A pointer to a variable that receives the maximum value of TriggerDelay in microseconds.

[Return value]

Returns result status. Refer to 5.8 Status code.

[Remarks]

This function will return error status if TriggerDelay register (or node) is not implemented in the camera.

Including TeliCamAPI.h is required.

5.5.10.11. GetCamTriggerDelay

Gets the TriggerDelay value of the camera.

TriggerDelay is the delay to apply after the trigger signal reception before effectively activating it.

[Syntax]

```
CAM_API_STATUS  GetCamTriggerDelay (
  CAM_HANDLE        hCam,
  float64_t         *pdDelayUs
);
```

[Parameters]

Parameter		Description
hCam	[in]	Camera-Handle of target camera.
pdDelayUs	[out]	A pointer to a variable that receives the value of TriggerDelay in microseconds.

[Return value]

Returns result status. Refer to 5.8 Status code.

[Remarks]

This function will return error status if TriggerDelay register (or node) is not implemented in the camera.

Including TeliCamAPI.h is required.

5.5.10.12. SetCamTriggerDelay

Sets the TriggerDelay value of the camera.

TriggerDelay is the delay to apply after the trigger signal reception before effectively activating it.

[Syntax]

```
CAM_API_STATUS  SetCamTriggerDelay (
  CAM_HANDLE        hCam,
  float64_t         dDelayUs
);
```

[Parameters]

Parameter		Description
hCam	[in]	Camera-Handle of target camera.
dDelayUs	[in]	The value of TriggerDelay in microseconds.

[Return value]

Returns result status. Refer to 5.8 Status code.

[Remarks]

This function will return error status if TriggerDelay register (or node) is not implemented in the camera.

Including TeliCamAPI.h is required.

5.5.10.13. ExecuteCamSoftwareTrigger

Sends a software trigger command to the camera.

[Syntax]

```
CAM_API_STATUS  ExecuteCamSoftwareTrigger (
  CAM_HANDLE        hCam
);
```

[Parameters]

Parameter		Description
hCam	[in]	Camera-Handle of target camera.

[Return value]

Returns result status. Refer to 5.8 Status code.

[Remarks]

This function will return error status if SoftwareTrigger register (or node) is not implemented in the camera.

If this function is called when the camera is not ready for accepting software trigger command, the camera may return a success status without doing anything.

Including TeliCamAPI.h is required.

5.5.10.14. GetCamTriggerActivation

Gets the trigger activation of hardware trigger.

TriggerActivation is the activation mode of the trigger.

[Syntax]

```
CAM_API_STATUS  GetCamTriggerActivation (
  CAM_HANDLE                  hCam,
  CAM_TRIGGER_ACTIVATION_TYPE  *peTriggerActivation
);
```

[Parameters]

Parameter		Description
hCam	[in]	Camera-Handle of target camera.
peTriggerActivation	[out]	A pointer to a variable that receives the type of trigger activation.

[*CAM_TRIGGER_ACTIVATION_TYPE* Enumeration]

Member	Description
CAM_TRIGGER_FALLING_EDGE	Falling Edge mode
CAM_TRIGGER_RISING_EDGE	Rising Edge mode

[Return value]

Returns result status. Refer to 5.8 Status code.

[Remarks]

This function will return error status if TriggerActivation, or LineInverterAll register (or node) is not implemented in the camera.

The value to be acquired is the setting value for the currently trigger source of random trigger shutter setting (TriggerSource). Before calling this function, set the trigger source of random trigger shutter by calling SetCamTriggerSource() function.

If the camera has LineModeAll register, this function reads the value of the register and returns the value of the target line(trigger source).

For details about TriggerControl features, refer to instruction manual of the camera.

Including TeliCamAPI.h is required.

5.5.10.15. SetCamTriggerActivation

Sets the trigger activation of hardware trigger.

TriggerActivation is the activation mode of the trigger.

[Syntax]

```
CAM_API_STATUS  SetCamTriggerActivation (
  CAM_HANDLE                  hCam,
  CAM_TRIGGER_ACTIVATION_TYPE  eTriggerActivation
);
```

[Parameters]

Parameter		Description
hCam	[in]	Camera-Handle of target camera.
eTriggerActivation	[in]	The type of trigger activation.

[Return value]

Returns result status. Refer to 5.8 Status code.

[Remarks]

This function will return error status if TriggerActivation, or LineInverterAll register (or node) is not implemented in the camera.

The value to be set is the setting value for the currently trigger source of random trigger shutter setting (TriggerSource). Before calling this function, set the trigger source of random trigger shutter by calling SetCamTriggerSource() function.

If the camera has LineModeAll register, this function reads the value of the register, changes only the value of the target line(trigger source), and sets the value in the register.
When SetCamLineInverterAll() or SetCamLineInverter() is called, the value set by this function may be changed.

For details about TriggerControl features, refer to instruction manual of the camera.

Including TeliCamAPI.h is required.

5.5.11.ExposureTime

This method group performs control of exposure.

For details about ExposureTime features, refer to instruction manual of the camera.

5.5.11.1. GetCamExposureTimeControl

Gets the exposure time control mode of the camera.

[Syntax]

```
CAM_API_STATUS  GetCamExposureTimeControl (
  CAM_HANDLE                    hCam,
  CAM_EXPOSURE_TIME_CONTROL_TYPE  *peExpControl
);
```

[Parameters]

Parameter		Description
hCam	[in]	Camera-Handle of target camera.
peExpControl	[out]	A pointer to a variable that receives the type of exposure time control mode.

[*CAM_EXPOSURE_TIME_CONTROL_TYPE* Enumeration]

Member	Description
CAM_EXPOSURE_TIME_CONTROL_AUTO	The camera will control exposure time automatically. • The camera that complie with IIDC2 : ExposureTimeControl register = Auto • The camera that does not comply with IIDC2 : ExposureAuto register = Continuous
CAM_EXPOSURE_TIME_CONTROL_MANUAL	The camera will determine exposure time using ExposureTime register value. • The camera that complie with IIDC2 : ExposureTimeControl register = Manula • The camera that does not comply with IIDC2 : ExposureAuto register = Off
CAM_EXPOSURE_TIME_CONTROL_NO_SPECIFY	The camera will determine exposure time using registers other than ExposureTime register. • The camera that complie with IIDC2 : ExposureTimeControl register = NoSpecify • The camera that does not comply with IIDC2 : Not Available

[Return value]

Returns result status. Refer to 5.8 Status code.

[Remarks]

This function will return error status if ExposureTimeControl or ExposureAuto register (or node) is not implemented in the camera.

The feature may be different depending on the camera..
For details about ExposureTime features, refer to instruction manual of the camera.

Including TeliCamAPI.h is required.

5.5.11.2. SetCamExposureTimeControl

Sets the exposure time control mode of the camera.

[Syntax]

```
CAM_API_STATUS  SetCamExposureTimeControl (
  CAM_HANDLE                    hCam,
  CAM_EXPOSURE_TIME_CONTROL_TYPE   eExpControl
);
```

[Parameters]

Parameter		Description
hCam	[in]	Camera-Handle of target camera.
eExpControl	[in]	The type of exposure time control mode.

[Return value]

Returns result status. Refer to 5.8 Status code.

[Remarks]

This method will return error status if TriggerAdditionalParameter or AcquisitionBurstFrameCount register (or node) is not implemented in the camera.

The feature and writable lines may be different depending on the camera.
For details about TriggerControl features, refer to instruction manual of the camera.

5.5.11.3. GetCamExposureTimeMinMax

Gets the minimum and maximum available exposure time of the camera when ExposureTimeControl is set to CAM_EXPOSURE_TIME_CONTROL_MANUAL.

[Syntax]

```
CAM_API_STATUS  GetCamExposureTime (
  CAM_HANDLE        hCam,
  float64_t         *pdExpTimeUsMin,
  float64_t         *pdExpTimeUsMax,
);
```

[Parameters]

Parameter		Description
hCam	[in]	Camera-Handle of target camera.
pdExpTimeUsMin	[out]	A pointer to a variable that receives the minimum value of exposure time in microseconds.
pdExpTimeUsMax	[out]	A pointer to a variable that receives the maximum value of exposure time in microseconds.

[Return value]

Returns result status. Refer to 5.8 Status code.

[Remarks]

This function will return error status if ExposureTime register (or node) is not implemented in the camera.

Including TeliCamAPI.h is required.

5.5.11.4. GetCamExposureTime

Gets the exposure time of the camera when ExposureTimeControl is set to CAM_EXPOSURE_TIME_CONTROL_MANUAL.

[Syntax]

```
CAM_API_STATUS  GetCamExposureTime (
  CAM_HANDLE        hCam,
  float64_t         *pdExpTimeUs
);
```

[Parameters]

Parameter		Description
hCam	[in]	Camera-Handle of target camera.
pdExpTimeUs	[out]	A pointer to a variable that receives the value of exposure time in microseconds.

[Return value]

Returns result status. Refer to 5.8 Status code.

[Remarks]

This function will return error status if ExposureTime register (or node) is not implemented in the camera.

Including TeliCamAPI.h is required.

5.5.11.5. SetCamExposureTime

Sets the exposure time of the camera when ExposureTimeControl is set to CAM_EXPOSURE_TIME_CONTROL_MANUAL.

[Syntax]

```
CAM_API_STATUS  SetCamExposureTime (
  CAM_HANDLE        hCam,
  float64_t         dExpTimeUs
);
```

[Parameters]

Parameter		Description
hCam	[in]	Camera-Handle of target camera.
dExpTimeUs	[in]	The value of exposure time in microseconds.

[Return value]

Returns result status. Refer to 5.8 Status code.

[Remarks]

This function will return error status if ExposureTime register (or node) is not implemented in the camera.

Including TeliCamAPI.h is required.

5.5.11.6. GetCamShortExposureMode

Gets the ShortExposureMode value of the camera.

Some color cameras have ShortExposureMode which can set high-speed exposure time at the time of MANUAL setting.

[Syntax]

```
CAM_API_STATUS  GetCamShortExposureMode (
  CAM_HANDLE        hCam,
  bool8_t           *pbValue
);
```

[Parameters]

Parameter		Description
hCam	[in]	Camera-Handle of target camera.
pbValue	[out]	A pointer to a variable that receives the value of ShortExposureMode. If false, mode is OFF. If true, mode is ON.

[Return value]

Returns result status. Refer to 5.8 Status code.

[Remarks]

This function will return error status if ShortExposureMode register (or node) is not implemented in the camera.

For details about ShortExposureMode features, refer to instruction manual of the camera.

Including TeliCamAPI.h is required.

5.5.11.7. SetCamHighFramerateMode

Sets the ShortExposureMode value of the camera.

Some color cameras have ShortExposureMode which can set high-speed exposure time at the time of MANUAL setting.

[Syntax]

```
CAM_API_STATUS  SetCamShortExposureMode (
  CAM_HANDLE        hCam,
  bool8_t           bValue
);
```

[Parameters]

Parameter		Description
hCam	[in]	Camera-Handle of target camera.
bValue	[in]	The value of ShortExposureMode. If false, mode is OFF. If true, mode is ON.

[Return value]

Returns result status. Refer to 5.8 Status code.

[Remarks]

This function will return error status if ShortExposureMode register (or node) is not implemented in the camera.

For details about ShortExposureMode features, refer to instruction manual of the camera.

Including TeliCamAPI.h is required.

5.5.12.DigitalIoControl

This feature group performs control of digital I/O ports.

For details about DigitalIoContol features, refer to instruction manual of the camera.

5.5.12.1. GetCamLineModeAll

Gets the input / output value for all digital I/O lines of the camera.

[Syntax]

```
CAM_API_STATUS  GetCamLineModeAll (
  CAM_HANDLE        hCam,
  uint32_t          *puiValue
);
```

[Parameters]

Parameter		Description
hCam	[in]	Camera-Handle of target camera.
puiValue	[out]	A pointer to a variable that receives the value of line mode for all digital I/O lines. Each bit corresponds to each I/O line. (Line0 : Bit0 , Line1 : Bit1 , Line2 : Bit2 , .) If bit value is 0, the line is input line, If bit value is 1, the line is output line. For example, If `puiValue is 0x06,` Line0 : input, Line1 : output, Line2 : output , ...)

[Return value]

Returns result status. Refer to 5.8 Status code.

[Remarks]

This function will return error status if LineModeAll register (or node), or LineSelector and LineMode registers (or nodes) are not implemented in the camera.

If the camera does not have LineModeAll register, this function reads input / output setting values of each line, combines them, and outputs it.

For details about DigitalIoContol features, refer to instruction manual of the camera.

Including TeliCamAPI.h is required.

5.5.12.2. SetCamLineModeAll

Sets the input / output value for all digital I/O lines of the camera.

[Syntax]

```
CAM_API_STATUS  SetCamLineModeAll (
  CAM_HANDLE        hCam,
  uint32_t          uiValue
);
```

[Parameters]

Parameter		Description
hCam	[in]	Camera-Handle of target camera.
uiValue	[in]	The value of line mode for all digital I/O lines. Each bit corresponds to each I/O line. (Line0 : Bit0 , Line1 : Bit1 , Line2 : Bit2 , .) If bit value is 0, the line is input. If bit value is 1, the line is output. For example, If `uiValue is 0x06`, Line0 : input, Line1 : output, Line2 : output , ...)

[Return value]

Returns result status. Refer to 5.8 Status code.

[Remarks]

This function will return error status if LineModeAll register (or node), or LineSelector and LineMode registers (or nodes) are not implemented in the camera.

If the camera does not have LineModeAll register, this function sets the values of line mode for each line.

The feature and writable lines may be different depending on the camera.
For details about DigitalIoContol features, refer to instruction manual of the camera.

Including TeliCamAPI.h is required.

5.5.12.3. GetCamLineMode

Gets the input / output value for each digital I/O line of the camera.

[Syntax]

```
CAM_API_STATUS  GetCamLineMode (
  CAM_HANDLE                hCam,
  CAM_LINE_SELECTOR_TYPE    eLineSelector,
  CAM_LINE_MODE_TYPE        *peLineMode
);
```

[Parameters]

Parameter		Description
hCam	[in]	Camera-Handle of target camera.
eLineSelector	[in]	Target line.
peLineMode	[out]	A pointer to a variable that receives the type of line mode.

[*CAM_LINE_SELECTOR_TYPE* Enumeration]

メンバ	内容
CAM_LINE_SELECTOR_LINE0	Line0
CAM_LINE_SELECTOR_LINE1	Line1
CAM_LINE_SELECTOR_LINE2	Line2

[*CAM_LINE_MODE_TYPE* Enumeration]

メンバ	内容
CAM_LINE_MODE_INPUT	Input
CAM_LINE_MODE_OUTPUT	Output

[Return value]

Returns result status. Refer to 5.8 Status code.

[Remarks]

This function will return error status if LineModeAll register (or node), or LineSelector and LineMode registers (or nodes) are not implemented in the camera.

If the camera has LineModeAll register, this function reads the value of the register, and gets the value of the target line.

For details about DigitalIoContol features, refer to instruction manual of the camera.

Including TeliCamAPI.h is required.

5.5.12.4. SetCamLineMode

Sets the input / output value for each digital I/O line of the camera.

[Syntax]

```
CAM_API_STATUS  SetCamLineMode (
  CAM_HANDLE                hCam,
  CAM_LINE_SELECTOR_TYPE    eLineSelector,
  CAM_LINE_MODE_TYPE        eLineMode
);
```

[Parameters]

Parameter		Description
hCam	[in]	Camera-Handle of target camera.
eLineSelector	[in]	Target line.
eLineMode	[in]	The type of line mode.

[Return value]

Returns result status. Refer to 5.8 Status code.

[Remarks]

This function will return error status if LineModeAll register (or node), or LineSelector and LineMode registers (or nodes) are not implemented in the camera.

If the camera has LineModeAll register, this function reads the value of the register, changes only the value of the target line, and sets the value in the register.

The feature and writable lines may be different depending on the camera.
For details about DigitalIoContol features, refer to instruction manual of the camera.

Including TeliCamAPI.h is required.

5.5.12.5. GetCamLineInverterAll

Gets the polarity value for all digital I/O lines of the camera.

[Syntax]

```
CAM_API_STATUS  GetCamLineInverterAll (
  CAM_HANDLE        hCam,
  uint32_t          *puiValue
);
```

[Parameters]

Parameter		Description
hCam	[in]	Camera-Handle of target camera.
puiValue	[out]	A pointer to a variable that receives the value of polarities for all digital I/O lines. Each bit corresponds to each I/O line. (Line0 : Bit0 , Line1 : Bit1 , Line2 : Bit2 , .) If bit value is 0, the line is not inverted. If bit value is 1, the line is inverted. For example, If puiValue is 0x06, Line0 : not inverted, Line1 : inverted, Line2 : inverted , …)

[Return value]

Returns result status. Refer to 5.8 Status code.

[Remarks]

This function will return error status if LineInverterAll register (or node), or LineSelector and LineInverter registers (or nodes) are not implemented in the camera.

If the camera does not have LineInverterAll register, this function reads the polarity values of each line, combines them, and outputs it.

For details about DigitalIoContol features, refer to instruction manual of the camera.

Including TeliCamAPI.h is required.

5.5.12.6. SetCamLineInverterAll

Sets the polarity value for all digital I/O lines of the camera.

[Syntax]

```
CAM_API_STATUS  SetCamLineInverterAll (
  CAM_HANDLE        hCam,
  uint32_t          uiValue
);
```

[Parameters]

Parameter		Description
hCam	[in]	Camera-Handle of target camera.
uiValue	[in]	The value of polarities for all digital I/O lines. Each bit corresponds to each I/O line. (Line0 : Bit0 , Line1 : Bit1 , Line2 : Bit2 , .) If bit value is 0, the line is not inverted, If bit value is 1, the line is inverted. For example, If `uiValue is 0x02`, Line0 : not inverted, Line1 : inverted, Line2 : not inverted, …)

[Return value]

Returns result status. Refer to 5.8 Status code.

[Remarks]

This function will return error status if LineInverterAll register (or node), or LineSelector and LineInverter registers (or nodes) are not implemented in the camera.

If the camera does not have LineInverterAll register, this function sets the values of polarity for each line.

The feature and writable lines may be different depending on the camera.
If 1 (Inverted) is set in the bit data of the Line that can not be written,
CAM_API_STS_INVALID_PARAMETER may be returned.
For details about DigitalIoContol features, refer to instruction manual of the camera.

Including TeliCamAPI.h is required.

5.5.12.7. GetCamLineInverter

Gets the polarity value for each digital I/O line of the camera.

[Syntax]

```
CAM_API_STATUS  GetCamLineInverter (
  CAM_HANDLE              hCam,
  CAM_LINE_SELECTOR_TYPE  eLineSelector,
  bool8_t                 *pbInvert
);
```

[Parameters]

Parameter		Description
hCam	[in]	Camera-Handle of target camera.
eLineSelector	[in]	Target line.
pbInvert	[out]	A pointer to a variable that receives the value of polarity. If the value is false, the polarity is not inverted. If the value is true, the polarity is inverted.

[Return value]

Returns result status. Refer to 5.8 Status code.

[Remarks]

This function will return error status if LineInverterAll register (or node), or LineSelector and LineInverter registers (or nodes) are not implemented in the camera.

If the camera has LineInverterAll register, this function reads the value of the register, and gets the value of the target line.

For details about DigitalIoContol features, refer to instruction manual of the camera.

Including TeliCamAPI.h is required.

5.5.12.8. SetCamLineInverter

Sets the polarity value for each digital I/O line of the camera.

[Syntax]

```
CAM_API_STATUS  SetCamLineInverter (
  CAM_HANDLE              hCam,
  CAM_LINE_SELECTOR_TYPE  eLineSelector,
  bool8_t                 pbInvert
);
```

[Parameters]

Parameter		Description
hCam	[in]	Camera-Handle of target camera.
eLineSelector	[in]	Target line.
pbInvert	[in]	The value of polarity. If the value is false, the polarity is not inverted. If the value is true, the polarity is inverted.

[Return value]

Returns result status. Refer to 5.8 Status code.

[Remarks]

This function will return error status if LineInverterAll register (or node), or LineSelector and LineInverter registers (or nodes) are not implemented in the camera.

If the camera has LineInverterAll register, this function reads the value of the register, changes only the value of the target line, and sets the value in the register.

The feature and writable lines may be different depending on the camera.
For details about DigitalIoContol features, refer to instruction manual of the camera.

Including TeliCamAPI.h is required.

5.5.12.9. GetCamLineStatusAll

Gets the line status for all digital I/O lines of the camera.

[Syntax]

```
CAM_API_STATUS  GetCamLineStatusAll (
  CAM_HANDLE        hCam,
  uint32_t          *puiValue
);
```

[Parameters]

Parameter		Description
hCam	[in]	Camera-Handle of target camera.
puiValue	[out]	A pointer to a variable that receives the value of line status for all digital I/O lines. Each bit corresponds to each I/O line. (Line0 : Bit0 , Line1 : Bit1 , Line2 : Bit2 , .) If bit value is 0, the line is Low. If bit value is 1, the line is High. For example, If `puiValue is 0x02,` Line0 : Low, Line1 : High, Line2 : Low, …)

[Return value]

Returns result status. Refer to 5.8 Status code.

[Remarks]

This function will return error status if LineStatusAll register (or node), or LineSelector and LineStatus registers (or nodes) are not implemented in the camera.

Including TeliCamAPI.h is required.

5.5.12.10. GetCamLineStatus

Gets the line status for each digital I/O line of the camera.

[Syntax]

```
CAM_API_STATUS  GetCamLineStatus (
  CAM_HANDLE                hCam,
  CAM_LINE_SELECTOR_TYPE    eLineSelector,
  bool8_t                   *pbValue
);
```

[Parameters]

Parameter		Description
hCam	[in]	Camera-Handle of target camera.
eLineSelector	[in]	Target line.
pbValue	[out]	A pointer to a variable that receives the value of line status. If the value is false, the line status is Low. If the value is true, the line status is High.

[Return value]

Returns result status. Refer to 5.8 Status code.

[Remarks]

This function will return error status if LineStatusAll register (or node), or LineSelector and LineStatus registers (or nodes) are not implemented in the camera.

Including TeliCamAPI.h is required.

5.5.12.11. GetCamUserOutputValueAll

Gets the user output value for all user output lines in the camera.

[Syntax]

```
CAM_API_STATUS  GetCamUserOutputValueAll (
  CAM_HANDLE        hCam,
  uint32_t          *puiValue
);
```

[Parameters]

Parameter		Description
hCam	[in]	Camera-Handle of target camera.
puiValue	[out]	A pointer to a variable that receives the user output value for all user output lines. Each bit corresponds to each I/O line. (Line0 : Bit0 , Line1 : Bit1 , Line2 : Bit2 , .) If bit value is 0, the line is Low. If bit value is 1, the line is High.

[Return value]

Returns result status. Refer to 5.8 Status code.

[Remarks]

This function will return error status if UserOutputValueAll register (or node), or LineSelector and UserOutputValue registers (or nodes) are not implemented in the camera.

Including TeliCamAPI.h is required.

5.5.12.12. SetCamUserOutputValueAll

Sets the user output value for all user output lines in the camera.

[Syntax]

```
CAM_API_STATUS  SetCamUserOutputValueAll (
  CAM_HANDLE        hCam,
  uint32_t          uiValue
);
```

[Parameters]

Parameter		Description
hCam	[in]	Camera-Handle of target camera.
uiValue	[in]	The user output value for all user output lines. Each bit corresponds to each I/O line. (Line0 : Bit0 , Line1 : Bit1 , Line2 : Bit2 , .) If bit value is 0, the line is Low. If bit value is 1, the line is High.

[Return value]

Returns result status. Refer to 5.8 Status code.

[Remarks]

This function will return error status if UserOutputValueAll register (or node), or LineSelector and UserOutputValue registers (or nodes) are not implemented in the camera.

The user output line is the line whose line source is set to UserOutput (`CAM_LINE_SOURCE_USER_OUTPUT`).
User application can use SetCamLineSource() and set the line source to UserOutput.

The bits of the line whose line source is not set to UserOutput are ignored.

Including TeliCamAPI.h is required.

5.5.12.13. GetCamUserOutputValue

Gets the user output value for each user output line of the camera.

[Syntax]

```
CAM_API_STATUS  GetCamUserOutputValue (
  CAM_HANDLE               hCam,
  CAM_LINE_SELECTOR_TYPE   eLineSelector,
  bool8_t                  *pbValue
);
```

[Parameters]

Parameter		Description
hCam	[in]	Camera-Handle of target camera.
eLineSelector	[in]	Target line.
puiValue	[out]	A pointer to a variable that receives the user output value. If the value is false, status is Low. If the value is true, status is High.

[Return value]

Returns result status. Refer to 5.8 Status code.

[Remarks]

This function will return error status if UserOutputValueAll register (or node), or LineSelector and UserOutputValue registers (or nodes) are not implemented in the camera.

Including TeliCamAPI.h is required.

5.5.12.14. SetCamUserOutputValue

Sets the user output value for each user output line of the camera.

[Syntax]

```
CAM_API_STATUS  SetCamUserOutputValue (
  CAM_HANDLE                hCam,
  CAM_LINE_SELECTOR_TYPE    eLineSelector,
  bool8_t                   pbValue
);
```

[Parameters]

Parameter		Description
hCam	[in]	Camera-Handle of target camera.
eLineSelector	[in]	Target line.
uiValue	[in]	The user output value. If the value is false, status is Low. If the value is true, status is High.

[Return value]

Returns result status. Refer to 5.8 Status code.

[Remarks]

This function will return error status if UserOutputValueAll register (or node), or LineSelector and UserOutputValue registers (or nodes) are not implemented in the camera.

The user output line is the line whose line source is set to UserOutput (CAM_LINE_SOURCE_USER_OUTPUT).
User application can use SetCamLineSource() and set the line source to UserOutput.

Including TeliCamAPI.h is required.

5.5.12.15. GetCamLineSource

Gets the source of the output signal.

[Syntax]

```
CAM_API_STATUS  GetCamLineSource (
  CAM_HANDLE                hCam,
  CAM_LINE_SELECTOR_TYPE    eLineSelector,
  CAM_LINE_SOURCE_TYPE      *peLineSource
);
```

[Parameters]

Parameter		Description
hCam	[in]	Camera-Handle of target camera.
eLineSelector	[in]	Target line.
peLineSource	[out]	A pointer to a variable that receives the source type of the output signal.

[*CAM_LINE_SOURCE_TYPE* Enumeration]

Member	Description
CAM_LINE_SOURCE_OFF	Off
CAM_LINE_SOURCE_VD	Internal VD sync signal.(Only for GigE Vision Camera)
CAM_LINE_SOURCE_USER_OUTPUT	Outputs the value set in “UserOutputValue”.
CAM_LINE_SOURCE_TIMER0_ACTIVE	This signal can be used as strobe control signal. The delay time and pulse width of this signal are configurable.
CAM_LINE_SOURCE_ACQUISITION_ACTIVE	Indicates AcquisitionStart state of camera.
CAM_LINE_SOURCE_FRAME_TRIGGER_WAIT	Indicates that camera is ready to accept trigger signal. (both hardware and software)
CAM_LINE_SOURCE_FRAME_ACTIVE	Period from exposure start to sensor read-out completion.
CAM_LINE_SOURCE_FRAME_TRANSFER_ACTIVE	Period of transferring image streaming data on interface bus.
CAM_LINE_SOURCE_EXPOSURE_ACTIVE	Period from exposure start to exposure end.

[Return value]

Returns result status. Refer to 5.8 Status code.

[Remarks]

This function will return error status if LineSelector and LineSource registers (or nodes) are not implemented in the camera.

Including TeliCamAPI.h is required.

5.5.12.16. SetCamLineSource

Sets the source of the output signal.

[Syntax]

```
CAM_API_STATUS  SetCamLineSource (
  CAM_HANDLE                hCam,
  CAM_LINE_SELECTOR_TYPE    eLineSelector,
  CAM_LINE_SOURCE_TYPE      eLineSource
);
```

[Parameters]

Parameter		Description
hCam	[in]	Camera-Handle of target camera.
eLineSelector	[in]	Target line.
eLineSource	[in]	The source type of the output signal.

[Return value]

Returns result status. Refer to 5.8 Status code.

[Remarks]

This function will return error status if LineSelector and LineSource registers (or nodes) are not implemented in the camera.

The user application can write to LineSource register and select which internal acquisition or I/O source signal to output on the selected output line.

The source types that can be set may be different depending on the camera.
For details about DigitalIoContol features, refer to instruction manual of the camera.

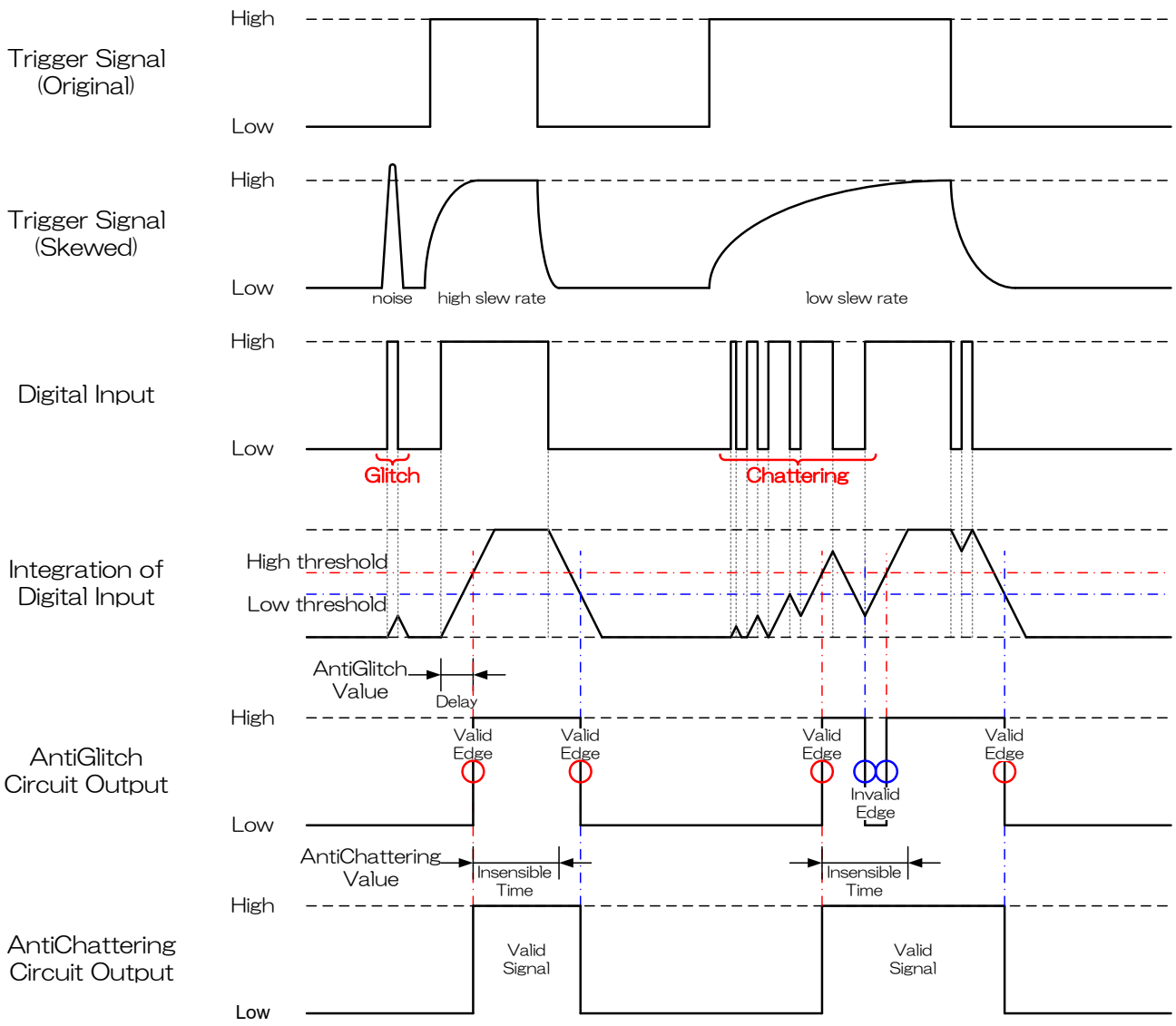
Including TeliCamAPI.h is required.

5.5.13.AntiGlitch / AntiChattering

This feature group performs control of AntiGlitch and AntiChattering function of the camera.

AntiGlitch and AntiChattering functions filter noise and unstable state of the digital input (trigger signal).
AntiGlitch circuit performs the digital integration of the trigger signal.
It is effective to remove impulsive noise.

AntiChattering circuit sets the edge insensible time to avoid trigger malfunction.
It is effective to remove unstable logic state and switch-chattering.



For details about AntiGlitch and AntiChattering features, refer to instruction manual of the camera.

5.5.13.1. GetCamAntiGlitchMinMax

Gets the minimum and maximum integration time of digital input signal in the camera.

[Syntax]

```
CAM_API_STATUS  GetCamAntiGlitchMinMax (
  CAM_HANDLE        hCam,
  float64_t         *pdTimeUsMin,
  float64_t         *pdTimeUsMax,
);
```

[Parameters]

Parameter		Description
hCam	[in]	Camera-Handle of target camera.
pdTimeUsMin	[out]	A pointer to a variable that receives the minimum integration time of digital input signal, in microseconds.
pdTimeUsMax	[out]	A pointer to a variable that receives the maximum integration time of digital input signal, in microseconds.

[Return value]

Returns result status. Refer to 5.8 Status code.

[Remarks]

This function will return error status if AntiGlitch register (or node) is not implemented in the camera.

Please note that the unit is microsecond in this function.

Including TeliCamAPI.h is required.

5.5.13.2. GetCamAntiGlitch

Gets the integration time of digital input signal in the camera.

[Syntax]

```
CAM_API_STATUS  GetCamAntiGlitch (
  CAM_HANDLE        hCam,
  float64_t         *pdTimeUs
);
```

[Parameters]

Parameter		Description
hCam	[in]	Camera-Handle of target camera.
pdTimeUs	[out]	A pointer to a variable that receives the integration time of digital input signal, in microseconds.

[Return value]

Returns result status. Refer to 5.8 Status code.

[Remarks]

This function will return error status if AntiGlitch register (or node) is not implemented in the camera.

Please note that the unit is microsecond in this function.
Including TeliCamAPI.h is required.

5.5.13.3. SetCamAntiGlitch

Sets the integration time of digital input signal in the camera.

[Syntax]

```
CAM_API_STATUS  SetCamAntiGlitch (
  CAM_HANDLE        hCam,
  float64_t         dTimeUs
);
```

[Parameters]

Parameter		Description
hCam	[in]	Camera-Handle of target camera.
dTimeUs	[in]	The integration time of digital input signal, in microseconds.

[Return value]

Returns result status. Refer to 5.8 Status code.

[Remarks]

This function will return error status if AntiGlitch register (or node) is not implemented in the camera.

Please note that the unit is microsecond in this function.
Including TeliCamAPI.h is required.

5.5.13.4. GetCamAntiChatteringMinMax

Gets the minimum and maximum insensible time of digital input signal in the camera.

[Syntax]

```
CAM_API_STATUS  GetCamAntiChatteringMinMax (
  CAM_HANDLE        hCam,
  float64_t         *pdTimeUsMin,
  float64_t         *pdTimeUsMax,
);
```

[Parameters]

Parameter		Description
hCam	[in]	Camera-Handle of target camera.
pdTimeUsMin	[out]	A pointer to a variable that receives the minimum insensible time of digital input signal, in microseconds.
pdTimeUsMax	[out]	A pointer to a variable that receives the maximum insensible time of digital input signal, in microseconds.

[Return value]

Returns result status. Refer to 5.8 Status code.

[Remarks]

This function will return error status if AntiChattering register (or node) is not implemented in the camera.

Please note that the unit is microsecond in this function.

Including TeliCamAPI.h is required.

5.5.13.5. GetCamAntiChattering

Gets the insensible time of digital input signal in the camera.

[Syntax]

```
CAM_API_STATUS  GetCamAntiChattering (
  CAM_HANDLE        hCam,
  float64_t         *pdTimeUs
);
```

[Parameters]

Parameter		Description
hCam	[in]	Camera-Handle of target camera.
pdTimeUs	[out]	A pointer to a variable that receives the insensible time of digital input signal, in microseconds.

[Return value]

Returns result status. Refer to 5.8 Status code.

[Remarks]

This function will return error status if AntiChattering register (or node) is not implemented in the camera.

Please note that the unit is microsecond in this function.
Including TeliCamAPI.h is required.

5.5.13.6. SetCamAntiChattering

Sets the insensible time of digital input signal in the camera.

[Syntax]

```
CAM_API_STATUS  SetCamAntiChattering (
  CAM_HANDLE        hCam,
  float64_t         dTimeUs
);
```

[Parameters]

Parameter		Description
hCam	[in]	Camera-Handle of target camera.
dTimeUs	[in]	The insensible time of digital input signal, in microseconds.

[Return value]

Returns result status. Refer to 5.8 Status code.

[Remarks]

This function will return error status if AntiChattering register (or node) is not implemented in the camera.

Please note that the unit is microsecond in this function.
Including TeliCamAPI.h is required.

5.5.14.TimerConrtol

This feature group performs control of timer.

Timer in the camera is used to output TimerActive signal.

Target timer signal of functions in this section is fixed to Timer0Active because Current BU and BG series has only one timer.

For details about TimerControl features, refer to instruction manual of the camera.

5.5.14.1. GetCamTimerDurationMinMax

Gets the minimum and maximum duration of Timer0Active signal.

[Syntax]

```
CAM_API_STATUS  GetCamTimerDurationMinMax (
  CAM_HANDLE        hCam,
  float64_t         *pdTimerDurationMin,
  float64_t         *pdTimerDurationMax
);
```

[Parameters]

Parameter		Description
hCam	[in]	Camera-Handle of target camera.
pdTimerDurationMin	[out]	A pointer to a variable that receives the minimum duration of Timer0Active signal, in microseconds.
pdTimerDurationMax	[out]	A pointer to a variable that receives the maximum duration of Timer0Active signal, in microseconds.

[Return value]

Returns result status. Refer to 5.8 Status code.

[Remarks]

This function will return error status if TimerDuraiton register (or node) is not implemented in the camera.

Including TeliCamAPI.h is required.

5.5.14.2. GetCamTimerDuration

Gets the duration of Timer0Active signal.

[Syntax]

```
CAM_API_STATUS  GetCamTimerDuration (
  CAM_HANDLE        hCam,
  float64_t         *pdTimerDuration
);
```

[Parameters]

Parameter		Description
hCam	[in]	Camera-Handle of target camera.
pdTimerDuration	[out]	A pointer to a variable that receives the duration of Timer0Active signal, in microseconds.

[Return value]

Returns result status. Refer to 5.8 Status code.

[Remarks]

This function will return error status if TimerDuraiton register (or node) is not implemented in the camera.

Including TeliCamAPI.h is required.

5.5.14.3. SetCamTimerDuration

Sets the duration of Timer0Active signal.

[Syntax]

```
CAM_API_STATUS  SetCamTimerDuration (
  CAM_HANDLE        hCam,
  float64_t         dTimerDuration
);
```

[Parameters]

Parameter		Description
hCam	[in]	Camera-Handle of target camera.
dTimerDuration	[in]	The duration of Timer0Active signal, in microseconds.

[Return value]

Returns result status. Refer to 5.8 Status code.

[Remarks]

This function will return error status if TimerDuraiton register (or node) is not implemented in the camera.

Including TeliCamAPI.h is required.

5.5.14.4. GetCamTimerDelayMinMax

Gets the minimum and maximum delay of Timer0Active signal.

[Syntax]

```
CAM_API_STATUS  GetCamTimerDelayMinMax (
  CAM_HANDLE        hCam,
  float64_t         *pdTimerDelayMin,
  float64_t         *pdTimerDelayMax
);
```

[Parameters]

Parameter		Description
hCam	[in]	Camera-Handle of target camera.
pdTimerDelayMin	[out]	A pointer to a variable that receives the minimum delay of Timer0Active signal, in microseconds.
pdTimerDelayMax	[out]	A pointer to a variable that receives the maximum delay of Timer0Active signal, in microseconds.

[Return value]

Returns result status. Refer to 5.8 Status code.

[Remarks]

This function will return error status if TimerDelay register (or node) is not implemented in the camera.

Including TeliCamAPI.h is required.

5.5.14.5. GetCamTimerDelay

Gets the delay of Timer0Active signal.

[Syntax]

```
CAM_API_STATUS  GetCamTimerDelay (
  CAM_HANDLE        hCam,
  float64_t         *pdTimerDelay
);
```

[Parameters]

Parameter		Description
hCam	[in]	Camera-Handle of target camera.
pdTimerDelay	[out]	A pointer to a variable that receives the delay of Timer0Active signal, in microseconds.

[Return value]

Returns result status. Refer to 5.8 Status code.

[Remarks]

This function will return error status if TimerDelay register (or node) is not implemented in the camera.

Including TeliCamAPI.h is required.

5.5.14.6. SetCamTimerDelay

Sets the delay of Timer0Active signal.

[Syntax]

```
CAM_API_STATUS  SetCamTimerDelay (
  CAM_HANDLE        hCam,
  float64_t         dTimerDelay
);
```

[Parameters]

Parameter		Description
hCam	[in]	Camera-Handle of target camera.
dTimerDelay	[in]	The delay of Timer0Active signal, in microseconds.

[Return value]

Returns result status. Refer to 5.8 Status code.

[Remarks]

This function will return error status if TimerDelay register (or node) is not implemented in the camera.

Including TeliCamAPI.h is required.

5.5.14.7. GetCamTimerTriggerSource

Gets the source of Timer0Active pulse.

[Syntax]

```
CAM_API_STATUS  GetCamTimerTriggerSource (
  CAM_HANDLE                    hCam,
  CAM_TIMER_TRIGGER_SOURCE_TYPE *peTimerTriggerSource
);
```

[Parameters]

Parameter		Description
hCam	[in]	Camera-Handle of target camera.
peTimerTriggerSource	[out]	A pointer to a variable that receives the source type of Timer0Active pulse.

[*CAM_TIMER_TRIGGER_SOURCE_TYPE* Enumeration]

Member	Description
CAM_TIMER_TRIGGER_SOURCE_OFF	DisablesTimer0Active signal.
CAM_TIMER_TRIGGER_SOURCE_LINE0_ACTIVE	Starts when Line0 is active.
CAM_TIMER_TRIGGER_SOURCE_FRAME_TRIGGER	Starts with the reception of the Frame Start Trigger.
CAM_TIMER_TRIGGER_SOURCE_EXPOSURE_START	Starts with the reception of the Exposure Start.
CAM_TIMER_TRIGGER_SOURCE_EXPOSURE_END	Starts with the reception of the Exposure End.

[Return value]

Returns result status. Refer to 5.8 Status code.

[Remarks]

This function will return error status if TimerTriggerSource register (or node) is not implemented in the camera.

Including TeliCamAPI.h is required.

5.5.14.8. SetCamTimerTriggerSource

Sets the source of Timer0Active pulse.

[Syntax]

```
CAM_API_STATUS  SetCamTimerTriggerSource (
  CAM_HANDLE                    hCam,
  CAM_TIMER_TRIGGER_SOURCE_TYPE    eTimerTriggerSource
);
```

[Parameters]

Parameter		Description
hCam	[in]	Camera-Handle of target camera.
eTimerTriggerSource	[in]	The source type of Timer0Active pulse.

[Return value]

Returns result status. Refer to 5.8 Status code.

[Remarks]

This function will return error status if TimerTriggerSource register (or node) is not implemented in the camera.

The source types that can be set may be different depending on the camera.
For details about TimerControl features, refer to instruction manual of the camera

Including TeliCamAPI.h is required.

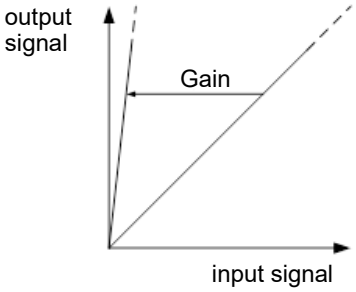
5.5.15.Gain

This feature group performs control of Gain.

Gain control adjusts an amplification factor applied to the output signal.

Gain feature adjusts manual gain.
GainAuto feature adjusts gain automatically.

For details about Gain features, refer to instruction manual of the camera.



5.5.15.1. GetCamGainMinMax

Gets the minimum and maximum Gain value of the camera.

[Syntax]

```
CAM_API_STATUS  GetCamGainMinMax (
  CAM_HANDLE        hCam,
  float64_t         *pdGainMin,
  float64_t         *pdGainMax
);
```

[Parameters]

Parameter		Description
hCam	[in]	Camera-Handle of target camera.
pdGainMin	[out]	A pointer to a variable that receives the minimum value of gain, in dB or times.
pdGainMax	[out]	A pointer to a variable that receives the maximum value of gain, in dB or times.

[Return value]

Returns result status. Refer to 5.8 Status code.

[Remarks]

This function will return error status if Gain register (or node) is not implemented in the camera.

Including TeliCamAPI.h is required.

5.5.15.2. GetCamGain

Gets the Gain value of the camera.

[Syntax]

```
CAM_API_STATUS  GetCamGain (
  CAM_HANDLE        hCam,
  float64_t         *pdGain
);
```

[Parameters]

Parameter		Description
hCam	[in]	Camera-Handle of target camera.
pdGain	[out]	A pointer to a variable that receives the value of gain, in dB or times.

[Return value]

Returns result status. Refer to 5.8 Status code.

[Remarks]

This function will return error status if Gain register (or node) is not implemented in the camera.

Including TeliCamAPI.h is required.

5.5.15.3. SetCamGain

Sets the Gain value of the camera.

[Syntax]

```
CAM_API_STATUS  SetCamGain (
  CAM_HANDLE        hCam,
  float64_t         dGain
);
```

[Parameters]

Parameter		Description
hCam	[in]	Camera-Handle of target camera.
dGain	[in]	The value of gain, in dB or times.

[Return value]

Returns result status. Refer to 5.8 Status code.

[Remarks]

This function will return error status if Gain register (or node) is not implemented in the camera.

Including TeliCamAPI.h is required.

5.5.15.4. GetCamGainAuto

Gets the AGC (automatic gain control) mode of the camera.

[Syntax]

```
CAM_API_STATUS  GetCamGainAuto (
  CAM_HANDLE            hCam,
  CAM_GAIN_AUTO_TYPE    *peGainAuto
);
```

[Parameters]

Parameter		Description
hCam	[in]	Camera-Handle of target camera.
peGainAuto	[out]	A pointer to a variable that receives the type of AGC mode.

[*CAM_GAIN_AUTO_TYPE* Enumeration]

Member	Description
CAM_GAIN_AUTO_OFF	Manual Gain Control (MANUAL)
CAM_GAIN_AUTO_AUTO	Automatic Gain Control (AGC)

[Return value]

Returns result status. Refer to 5.8 Status code.

[Remarks]

This function will return error status if Gain or GainAuto register (or node) is not implemented in the camera.

Including TeliCamAPI.h is required.

5.5.15.5. SetCamGainAuto

Sets the AGC (automatic gain control) mode of the camera.

[Syntax]

```
CAM_API_STATUS  SetCamGainAuto (
  CAM_HANDLE            hCam,
  CAM_GAIN_AUTO_TYPE    eGainAuto
);
```

[Parameters]

Parameter		Description
hCam	[in]	Camera-Handle of target camera.
eGainAuto	[in]	The type of AGC mode.

[Return value]

Returns result status. Refer to 5.8 Status code.

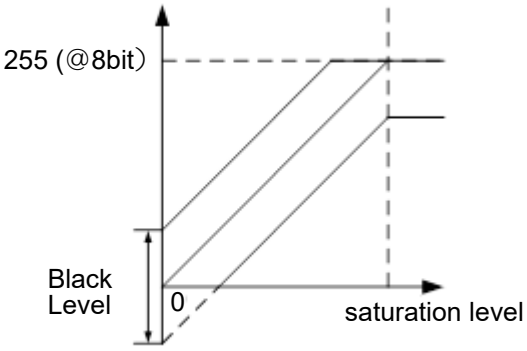
[Remarks]

This function will return error status if Gain or GainAuto register (or node) is not implemented in the camera.

Including TeliCamAPI.h is required.

5.5.16.BlackLevel

This feature group performs control of black level control.



For details about BlackLevel features, refer to instruction manual of the camera.

5.5.16.1. GetCamBlackLevelMinMax

Gets the minimum and maximum black level value of the camera.

[Syntax]

```
CAM_API_STATUS  GetCamBlackLevelMinMax (
  CAM_HANDLE        hCam,
  float64_t         *pdBlackLevelMin,
  float64_t         *pdBlackLevelMax
);
```

[Parameters]

Parameter		Description
hCam	[in]	Camera-Handle of target camera. d.
pdBlackLevelMin	[out]	A pointer to a variable that receives the minimum value of balck level, in %.
pdBlackLevelMax	[out]	A pointer to a variable that receives the maximum value of balck level, in %.

[Return value]

Returns result status. Refer to 5.8 Status code.

[Remarks]

This function will return error status if BlackLevel register (or node) is not implemented in the camera.

Including TeliCamAPI.h is required.

5.5.16.2. GetCamBlackLevel

Gets the black level value of the camera.

[Syntax]

```
CAM_API_STATUS  GetCamBlackLevel (
  CAM_HANDLE        hCam,
  float64_t         *pdBlackLevel
);
```

[Parameters]

Parameter		Description
hCam	[in]	Camera-Handle of target camera.
pdBlackLevel	[out]	A pointer to a variable that receives the value of balck level, in %.

[Return value]

Returns result status. Refer to 5.8 Status code.

[Remarks]

This function will return error status if BlackLevel register (or node) is not implemented in the camera.

Including TeliCamAPI.h is required.

5.5.16.3. SetCamBlackLevel

Sets the black level value of the camera.

[Syntax]

```
CAM_API_STATUS  SetCamBlackLevel (
  CAM_HANDLE        hCam,
  float64_t         dBlackLevel
);
```

[Parameters]

Parameter		Description
hCam	[in]	Camera-Handle of target camera.
dBlackLevel	[in]	The value of balck level, in %.

[Return value]

Returns result status. Refer to 5.8 Status code.

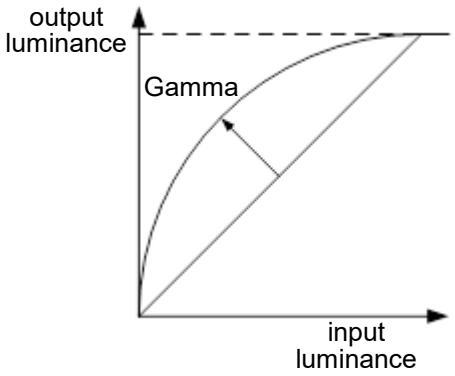
[Remarks]

This function will return error status if BlackLevel register (or node) is not implemented in the camera.

Including TeliCamAPI.h is required.

5.5.17.Gamma

This feature group performs control of gamma correction.



For details about Gamma features, refer to instruction manual of the camera.

5.5.17.1. GetCamGammaMinMax

Gets the minimum and maximum gamma correction value of the camera.

[Syntax]

```
CAM_API_STATUS  GetCamGammaMinMax (
  CAM_HANDLE        hCam,
  float64_t         *pdGammaMin,
  float64_t         *pdGammaMax
);
```

[Parameters]

Parameter		Description
hCam	[in]	Camera-Handle of target camera.
pdGammaMin	[out]	A pointer to a variable that receives the minimum value of gamma correction.
pdGammaMax	[out]	A pointer to a variable that receives the maximum value of gamma correction.

[Return value]

Returns result status. Refer to 5.8 Status code.

[Remarks]

This function will return error status if Gamma register (or node) is not implemented in the camera.

Including TeliCamAPI.h is required.

5.5.17.2. GetCamGamma

Gets the gamma correction value of the camera.

[Syntax]

```
CAM_API_STATUS  GetCamGamma (
  CAM_HANDLE        hCam,
  float64_t         *pdGamma
);
```

[Parameters]

Parameter		Description
hCam	[in]	Camera-Handle of target camera.
pdGamma	[out]	A pointer to a variable that receives the value of gamma correction.

[Return value]

Returns result status. Refer to 5.8 Status code.

[Remarks]

This function will return error status if Gamma register (or node) is not implemented in the camera.

Including TeliCamAPI.h is required.

5.5.17.3. SetCamGamma

Sets the gamma correction value of the camera.

[Syntax]

```
CAM_API_STATUS  SetCamGamma (
  CAM_HANDLE        hCam,
  float64_t         dGamma
);
```

[Parameters]

Parameter		Description
hCam	[in]	Camera-Handle of target camera.
dGamma	[in]	The value of gamma correction.

[Return value]

Returns result status. Refer to 5.8 Status code.

[Remarks]

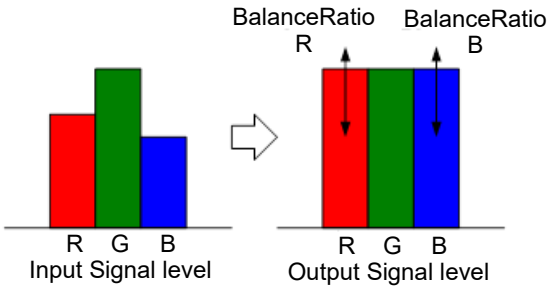
This function will return error status if Gamma register (or node) is not implemented in the camera.

Including TeliCamAPI.h is required.

5.5.18.WhiteBalance (BalanceRatio / BalanceWhiteAuto)

This feature group performs control of White balance control.
It is available only in color model cameras.

Camera will control White balance, adjusting relative gain value of R/G and B/G, manually or automatically.



For details about BalanceRatio and BalanceWhiteAuto features, refer to instruction manual of the camera.

5.5.18.1. GetCamBalanceRatioMinMax

Gets the minimum and maximum white balance gain of the camera.

[Syntax]

```
CAM_API_STATUS  GetCamBalanceRatioMinMax (
  CAM_HANDLE                        hCam,
  CAM_BALANCE_RATIO_SELECTOR_TYPE   eSelector,
  float64_t                         *pdBalanceRatioMin,
  float64_t                         *pdBalanceRatioMax
);
```

[Parameters]

Parameter		Description
hCam	[in]	Camera-Handle of target camera.
eSelector	[in]	Target color component of balance gain to control (R or B).
pdBalanceRatioMin	[out]	A pointer to a variable that receives the minimum value of white balance gain.
pdBalanceRatioMax	[out]	A pointer to a variable that receives the maximum value of white balance gain.

[*CAM_BALANCE_RATIO_SELECTOR_TYPE* Enumeration]

Member	Description
CAM_BALANCE_RATIO_SELECTOR_RED	BalanceRatio = Red Gain
CAM_BALANCE_RATIO_SELECTOR_BLUE	BalanceRatio = Blue Gain

[Return value]

Returns result status. Refer to 5.8 Status code.

[Remarks]

This function will return error status if WhiteBalance* register (or BalanceRatioSelector / BalanceRatio node) is not implemented in the camera.

Including TeliCamAPI.h is required.

5.5.18.2. GetCamBalanceRatio

Gets the white balance gain of the camera.

[Syntax]

```
CAM_API_STATUS  GetCamBalanceRatio (
  CAM_HANDLE                        hCam,
  CAM_BALANCE_RATIO_SELECTOR_TYPE   eSelector,
  float64_t                         *pdBalanceRatio
);
```

[Parameters]

Parameter		Description
hCam	[in]	Camera-Handle of target camera.
eSelector	[in]	Target color component of balance gain to control (R or B).
pdBalanceRatio	[out]	A pointer to a variable that receives the value of white balance gain.

[Return value]

Returns result status. Refer to 5.8 Status code.

[Remarks]

This function will return error status if WhiteBalance* register (or BalanceRatioSelector / BalanceRatio node) is not implemented in the camera.

Including TeliCamAPI.h is required.

5.5.18.3. SetCamBalanceRatio

Sets the white balance gain of the camera.

[Syntax]

```
CAM_API_STATUS  SetCamBalanceRatio (
  CAM_HANDLE                        hCam,
  CAM_BALANCE_RATIO_SELECTOR_TYPE   eSelector,
  float64_t                         dBalanceRatio
);
```

[Parameters]

Parameter		Description
hCam	[in]	Camera-Handle of target camera.
eSelector	[in]	Target color component of balance gain to control (R or B).
dBalanceRatio	[in]	The value of white balance gain.

[Return value]

Returns result status. Refer to 5.8 Status code.

[Remarks]

Refer to TeliCamAPI about `CAM_BALANCE_RATIO_SELECTOR_TYPE`.

This function will return error status if BalanceRatioSelector and BalanceRatio registers (or nodes) are not implemented in the camera.

Including TeliCamAPI.h is required.

5.5.18.4. GetCamBalanceWhiteAuto

Gets the auto white balance mode of the camera.

[Syntax]

```
CAM_API_STATUS  GetCamBalanceWhiteAuto (
  CAM_HANDLE                    hCam,
  CAM_BALANCE_WHITE_AUTO_TYPE   *peBalanceWhiteAuto
);
```

[Parameters]

Parameter		Description
hCam	[in]	Camera-Handle of target camera.
peBalanceWhiteAuto	[out]	A pointer to a variable that receives the type of auto white balance mode.

[*CAM_BALANCE_WHITE_AUTO_TYPE* Enumeration]

Member	Description
CAM_BALANCE_WHITE_AUTO_OFF	No operation.
CAM_BALANCE_WHITE_AUTO_CONTINUOUS	Execute auto white balance continuously.
CAM_BALANCE_WHITE_AUTO_ONCE	Execute auto white balance once.

[Return value]

Returns result status. Refer to 5.8 Status code.

[Remarks]

This function will return error status if WhiteBalance* register (or BalanceWhiteAuto node) is not implemented in the camera.

Including TeliCamAPI.h is required.

5.5.18.5. SetCamBalanceWhiteAuto

Sets the auto white balance mode of the camera.

[Syntax]

```
CAM_API_STATUS  SetCamBalanceWhiteAuto (
  CAM_HANDLE                    hCam,
  CAM_BALANCE_WHITE_AUTO_TYPE   eBalanceWhiteAuto
);
```

[Parameters]

Parameter		Description
hCam	[in]	Camera-Handle of target camera.
BalanceWhiteAuto	[in]	The type of auto white balance mode.

[Return value]

Returns result status. Refer to 5.8 Status code.

[Remarks]

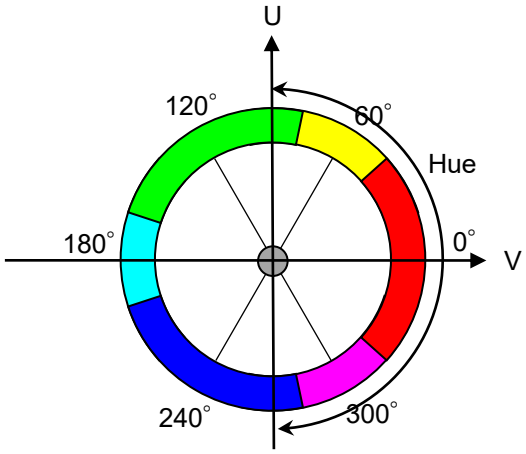
This function will return error status if WhiteBalance* register (or BalanceWhiteAuto node) is not implemented in the camera.

Including TeliCamAPI.h is required.

5.5.19.Hue

This feature group performs control of Hue
It is available only in color model cameras.

For details about Hue features, refer to instruction manual of the camera.



5.5.19.1. GetCamHueMinMax

Gets the minimum and maximum hue value of the camera.

[Syntax]

```
CAM_API_STATUS  GetCamHueMinMax (
  CAM_HANDLE        hCam,
  float64_t         *pdHueMin,
  float64_t         *pdHueMax
);
```

[Parameters]

Parameter		Description
hCam	[in]	Camera-Handle of target camera.
pdHueMin	[out]	A pointer to a variable that receives the minimum value of hue, in degrees.
pdHueMax	[out]	A pointer to a variable that receives the maximum value of hue, in degrees.

[Return value]

Returns result status. Refer to 5.8 Status code.

[Remarks]

This function will return error status if Hue register (or node) is not implemented in the camera.

Including TeliCamAPI.h is required.

5.5.19.2. GetCamHue

Gets the hue value of the camera.

[Syntax]

```
CAM_API_STATUS  GetCamHue (
  CAM_HANDLE        hCam,
  float64_t         *pdHue,
);
```

[Parameters]

Parameter		Description
hCam	[in]	Camera-Handle of target camera.
pdHue	[out]	A pointer to a variable that receives the value of hue, in degrees.

[Return value]

Returns result status. Refer to 5.8 Status code.

[Remarks]

This function will return error status if Hue register (or node) is not implemented in the camera.

Including TeliCamAPI.h is required.

5.5.19.3. SetCamHue

Sets the hue value of the camera.

[Syntax]

```
CAM_API_STATUS  GetCamHue (
  CAM_HANDLE        hCam,
  float64_t         dHue,
);
```

[Parameters]

Parameter		Description
hCam	[in]	Camera-Handle of target camera.
dHue	[in]	The value of hue, in degrees.

[Return value]

Returns result status. Refer to 5.8 Status code.

[Remarks]

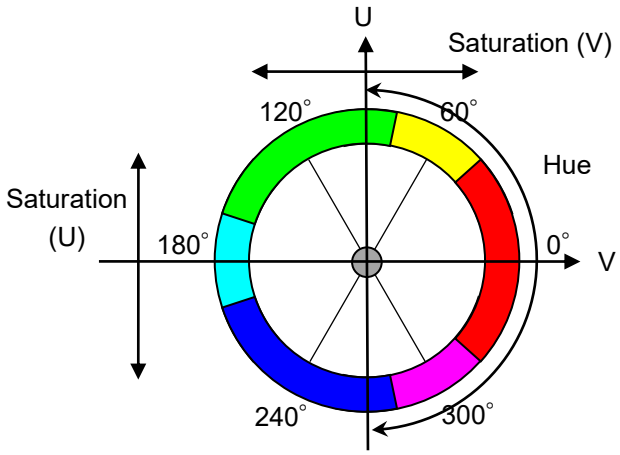
This function will return error status if Hue register (or node) is not implemented in the camera.

Including TeliCamAPI.h is required.

5.5.20.Saturation

This feature group performs control of Saturation.
It is available only in color model cameras.

There are two types of cameras.
One is a camera that selects the target element (U / V) and sets the value in units of [times].
The other is a camera that sets the value in units of [%] common to U / V elements.
For details about Saturation features, refer to instruction manual of the camera.



5.5.20.1. GetCamSaturationSelector

Gets the target element of saturation.

[Syntax]

```
CAM_API_STATUS  GetCamSaturationSelector (
  CAM_HANDLE                    hCam,
  CAM_SATURATION_SELECTOR_TYPE  *peSelector
);
```

[Parameters]

Parameter		Description
hCam	[in]	Camera-Handle of target camera.
peSelector	[out]	A pointer to a variable that receives the element type of saturation.

[*CAM_SATURATION_SELECTOR_TYPE* Enumeration]

Member	Description
CAM_SATURATION_SELECTOR_U	Saturation = U Gain
CAM_SATURATION_SELECTOR_V	Saturation = V Gain

[Return value]

Returns result status. Refer to 5.8 Status code.

[Remarks]

This function will return error status if SaturationSelector register (or node) is not implemented in the camera.

Including TeliCamAPI.h is required.

5.5.20.2. SetCamSaturationSelector

Sets the target element of saturation.

[Syntax]

```
CAM_API_STATUS  SetCamSaturationSelector (
  CAM_HANDLE                    hCam,
  CAM_SATURATION_SELECTOR_TYPE  eSelector
);
```

[Parameters]

Parameter		Description
hCam	[in]	Camera-Handle of target camera.
eSelector	[in]	The element type of saturation.

[Return value]

Returns result status. Refer to 5.8 Status code.

[Remarks]

This function will return error status if SaturationSelector register (or node) is not implemented in the camera.

Including TeliCamAPI.h is required.

5.5.20.3. GetCamSaturationMinMax

Gets the minimum and maximum saturation value of the camera.

[Syntax]

```
CAM_API_STATUS  GetCamSaturationMinMax (
  CAM_HANDLE      hCam,
  float64_t       *pdSaturationMin,
  float64_t       *pdSaturationMax
);
```

[Parameters]

Parameter		Description
hCam	[in]	Camera-Handle of target camera.
pdSaturationMin	[out]	A pointer to a variable that receives the minimum value of saturation.
pdSaturationMax	[out]	A pointer to a variable that receives the maximum value of saturation.

[Return value]

Returns result status. Refer to 5.8 Status code.

[Remarks]

This function will return error status if Saturation register (or node) is not implemented in the camera.

There are two types of cameras.
One is a camera that selects the target element (U / V) and sets the value in units of [times].
The other is a camera that sets the value in units of [%] common to U / V elements.
If the camera is the type that selects the target element (U / V), set the target element (U / V) with SetCamSaturationSelector() before calling this function.
For details about Saturation features, refer to instruction manual of the camera.

Including TeliCamAPI.h is required.

5.5.20.4. GetCamSaturation

Gets the saturation value of the camera.

[Syntax]

```
CAM_API_STATUS  GetCamSaturation (
  CAM_HANDLE        hCam,
  float64_t         *pdSaturation
);
```

[Parameters]

Parameter		Description
hCam	[in]	Camera-Handle of target camera.
pdSaturation	[out]	A pointer to a variable that receives the value of saturation.

[Return value]

Returns result status. Refer to 5.8 Status code.

[Remarks]

This function will return error status if Saturation register (or node) is not implemented in the camera.

There are two types of cameras.
One is a camera that selects the target element (U / V) and sets the value in units of [times].
The other is a camera that sets the value in units of [%] common to U / V elements.
If the camera is the type that selects the target element (U / V), set the target element (U / V) with SetCamSaturationSelector() before calling this function.
For details about Saturation features, refer to instruction manual of the camera.

Including TeliCamAPI.h is required.

5.5.20.5. SetCamSaturation

Sets the saturation value of the camera.

[Syntax]

```
CAM_API_STATUS  SetCamSaturation (
  CAM_HANDLE        hCam,
  float64_t         dSaturation
);
```

[Parameters]

Parameter		Description
hCam	[in]	Camera-Handle of target camera.
dSaturation	[in]	The value of saturation.

[Return value]

Returns result status. Refer to 5.8 Status code.

[Remarks]

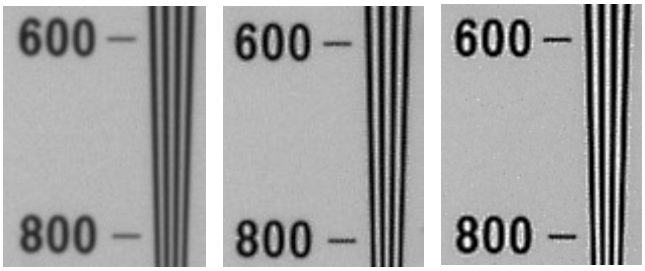
This function will return error status if Saturation register (or node) is not implemented in the camera.

There are two types of cameras.
One is a camera that selects the target element (U / V) and sets the value in units of [times].
The other is a camera that sets the value in units of [%] common to U / V elements.
If the camera is the type that selects the target element (U / V), set the target element (U / V) with SetCamSaturationSelector() before calling this function.
For details about Saturation features, refer to instruction manual of the camera.

Including TeliCamAPI.h is required.

5.5.21.Sharpness

This feature group performs control of Sharpness.



Sharpness=0 Sharpness=4 Sharpness=7

For details about Sharpness features, refer to instruction manual of the camera.

5.5.21.1. GetCamSharpnessMinMax

Gets the minimum and maximum sharpness value of the camera.

[Syntax]

```
CAM_API_STATUS  GetCamSharpnessMinMax (
  CAM_HANDLE        hCam,
  uint32_t          *puiSharpnessMin,
  uint32_t          *puiSharpnessMax
);
```

[Parameters]

Parameter		Description
hCam	[in]	Camera-Handle of target camera.
puiSharpnessMin	[out]	A pointer to a variable that receives the minimum value of sharpness.
puiSharpnessMax	[out]	A pointer to a variable that receives the maximum value of sharpness.

[Return value]

Returns result status. Refer to 5.8 Status code.

[Remarks]

This function will return error status if Sharpness register (or node) is not implemented in the camera.

Including TeliCamAPI.h is required.

5.5.21.2. GetCamSharpness

Gets the sharpness value of the camera.

[Syntax]

```
CAM_API_STATUS  GetCamSharpness (
  CAM_HANDLE        hCam,
  uint32_t          *puiSharpness
);
```

[Parameters]

Parameter		Description
hCam	[in]	Camera-Handle of target camera.
puiSharpness	[out]	A pointer to a variable that receives the value of sharpness.

[Return value]

Returns result status. Refer to 5.8 Status code.

[Remarks]

This function will return error status if Sharpness register (or node) is not implemented in the camera.

Including TeliCamAPI.h is required.

5.5.21.3. SetCamSharpness

Sets the sharpness value of the camera.

[Syntax]

```
CAM_API_STATUS  SetCamSharpness (
  CAM_HANDLE        hCam,
  uint32_t          uiSharpness
);
```

[Parameters]

Parameter		Description
hCam	[in]	Camera-Handle of target camera.
uiSharpness	[in]	The value of sharpness.

[Return value]

Returns result status. Refer to 5.8 Status code.

[Remarks]

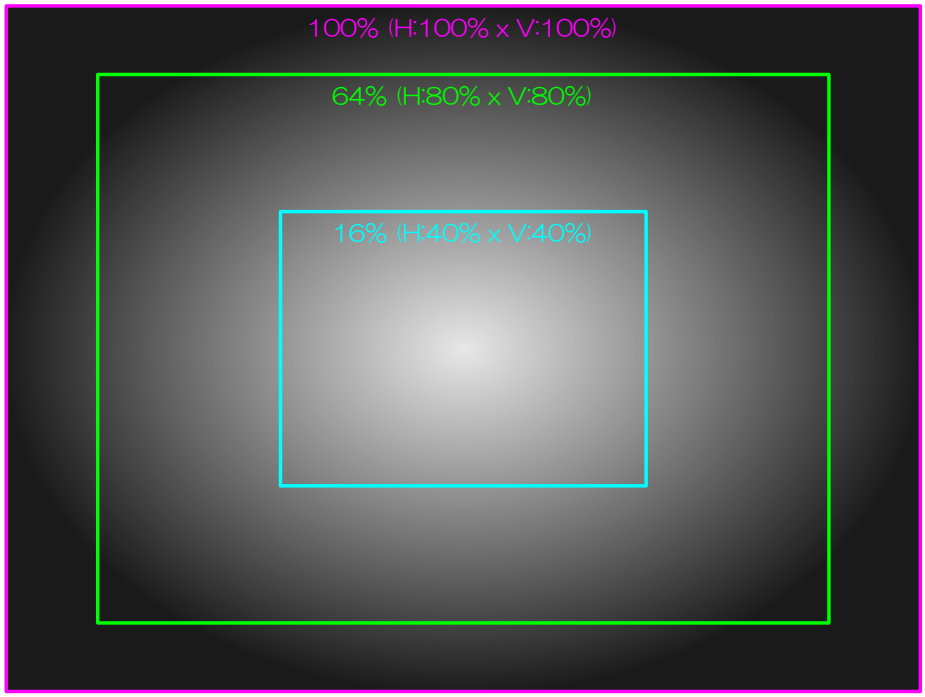
This function will return error status if Sharpness register (or node) is not implemented in the camera.

Including TeliCamAPI.h is required.

5.5.22.ALCControl

This feature group performs control of ALC.

- ALCPhotometricAreaSize defines photometric area size for measuring luminance.



ALCPhotometricAreaSize (e.g. 100%, 64%, 16%)

- ALCExposureValue defines a correction value for a convergence value.
  Final convergence value is determined by the following formula.

$$\text{Final convergence value} = 84(\text{Reference Luminance}) \times 2^{\text{ALCExposureValue}}$$

For details about ALCControl features, refer to instruction manual of the camera.

5.5.22.1. GetCamALCPhotometricAreaSizeMinMax

Gets the minimum and maximum ALC photometric area size of the camera.

[Syntax]

```
CAM_API_STATUS  GetCamALCPhotometricAreaSizeMinMax (
  CAM_HANDLE        hCam,
  float64_t         *pdPercentMin,
  float64_t         *pdPercentMax
);
```

[Parameters]

Parameter		Description
hCam	[in]	Camera-Handle of target camera.
pdPercentMin	[out]	A pointer to a variable that receives the minimum value of ALC photometric area size, in %.
pdPercentMax	[out]	A pointer to a variable that receives the maximum value of ALC photometric area size, in %.

[Return value]

Returns result status. Refer to 5.8 Status code.

[Remarks]

This function will return error status if ALCPhotometricAreaSize register (or node) is not implemented in the camera.

Including TeliCamAPI.h is required.

5.5.22.2. GetCamALCPhotometricAreaSize

Gets the ALC photometric area size of the camera.

[Syntax]

```
CAM_API_STATUS  GetCamALCPhotometricAreaSize (
  CAM_HANDLE        hCam,
  float64_t         *pdPercent
);
```

[Parameters]

Parameter		Description
hCam	[in]	Camera-Handle of target camera.
pdPercent	[out]	A pointer to a variable that receives the value of ALC photometric area size, in %.

[Return value]

Returns result status. Refer to 5.8 Status code.

[Remarks]

This function will return error status if ALCPhotometricAreaSize register (or node) is not implemented in the camera.

Including TeliCamAPI.h is required.

5.5.22.3. SetCamALCPhotometricAreaSize

Sets the ALC photometric area size of the camera.

[Syntax]

```
CAM_API_STATUS  SetCamALCPhotometricAreaSize (
  CAM_HANDLE        hCam,
  float64_t         dPercent
);
```

[Parameters]

Parameter		Description
hCam	[in]	Camera-Handle of target camera.
dPercent	[in]	The value of ALC photometric area size, in %.

[Return value]

Returns result status. Refer to 5.8 Status code.

[Remarks]

This function will return error status if ALCPhotometricAreaSize register (or node) is not implemented in the camera.

Including TeliCamAPI.h is required.

5.5.22.4. GetCamALCExposureValueMinMax

Gets the minimum and maximum ALC Exposure Value of the camera.

[Syntax]

```
CAM_API_STATUS  GetCamALCExposureValueMinMax (
  CAM_HANDLE        hCam,
  float64_t         *pdExposureValueMin,
  float64_t         *pdExposureValueMax
);
```

[Parameters]

Parameter		Description
hCam	[in]	Camera-Handle of target camera.
pdExposureValueMin	[out]	A pointer to a variable that receives the minimum value of ALC Exposure Value, in EV.
pdExposureValueMax	[out]	A pointer to a variable that receives the maximum value of ALC Exposure Value, in EV.

[Return value]

Returns result status. Refer to 5.8 Status code.

[Remarks]

This function will return error status if ALCExposureValue register (or node) is not implemented in the camera.

Including TeliCamAPI.h is required.

5.5.22.5. GetCamALCExposureValue

Gets the ALC Exposure Value of the camera.

[Syntax]

```
CAM_API_STATUS  GetCamALCExposureValue (
  CAM_HANDLE        hCam,
  float64_t         *pdExposureValue
);
```

[Parameters]

Parameter		Description
hCam	[in]	Camera-Handle of target camera.
pdExposureValue	[out]	A pointer to a variable that receives the value of ALC Exposure Value, in EV.

[Return value]

Returns result status. Refer to 5.8 Status code.

[Remarks]

This function will return error status if ALCExposureValue register (or node) is not implemented in the camera.

Including TeliCamAPI.h is required.

5.5.22.6. SetCamALCExposureValue

Sets the ALC Exposure Value of the camera.

[Syntax]

```
CAM_API_STATUS  SetCamALCExposureValue (
  CAM_HANDLE        hCam,
  float64_t         dExposureValue
);
```

[Parameters]

Parameter		Description
hCam	[in]	Camera-Handle of target camera.
dExposureValue	[in]	The value of ALC Exposure Value, in EV.

[Return value]

Returns result status. Refer to 5.8 Status code.

[Remarks]

This function will return error status if ALCExposureValue register (or node) is not implemented in the camera.

Including TeliCamAPI.h is required.

5.5.23.ColorCorrectionMatrix

This feature group performs control of Color Correction Matrix for correcting pixel color value.
It is available only in color model cameras.

The relationship between original pixel data (R, G, and B) and corrected pixel data (R', G', and B') are represented in the following formula.

$$\begin{pmatrix} R' \\ G' \\ B' \end{pmatrix} = \begin{pmatrix} 1 & -mask\_rg & -mask\_rb \\ -mask\_gr & 1 & -mask\_gb \\ -mask\_br & -mask\_bg & 1 \end{pmatrix} \begin{pmatrix} R & (G-R) & (B-R) \\ (R-G) & G & (B-G) \\ (R-B) & (G-B) & B \end{pmatrix}$$

$$R' = (1 - mask\_rg - mask\_rb) \times R + mask\_rg \times G + mask\_rb \times B$$

$$G' = mask\_gr \times R + (1 - mask\_gr - mask\_gb) \times G + mask\_gb \times B$$

$$B' = mask\_br \times R + mask\_bg \times G + (1 - mask\_br \hat{} mask\_bg) \times B$$

The correspondence of “SelectorI” and “SelectorJ” to color correction matrix element is as follows.

	SelectorJ = R	SelectorJ = G	SelectorJ = B
SelectorI = R		mask_rg	mask_rb
SelectorI = G	mask_gr		mask_gb
SelectorI = B	mask_br	mask_bg	

The following functions use `CAM_COLOR_CORRECTION_MATRIX_TYPE` value as element selector, which specifies both “`SelectorI`” and “`SelectorJ`” in an enumeration member value

For details about ColorCorrectionMatrix features, refer to instruction manual of the camera.

5.5.23.1. GetCamColorCorrectionMatrixMinMax

Gets the minimum and maximum coefficient value of color correction matrix in the camera.

[Syntax]

```
CAM_API_STATUS  GetCamColorCorrectionMatrixMinMax (
  CAM_HANDLE                          hCam,
  CAM_COLOR_CORRECTION_MATRIX_TYPE    eType,
  float64_t                           *pdMatrixMin,
  float64_t                           *pdMatrixMax
);
```

[Parameters]

Parameter		Description
hCam	[in]	Camera-Handle of target camera.
eType	[in]	Target column element of color correction matrix.
pdMatrixMin	[out]	A pointer to a variable that receives the minimum coefficient value of color correction matrix.
pdMatrixMax	[out]	A pointer to a variable that receives the maximum coefficient value of color correction matrix.

[*CAM_COLOR_CORRECTION_MATRIX_TYPE* Enumeration]

Member	Description
CAM_COLOR_CORRECTION_MATRIX_RG	SelectorI = R , SelectorJ = G
CAM_COLOR_CORRECTION_MATRIX_RB	SelectorI = R , SelectorJ = B
CAM_COLOR_CORRECTION_MATRIX_GR	SelectorI = G , SelectorJ = R
CAM_COLOR_CORRECTION_MATRIX_GB	SelectorI = G , SelectorJ = B
CAM_COLOR_CORRECTION_MATRIX_BR	SelectorI = B , SelectorJ = R
CAM_COLOR_CORRECTION_MATRIX_BG	SelectorI = B , SelectorJ = G

[Return value]

Returns result status. Refer to 5.8 Status code.

[Remarks]

This function will return error status if Masking* register, or ColorCorrectionMatrix and ColorCorrectionMatrixSelectorI and ColorCorrectionMatrixSelectorJ registers (or nodes) are not implemented in the camera.

Including TeliCamAPI.h is required.

5.5.23.2. GetCamColorCorrectionMatrix

Gets the coefficient value of color correction matrix in the camera.

[Syntax]

```
CAM_API_STATUS  GetCamColorCorrectionMatrix (
  CAM_HANDLE                          hCam,
  CAM_COLOR_CORRECTION_MATRIX_TYPE    eType,
  float64_t                           *pdMatrix
);
```

[Parameters]

Parameter		Description
hCam	[in]	Camera-Handle of target camera.
eType	[in]	Target column element of color correction matrix.
pdMatrix	[out]	A pointer to a variable that receives the coefficient value of color correction matrix.

[Return value]

Returns result status. Refer to 5.8 Status code.

[Remarks]

This function will return error status if Masking* register, or ColorCorrectionMatrix and ColorCorrectionMatrixSelectorI and ColorCorrectionMatrixSelectorJ registers (or nodes) are not implemented in the camera.

Including TeliCamAPI.h is required.

5.5.23.3. SetCamColorCorrectionMatrix

Sets the coefficient value of color correction matrix in the camera.

[Syntax]

```
CAM_API_STATUS  SetCamColorCorrectionMatrix (
  CAM_HANDLE                          hCam,
  CAM_COLOR_CORRECTION_MATRIX_TYPE    eType,
  float64_t                           dMatrix
);
```

[Parameters]

Parameter		Description
hCam	[in]	Camera-Handle of target camera.
eType	[in]	Target column element of color correction matrix.
dMatrix	[in]	The coefficient value of color correction matrix.

[Return value]

Returns result status. Refer to 5.8 Status code.

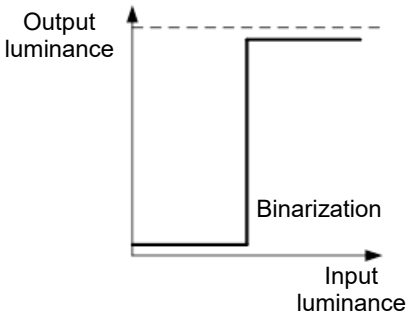
[Remarks]

This function will return error status if Masking* register, or ColorCorrectionMatrix and ColorCorrectionMatrixSelectorI and ColorCorrectionMatrixSelectorJ registers (or nodes) are not implemented in the camera.

Including TeliCamAPI.h is required.

5.5.24.LUT

This feature group performs control of LUT (Look up table) feature for correcting pixel value.



example of LUT setting

For details about LUT features, refer to instruction manual of the camera.

5.5.24.1. GetCamLutEnable

Gets the enable state of LUT function in the camera.

[Syntax]

```
CAM_API_STATUS GetCamLutEnable (
  CAM_HANDLE        hCam,
  bool8_t           *pbEnable
);
```

[Parameters]

Parameter		Description
hCam	[in]	Camera-Handle of target camera.
pbEnable	[out]	A pointer to a variable that receives the enable state of LUT function. If false, the LUT function is disable. If true, the LUT function is enable.

[Return value]

Returns result status. Refer to 5.8 Status code.

[Remarks]

This function will return error status if LUTEnable register (or node) is not implemented in the camera.

Including TeliCamAPI.h is required.

5.5.24.2. SetCamLutEnable

Sets the enable state of LUT function in the camera.

[Syntax]

```
CAM_API_STATUS  SetCamLutEnable (
  CAM_HANDLE        hCam,
  bool8_t           bEnable
);
```

[Parameters]

Parameter		Description
hCam	[in]	Camera-Handle of target camera.
bEnable	[in]	The enable state of LUT function. If false, the LUT function is disable. If true, the LUT function is enable.

[Return value]

Returns result status. Refer to 5.8 Status code.

[Remarks]

This function will return error status if LUTEnable register (or node) is not implemented in the camera.

Including TeliCamAPI.h is required.

5.5.24.3. GetCamLutValue

Gets the output level of LUT in the camera.

[Syntax]

```
CAM_API_STATUS  GetCamLutValue (
  CAM_HANDLE        hCam,
  uint32_t          uiLutIndex,
  uint32_t          *puiLutValue
);
```

[Parameters]

Parameter		Description
hCam	[in]	Camera-Handle of target camera.
uiLutIndex	[in]	The target input level of LUT. The range of the level is from 0 up to 1023, or from 0 up to 4095.
puiLutValue	[out]	A pointer to a variable that receives the output level of LUT. The range of the level is from 0 up to 1023, or from 0 up to 4095.

[Return value]

Returns result status. Refer to 5.8 Status code.

[Remarks]

This function will return error status if LUTValueAll, or LUTIndex and LUTValue registers (or nodes) are not implemented in the camera.

The range of input and output level may be different depending on the camera.
For details about LUT features, refer to instruction manual of the camera.

Including TeliCamAPI.h is required.

5.5.24.4. SetCamLutValue

Sets the output level of LUT in the camera.

[Syntax]

```
CAM_API_STATUS  SetCamLutValue (
  CAM_HANDLE        hCam,
  uint32_t          uiLutIndex,
  uint32_t          uiLutValue
);
```

[Parameters]

Parameter		Description
hCam	[in]	Camera-Handle of target camera.
uiLutIndex	[in]	The target input level of LUT. The level should be from 0 up to 1023, or from 0 up to 4095.
puiLutValue	[in]	The output level of LUT. The level should be from 0 up to 1023, or from 0 up to 4095.

[Return value]

Returns result status. Refer to 5.8 Status code.

[Remarks]

This function will return error status if LUTValueAll, or LUTIndex and LUTValue registers (or nodes) are not implemented in the camera.

The range of input and output level may be different depending on the camera.
For details about LUT features, refer to instruction manual of the camera.

Including TeliCamAPI.h is required.

5.5.25.UserSetControl

This feature group performs control of UserSet.

User application can save current settings of the camera to non-volatile user memory, load saved settings to camera registers using this control.

For details about UserSetControl features, refer to instruction manual of the camera.

5.5.25.1. ExecuteCamUserSetLoad

Loads parameters saved in non-volatile memory (user memory) of the camera and save them to registers in the camera.

[Syntax]

```
CAM_API_STATUS  ExecuteCamUserSetLoad (
  CAM_HANDLE                  hCam,
  CAM_USER_SET_SELECTOR_TYPE  eSelector
);
```

[Parameters]

Parameter		Description
hCam	[in]	Camera-Handle of target camera.
eSelector	[in]	The type of user memory channel to load.

[*CAM_USER_SET_SELECTOR_TYPE* Enumeration]

Member	Description
CAM_USER_SET_SELECTOR_DEFAULT	Initial factory setting. This channel is read-only.
CAM_USER_SET_SELECTOR_USER_SET1	Memory channel 1 for user setting.
CAM_USER_SET_SELECTOR_USER_SET2	Memory channel 2 for user setting.
CAM_USER_SET_SELECTOR_USER_SET3	Memory channel 3 for user setting.
CAM_USER_SET_SELECTOR_USER_SET4	Memory channel 4 for user setting.
CAM_USER_SET_SELECTOR_USER_SET5	Memory channel 5 for user setting.
CAM_USER_SET_SELECTOR_USER_SET6	Memory channel 6 for user setting.
CAM_USER_SET_SELECTOR_USER_SET7	Memory channel 7 for user setting.
CAM_USER_SET_SELECTOR_USER_SET8	Memory channel 8 for user setting.
CAM_USER_SET_SELECTOR_USER_SET9	Memory channel 9 for user setting.
CAM_USER_SET_SELECTOR_USER_SET10	Memory channel 10 for user setting.
CAM_USER_SET_SELECTOR_USER_SET11	Memory channel 11 for user setting.
CAM_USER_SET_SELECTOR_USER_SET12	Memory channel 12 for user setting.
CAM_USER_SET_SELECTOR_USER_SET13	Memory channel 13 for user setting.
CAM_USER_SET_SELECTOR_USER_SET14	Memory channel 14 for user setting.
CAM_USER_SET_SELECTOR_USER_SET15	Memory channel 15 for user setting.

[Return value]

Returns result status. Refer to 5.8 Status code.

[Remarks]

This function will return error status if UserSetSelector and UserSetCommand, or UserSetLoad registers (or nodes) are not implemented in the camera.

Refer to instruction manual of the camera, to check target registers for loading from user memory.

Including TeliCamAPI.h is required.

5.5.25.2. ExecuteCamUserSetSave

Saves the current parameters to non-volatile memory (user memory) of the camera.

[Syntax]

```
CAM_API_STATUS  ExecuteCamUserSetSave (
  CAM_HANDLE                hCam,
  CAM_USER_SET_SELECTOR_TYPE  eSelector
);
```

[Parameters]

Parameter		Description
hCam	[in]	Camera-Handle of target camera.
eSelector	[in]	The type of user memory channel to save. CAM_USER_SET_SELECTOR_DEFAULT is not available for saving target.

[Return value]

Returns result status. Refer to 5.8 Status code.

[Remarks]

This function will return error status if UserSetSelector and UserSetCommand, or UserSetSave registers (or nodes) are not implemented in the camera.

Refer to instruction manual of the camera, to check target registers for saving data to user memory. The difference between UserSetSave and UserSetQuickSave is described in instruction manual of the camera.

Including TeliCamAPI.h is required.

5.5.25.3. ExecuteCamUserSetQuickSave

Saves current parameters to volatile memory of the camera.

[Syntax]

```
CAM_API_STATUS  ExecuteCamUserSetQuickSave (
  CAM_HANDLE                  hCam,
  CAM_USER_SET_SELECTOR_TYPE  eSelector
);
```

[Parameters]

Parameter		Description
hCam	[in]	Camera-Handle of target camera.
eSelector	[in]	The type of user memory channel to save. CAM_USER_SET_SELECTOR_DEFAULT is not available for saving target.

[Return value]

Returns result status. Refer to 5.8 Status code.

[Remarks]

This function will return error status if UserSetSelector and UserSetCommand, or UserSetQuickSave registers (or nodes) are not implemented in the camera.

UserSetQuickSave can reduce the overhead time of UserSetSave greatly because it stored to internal RAM.
You can also save UserSets to non-volatile memory(Serial Flash) if necessary by UserSetSave. (backward compatible)

Refer to instruction manual of the camera, to check target registers for saving parameters to user memory. The difference between UserSetSave and UserSetQuickSave is described in instruction manual of the camera.

Including TeliCamAPI.h is required.

5.5.25.4. GetCamUserSetDefault

Gets the user memory channel to be loaded when the camera is powered on.

[Syntax]

```
CAM_API_STATUS  GetCamUserSetDefault (
  CAM_HANDLE                  hCam,
  CAM_USER_SET_SELECTOR_TYPE  *peSelector
);
```

[Parameters]

Parameter		Description
hCam	[in]	Camera-Handle of target camera.
peSelector	[out]	A pointer to a variable that receives the typelof user memory channel.

[Return value]

Returns result status. Refer to 5.8 Status code.

[Remarks]

This function will return error status if UserSetDefault register (or node) is not implemented in the camera.

Including TeliCamAPI.h is required.

5.5.25.5. SetCamUserSetDefault

Sets the user memory channel to be loaded when the camera is powered on.

[Syntax]

```
CAM_API_STATUS  SetCamUserSetDefault (
  CAM_HANDLE                  hCam,
  CAM_USER_SET_SELECTOR_TYPE  eSelector
);
```

[Parameters]

Parameter		Description
hCam	[in]	Camera-Handle of target camera.
eSelector	[in]	The type of user memory channel.

[Return value]

Returns result status. Refer to 5.8 Status code.

[Remarks]

This function will return error status if UserSetDefault register (or node) is not implemented in the camera.

When you write a value to UserSetDefault register, some cameras are saved in non-volatile memory, but some cameras are not saved. (Depending on the firmware version of the camera.) If you use the camera that is not saved in non-volatile memory, please use ExecuteCamUserSetSaveAndSetDefault() function.
For details about UserSetControl features, refer to instruction manual of the camera.

Including TeliCamAPI.h is required.

5.5.25.6. ExecuteCamUserSetSaveAndSetDefault

Saves the current parameters to non-volatile memory (user memory) of the camera, and sets the user memory channel to be loaded when the camera is powered on.

[Syntax]

```
CAM_API_STATUS  ExecuteCamUserSetSaveAndSetDefault (
  CAM_HANDLE                  hCam,
  CAM_USER_SET_SELECTOR_TYPE  eSelector
);
```

[Parameters]

Parameter		Description
hCam	[in]	Camera-Handle of target camera.
eSelector	[in]	The type of user memory channel to save and set.

[Return value]

Returns result status. Refer to 5.8 Status code.

[Remarks]

This function will return error status if UserSetDefault and UserSetCommand, or UserSetDefault and UserSetSave registers (or nodes) are not implemented in the camera.

When other than `CAM_USER_SET_SELECTOR_DEFAULT` value is specified to `eSelector`, current parameters will be written to specified channel of non-volatile memory (user memory) in the camera. The camera will start up with parameters saved in the specified channed from the next power-on.

When `CAM_USER_SET_SELECTOR_DEFAULT` is specified to `eSelector`, current parameters will be discarded and initial factory settings will be loaded immediately. The camera will start up with initial factory settings from the next power-on.

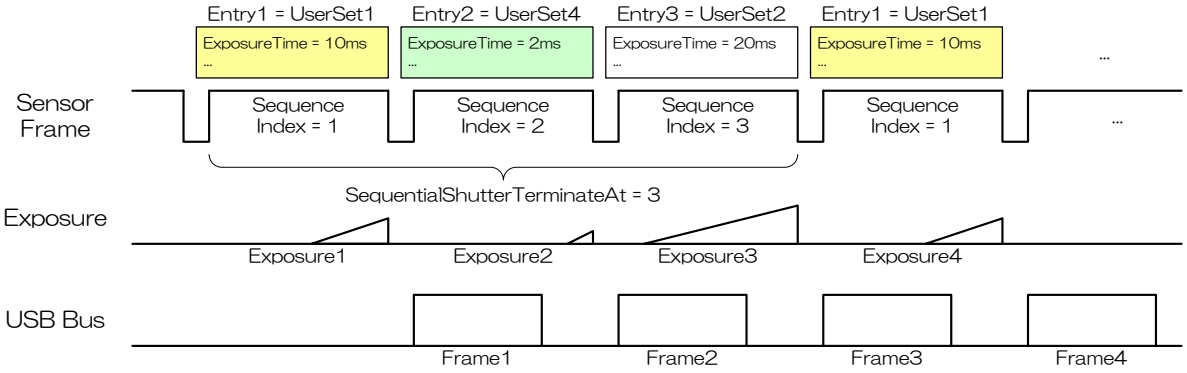
Refer to instruction manual of the camera, to check target registers for saving data to user memory.

Including TeliCamAPI.h is required.

5.5.26.SequentialShutter

This feature group performs control of Sequential shutter.

Sequential Shutter function performs sequential capturing with applying the settings of UserSet that have been made entry in advance.



For details about SequentialShutter features, refer to instruction manual of the camera.

5.5.26.1. GetCamSequentialShutterEnable

Gets the enable state of Sequential Shutter function in the camera.

[Syntax]

```
CAM_API_STATUS  GetCamSequentialShutterEnable (
  CAM_HANDLE        hCam,
  bool8_t           *pbEnable
);
```

[Parameters]

Parameter		Description
hCam	[in]	Camera-Handle of target camera.
pbEnable	[out]	A pointer to a variable that receives the enable state of Sequential Shutter function If false, the Sequential Shutter function is disable. If true, the Sequential Shutter function is enable.

[Return value]

Returns result status. Refer to 5.8 Status code.

[Remarks]

This function will return error status if SequentialShutterEnable register (or node) is not implemented in the camera.

Including TeliCamAPI.h is required.

5.5.26.2. SetCamSequentialShutterEnable

Sets the enable state of Sequential Shutter function in the camera.

[Syntax]

```
CAM_API_STATUS  SetCamSequentialShutterEnable (
  CAM_HANDLE        hCam,
  bool8_t           bEnable
);
```

[Parameters]

Parameter		Description
hCam	[in]	Camera-Handle of target camera.
bEnable	[in]	The enable state of Sequential Shutter function If false, the Sequential Shutter function is disable. If true, the Sequential Shutter function is enable.

[Return value]

Returns result status. Refer to 5.8 Status code.

[Remarks]

This function will return error status if SequentialShutterEnable register (or node) is not implemented in the camera.

Including TeliCamAPI.h is required.

5.5.26.3. GetCamSequentialShutterTerminateAtMinMax

Gets the minimum and maximum number of index to repeat the sequence.

[Syntax]

```
CAM_API_STATUS  GetCamSequentialShutterTerminateAtMinMax (
  CAM_HANDLE        hCam,
  uint32_t          *puiMin,
  uint32_t          *puiMax
);
```

[Parameters]

Parameter		Description
hCam	[in]	Camera-Handle of target camera.
puiMin	[out]	A pointer to a variable that receives the minimum number of index to repeat the sequence.
puiMax	[out]	A pointer to a variable that receives the maximum number of index to repeat the sequence.

[Return value]

Returns result status. Refer to 5.8 Status code.

[Remarks]

This function will return error status if SequentialShutterTerminateAt register (or node) is not implemented in the camera.
Including TeliCamAPI.h is required.

5.5.26.4. GetCamSequentialShutterTerminateAt

Gets the number of index to repeat the sequence.

[Syntax]

```
CAM_API_STATUS  GetCamSequentialShutterTerminateAt (
  CAM_HANDLE        hCam,
  uint32_t          *puiValue
);
```

[Parameters]

Parameter		Description
hCam	[in]	Camera-Handle of target camera.
puiValue	[out]	A pointer to a variable that receives the number of index to repeat the sequence.

[Return value]

Returns result status. Refer to 5.8 Status code.

[Remarks]

This function will return error status if SequentialShutterTerminateAt register (or node) is not implemented in the camera.

Including TeliCamAPI.h is required.

5.5.26.5. SetCamSequentialShutterTerminateAt

Sets the number of index to repeat the sequence.

[Syntax]

```
CAM_API_STATUS  SetCamSequentialShutterTerminateAt (
  CAM_HANDLE        hCam,
  uint32_t          uiValue
);
```

[Parameters]

Parameter		Description
hCam	[in]	Camera-Handle of target camera.
uiValue	[in]	The number of index.

[Return value]

Returns result status. Refer to 5.8 Status code.

[Remarks]

This function will return error status if SequentialShutterTerminateAt register (or node) is not implemented in the camera.

Including TeliCamAPI.h is required.

5.5.26.6. GetCamSequentialShutterIndexMinMax

Gets the minimum and maximum value of sequence number of sequential shutter function.

[Syntax]

```
CAM_API_STATUS  GetCamSequentialShutterIndexMinMax (
  CAM_HANDLE        hCam,
  uint32_t          *puiMin,
  uint32_t          *puiMax
);
```

[Parameters]

Parameter		Description
hCam	[in]	Camera-Handle of target camera.
puiMin	[out]	A pointer to a variable that receives the minimum value of sequence number.
puiMax	[out]	A pointer to a variable that receives the maximum value of sequence number.

[Return value]

Returns result status. Refer to 5.8 Status code.

[Remarks]

This function will return error status if SequentialShutterSequenceTable or SequentialShutterIndex register (or node) is not implemented in the camera.

Including TeliCamAPI.h is required.

5.5.26.7. GetCamSequentialShutterEntryMinMax

Gets the minimum and maximum value of UserSet number of sequential shutter function.

[Syntax]

```
CAM_API_STATUS  GetCamSequentialShutterEntryMinMax (
  CAM_HANDLE        hCam,
  uint32_t          *puiMin,
  uint32_t          *puiMax
);
```

[Parameters]

Parameter		Description
hCam	[in]	Camera-Handle of target camera.
puiMin	[out]	A pointer to a variable that receives the minimum value of UserSet number
puiMax	[out]	A pointer to a variable that receives the minimum value of UserSet number

[Return value]

Returns result status. Refer to 5.8 Status code.

[Remarks]

This function will return error status if SequentialShutterSequenceTable or SequentialShutterEntry register (or node) is not implemented in the camera.

Including TeliCamAPI.h is required.

5.5.26.8. GetCamSequentialShutterEntry

Gets the value of UserSet number of sequential shutter function.

[Syntax]

```
CAM_API_STATUS  GetCamSequentialShutterEntry (
  CAM_HANDLE        hCam,
  uint32_t          uiIndex,
  uint32_t          *puiEntry
);
```

[Parameters]

Parameter		Description
hCam	[in]	Camera-Handle of target camera.
uiIndex	[in]	Target sequence number. (SequentialShutterIndex)
puiEntry	[out]	A pointer to a variable that receives the value of UserSet number

[Return value]

Returns result status. Refer to 5.8 Status code.

[Remarks]

This function will return error status if SequentialShutterSequenceTable or SequentialShutterEntry register (or node) is not implemented in the camera.
Including TeliCamAPI.h is required.

5.5.26.9. SetCamSequentialShutterEntry

Sets the value of UserSet number of sequential shutter function.

[Syntax]

```
CAM_API_STATUS  SetCamSequentialShutterEntry (
  CAM_HANDLE        hCam,
  uint32_t          uiIndex,
  uint32_t          uiEntry
);
```

[Parameters]

Parameter		Description
hCam	[in]	Camera-Handle of target camera.
uiIndex	[in]	Target sequence number. (SequentialShutterIndex)
uiEntry	[in]	The value of UserSet number.

[Return value]

Returns result status. Refer to 5.8 Status code.

[Remarks]

This function will return error status if SequentialShutterSequenceTable or SequentialShutterEntry register (or node) is not implemented in the camera.

Including TeliCamAPI.h is required.

5.5.27.UserDefinedName (DeviceUserID)

This feature group controls user-defined name (“UserDefinedName” or “DeviceUserID” register) of the camera.

User application can set any string to “UserDefinedName” or “DeviceUserID” register for identifying it. The register value is stored in non0volatile memory of the camera,

For details about UserDefinedName features, refer to instruction manual of the camera.

5.5.27.1. GetCamUserDefinedName

Gets the user-defined name of the camera.

[Syntax]

```
CAM_API_STATUS  GetCamUserDefinedName (
  CAM_HANDLE        hCam,
  char              *pszName,
  uint32_t          *puiSize
);
```

[Parameters]

Parameter		Description
hCam	[in]	Camera-Handle of target camera.
pszName	[out]	A pointer to a buffer that receives the user-defined name. If NULL is specified to pszName, the necessary buffer size will be written to the variable pointed by puiSize.
puiSize	[in,out]	A pointer to a variable that contains size of buffer pointed by pszName, in bytes. The length of user-defined name actually written to pszName will be written to this variable. If NULL is specified to pszName, the necessary buffer size will be written to this variable.

[Return value]

Returns result status. Refer to 5.8 Status code.

[Remarks]

This function will return error status if UserDefinedName or DeviceUserID register (or node) is not implemented in the camera.

If user-defined name registered in the camera is not NULL-terminated, the last character may be replaced with NULL.

When user application sets the new user-defined name, the old user-defined name may remain until “Sys_GetNumOfCameras()” is called.

Including TeliCamAPI.h is required.

5.5.27.2. SetCamUserDefinedName

Sets the user-defined name of the camera.

[Syntax]

```
CAM_API_STATUS  SetCamUserDefinedName (
  CAM_HANDLE        hCam,
  char              *pszName,
);
```

[Parameters]

Parameter		Description
hCam	[in]	Camera-Handle of target camera.
pszName	[in]	A pointer to NULL-terminated user-defined name.

[Return value]

Returns result status. Refer to 5.8 Status code.

[Remarks]

This function will return error status if UserDefinedName or DeviceUserID register (or node) is not implemented in the camera.

The length of user-defined name register in GigE Vision camera is 16 bytes, that in USB3 Vision camera is 64 bytes.

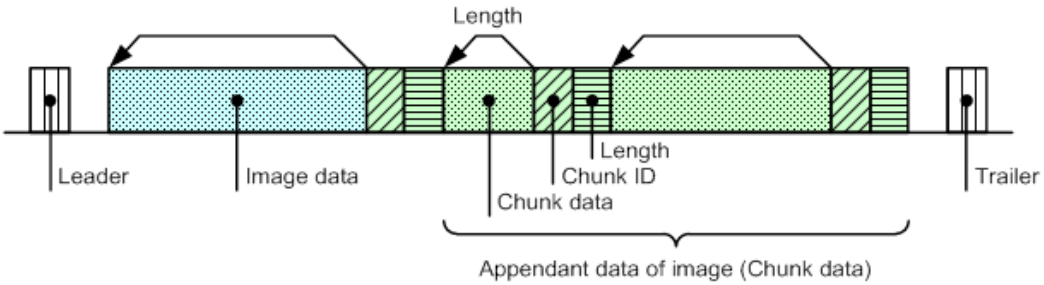
Note that NULL character for termination is counted as one character.

Including TeliCamAPI.h is required.

5.5.28.Chunk

This feature group performs control of Chunk feature.

Chunk data means tagged blocks of data.
The tags allow a chunk parser to dissect the data payload into its elements and to identify the content.
The length of a frame varies depending on the number of activated chunks.



For details about Chunk features, refer to instruction manual of the camera.

When using GenICam to acquire chunk data, it is necessary to attach the buffer storing the payload data to GenICam chunk adaptor.
For details, refer to the description of the Chunk_AttachBuffer () function.

5.5.28.1. GetCamChunkModeActive

Gets the activation state of Chunk function in the camera.

[Syntax]

```
CAM_API_STATUS  GetCamChunkModeActive (
  CAM_HANDLE        hCam,
  bool8_t           *pbValue
);
```

[Parameters]

Parameter		Description
hCam	[in]	Camera-Handle of target camera.
pbValue	[out]	A pointer to a variable that receives the activation state of Chunk function. If false, the Chunk function is inactive. If true, the Chunk function is active.

[Return value]

Returns result status. Refer to 5.8 Status code.

[Remarks]

This function will return error status if ChunkModeActive register (or node) is not implemented in the camera.

Including TeliCamAPI.h is required.

5.5.28.2. SetCamChunkModeActive

Sets the activation value of Chunk function in the camera.

[Syntax]

```
CAM_API_STATUS  SetCamChunkModeActive (
  CAM_HANDLE        hCam,
  bool8_t           bValue
);
```

[Parameters]

Parameter		Description
hCam	[in]	Camera-Handle of target camera.
bValue	[in]	The activation value of Chunk function. If false, the Chunk function is inactive. If true, the Chunk function is active.

[Return value]

Returns result status. Refer to 5.8 Status code.

[Remarks]

This function will return error status if ChunkModeActive register (or node) is not implemented in the camera.

Including TeliCamAPI.h is required.

5.5.28.3. GetCamChunkEnable

Gets the enable state of Chunk data in the camera.

[Syntax]

```
CAM_API_STATUS  GetCamChunkEnable (
  CAM_HANDLE                hCam,
  CAM_CHUNK_SELECTOR_TYPE   eSelector,
  bool8_t                   *pbEnable
);
```

[Parameters]

Parameter		Description
hCam	[in]	Camera-Handle of target camera.
eSelector	[in]	Target element of Chunk data.
pbValue	[out]	A pointer to a variable that receives the enable state of the selected Chunk data. If false, the selected Chunk data is disable. If ture, the selected Chunk data is enable.

[*CAM_CHUNK_SELECTOR_TYPE* Enumeration]

Member	Description
CAM_CAM_CHUNK_SELECTOR_BLOCK_ID	BlockID
CAM_CAM_CHUNK_SELECTOR_FRAME_BURST_TRIGGER_COUNT	FrameBurstTriggerCount
CAM_CAM_CHUNK_SELECTOR_SEQUENTIAL_SHUTTER_NUMBER	SequentialShutter number
CAM_CAM_CHUNK_SELECTOR_SEQUENTIAL_SHUTTER_ELEMENT	SequentialShutter element
CAM_CAM_CHUNK_SELECTOR_USER_AREA	User area
CAM_CAM_CHUNK_SELECTOR_EXPOSURE_TIME	Exposure time
CAM_CAM_CHUNK_SELECTOR_GAIN	Gain
CAM_CAM_CHUNK_SELECTOR_WHITE_BALANCE_R	WhiteBalanceR
CAM_CAM_CHUNK_SELECTOR_WHITE_BALANCE_B	WhiteBalanceB
CAM_CAM_CHUNK_SELECTOR_LINE_STATUS_ALL	LineStatusAll

[Return value]

Returns result status. Refer to 5.8 Status code.

[Remarks]

This function will return error status if ChunkEnableOf*, or ChunkSelector and ChunkEnable registers (or nodes) are not implemented in the camera.

Including TeliCamAPI.h is required.

5.5.28.4. SetCamChunkEnable

Sets the enable state of Chunk data in the camera.

[Syntax]

```
CAM_API_STATUS  SetCamChunkEnable (
  CAM_HANDLE                hCam,
  CAM_CHUNK_SELECTOR_TYPE   eSelector,
  bool8_t                   bEnable
);
```

[Parameters]

Parameter		Description
hCam	[in]	Camera-Handle of target camera.
eSelector	[in]	Target element of Chunk data.
bValue	[in]	The enable state of the selected chunk data. If false, the selected Chunk data is disable. If ture, the selected Chunk data is enable.

[Return value]

Returns result status. Refer to 5.8 Status code.

[Remarks]

This function will return error status if ChunkEnableOf*, or ChunkSelector and ChunkEnable registers (or nodes) are not implemented in the camera.

Including TeliCamAPI.h is required.

5.5.28.5. GetCamChunkUserAreaLength

Gets the length of ChunkUserAreaTable in the camera.

[Syntax]

```
CAM_API_STATUS  GetCamChunkUserAreaLength (
  CAM_HANDLE        hCam,
  uint32_t          *puiLength
);
```

[Parameters]

Parameter		Description
hCam	[in]	Camera-Handle of target camera.
puiLength	[out]	A pointer to a variable that receives the length of ChunkUserAreaTable, in byte(s).

[Return value]

Returns result status. Refer to 5.8 Status code.

[Remarks]

This function will return error status if ChunkUserArea or ChunkUserAreaTable register (or node) is not implemented in the camera.

Including TeliCamAPI.h is required.

5.5.28.6. GetCamChunkUserAreaTable

Gets the data of ChunkUserAreaTable in the camera.

[Syntax]

```
CAM_API_STATUS  GetCamChunkUserAreaTable (
  CAM_HANDLE        hCam,
  char              *pvBuffer,
  uint32_t          uiOffset,
  uint32_t          uiSize
);
```

[Parameters]

Parameter		Description
hCam	[in]	Camera-Handle of target camera.
pvBuffer	[out]	A pointer to an array that receives the data of ChunkUserAreaTable.
uiOffset	[in]	The offset value of ChunkUserArea, in byte(s). Set "0" when acquiring data from the beginning of ChunkUserAreaTable.
puiSize	[in]	The size of the data to be acquired, in byte(s). Do not set a value larger than the size of the buffer specified by pvBuffer.

[Return value]

Returns result status. Refer to 5.8 Status code.

[Remarks]

This function will return error status if ChunkUserArea or ChunkUserAreaTable register (or node) is not implemented in the camera.

Including TeliCamAPI.h is required.

5.5.28.7. SetCamChunkUserAreaTable

Sets the data of ChunkUserAreaTable in the camera.

[Syntax]

```
CAM_API_STATUS  SetCamChunkUserAreaTable (
  CAM_HANDLE        hCam,
  char              *pvBuffer,
  uint32_t          uiOffset,
  uint32_t          uiSize
);
```

[Parameters]

Parameter		Description
hCam	[in]	Camera-Handle of target camera.
pvBuffer	[in]	A pointer to an array to be set.
uiOffset	[in]	The offset value of ChunkUserArea, in byte(s). Set "0" when setting data from the beginning of ChunkUserArea.
puiSize	[in]	The size of the data to be acquired, in byte(s). Do not set a value larger than the size of the buffer specified by pvBuffer.

[Return value]

Returns result status. Refer to 5.8 Status code.

[Remarks]

This function will return error status if ChunkUserArea or ChunkUserAreaTable register (or node) is not implemented in the camera.

Including TeliCamAPI.h is required.

5.5.29.FrameSynchronization

This feature group performs control of frame synchronization method of the camera.

For details about FrameSynchronization features, refer to instruction manual of the camera.

5.5.29.1. GetCamFrameSynchronization

Gets the camera frame synchronization method.

[Syntax]

```
CAM_API_STATUS  GetCamFrameSynchronization (
  CAM_HANDLE                    hCam,
  CAM_FRAME_SYNCHRONIZATION_TYPE   *peSync
);
```

[Parameters]

Parameter		Description
hCam	[in]	Camera-Handle of target camera.
pbValue	[out]	A pointer to a variable that receives the type of the camera frame synchronization method.

[*CAM_FRAME_SYNCHRONIZATION_TYPE* Enumeration]

メンバ	内容
CAM_FRAME_SYNCHRONIZATION_OFF	Internal synchronization
CAM_FRAME_SYNCHRONIZATION_BUS	Bus synchronization

[Return value]

Returns result status. Refer to 5.8 Status code.

[Remarks]

This function will return error status if FrameSynchronization register (or node) is not implemented in the camera.

Including TeliCamAPI.h is required.

5.5.29.2. SetCamFrameSynchronization

Sets the camera frame synchronization method.

[Syntax]

```
CAM_API_STATUS  SetCamFrameSynchronization (
  CAM_HANDLE                    hCam,
  CAM_FRAME_SYNCHRONIZATION_TYPE   eSync
);
```

[Parameters]

Parameter		Description
hCam	[in]	Camera-Handle of target camera.
eSync	[in]	The type of the camera frame synchronization method.

[Return value]

Returns result status. Refer to 5.8 Status code.

[Remarks]

This function will return error status if FrameSynchronization register (or node) is not implemented in the camera.

Including TeliCamAPI.h is required.

5.5.30.Other functions

5.5.30.1. GetCamIndexFromHandle

This function reports index of camera which has specified Camera-Handle.
Index value of the timing when the camera was opened will be reported.

[Syntax]

```
CAM_API_STATUS  GetCamIndexFromHandle (
  CAM_HANDLE        hCam,
  uint32_t          *puiCamIndex
);
```

[Parameters]

Parameter		Description
hCam	[in]	Camera-Handle of target camera.
puiCamIndex	[out]	A pointer to a variable that receives index of the camera.

[Return value]

Returns result status. Refer to 5.8 Status code.

[Remarks]

If "Sys_GetNumOfCameras()" is called after the target camera was opened, re-numbering of camera index is performed, which may change camera index value assignment if cameras are disconnected or new cameras are connected.

TeliCamAPI saves camera index when the camera is opened. This function reports the saved camera index.

Including TeliCamAPI.h is required.

5.5.30.2. GetCamTypeFromCamHandle

This function reports interface type of the camera whose Camera-Handle is same as the argument value.

[Syntax]

```
CAM_API_STATUS  GetCamTypeFromCamHandle (
  CAM_HANDLE        hCam,
  CAM_TYPE          *peType
);
```

[Parameters]

Parameter		Description
hCam	[in]	Camera-Handle of target camera.
peType	[out]	A pointer to a variable that receives interface type of the camera.

[Return value]

Returns result status. Refer to 5.8 Status code.

[Remarks]

Refer to TeliCamAPI.h about CAM_TYPE.

Including TeliCamAPI.h is required.

5.5.30.3. GetCamSupportIIDC2

This function reports whether the camera complies with IIDC2 standard or not.

[Syntax]

```
CAM_API_STATUS  GetCamSupportIIDC2 (
  CAM_HANDLE        hCam,
  bool8_t           *pbValue
);
```

[Parameters]

Parameter		Description
hCam	[in]	Camera-Handle of target camera.
pbValue	[out]	A pointer to a variable that receives value. If the value is true, the camera complies with IIDC2 standard. If the value is false, the camera does not comply with IIDC2 standard.

[Return value]

Returns result status. Refer to 5.8 Status code.

[Remarks]

Including TeliCamAPI.h is required.

5.5.30.4. GetCamTLParamsLocked

This function reports value of TLParamsLocked node.

Variable TLParamsLocked is used for protecting critical transport layer function register from changing value during streaming is active,

When streaming start function of TeliCanAPI is called, TeliCamAPI write 1 to TLParamsLocked.

When streaming stop function of TeliCamAPI is called, TeliCamAPI writes0 to TLParamsLocked.

During TLParamsLocked is 1, the camera and / or GenAPI system rejects writing data to critical transport layer function register.

[Syntax]

```
CAM_API_STATUS  GetCamTLParamsLocked (
  CAM_HANDLE        hCam,
  uint32_t          *puiValue
);
```

[Parameters]

Parameter		Description
hCam	[in]	Camera-Handle of target camera.
puiValue	[out]	A pointer to a variable that receives current TLParamsLocked value.

[Return value]

Returns result status. Refer to 5.8 Status code.

[Remarks]

TeliCamAPI controls TLParamsLocked value inside it if user application used functions in section 5.3 for controlling stream. In this case, user application can forget controlling TLParamsLocked.

If user application writes new value to camera registers protected by TLParamsLocked using "Cam_WriteReg()", the function may receive success status. Actually, internal writing action may be skipped or writing action may be reserved until TLParamsLocked is cleared.

Including TeliCamAPI.h is required.

5.6. GenICam functions

Functions in this section wrapper functions of GenICam GenApi, which allow user application to access various information of camera register easier.

Refer to http://www.genicam.org about detail information of GenICam.

5.6.1. INode functions

5.6.1.1. GenApi_GetType

This function reports node type of the specified node name.

[Syntax]

```
CAM_API_STATUS  GenApi_GetType (
  CAM_HANDLE        hCam,
  const char        *pszNodeName,
  TC_NODE_TYPE      *peNodeType
);
```

[Parameters]

Parameter		Description
hCam	[in]	Camera-Handle of target camera.
pszNodeName	[in]	NULL terminated node name.
peNodeType	[out]	A pointer to a variable that receives node type of the specified node.

[*TC_NODE_TYPE* Enumeration]

Member	Description
TC_NODE_TYPE_VALUE	IValue interface
TC_NODE_TYPE_BASE	IBase interface
TC_NODE_TYPE_INTEGER	IInteger interface
TC_NODE_TYPE_BOOLEAN	IBoolean interface
TC_NODE_TYPE_COMMAND	ICommand interface
TC_NODE_TYPE_FLOAT	IFloat interface
TC_NODE_TYPE_STRING	IString interface
TC_NODE_TYPE_REGISTER	IRegister interface
TC_NODE_TYPE_CATEGORY	ICategory interface
TC_NODE_TYPE_ENUMERATION	IEnumeration interface
TC_NODE_TYPE_ENUM_ENTRY	IEnumEntry interface
TC_NODE_TYPE_PORT	IPort interface
TC_NODE_TYPE_UNKNOWN	Unknown interface

[Return value]

Returns result status. Refer to 5.8 Status code.

[Remarks]

TC_NODE_TYPE is declared in TeliCamNode.h.

Enum type “EInterfaceType” declared in Types.h of GenApi reference implementation corresponds to

TC_NODE_TYPE.

Including TeliCamAPI.h is required.

[Example]

C++

```
CAM_API_STATUS            uiStatus;
uint32_t                  uiNum, uiSize;
CAM_HANDLE                hCam = (CAM_HANDLE)NULL;
TC_NODE_TYPE              eNodeType = TC_NODE_TYPE_UNKNOWN;
TC_NODE_ACCESS_MODE       eNodeAccessMode = TC_NODE_ACCESS_MODE_UNKNOWN;
TC_NODE_VISIBILITY        eNodeVisibility = TC_NODE_VISIBILITY_UNKNOWN;
TC_NODE_CACHING_MODE      eNodeCachingMode = TC_NODE_CACHING_MODE_UNKNOWN;
TC_NODE_REPRESENTATION    nodeRepresentation = TC_NODE_REPRESENTATION_UNKNOWN;
char                      *pszBuf = NULL;

// Initialize system.
uiStatus = Sys_Initialize();
if (uiStatus != CAM_API_STS_SUCCESS)
    return -1;

// Get number of cameras.
uiStatus = Sys_GetNumOfCameras(&uiNum);
if ((uiStatus != CAM_API_STS_SUCCESS) || (uiNum == 0))
    return -1;

// Open camera that is detected first, in this sample code.
uiStatus = Cam_Open(0, &hCam, NULL);
if (uiStatus != CAM_API_STS_SUCCESS)
    return -1;

__try
{
    // Get type.
    uiStatus = GenApi_GetType(hCam, "Gain", &eNodeType);
    if (uiStatus != CAM_API_STS_SUCCESS)
        return -1;
    printf("Type = %d\n", (uint32_t)eNodeType);

    // Get access mode.
    uiStatus = GenApi_GetAccessMode(hCam, "Gain", &eNodeAccessMode);
    if (uiStatus != CAM_API_STS_SUCCESS)
        return -1;
    printf("AccessMode = %d\n", (uint32_t)eNodeAccessMode);

    // Get visibility.
    uiStatus = GenApi_GetVisibility(hCam, "Gain", &eNodeVisibility);
    if (uiStatus != CAM_API_STS_SUCCESS)
        return -1;
    printf("Visibility = %d\n", (uint32_t)eNodeVisibility);

    // Get cachingMode.
    uiStatus = GenApi_GetCachingMode(hCam, "Gain", &eNodeCachingMode);
    if (uiStatus != CAM_API_STS_SUCCESS)
        return -1;
    printf("CachingMode = %d\n", (uint32_t)eNodeCachingMode);

    // Get description.
    uiSize = 0;
    uiStatus = GenApi_GetDescription(hCam, "Gain", NULL, &uiSize);
    if (uiStatus != CAM_API_STS_SUCCESS)
        return -1;

    // Allocate buffer for description data.
```

```
    pszBuf = (char *)VirtualAlloc(NULL,
                                  uiSize,
                                  MEM_RESERVE | MEM_COMMIT,
                                  PAGE_EXECUTE_READWRITE);
    if (pszBuf == NULL) {
        return -1;
    }

    uiStatus = GenApi_GetDescription(hCam, "Gain", pszBuf, &uiSize);
    if (uiStatus != CAM_API_STS_SUCCESS)
        return -1;
    printf("Description = %s\n", pszBuf);

    // Release buffer for description data.
    VirtualFree(pszBuf, 0, MEM_RELEASE);
    pszBuf = NULL;

    // Get tooltip.
    uiSize = 0;
    uiStatus = GenApi_GetToolTip(hCam, "Gain", NULL, &uiSize);
    if (uiStatus != CAM_API_STS_SUCCESS)
        return -1;

    // Allocate buffer for tooltip data.
    pszBuf = (char *)VirtualAlloc(NULL,
                                  uiSize,
                                  MEM_RESERVE | MEM_COMMIT,
                                  PAGE_EXECUTE_READWRITE);
    if (pszBuf == NULL) {
        return -1;
    }

    uiStatus = GenApi_GetToolTip(hCam, "Gain", pszBuf, &uiSize);
    if (uiStatus != CAM_API_STS_SUCCESS)
        return -1;

    printf("ToolTip = %s\n", pszBuf);

    // Release buffer for tooltip data.
    VirtualFree(pszBuf, 0, MEM_RELEASE);
    pszBuf = NULL;

    // Get representation.
    uiStatus = GenApi_GetRepresentation(hCam, "Gain", &nodeRepresentation);
    if (uiStatus != CAM_API_STS_SUCCESS)
        return -1;
    printf("Representation = %d\n", (uint32_t)nodeRepresentation);

    // Get unit.
    uiSize = 0;
    uiStatus = GenApi_GetUnit(hCam, "Gain", NULL, &uiSize);
    if (uiStatus != CAM_API_STS_SUCCESS)
        return -1;

    // Allocate buffer for unit data.
    pszBuf = (char *)VirtualAlloc(NULL,
                                  uiSize,
                                  MEM_RESERVE | MEM_COMMIT,
                                  PAGE_EXECUTE_READWRITE);
    if (pszBuf == NULL) {
        return -1;
    }

    uiStatus = GenApi_GetUnit(hCam, "Gain", pszBuf, &uiSize);
    if (uiStatus != CAM_API_STS_SUCCESS)
        return -1;
```

```
    printf("Unit = %s\n", pszBuf);

    // Release buffer for unit data.
    VirtualFree(pszBuf, 0, MEM_RELEASE);
    pszBuf = NULL;

    return 0;
}
__finally
{
    if (pszBuf != NULL) {
        free( pszBuf);
    }

    // Close camera.
    if (hCam != (CAM_HANDLE)NULL) {
        uiStatus = Cam_Close(hCam);
        if (uiStatus != CAM_API_STS_SUCCESS)
            printf("Cam_Close error! (0x%x)", uiStatus);
    }

    // Terminate system.
    Sys_Terminate();
}
```

5.6.1.2. GenApi_GetAccessMode

This function reports access mode of the specified node name.

[Syntax]

```
CAM_API_STATUS  GenApi_GetAccessMode (
  CAM_HANDLE            hCam,
  const char            *pszNodeName,
  TC_NODE_ACCESS_MODE   *peAccessMode
);
```

[Parameters]

Parameter		Description
hCam	[in]	Camera-Handle of target camera.
pszNodeName	[in]	NULL terminated node name.
peAccessMode	[out]	A pointer to a variable that receives current access mode of the node.

[*TC_NODE_ACCESS_MODE* Enumeration]

Member	Description
TC_NODE_ACCESS_MODE_NI	Not implemented
TC_NODE_ACCESS_MODE_NA	Not available
TC_NODE_ACCESS_MODE_WO	Write Only
TC_NODE_ACCESS_MODE_RO	Read Only
TC_NODE_ACCESS_MODE_RW	Read and Write
TC_NODE_ACCESS_MODE_Undefined	Object is not yet initialized
TC_NODE_ACCESS_MODE_UNKNOWN	Unknown mode

[Return value]

Returns result status. Refer to 5.8 Status code.

[Remarks]

TC_NODE_ACCESS_MODE is declared in TeliCamNode.h.

Enum type “EAccessMode” declared in Types.h of GenApi reference implementation corresponds to TC_NODE_ACCESS_MODE.

Including TeliCamAPI.h is required.

[Example]

Refer to example of 5.6.1.1 GenApi_GetType.

5.6.1.3. GenApi_GetVisibility

This functio reports visibility of the specified node name.

[Syntax]

```
CAM_API_STATUS  GenApi_GetVisibility (
  CAM_HANDLE            hCam,
  const char            *pszNodeName,
  TC_NODE_VISIBILITY    *peVisibility
);
```

[Parameters]

Parameter		Description
hCam	[in]	Camera-Handle of target camera.
pszNodeName	[in]	NULL terminated node name.
peVisibility	[out]	A pointer to a variable that receives visibility of the node.

[*TC_NODE_VISIBILITY* Enumeration]

Member	Description
TC_NODE_VISIBILITY_BEGINNER	Always visible
TC_NODE_VISIBILITY_EXPERT	Visible for experts or Gurus
TC_NODE_VISIBILITY_GURU	Visible for Gurus
TC_NODE_VISIBILITY_INVISIBLE	Not Visible
TC_NODE_VISIBILITY_Undefined	Object is not yet initialized
TC_NODE_VISIBILITY_UNKNOWN	Unknown

[Return value]

Returns result status. Refer to 5.8 Status code.

[Remarks]

TC_NODE_VISIBILITY is declared in TeliCamNode.h.

Enum type “EVisibility” declared in Types.h of GenApi reference implementation corresponds to TC_NODE_VISIBILITY.

Including TeliCamAPI.h is required.

[Example]

Refer to example of 5.6.1.1 GenApi_GetType.

5.6.1.4. GenApi_GetCachingMode

This function reports caching mode of the specified node name.

[Syntax]

```
CAM_API_STATUS  GenApi_GetCachingMode (
  CAM_HANDLE            hCam,
  const char            *pszNodeName,
  TC_NODE_CACHING_MODE  *peCachingMode
);
```

[Parameters]

Parameter		Description
hCam	[in]	Camera-Handle of target camera.
pszNodeName	[in]	NULL terminated node name.
peCachingMode	[out]	A pointer to a variable that receives caching mode of the node.

[*TC_NODE_CACHING_MODE* Enumeration]

Member	Description
TC_NODE_CACHING_MODE_NO_CACHE	Do not use cache.
TC_NODE_CACHING_MODE_WRITE_THROUGH	Write to cache and register.
TC_NODE_CACHING_MODE_WRITE_AROUND	Write to register, write to cache on read.
TC_NODE_CACHING_MODE_UNDEFINED	Not yet initialized.
TC_NODE_CACHING_MODE_UNKNOWN	Unknown.

[Return value]

Returns result status. Refer to 5.8 Status code.

[Remarks]

TC_NODE_CACHING_MODE is declared in TeliCamNode.h. Enum type “ECachingMode” declared in Types.h of GenApi reference implementation corresponds to TC_NODE_CACHING_MODE.

Including TeliCamAPI.h is required.

[Example]

Refer to example of 5.6.1.1 GenApi_GetType.

5.6.1.5. GenApi_GetDescription

This function reports description of the specified node name.

[Syntax]

```
CAM_API_STATUS  GenApi_GetDescription (
  CAM_HANDLE            hCam,
  const char            *pszNodeName,
  char                  *pszBuf,
  uint32_t              *puiSize
);
```

[Parameters]

Parameter		Description
hCam	[in]	Camera-Handle of target camera.
pszNodeName	[in]	NULL terminated node name.
pszBuf	[out]	A pointer to a variable that receives description of the node. If NULL is specified to this argument, this function will write buffer size necessary for writing description of the node to variable pointed by puiSize, without reporting description string.
puiSize	[in, out]	A pointer to a variable that contains size of buffer pointed by pszBuf If NULL is specified to pszBuf, this function will write buffer size necessary for writing description of the node to variable pointed by this argument.

[Return value]

Returns result status. Refer to 5.8 Status code.

[Remarks]

Including TeliCamAPI.h is required.

[Example]

Refer to example of 5.6.1.1 GenApi_GetType.

5.6.1.6. GenApi_GetToolTip

This function reports tool tip of the specified node name.

[Syntax]

```
CAM_API_STATUS  GenApi_GetToolTip (
  CAM_HANDLE          hCam,
  const char          *pszNodeName,
  char                *pszBuf,
  uint32_t            *puiSize
);
```

[Parameters]

Parameter		Description
hCam	[in]	Camera-Handle of target camera.
pszNodeName	[in]	NULL terminated node name.
pszBuf	[out]	A pointer to a variable that receives tool tip of the node. If NULL is specified to this argument, this function will write buffer size necessary for writing tool tip of the node to variable pointed by `puiSize`, without reporting tool tip string.
puiSize	[in, out]	A pointer to a variable that contains size of buffer pointed by `pszBuf`. If NULL is specified to `pszBuf`, this function will write buffer size necessary for writing tool tip of the node to variable pointed by this argument.

[Return value]

Returns result status. Refer to 5.8 Status code.

[Remarks]

Including TeliCamAPI.h is required.

[Example]

Refer to example of 5.6.1.1 GenApi_GetType.

5.6.1.7. GenApi_GetRepresentation

This function reports recommended representation of the specified node name.

[Syntax]

```
CAM_API_STATUS  GenApi_GetRepresentation (
  CAM_HANDLE            hCam,
  const char            *pszNodeName,
  TC_NODE_REPRESENTATION   *peRepresentation
);
```

[Parameters]

Parameter		Description
hCam	[in]	Camera-Handle of target camera.
pszNodeName	[in]	NULL terminated node name.
peRepresentation	[out]	A pointer to a variable that receives recommended representation of the node.

[*TC_NODE_REPRESENTATION* Enumeration]

Member	Description
TC_NODE_REPRESENTATION_LINEAR	Slider with linear behavior
TC_NODE_REPRESENTATION_LOGARITHMIC	Slider with logarithmic behaviour
TC_NODE_REPRESENTATION_BOOLEAN	Check box
TC_NODE_REPRESENTATION_PURE_NUMBER	Decimal number in an edit control
TC_NODE_REPRESENTATION_HEX_NUMBER	Hex number in an edit control
TC_NODE_REPRESENTATION_IPV4_ADDRESS	IPV4-Address
TC_NODE_REPRESENTATION_MAC_ADDRESS	MAC Address
TC_NODE_REPRESENTATION_UNDEFINED	Not yet initialized
TC_NODE_REPRESENTATION_UNKNOWN	Unknown

[Return value]

Returns result status. Refer to 5.8 Status code.

[Remarks]

TC_NODE_REPRESENTATION is declared in TeliCamNode.h. Enum type “ERepresentation” declared in Types.h of GenApi reference implementation corresponds to TC_NODE_REPRESENTATION.

Including TeliCamAPI.h is required.

[Example]

Refer to example of 5.6.1.1 GenApi_GetType.

5.6.1.8. GenApi_GetUnit

This function reports unit name of the specified node name.

[Syntax]

```
CAM_API_STATUS  GenApi_GetUnit (
  CAM_HANDLE          hCam,
  const char          *pszNodeName,
  char                *pszBuf,
  uint32_t            *puiSize
);
```

[Parameters]

Parameter		Description
hCam	[in]	Camera-Handle of target camera.
pszNodeName	[in]	NULL terminated node name.
pszBuf	[out]	A pointer to a variable that receives unit name of target node value. If NULL is specified to this argumen , this function will write buffer size necessary for writing unit name of the node to variable pointed by puiSize, without reporting unit name string.
puiSize	[in, out]	A pointer to a variable that contains size of buffer pointed by pszBuf. If NULL is specified to pszBuf, this function will write buffer size necessary for writing unit name of the node to variable pointed by this argument.

[Return value]

Returns result status. Refer to 5.8 Status code.

[Remarks]

Including TeliCamAPI.h is required.

[Example]

Refer to example of 5.6.1.1 GenApi_GetType.

5.6.2. ICategory node functions

5.6.2.1. GenApi_GetNumOfFeatures

This function reports number of features that the specified category node contains.

[Syntax]

```
CAM_API_STATUS  GenApi_GetNumOfFeatures (
  CAM_HANDLE        hCam,
  const char        *pszCategoryName,
  uint32_t          *puiNum
);
```

[Parameters]

Parameter		Description
hCam	[in]	Camera-Handle of target camera.
pszCategoryName	[in]	NULL terminated category name.
puiNum	[out]	A pointer to a variable that receives number of features.

[Return value]

Returns result status. Refer to 5.8 Status code.

[Remarks]

This function is available only for ICategory type node (`TC_NODE_TYPE_CATEGORY`).
This function will return error status if node type of target node is not ICategory type.
Including TeliCamAPI.h is required.

[Example]

C++

```
CAM_API_STATUS   uiStatus;
uint32_t         uiNum, uiFeatureNum, i;
CAM_HANDLE       hCam = (CAM_HANDLE)NULL;
CAM_NODE_NAME    sFeatureName;

// Initialize system.
uiStatus = Sys_Initialize();
if (uiStatus != CAM_API_STS_SUCCESS)
    return -1;

// Get number of cameras.
uiStatus = Sys_GetNumOfCameras(&uiNum);
if ((uiStatus != CAM_API_STS_SUCCESS) || (uiNum == 0))
    return -1;

// Open camera that is detected first, in this sample code.
uiStatus = Cam_Open(0, &hCam, NULL);
if (uiStatus != CAM_API_STS_SUCCESS)
    return -1;

__try
{
    // Get num of features.
    uiStatus = GenApi_GetNumOfFeatures(hCam, "DeviceControl", &uiFeatureNum);
    if (uiStatus != CAM_API_STS_SUCCESS)
        return -1;
    printf("Num of features = %d\n", uiFeatureNum);
```

```
    for (i = 0; i < uiFeatureNum; i++) {
        // Get node name.
        uiStatus = GenApi_GetName(hCam, "DeviceControl", i, &sFeatureName);
        if (uiStatus != CAM_API_STS_SUCCESS)
            return -1;
        printf("No.%d : %s\n", i, sFeatureName.szNodeName);
    }

    return 0;
}
__finally
{
    // Close camera.
    if (hCam != (CAM_HANDLE)NULL) {
        uiStatus = Cam_Close(hCam);
        if (uiStatus != CAM_API_STS_SUCCESS)
            printf("Cam_Close error! (0x%x)", uiStatus);
    }

    // Terminate system.
    Sys_Terminate();
}
```


5.6.2.2. GenApi_GetFeatureName

This function reports the feature name of the index in the specified category node.

[Syntax]

```
CAM_API_STATUS  GenApi_GetFeatureName (
  CAM_HANDLE        hCam,
  const char        *pszCategoryName,
  uint32_t          uiNodeIndex,
  CAM_NODE_NAME     *peFeatureName
);
```

[Parameters]

Parameter		Description
hCam	[in]	Camera-Handle of target camera.
pszCategoryName	[in]	NULL terminated category name.
uiNodeIndex	[in]	Index in feature list of the category node.
peFeatureName	[out]	A pointer to a variable that receives feature name.

[CAM_NODE_NAME structure]

```
typedef struct _UNI_NODE_NAME
{
      char    szNodeName[NODE_NAME_LENGTH_MAX];
} UNI_NODE_NAME, *PUNI_NODE_NAME;
```

Member		Description
szNodeName	[out]	name of the node.

[Return value]

Returns result status. Refer to 5.8 Status code.

[Remarks]

CAM_NODE_NAME is declared in TeliCamAPI.h.

Including TeliCamAPI.h is required.

[Example]

Refer to example of 5.6.2.1 GenApi_GetNumOfFeatures.

5.6.3. IInteger node functions

5.6.3.1. GenApi_GetIntMin

This function reports the minimum valid value of the specified feature name.
This function assumes that target node is IInteger type node.

[Syntax]

```
CAM_API_STATUS  GenApi_GetIntMin (
  CAM_HANDLE        hCam,
  const char        *pszFeatureName,
  int64_t           *pllMin
);
```

[Parameters]

Parameter		Description
hCam	[in]	Camera-Handle of target camera.
pszFeatureName	[in]	NULL terminated feature name.
pllMin	[out]	A pointer to a variable that receives the minimum valid value.

[Return value]

Returns result status. Refer to 5.8 Status code.

[Remarks]

This function is available only for IInteger type node (TC_NODE_TYPE_INTEGER).

This function will return error status if node type of target node is not IInteger type.

Including TeliCamAPI.h is required.

[Example]

C++

```
CAM_API_STATUS   uiStatus;
uint32_t         uiNum;
CAM_HANDLE       hCam = (CAM_HANDLE)NULL;
int64_t          llMin, llMax, llInc, llRdValue, llWrValue;

// Initialize system.
uiStatus = Sys_Initialize();
if (uiStatus != CAM_API_STS_SUCCESS)
    return -1;

// Get number of cameras.
uiStatus = Sys_GetNumOfCameras(&uiNum);
if ((uiStatus != CAM_API_STS_SUCCESS) || (uiNum == 0))
    return -1;

// Open camera that is detected first, in this sample code.
uiStatus = Cam_Open(0, &hCam, NULL);
if (uiStatus != CAM_API_STS_SUCCESS)
    return -1;

__try
{
    // Get minimum value.
    uiStatus = GenApi_GetIntMin(hCam, "Width", &llMin);
```

```
    if (uiStatus != CAM_API_STS_SUCCESS)
        return -1;
    printf("Width Min   : %d\n", (uint32_t)llMin);

    // Get maximum value.
    uiStatus = GenApi_GetIntMax(hCam, "Width", &llMax);
    if (uiStatus != CAM_API_STS_SUCCESS)
        return -1;
    printf("      Max   : %d\n", (uint32_t)llMax);

    // Get increment value.
    uiStatus = GenApi_GetIntInc(hCam, "Width", &llInc);
    if (uiStatus != CAM_API_STS_SUCCESS)
        return -1;
    printf("      Inc   : %d\n", (uint32_t)llInc);

    // Get value.
    uiStatus = GenApi_GetIntValue(hCam, "Width", &llRdValue);
    if (uiStatus != CAM_API_STS_SUCCESS)
        return -1;
    printf("      Before Value : %d\n", (uint32_t)llRdValue);

    // Set value.
    llWrValue = llMin + llInc;
    uiStatus = GenApi_SetIntValue(hCam, "Width", llWrValue);
    if (uiStatus != CAM_API_STS_SUCCESS)
        return -1;

    // Get value.
    uiStatus = GenApi_GetIntValue(hCam, "Width", &llRdValue);
    if (uiStatus != CAM_API_STS_SUCCESS)
        return -1;
    printf("      After Value : %d\n", (uint32_t)llRdValue);

    return 0;
}
__finally
{
    // Close camera.
    if (hCam != (CAM_HANDLE)NULL) {
        uiStatus = Cam_Close(hCam);
        if (uiStatus != CAM_API_STS_SUCCESS)
            printf("Cam_Close error! (0x%x)", uiStatus);
    }

    // Terminate system.
    Sys_Terminate();
}
```


5.6.3.2. GenApi_GetIntMax

This function reports the maximum valid value of the specified feature name.
This function assumes that target node is IInteger type node.

[Syntax]

```
CAM_API_STATUS  GenApi_GetIntMax (
  CAM_HANDLE        hCam,
  const char        *pszFeatureName,
  int64_t           *pllMax
);
```

[Parameters]

Parameter		Description
hCam	[in]	Camera-Handle of target camera.
pszFeatureName	[in]	NULL terminated feature name.
pllMax	[out]	A pointer to a variable that receives the maximum valid value .

[Return value]

Returns result status. Refer to 5.8 Status code.

[Remarks]

This function is available only for IInteger type node (`TC_NODE_TYPE_INTEGER`).

This function will return error status if node type of target node is not IInteger type.

Including TeliCamAPI.h is required.

[Example]

Refer to example of 5.6.3.1 GenApi_GetIntMin.

5.6.3.3. GenApi_GetIntInc

This function reports the Increment value of the specified feature name.
This function assumes that target node is IInteger type node.

[Syntax]

```
CAM_API_STATUS  GenApi_GetIntInc (
  CAM_HANDLE        hCam,
  const char        *pszFeatureName,
  int64_t           *pllInc
);
```

[Parameters]

Parameter		Description
hCam	[in]	Camera-Handle of target camera.
pszFeatureName	[in]	NULL terminated feature name.
pllInc	[out]	A pointer to a variable that receives the Increment value .

[Return value]

Returns result status. Refer to 5.8 Status code.

[Remarks]

This function is available only for IInteger type node (`TC_NODE_TYPE_INTEGER`).

This function will return error status if node type of target node is not IInteger type.

Including TeliCamAPI.h is required.

[Example]

Refer to example of 5.6.3.1 GenApi_GetIntMin.

5.6.3.4. GenApi_GetIntValue

This function reports current value of the specified feature name.
This function assumes that target node is IInteger type node.

[Syntax]

```
CAM_API_STATUS  GenApi_GetIntValue (
  CAM_HANDLE        hCam,
  const char        *pszFeatureName,
  int64_t           *pllValue,
  bool8_t           bVerify = false,
  bool8_t           bIgnoreCache = false,
);
```

[Parameters]

Parameter		Description
hCam	[in]	Camera-Handle of target camera.
pszFeatureName	[in]	NULL terminated feature name.
pllValue	[out]	A pointer to a variable that receives current value.
bVerify	[in]	Specify true to check access mode and valid value range. Specify false to simply read register value. This argument is optional. If nothing is specified to this argument, false will be selected.
bIgnoreCache	[in]	Specify true to read camera register value neglecting cached data. Specify false to use cached data if it is available. This argument is optional. If nothing is specified to this argument, false will be selected.

[Return value]

Returns result status. Refer to 5.8 Status code.

[Remarks]

This function is available only for IInteger type node (`TC_NODE_TYPE_INTEGER`).

This function will return error status if node type of target node is not IInteger type.

Including TeliCamAPI.h is required.

[Example]

Refer to example of 5.6.3.1 GenApi_GetIntMin.

5.6.3.5. GenApi_SetIntValue

This function writes new value to the node specified by the argument feature name.
This function assumes that target node is IInteger type node.

[Syntax]

```
CAM_API_STATUS  GenApi_SetIntValue (
  CAM_HANDLE        hCam,
  const char        *pszFeatureName,
  int64_t           llValue,
  bool8_t           bVerify = true
);
```

[Parameters]

Parameter		Description
hCam	[in]	Camera-Handle of target camera.
pszFeatureName	[in]	NULL terminated feature name.
llValue	[in]	New value.
bVerify	[in]	Specify true to check access mode and valid value range before writing data. Specify false to skip checking. This argument is oprional. If nothing is specified to this argument, true will be selected.

[Return value]

Returns result status. Refer to 5.8 Status code.

[Remarks]

This function is available only for IInteger type node (`TC_NODE_TYPE_INTEGER`).

This function will return error status if node type of target node is not IInteger type.

Including TeliCamAPI.h is required.

[Example]

Refer to example of 5.6.3.1 GenApi_GetIntMin.

5.6.4. IFloat node functions

5.6.4.1. GenApi_GetFloatMin

This function reports the minimum valid value of the specified feature name.
This function assumes that target node is IFloat type node.

[Syntax]

```
CAM_API_STATUS  GenApi_GetFloatMin (
  CAM_HANDLE        hCam,
  const char        *pszFeatureName,
  float64_t         *pdMin
);
```

[Parameters]

Parameter		Description
hCam	[in]	Camera-Handle of target camera.
pszFeatureName	[in]	NULL terminated feature name.
pdMin	[out]	A pointer to a variable that receives the minimum valid value.

[Return value]

Returns result status. Refer to 5.8 Status code.

[Remarks]

This function is available only for IFloat type node (TC_NODE_TYPE_FLOAT).

This function will return error status if node type of target node is not IFloat type.

Including TeliCamAPI.h is required.

[Example]

C++

```
CAM_API_STATUS           uiStatus;
uint32_t                 uiNum;
CAM_HANDLE               hCam = (CAM_HANDLE)NULL;
float64_t                dMin, dMax, dInc, dRdValue, dWrValue;
bool8_t                  bInc;
TC_NODE_DISPLAY_NOTATION eDisplayNotation;
int64_t                  llPrecision;

// Initialize system.
uiStatus = Sys_Initialize();
if (uiStatus != CAM_API_STS_SUCCESS)
    return -1;

// Get number of cameras.
uiStatus = Sys_GetNumOfCameras(&uiNum);
if ((uiStatus != CAM_API_STS_SUCCESS) || (uiNum == 0))
    return -1;

// Open camera that is detected first, in this sample code.
uiStatus = Cam_Open(0, &hCam, NULL);
if (uiStatus != CAM_API_STS_SUCCESS)
    return -1;

__try
{
```

```
    // Get minimum value.
    uiStatus = GenApi_GetFloatMin(hCam, "Gain", &dMin);
    if (uiStatus != CAM_API_STS_SUCCESS)
        return -1;
    printf("Width Min   : %f\n", dMin);

    // Get maximum value.
    uiStatus = GenApi_GetFloatMax(hCam, "Gain", &dMax);
    if (uiStatus != CAM_API_STS_SUCCESS)
        return -1;
    printf("      Max   : %f\n", dMax);

    // Confirm whether the float node has a constant increment.
    uiStatus = GenApi_GetFloatHasInc(hCam, "Gain", &bInc);
    if (uiStatus != CAM_API_STS_SUCCESS)
        return -1;

    if (bInc) {
        // Get increment value.
        uiStatus = GenApi_GetFloatInc(hCam, "Gain", &dInc);
        if (uiStatus != CAM_API_STS_SUCCESS)
            return -1;
        printf("      Inc   : %f\n", dInc);
    }

    // Get display notation.
    uiStatus = GenApi_GetFloatDisplayNotation(hCam, "Gain", &eDisplayNotation);
    if (uiStatus != CAM_API_STS_SUCCESS)
        return -1;
    printf("      Display Notation : %d\n", eDisplayNotation);

    // Get display precision.
    uiStatus = GenApi_GetFloatDisplayPrecision(hCam, "Gain", &llPrecision);
    if (uiStatus != CAM_API_STS_SUCCESS)
        return -1;
    printf("      Display Precision : %d\n", (uint32_t)llPrecision);

    // Get value.
    uiStatus = GenApi_GetFloatValue(hCam, "Gain", &dRdValue);
    if (uiStatus != CAM_API_STS_SUCCESS)
        return -1;
    printf("      Before Value : %f\n", dRdValue);

    // Set value.
    dWrValue = (dMin + dMax) / 2.0f;
    uiStatus = GenApi_SetFloatValue(hCam, "Gain", dWrValue);
    if (uiStatus != CAM_API_STS_SUCCESS)
        return -1;

    // Get value.
    uiStatus = GenApi_GetFloatValue(hCam, "Gain", &dRdValue);
    if (uiStatus != CAM_API_STS_SUCCESS)
        return -1;
    printf("      After Value : %f\n", dRdValue);

    return 0;
}
__finally
{
    // Close camera.
    if (hCam != (CAM_HANDLE)NULL) {
        uiStatus = Cam_Close(hCam);
        if (uiStatus != CAM_API_STS_SUCCESS)
            printf("Cam_Close error! (0x%x)", uiStatus);
    }
```



```
    // Terminate system.
    Sys_Terminate();
}
```

5.6.4.2. GenApi_GetFloatMax

This function reports the maximum valid value of the specified feature name.
This function assumes that target node is IFloat type node.

[Syntax]

```
CAM_API_STATUS  GenApi_GetFloatMax (
  CAM_HANDLE        hCam,
  const char        *pszFeatureName,
  float64_t         *pdMax
);
```

[Parameters]

Parameter		Description
hCam	[in]	Camera-Handle of target camera.
pszFeatureName	[in]	NULL terminated feature name.
dMax	[out]	A pointer to a variable that receives the maximum valid value.

[Return value]

Returns result status. Refer to 5.8 Status code.

[Remarks]

This function is available only for IFloat type node (`TC_NODE_TYPE_FLOAT`).

This function will return error status if node type of target node is not IFloat type.

Including TeliCamAPI.h is required.

[Example]

Refer to example of 5.6.4.1 GenApi_GetFloatMin.

5.6.4.3. GenApi_GetFloatHasInc

This function reports whether the node of the specified feature name has a valid Increment value.
This function assumes that target node is IFloat type node.

[Syntax]

```
CAM_API_STATUS  GenApi_GetFloatHasInc (
  CAM_HANDLE        hCam,
  const char        *pszFeatureName,
  bool8_t           *pbInc
);
```

[Parameters]

Parameter		Description
hCam	[in]	Camera-Handle of target camera.
pszFeatureName	[in]	NULL terminated feature name.
pbInc	[out]	A pointer to a variable that receives flag which describes that the node has valid Increment value or not. If true, valid Increment value exists, otherwise valid Increment value does not exist.

[Return value]

Returns result status. Refer to 5.8 Status code.

[Remarks]

This function is available only for IFloat type node (`TC_NODE_TYPE_FLOAT`).

This function will return error status if node type of target node is not IFloat type.

Including TeliCamAPI.h is required.

[Example]

Refer to example of 5.6.4.1 GenApi_GetFloatMin.

5.6.4.4. GenApi_GetFloatInc

This function reports the Increment value of the specified feature name.
This function assumes that target node is IFloat type node.

[Syntax]

```
CAM_API_STATUS  GenApi_GetFloatInc (
  CAM_HANDLE        hCam,
  const char        *pszFeatureName,
  float64_t         *pdInc
);
```

[Parameters]

Parameter		Description
hCam	[in]	Camera-Handle of target camera.
pszFeatureName	[in]	NULL terminated feature name.
pdInc	[out]	A pointer to a variable that receives the Increment value.

[Return value]

Returns result status. Refer to 5.8 Status code.

[Remarks]

This function is available only for IFloat type node (`TC_NODE_TYPE_FLOAT`).

This function will return error status if node type of target node is not IFloat type.

This function will return error status if the node does not have valid Increment value.

Including TeliCamAPI.h is required.

[Example]

Refer to example of 5.6.4.1 GenApi_GetFloatMin.

5.6.4.5. GenApi_GetFloatDisplayNotation

This function reports the display notation of the specified feature name.
This function assumes that target node is IFloat type node.

[Syntax]

```
CAM_API_STATUS  GenApi_GetFloatDisplayNotation (
  CAM_HANDLE                hCam,
  const char                *pszFeatureName,
  TC_NODE_DISPLAY_NOTATION  *peDisplayNotation
);
```

[Parameters]

Parameter		Description
hCam	[in]	Camera-Handle of target camera.
pszFeatureName	[in]	NULL terminated feature name.
peDisplayNotation	[out]	A pointer to a variable that receives display notation.

[*TC_NODE_DISPLAY_NOTATION* Enumeration]

Member	Description
TC_NODE_DISPLAY_NOTATION_AUTOMATIC	the notation if either scientific or fixed depending on what is shorter
TC_NODE_DISPLAY_NOTATION_FIXED	the notation is fixed, e.g. 123.4
TC_NODE_DISPLAY_NOTATION_SCIENTIFIC	the notation is scientific, e.g. 1.234e2
TC_NODE_DISPLAY_NOTATION_UNDEFINED	Object is not yet initialized
TC_NODE_DISPLAY_NOTATION_UNKNOWN	Unknown

[Return value]

Returns result status. Refer to 5.8 Status code.

[Remarks]

TC_NODE_DISPLAY_NOTATION is declared in TeliCamNode.h. Enum type “EDisplayNotation” declared in Types.h of GenApi reference implementation corresponds to TC_NODE_DISPLAY_NOTATION.

This function is available only for IFloat type node (TC_NODE_TYPE_FLOAT).

This function will return error status if node type of target node is not IFloat type.

Including TeliCamAPI.h is required.

[Example]

Refer to example of 5.6.4.1 GenApi_GetFloatMin.

5.6.4.6. GenApi_GetFloatDisplayPrecision

This function reports the recommended display precision value of the specified feature name.
This function assumes that target node is IFloat type node.

[Syntax]

```
CAM_API_STATUS  GenApi_GetFloatDisplayPrecision (
  CAM_HANDLE        hCam,
  const char        *pszFeatureName,
  int64_t           *pllPrecision
);
```

[Parameters]

Parameter		Description
hCam	[in]	Camera-Handle of target camera.
pszFeatureName	[in]	NULL terminated feature name.
pllPrecision	[out]	A pointer to a variable that receives the recommended display precision value.

[Return value]

Returns result status. Refer to 5.8 Status code.

[Remarks]

This function is available only for IFloat type node (`TC_NODE_TYPE_FLOAT`).

This function will return error status if node type of target node is not IFloat type.

Including TeliCamAPI.h is required.

[Example]

Refer to example of 5.6.4.1 GenApi_GetFloatMin.

5.6.4.7. GenApi_GetFloatValue

This function reports current value of the specified feature name.
This function assumes that target node is IFloat type node.

[Syntax]

```
CAM_API_STATUS  GenApi_GetFloatValue (
  CAM_HANDLE        hCam,
  const char        *pszFeatureName,
  float64_t         *pdValue,
  bool8_t           bVerify = false,
  bool8_t           bIgnoreCache = false,
);
```

[Parameters]

Parameter		Description
hCam	[in]	Camera-Handle of target camera.
pszFeatureName	[in]	NULL terminated feature name.
pdValue	[out]	A pointer to a variable that receives current value.
bVerify	[in]	Specify true to check access mode and valid value range. Specify false to simply read register value. This argument is optional. If nothing is specified to this argument, false will be selected.
bIgnoreCache	[in]	Specify true to read camera register value neglecting cached data. Specify false to use cached data if it is available. This argument is optional. If nothing is specified to this argument, false will be selected.

[Return value]

Returns result status. Refer to 5.8 Status code.

[Remarks]

This function is available only for IFloat type node (`TC_NODE_TYPE_FLOAT`).

This function will return error status if node type of target node is not IFloat type.

Including TeliCamAPI.h is required.

[Example]

Refer to example of 5.6.4.1 GenApi_GetFloatMin.

5.6.4.8. GenApi_SetFloatValue

This function writes new value to the node specified by the argument feature name.
This function assumes that target node is IFloat type node.

[Syntax]

```
CAM_API_STATUS  GenApi_SetFloatValue (
  CAM_HANDLE        hCam,
  const char        *pszFeatureName,
  float64_t         dValue,
  bool8_t           bVerify = true
);
```

[Parameters]

Parameter		Description
hCam	[in]	Camera-Handle of target camera.
pszFeatureName	[in]	NULL terminated feature name.
dValue	[in]	New value.
bVerify	[in]	Specify true to check access mode and valid value range before writing data. Specify false to skip checking. This argument is oprional. If nothing is specified to this argument, true will be selected.

[Return value]

Returns result status. Refer to 5.8 Status code.

[Remarks]

This function is available only for IFloat type node (`TC_NODE_TYPE_FLOAT`).

This function will return error status if node type of target node is not IFloat type.

Including TeliCamAPI.h is required.

[Example]

Refer to example of 5.6.4.1 GenApi_GetFloatMin.

5.6.5. IBoolean node functions

5.6.5.1. GenApi_GetBoolValue

This function reports current value of the specified feature name.
This function assumes that target node is IBoolean type node.

[Syntax]

```
CAM_API_STATUS  GenApi_GetBoolValue (
  CAM_HANDLE        hCam,
  const char        *pszFeatureName,
  bool8_t           *pbValue,
  bool8_t           bVerify = false,
  bool8_t           bIgnoreCache = false
);
```

[Parameters]

Parameter		Description
hCam	[in]	Camera-Handle of target camera.
pszFeatureName	[in]	NULL terminated feature name.
pbValue	[out]	A pointer to a variable that receives current value.
bVerify	[in]	Specify true to check access mode and valid value range. Specify false to simply read register value. This argument is optional. If nothing is specified to this argument, false will be selected.
bIgnoreCache	[in]	Specify true to read camera register value neglecting cached data. Specify false to use cached data if it is available. This argument is optional. If nothing is specified to this argument, false will be selected.

[Return value]

Returns result status. Refer to 5.8 Status code.

[Remarks]

This function is available only for IBoolean type node (`TC_NODE_TYPE_BOOLEAN`).

This function will return error status if node type of target node is not IBoolean type.

Including TeliCamAPI.h is required.

[Example]

C++

```
CAM_API_STATUS       uiStatus;
uint32_t             uiNum;
CAM_HANDLE           hCam = (CAM_HANDLE)NULL;
bool8_t              bRdValue, bWrValue;

// Initialize system.
uiStatus = Sys_Initialize();
if (uiStatus != CAM_API_STS_SUCCESS)
    return -1;

// Get number of cameras.
uiStatus = Sys_GetNumOfCameras(&uiNum);
if ((uiStatus != CAM_API_STS_SUCCESS) || (uiNum == 0))
    return -1;

// Open camera that is detected first, in this sample code.
uiStatus = Cam_Open(0, &hCam, NULL);
if (uiStatus != CAM_API_STS_SUCCESS)
    return -1;

__try
{
    // Get value.
    uiStatus = GenApi_GetBoolValue(hCam, "ReverseX", &bRdValue);
    if (uiStatus != CAM_API_STS_SUCCESS)
        return -1;
    printf("      Before Value : %d\n", (uint32_t)bRdValue);

    // Set value.
    bWrValue = !bRdValue;
    uiStatus = GenApi_SetBoolValue(hCam, "ReverseX", bWrValue);
    if (uiStatus != CAM_API_STS_SUCCESS)
        return -1;

    // Get value.
    uiStatus = GenApi_GetBoolValue(hCam, "ReverseX", &bRdValue);
    if (uiStatus != CAM_API_STS_SUCCESS)
        return -1;
    printf("      After Value : %d\n", (uint32_t)bRdValue);

    return 0;
}
__finally
{
    // Close camera.
    if (hCam != (CAM_HANDLE)NULL) {
        uiStatus = Cam_Close(hCam);
        if (uiStatus != CAM_API_STS_SUCCESS)
            printf("Cam_Close error! (0x%x)", uiStatus);
    }

    // Terminate system.
    Sys_Terminate();
}
```

5.6.5.2. GenApi_SetBoolValue

This function writes new value to the node specified by the argument feature name.
This function assumes that target node is IBoolean type node.

[Syntax]

```
CAM_API_STATUS  GenApi_SetBoolValue (
  CAM_HANDLE        hCam,
  const char        *pszFeatureName,
  bool8_t           bValue,
  bool8_t           bVerify = true
);
```

[Parameters]

Parameter		Description
hCam	[in]	Camera-Handle of target camera.
pszFeatureName	[in]	NULL terminated feature name.
bValue	[in]	New value.
bVerify	[in]	Specify true to check access mode and valid value range before writing data. Specify false to skip checking. This argument is oprional. If nothing is specified to this argument, true will be selected.

[Return value]

Returns result status. Refer to 5.8 Status code.

[Remarks]

This function is available only for IBoolean type node (`TC_NODE_TYPE_BOOLEAN`).

This function will return error status if node type of target node is not IBoolean type.

Including TeliCamAPI.h is required.

[Example]

Refer to example of 5.6.5.1 GenApi_GetBoolValue.

5.6.6. IEnumeration and IEnumEntry node functions

5.6.6.1. GenApi_GetEnumIntValue

This function reports current value of the target enumeration node as an integer value.
This function assumes that target node is IEnumeration type node.

[Syntax]

```
CAM_API_STATUS  GenApi_GetEnumIntValue (
  CAM_HANDLE        hCam,
  const char        *pszFeatureName,
  int64_t           *pllValue,
  bool8_t           bVerify = false,
  bool8_t           bIgnoreCache = false,
);
```

[Parameters]

Parameter		Description
hCam	[in]	Camera-Handle of target camera.
pszFeatureName	[in]	NULL terminated feature name.
pllValue	[out]	A pointer to a variable that receives current value.
bVerify	[in]	Specify true to check access mode and validity of the value. Specify false to simply read register value. This argument is optional. If nothing is specified to this argument, false will be selected.
bIgnoreCache	[in]	Specify true to read camera register value neglecting cached data. Specify false to use cached data if it is available, This argument is optional. If nothing is specified to this argument, false will be selected.

[Return value]

Returns result status. Refer to 5.8 Status code.

[Remarks]

This function is available only for IEnumeration type node (`TC_NODE_TYPE_ENUMERATION`).

This function will return error status if node type of target node is not IEnumeration type.

Including TeliCamAPI.h is required.

[Example]

C++

```
CAM_API_STATUS       uiStatus;
uint32_t             uiNum;
CAM_HANDLE           hCam = (CAM_HANDLE)NULL;
int64_t              llRdValue, llWrValue;

// Initialize system.
uiStatus = Sys_Initialize();
if (uiStatus != CAM_API_STS_SUCCESS)
    return -1;

// Get number of cameras.
uiStatus = Sys_GetNumOfCameras(&uiNum);
```

```
if ((uiStatus != CAM_API_STS_SUCCESS) || (uiNum == 0))
    return -1;

// Open camera that is detected first, in this sample code.
uiStatus = Cam_Open(0, &hCam, NULL);
if (uiStatus != CAM_API_STS_SUCCESS)
    return -1;

__try
{
    // Set int value.
    llWrValue = 64;      // 64 : Software
    uiStatus = GenApi_SetEnumIntValue(hCam, "TriggerSource", llWrValue);
    if (uiStatus != CAM_API_STS_SUCCESS)
        return -1;       // Not impremented, or any other error

    // Readback.
    uiStatus = GenApi_GetEnumIntValue(hCam, "TriggerSource", &llRdValue);
    if (uiStatus != CAM_API_STS_SUCCESS)
        return -1;
    printf("      Read Int value : %d\n", (uint32_t)llRdValue);

    return 0;
}
__finally
{
    // Close camera.
    if (hCam != (CAM_HANDLE)NULL) {
        uiStatus = Cam_Close(hCam);
        if (uiStatus != CAM_API_STS_SUCCESS)
            printf("Cam_Close error! (0x%x)", uiStatus);
    }

    // Terminate system.
    Sys_Terminate();
}
```


5.6.6.2. GenApi_SetEnumIntValue

This function writes new value expressed as an integer value to the target enumeration node.
This function assumes that target node is IEnumeration type node.

[Syntax]

```
CAM_API_STATUS  GenApi_SetEnumIntValue (
  CAM_HANDLE        hCam,
  const char        *pszFeatureName,
  int64_t           llValue,
  bool8_t           bVerify = true
);
```

[Parameters]

Parameter		Description
hCam	[in]	Camera-Handle of target camera.
pszFeatureName	[in]	NULL terminated feature name.
llValue	[in]	New value expressed in integer value.
bVerify	[in]	Specify true to check access mode and validity of the value before writing data. Specify false to skip checking. This argument is oprional. If nothing is specified to this argument, true will be selected.

[Return value]

Returns result status. Refer to 5.8 Status code.

[Remarks]

This function is available only for IEnumeration type node (`TC_NODE_TYPE_ENUMERATION`).

This function will return error status if node type of target node is not IEnumeration type.

Including TeliCamAPI.h is required.

[Example]

Refer to example of 5.6.6.1 GenApi_GetEnumIntValue.

5.6.6.3. GenApi_GetEnumStrValue

This function reports current value of the target enumeration node as a string.
This function assumes that target node is IEnumeration type node.

[Syntax]

```
CAM_API_STATUS  GenApi_GetEnumStrValue (
  CAM_HANDLE        hCam,
  const char        *pszFeatureName,
  char              *pszBuf,
  uint32_t          *puiSize,
  bool8_t           bVerify = false,
  bool8_t           bIgnoreCache = false
);
```

[Parameters]

Parameter		Description
hCam	[in]	Camera-Handle of target camera.
pszFeatureName	[in]	NULL terminated feature name.
pszBuf	[out]	A pointer to a variable that receives string value of the target node. If NULL is specified to this argument, this function will writes buffer size necessary for writing string value of the node to variable pointed by puiSize, without reporting string value.
puiSize	[in, out]	A pointer to a variable that contains size of buffer pointed by `pszBuf`. If NULL is specified to `pszBuf`, this function will writes buffer size necessary for writing string value of the node to variable pointed by this argument.
bVerify	[in]	Specify true to check access mode and validity of the value. Specify false to simply read register value. This argument is optional. If nothing is specified to this argument, false will be selected.
bIgnoreCache	[in]	Specify true to read camera register value ignoring cached data. Specify false to use cached data, if it is available, This argument is optional. If nothing is specified to this argument, false will be selected.

[Return value]

Returns result status. Refer to 5.8 Status code.

[Remarks]

This function is available only for IEnumeration type node (`TC_NODE_TYPE_ENUMERATION`).

This function will return error status if node type of target node is not IEnumeration type.

Including TeliCamAPI.h is required.

[Example]

C++

```
CAM_API_STATUS       uiStatus;
uint32_t             uiNum, uiSize;
CAM_HANDLE           hCam = (CAM_HANDLE)NULL;
char                 szbuf[32];

// Initialize system.
uiStatus = Sys_Initialize();
if (uiStatus != CAM_API_STS_SUCCESS)
    return -1;

// Get number of cameras.
uiStatus = Sys_GetNumOfCameras(&uiNum);
if ((uiStatus != CAM_API_STS_SUCCESS) || (uiNum == 0))
    return -1;

// Open camera that is detected first, in this sample code.
uiStatus = Cam_Open(0, &hCam, NULL);
if (uiStatus != CAM_API_STS_SUCCESS)
    return -1;

__try
{
    // Read string.
    uiSize = 32;
    uiStatus = GenApi_GetEnumStrValue(hCam, "TriggerSource", szbuf, &uiSize);
    if (uiStatus != CAM_API_STS_SUCCESS)
        return -1;
    printf("Before read str value : %s\n", szbuf);

    // Set string.
    if (strncmp((const char*)szbuf, "Line0", uiSize) == 0) {
        uiStatus = GenApi_SetEnumStrValue(hCam, "TriggerSource", "Software");
    } else {
        uiStatus = GenApi_SetEnumStrValue(hCam, "TriggerSource", "Line0");
    }
    if (uiStatus != CAM_API_STS_SUCCESS)
        return -1;       // Not impremented, or any other error

    // Readback.
    uiSize = 32;
    uiStatus = GenApi_GetEnumStrValue(hCam, "TriggerSource", szbuf, &uiSize);
    if (uiStatus != CAM_API_STS_SUCCESS)
        return -1;
    printf("After  read str value : %s\n", szbuf);

    return 0;
}
__finally
{
    // Close camera.
    if (hCam != (CAM_HANDLE)NULL) {
        uiStatus = Cam_Close(hCam);
        if (uiStatus != CAM_API_STS_SUCCESS)
            return -1;
    }

    // Terminate system.
    Sys_Terminate();
}
```

5.6.6.4. GenApi_SetEnumStrValue

This function writes new value expressed as a string to the target enumeration node.
This function assumes that target node is IEnumeration type node.

[Syntax]

```
CAM_API_STATUS  GenApi_SetEnumStrValue (
  CAM_HANDLE        hCam,
  const char        *pszFeatureName,
  const char        *pszBuf,
  bool8_t           bVerify = true
);
```

[Parameters]

Parameter		Description
hCam	[in]	Camera-Handle of target camera.
pszFeatureName	[in]	NULL terminated feature name.
pszBuf	[in]	A pointer to new string value.
bVerify	[in]	Specify true to check access mode and validity of the value before writing data. Specify false to skip checking. This argument is oprional. If nothing is specified to this argument, true will be selected.

[Return value]

Returns result status. Refer to 5.8 Status code.

[Remarks]

This function is available only for IEnumeration type node (`TC_NODE_TYPE_ENUMERATION`).

This function will return error status if node type of target node is not IEnumeration type.

Including TeliCamAPI.h is required.

[Example]

Refer to example of 5.6.6.3 GenApi_GetEnumStrValue.

5.6.6.5. GenApi_GetNumOfEnumEntries

This function reports number of entries of the IEnumeration node specified by the argument feature name.
This function assumes that target node is IEnumeration type node.

[Syntax]

```
CAM_API_STATUS  GenApi_GetNumOfEnumEntries (
  CAM_HANDLE        hCam,
  const char        *pszFeatureName,
  uint32_t          *puiNum
);
```

[Parameters]

Parameter		Description
hCam	[in]	Camera-Handle of target camera.
pszFeatureName	[in]	NULL terminated feature name.
puiNum	[out]	A pointer to a variable that receives the number of the entries.

[Return value]

Returns result status. Refer to 5.8 Status code.

[Remarks]

The number of entries is a number of entry nodes described in the camera description file (Xml file) which include not implemented entry nodes and not available entry nodes.

This function is available only for IEnumeration type node (TC_NODE_TYPE_ENUMERATION).

This function will return error status if node type of target node is not IEnumeration type.

Including TeliCamAPI.h is required.

[Example]

C++

```
CAM_API_STATUS       uiStatus;
uint32_t             uiNum, uiSize;
CAM_HANDLE           hCam = (CAM_HANDLE)NULL;
CAM_NODE_HANDLE      hEnumEntryNode = NULL;
CAM_NODE_HANDLE      hNode = NULL;
uint32_t             uiEntriesNum;
int64_t              llEntryValue;
char                 *pszBuf = NULL;

// Initialize system.
uiStatus = Sys_Initialize();
if (uiStatus != CAM_API_STS_SUCCESS)
    return -1;

// Get number of cameras.
uiStatus = Sys_GetNumOfCameras(&uiNum);
if ((uiStatus != CAM_API_STS_SUCCESS) || (uiNum == 0))
    return -1;

// Open camera that is detected first, in this sample code.
uiStatus = Cam_Open(0, &hCam, NULL);
if (uiStatus != CAM_API_STS_SUCCESS)
    return -1;

__try
```

```
{
    // Get node handle.
    uiStatus = GenApi_GetNode(hCam, "LineSelector", &hNode);
    if (uiStatus != CAM_API_STS_SUCCESS)
        return -1;   // Not impremented, or any other error
    printf("hNode = %I64x\n", hNode);

    // Get num of entries.
    uiStatus = GenApi_GetNumOfEnumEntries(hCam, hNode, &uiEntriesNum);
    if (uiStatus != CAM_API_STS_SUCCESS)
        return -1;
    printf("      Num of enum entries : %d\n", (uint32_t)uiEntriesNum);

    for (uint32_t i = 0; i < uiEntriesNum; i++) {
        // Get enum entry by index.
        uiStatus = GenApi_GetEnumEntryByIndex(hCam, hNode, i, &hEnumEntryNode);
        if (uiStatus != CAM_API_STS_SUCCESS)
            return -1;

        // Get int value.
        uiStatus =GenApi_GetEnumEntryIntValue(hCam, hEnumEntryNode, &llEntryValue);
        if (uiStatus != CAM_API_STS_SUCCESS)
            return -1;
        printf("Enum %d : EnumEntryIntValue = %d", i, llEntryValue);

        // Get size of the string.
        uiSize = 0;
        uiStatus = GenApi_GetEnumEntryStrValue(hCam, hEnumEntryNode, NULL, &uiSize);
        if (uiStatus != CAM_API_STS_SUCCESS)
            return -1;

        // Allocate buffer.
        pszBuf = (char *)VirtualAlloc(NULL,
                                      uiSize,
                                      MEM_RESERVE | MEM_COMMIT,
                                      PAGE_EXECUTE_READWRITE);
        if (pszBuf == NULL) {
            return -1;
        }

        // Get string.
        uiStatus =GenApi_GetEnumEntryStrValue(hCam, hEnumEntryNode, pszBuf, &uiSize);
        if (uiStatus != CAM_API_STS_SUCCESS)
            return -1;
        printf(", EnumEntryStrValue = %s\n", pszBuf);

        VirtualFree(pszBuf, 0, MEM_RELEASE);
        pszBuf = NULL;
    }

    return 0;
}
__finally
{
    if (pszBuf != NULL) {
        free( pszBuf);
    }

    // Close camera.
    if (hCam != (CAM_HANDLE)NULL) {
        uiStatus = Cam_Close(hCam);
        if (uiStatus != CAM_API_STS_SUCCESS)
            printf("Cam_Close error! (0x%x)", uiStatus);
    }

    // Terminate system.
    Sys_Terminate();
}
```


5.6.6.6. GenApi_GetEnumEntryAccessMode

This function reports access mode of enumeration entry node specified by index of the entry.
This function assumes that target node is IEnumeration type node.

[Syntax]

```
CAM_API_STATUS   GenApi_GetEnumEntryAccessMode (
  CAM_HANDLE           hCam,
  const char           *pszFeatureName,
  uint32_t             uiEnumIndex,
  TC_NODE_ACCESS_MODE  *peAccessMode
);
```

[Parameters]

Parameter		Description
hCam	[in]	Camera-Handle of target camera.
pszFeatureName	[in]	NULL terminated feature name.
uiEnumIndex	[in]	Index in entry list described in target node.
peAccessMode	[out]	A pointer to a variable that receives current access mode of the target enumeration entry node.

[Return value]

Returns result status. Refer to 5.8 Status code.

[Remarks]

This function is available only for IEnumeration type node (`TC_NODE_TYPE_ENUMERATION`).

This function will return error status if node type of target node is not IEnumeration type.

Including TeliCamAPI.h is required.

[Example]

Refer to example of 5.6.6.5 GenApi_GetNumOfEnumEntries.

5.6.6.7. GenApi_GetEnumEntryIntValue

This function reports value of enumeration entry node specified by index of entry, as an integer value.
This function assumes that target node is IEnumeration type node.

[Syntax]

```
CAM_API_STATUS  GenApi_GetEnumEntryIntValue (
  CAM_HANDLE        hCam,
  const char        *pszFeatureName,
  uint32_t          uiEnumIndex,
  int64_t           *pllValue
);
```

[Parameters]

Parameter		Description
hCam	[in]	Camera-Handle of target camera.
pszFeatureName	[in]	NULL terminated feature name.
uiEnumIndex	[in]	Index in entry list described in target node.
pllValue	[out]	A pointer to a variable that receives integer value of the target enumeration entry node.

[Return value]

Returns result status. Refer to 5.8 Status code.

[Remarks]

This function is available only for IEnumeration type node (`TC_NODE_TYPE_ENUMERATION`).

This function will return error status if node type of target node is not IEnumeration type.

Including TeliCamAPI.h is required.

[Example]

Refer to example of 5.6.6.5 GenApi_GetNumOfEnumEntries.

5.6.6.8. GenApi_GetEnumEntryStrValue

This function reports value of enumeration entry node specified by index of entry, as a string.
This function assumes that target node is IEnumeration type node.

[Syntax]

```
CAM_API_STATUS  GenApi_GetEnumEntryStrValue (
  CAM_HANDLE        hCam,
  const char        *pszFeatureName,
  uint32_t          uiEnumIndex,
  char              *pszBuf,
  uint32_t          *puiSize
);
```

[Parameters]

Parameter		Description
hCam	[in]	Camera-Handle of target camera.
pszFeatureName	[in]	NULL terminated feature name.
uiEnumIndex	[in]	Index in entry list described in target node.
pszBuf	[out]	A pointer to a variable that receives string value of the target enumeration entry node. If NULL is specified to this argument , this function will writes buffer size necessary for writing string value of the entry node to variable pointed by puiSize, without reporting string value.
puiSize	[in, out]	A pointer to a variable that contains size of buffer pointed by `pszBuf`. If NULL is specified to `pszBuf`, this function will writes buffer size necessary for writing string value of the entry node to variable pointed by this argument.

[Return value]

Returns result status. Refer to 5.8 Status code.

[Remarks]

This function is available only for IEnumeration type node (`TC_NODE_TYPE_ENUMERATION`).

This function will return error status if node type of target node is not IEnumeration type.

Including TeliCamAPI.h is required.

[Example]

Refer to example of 5.6.6.5 GenApi_GetNumOfEnumEntries.

5.6.7. ICommand node functions

5.6.7.1. GenApi_CmdExecute

This function issues the command of ICommand type node.
This function assumes that target node is ICommand type node.

[Syntax]

```
CAM_API_STATUS  GenApi_CmdExecute (
  CAM_HANDLE        hCam,
  const char        *pszFeatureName,
  bool8_t           bVerify = true
);
```

[Parameters]

Parameter		Description
hCam	[in]	Camera-Handle of target camera.
pszFeatureName	[in]	NULL terminated feature name.
bVerify	[in]	Specify true to check access mode. Specify false to skip checking. This argument is oprional. If nothing is specified to this argument, true will be selected.

[Return value]

Returns result status. Refer to 5.8 Status code.

[Remarks]

This function is available only for ICommand type node (TC_NODE_TYPE_COMMAND).
This function will return error status if node type of target node is not ICommand type.
Including TeliCamAPI.h is required.

[Example]

C++

```
...

bool bDone = false;

// Execute command.
uiStatus = GenApi_CmdExecute(hCam, "TriggerSoftware");
if (uiStatus != CAM_API_STS_SUCCESS)
    return -1;

while(1) {
    // Confirm whether the execution has been accomplished.
    uiStatus = GenApi_GetCmdIsDone(hCam, "TriggerSoftware", &bDone);
    if (uiStatus != CAM_API_STS_SUCCESS)
        return -1;

    if (bDone == true)
        break;

    Sleep(0);
}
...
```

5.6.7.2. GenApi_GetCmdIsDone

This function reports whether the command has finished or not.
This function assumes that target node is ICommand type node.

[Syntax]

```
CAM_API_STATUS  GenApi_GetCmdIsDone (
  CAM_HANDLE        hCam,
  const char        *pszFeatureName,
  bool8_t           *pbDone,
  bool8_t           bVerify = false
);
```

[Parameters]

Parameter		Description
hCam	[in]	Camera-Handle of target camera.
pszFeatureName	[in]	NULL terminated feature name.
pbDone	[out]	A pointer to a variable that receives current execution status . True means that the command has finished. False means that the command has not finished.
bVerify	[in]	Specify true to check access mode. Specify false to skip checking. This argument is oprional. If nothing is specified to this argument, false will be selected.

[Return value]

Returns result status. Refer to 5.8 Status code.

[Remarks]

This function is available only for ICommand type node (`TC_NODE_TYPE_COMMAND`).

This function will return error status if node type of target node is not ICommand type.

Including TeliCamAPI.h is required.

[Example]

Refer to example of 5.6.7.1 GenApi_CmdExecute.

5.6.8. IString node functions

5.6.8.1. GenApi_GetStrValue

This function reports string value of the node specified by the argument feature name.
This function assumes that target node is IString node.

[Syntax]

```
CAM_API_STATUS  GenApi_GetStrValue (
  CAM_HANDLE        hCam,
  const char        *pszFeatureName,
  char              *pszBuf,
  uint32_t          *puiSize,
  bool8_t           bVerify = false,
  bool8_t           bIgnoreCache = false,
);
```

[Parameters]

Parameter		Description
hCam	[in]	Camera-Handle of target camera.
pszFeatureName	[in]	NULL terminated feature name.
pszBuf	[out]	A pointer to a variable that receives string value of target node. If NULL is specified to this argument , this function will writes buffer size necessary for writing string value of the node to variable pointed by puiSize, without reporting string value.
puiSize	[in, out]	A pointer to a variable that contains size of buffer pointed by pszBuf. If NULL is specified to pszBuf, this function will writes buffer size necessary for writing string value of the node to variable pointed by this argument.
bVerify	[in]	Specify true to check access mode and validity of the value. Specify false to simply read register value. This argument is optional. If nothing is specified to this argument, false will be selected.
bIgnoreCache	[in]	Specify true to read camera register value neglecting cached data. Specify false to use cached data if it is available. This argument is optional. If nothing is specified to this argument, false will be selected.

[Return value]

Returns result status. Refer to 5.8 Status code.

[Remarks]

This function is available only for IString type node (TC_NODE_TYPE_STRING).
This function will return error status if node type of target node is not IString type.
Including TeliCamAPI.h is required.

[Example]

C++

```
...

char            szBuf[128];
uint32_t        uiSize = 128;


// Set string.
uiStatus = GenApi_SetStrValue(hCam, "UserDefinedName", "Test");
if (uiStatus != CAM_API_STS_SUCCESS)
    return -1;

// Get string.
uiStatus = GenApi_GetStrValue(hCam, "UserDefinedName", &szBuf[0], &uiSize);
if (uiStatus != CAM_API_STS_SUCCESS)
    return -1;
printf("  GenApi_GetStrValue : %s\n", szBuf);

...
```


5.6.8.2. GenApi_SetStrValue

This function writes new string value to the node specifid by the argument node handle.
This function assumes that target node is IString type node.

[Syntax]

```
CAM_API_STATUS  GenApi_SetStrValue (
  CAM_HANDLE        hCam,
  const char        *pszFeatureName,
  const char        *pszBuf,
  bool8_t           bVerify = true
);
```

[Parameters]

Parameter		Description
hCam	[in]	Camera-Handle of target camera.
pszFeatureName	[in]	NULL terminated feature name.
pszBuf	[in]	A pointer to a new string value.
bVerify	[in]	Specify true to check access mode and validity of the value. Specify false to skip checking. This argument is optional. If nothing is specified to this argument, false will be selected.

[Return value]

Returns result status. Refer to 5.8 Status code.

[Remarks]

This function is available only for IString type node (`TC_NODE_TYPE_STRING`).

This function will return error status if node type of target node is not IString type.

Including TeliCamAPI.h is required.

[Example]

Refer to example of 5.6.8.1 GenApi_GetStrValue.

5.6.9. Chunk functions

5.6.9.1. GenApi_ChunkAttachBuffer

This function attaches a buffer to the GenICam chunk adapter.
When acquiring chunk data from received payload data using GenICam, it is necessary to attach the buffer storing the payload data to GenICam chunk adapter.

[Syntax]

```
CAM_API_STATUS  GenApi_ChunkAttachBuffer (
  CAM_STRM_HANDLE   hStrm,
  void              *pvPayloadBuf,
  uint32_t          uiPayloadSize
);
```

[Parameters]

Parameter		Description
hStrm	[in]	Stream-Handle of the target stream interface.
pvPayloadBuf	[in]	A pointer to the buffer storing the received payload data.
uiPayloadSize	[in]	The size of the received payload data.

[Return value]

Returns result status. Refer to 5.8 Status code.

[Remarks]

This function will return error status if no chunk data is added to the payload data.
The chunk data is acquired using the GenICam function.

Including TeliCamAPI.h is required.

[Example]

C++

```
...

CAM_IMAGE_INFO   sImageInfo;
int64_t          lFrameID;
float64_t        dExposureTime;

// Get current image.
uiStatus = Strm_ReadCurrentImage(hStrm, pvPayloadBuf, &uiPyldSize, &sImageInfo);
if (uiStatus != CAM_API_STS_SUCCESS)
    return -1;

// Attach Buffer
uiStatus = Chunk_AttachBuffer(hStrm, pvPayloadBuf, uiPyldSize);
if (uiStatus != CAM_API_STS_SUCCESS)
    return -1;

// Get ChunkFrameID
uiStatus = GenApi_GetIntValue(hCam, "ChunkFrameID", &lFrameID);
if (uiStatus != CAM_API_STS_SUCCESS)
    return -1;
printf("ChunkFrameID = %d\n", lFrameID);

// Get ChunkExposureTime
uiStatus = GenApi_GetFloatValue(hCam, "ChunkExposureTime", &dExposureTime);
```

```
if (uiStatus != CAM_API_STS_SUCCESS)
    return -1;
printf("ChunkExposureTime = %f\n", dExposureTime);

...
```

5.6.9.2. GenApi_ChunkUpdateBuffer

This function updates the buffer attached to the GenICam chunk adapter.

[Syntax]

```
CAM_API_STATUS  GenApi_ChunkUpdateBuffer (
  CAM_STRM_HANDLE   hStrm,
  void              *pvPayloadBuf
);
```

[Parameters]

Parameter		Description
hStrm	[in]	Stream-Handle of the target stream interface.
pvPayloadBuf	[in]	A pointer to the buffer storing the received payload data.

[Return value]

Returns result status. Refer to 5.8 Status code.

[Remarks]

This function will return error status if no chunk data is added to the payload data.
The chunk data is acquired using the GenICam function.

If the buffer has not been attached to GenICam chunk adapter, or if the layout of the chunk data has changed, an error is returned.
It can process faster than GenApi_ChunkAttachBuffer().

Including TeliCamAPI.h is required.

5.6.9.3. GenApi_ChunkCheckBufferLayout

This function checks if a buffer contains chunks in a known format.

[Syntax]

```
CAM_API_STATUS  GenApi_ChunkCheckBufferLayout (
  CAM_STRM_HANDLE   hStrm,
  void              *pvPayloadBuf,
  uint32_t          uiPayloadSize,
  bool8_t           *pbValue
);
```

[Parameters]

Parameter		Description
hStrm	[in]	Stream-Handle of the target stream interface.
pvPayloadBuf	[in]	A pointer to the buffer storing the received payload data.
uiPayloadSize	[in]	The size of the received payload data.
pbValue	[out]	A pointer to a variable that receives value. If the value is true, the buffer contains chunks. If the value is false, the buffer does not contain chunks.

[Return value]

Returns result status. Refer to 5.8 Status code.

[Remarks]

Including TeliCamAPI.h is required.

5.6.10.Others

5.6.10.1. GenApi_GetLastError

This function reports detail error information about CAM_API_STS_GENICAM_ERR.

[Syntax]

```
CAM_API_STATUS  GenApi_GetLastError (
  CAM_GENICAM_ERR_MSG      *peErrMsg
);
```

[Parameters]

Parameter		Description
peErrMsg	[out]	A pointer to a variable that receives detail error information.

```
[CAM_GENICAM_ERR_MSG structure]
  typedef struct _CAM_GENICAM_ERR_MSG
  {
        const char  *pszDescription;   // Error description
        const char  *pszSourceFileName;// Filename in which the error occurred.
        uint32_t    uiSourceLine;      // Line number at which the error occurred.
  } CAM_GENICAM_ERR_MSG, *PCAM_GENICAM_ERR_MSG;
```

Member		Description
pszDescription	[out]	Error idescription.
pszSpurceFileName	[out]	The file name in which the error ocuurred.
uiSourceLine	[out]	The line number at which the error occurred.

[Return value]

Returns result status. Refer to 5.8 Status code.

[Remarks]

CAM_API_STS_GENICAM_ERR may occur when function in 5.6 GenICam functions is called.

Functions in 5.5 Controlling camera feature functions may also CAM_API_STS_GENICAM_ERR, because , some of functions in 5.5 Controlling camera feature functions internally call GenICam functions.

The last GenICam error covers all threads in the application.
This function may return an error information different from the last error information at the timing when this function was called, If another CAM_API_STS_GENICAM_ERR occurred in the other thread.

Including TeliCamAPI.h is required.

5.6.11.Old GenICam functions

You can use the functions in this chapter for compatibility with previous versions.
It is recommended that you use the new GenICam function (GenApi_ *) from Section 5.6.1 to 5.6.10 if backwards compatibility is not required.

5.6.11.1. INode functions

5.6.11.1.1. Nd_GetNode

This function retrieves a node which has name specified in the argument `pszName`, and writes its node handle to variable pointed by the argument `phNode`.

[Syntax]

```
CAM_API_STATUS  Nd_GetNode (
  CAM_HANDLE        hCam,
  const char        *pszName,
  CAM_NODE_HANDLE   *phNode
);
```

[Parameters]

Parameter		Description
hCam	[in]	Camera-Handle of target camera.
pszName	[in]	NULL terminated node name.
phNode	[out]	A pointer to a variable that receives node handle of the retrived node.

[Return value]

Returns result status. Refer to 5.8 Status code.

[Remarks]

This function remains for backward compatibility.

Use node handle retrieved with this function for accessing node in camera description data (XML data) of the camera.

The available node names of the camera depends on implemented functions and or interface type of the camera.
Refer to instruction manual of the camera for checking the node name available in the camera.

.

5.6.11.1.2. Nd_GetType

This function reports node type of the specified node.

[Syntax]

```
CAM_API_STATUS  Nd_GetType (
  CAM_HANDLE        hCam,
  CAM_NODE_HANDLE   hNode,
  TC_NODE_TYPE      *peNodeType
);
```

[Parameters]

Parameter		Description
hCam	[in]	Camera-Handle of target camera.
hNode	[in]	Node handle of target node.
peNodeType	[out]	A pointer to a variable that receives node type of the specified node.

[Return value]

Returns result status. Refer to 5.8 Status code.

[Remarks]

This function remains for backward compatibility.
When creating a new application, use of GenApi_GetType() is recommended.

5.6.11.1.3. Nd_GetName

This function reports name of the node specified by the argument node handle.

[Syntax]

```
CAM_API_STATUS  Nd_GetName (
  CAM_HANDLE        hCam,
  CAM_NODE_HANDLE   hNode,
  CAM_NODE_NAME     *pszNodeName
);
```

[Parameters]

Parameter		Description
hCam	[in]	Camera-Handle of target camera.
hNode	[in]	Node handle of target node.
pszNode	[out]	A pointer to a variable that receives node name of the node.

[Return value]

Returns result status. Refer to 5.8 Status code.

[Remarks]

This function remains for backward compatibility.
When creating a new application, use of GenApi_GetFeatureName() is recommended.

5.6.11.1.4. Nd_GetAccessMode

This function reports access mode of the node specified by the argument node handle.

[Syntax]

```
CAM_API_STATUS  Nd_GetAccessMode (
  CAM_HANDLE            hCam,
  CAM_NODE_HANDLE       hNode,
  TC_NODE_ACCESS_MODE   *peAccessMode
);
```

[Parameters]

Parameter		Description
hCam	[in]	Camera-Handle of target camera.
hNode	[in]	Node handle of target node.
peAccessMode	[out]	A pointer to a variable that receives current access mode of the node.

[Return value]

Returns result status. Refer to 5.8 Status code.

[Remarks]

This function remains for backward compatibility.
When creating a new application, use of GenApi_GetAccessMode() or GenApi_GetEnumEntryAccessMode() is recommended.

5.6.11.1.5. Nd_GetVisibility

This functio reports visibility of the node specified by the argument node handle.

[Syntax]

```
CAM_API_STATUS  Nd_GetVisibility (
  CAM_HANDLE            hCam,
  CAM_NODE_HANDLE       hNode,
  TC_NODE_VISIBILITY    *peVisibility
);
```

[Parameters]

Parameter		Description
hCam	[in]	Camera-Handle of target camera.
hNode	[in]	Node handle of target node.
peVisibility	[out]	A pointer to a variable that receives visibility of the node.

[Return value]

Returns result status. Refer to 5.8 Status code.

[Remarks]

This function remains for backward compatibility.
When creating a new application, use of GenApi_GetVisibility() is recommended.

5.6.11.1.6. Nd_GetCachingMode

This function reports caching mode of the node specified by the argument node handle

[Syntax]

```
CAM_API_STATUS  Nd_GetCachingMode (
  CAM_HANDLE            hCam,
  CAM_NODE_HANDLE       hNode,
  TC_NODE_CACHING_MODE  *peCachingMode
);
```

[Parameters]

Parameter		Description
hCam	[in]	Camera-Handle of target camera.
hNode	[in]	Node handle of target node.
peCachingMode	[out]	A pointer to a variable that receives caching mode of the node.

[Return value]

Returns result status. Refer to 5.8 Status code.

[Remarks]

This function remains for backward compatibility.
When creating a new application, use of GenApi_GetCachingMode() is recommended.

5.6.11.1.7. Nd_GetDescription

This function reports description of the node specified by the argument node handle.

[Syntax]

```
CAM_API_STATUS  Nd_GetDescription (
  CAM_HANDLE            hCam,
  CAM_NODE_HANDLE       hNode,
  char                  *pszBuf,
  uint32_t              *puiSize
);
```

[Parameters]

Parameter		Description
hCam	[in]	Camera-Handle of target camera.
hNode	[in]	Node handle of target node.
pszBuf	[out]	A pointer to a variable that receives description of the node. If NULL is specified to this argument, this function will write buffer size necessary for writing description of the node to variable pointed by `puiSize`, without reporting description string.
puiSize	[in, out]	A pointer to a variable that contains size of buffer pointed by `pszBuf` If NULL is specified to `pszBuf`, this function will write buffer size necessary for writing description of the node to variable pointed by this argument.

[Return value]

Returns result status. Refer to 5.8 Status code.

[Remarks]

This function remains for backward compatibility.

When creating a new application, use of GenApi_GetDescription() is recommended.

5.6.11.1.8. Nd_GetToolTip

This function reports tool tip of the node specified by the argument node handle.

[Syntax]

```
CAM_API_STATUS  Nd_GetToolTip (
  CAM_HANDLE          hCam,
  CAM_NODE_HANDLE     hNode,
  char                *pszBuf,
  uint32_t            *puiSize
);
```

[Parameters]

Parameter		Description
hCam	[in]	Camera-Handle of target camera.
hNode	[in]	Node handle of target node.
pszBuf	[out]	A pointer to a variable that receives tool tip of the node. If NULL is specified to this argument, this function will write buffer size necessary for writing tool tip of the node to variable pointed by puiSize, without reporting tool tip string.
puiSize	[in, out]	A pointer to a variable that contains size of buffer pointed by pszBuf. If NULL is specified to pszBuf, this function will write buffer size necessary for writing tool tip of the node to variable pointed by this argument.

[Return value]

Returns result status. Refer to 5.8 Status code.

[Remarks]

This function remains for backward compatibility.

When creating a new application, use of GenApi_GetToolTip() is recommended.

5.6.11.1.9. Nd_GetRepresentation

This function reports recommended representation of the node specified by the argument node handle.

[Syntax]

```
CAM_API_STATUS  Nd_GetRepresentation (
  CAM_HANDLE              hCam,
  CAM_NODE_HANDLE         hNode,
  TC_NODE_REPRESENTATION  *peRepresentation
);
```

[Parameters]

Parameter		Description
hCam	[in]	Camera-Handle of target camera.
hNode	[in]	Node handle of target node.
peRepresentation	[out]	A pointer to a variable that receives recommended representation of the node.

[Return value]

Returns result status. Refer to 5.8 Status code.

[Remarks]

This function remains for backward compatibility.
When creating a new application, use of GenApi_GetRepresentation() is recommended.

5.6.11.1.10. Nd_GetUnit

This function reports unit name of the node value, whose node is specified by the argument node handle.

[Syntax]

```
CAM_API_STATUS  Nd_GetUnit (
  CAM_HANDLE            Cam,
  CAM_NODE_HANDLE       hNode,
  char                  *pszBuf,
  uint32_t              *puiSize
);
```

[Parameters]

Parameter		Description
hCam	[in]	Camera-Handle of target camera.
hNode	[in]	Node handle of target node.
pszBuf	[out]	A pointer to a variable that receives unit name of target node value. If NULL is specified to this argumen , this function will write buffer size necessary for writing unit name of the node to variable pointed by `puiSize`, without reporting unit name string.
puiSize	[in, out]	A pointer to a variable that contains size of buffer pointed by `pszBuf`. If NULL is specified to `pszBuf`, this function will write buffer size necessary for writing unit name of the node to variable pointed by this argument.

[Return value]

Returns result status. Refer to 5.8 Status code.

[Remarks]

This function remains for backward compatibility.
When creating a new application, use of GenApi_GetUnit() is recommended.

5.6.11.2. ICategory node functions

5.6.11.2.1. Nd_GetNumOfFeatures

This function reports number of features that the specified node contains.

[Syntax]

```
CAM_API_STATUS  Nd_GetNumOfFeatures (
  CAM_HANDLE       hCam,
  CAM_NODE_HANDLE  hNode,
  uint32_t         *puiNum
);
```

[Parameters]

Parameter		Description
hCam	[in]	Camera-Handle of target camera.
hNode	[in]	Node handle of target node.
puiNum	[out]	A pointer to a variable that receives number of features.

[Return value]

Returns result status. Refer to 5.8 Status code.

[Remarks]

This function remains for backward compatibility.
When creating a new application, use of GenApi_GetNumOfFeatures() is recommended.

5.6.11.2.2. Nd_GetFeatureByIndex

This function reports node handle of a feature node whose index in feature list of the category node is same as index specified in the argument.

[Syntax]

```
CAM_API_STATUS  Nd_GetFeatureByIndex (
  CAM_HANDLE       hCam,
  CAM_NODE_HANDLE  hNode,
  uint32_t         uiNodeIndex,
  CAM_NODE_HANDLE  *phNode
);
```

[Parameters]

Parameter		Description
hCam	[in]	Camera-Handle of target camera.
hNode	[in]	Node handle of the target category node.
uiNodeIndex	[in]	Index in feature list of the category node.
phNode	[out]	A pointer to a variable that receives node handle of a feature node, whose index is same as index specified in the arggument .

[Return value]

Returns result status. Refer to 5.8 Status code.

[Remarks]

This function remains for backward compatibility.

5.6.11.3. IInteger node functions

5.6.11.3.1. Nd_GetIntMin

This function reports the minimum valid value of the specified node.
This function assumes that target node is IInteger type node.

[Syntax]

```
CAM_API_STATUS  Nd_GetIntMin (
  CAM_HANDLE        hCam,
  CAM_NODE_HANDLE   hNode,
  int64_t           *pllMin
);
```

[Parameters]

Parameter		Description
hCam	[in]	Camera-Handle of target camera.
hNode	[in]	Node handle of target node.
pllMin	[out]	A pointer to a variable that receives the minimum valid value.

[Return value]

Returns result status. Refer to 5.8 Status code.

[Remarks]

This function remains for backward compatibility.
When creating a new application, use of GenApi_GetIntMin() is recommended.

5.6.11.3.2. Nd_GetIntMax

This function reports the maximum valid value of the specified node.
This function assumes that target node is IInteger type node.

[Syntax]

```
CAM_API_STATUS  Nd_GetIntMax (
  CAM_HANDLE        hCam,
  CAM_NODE_HANDLE   hNode,
  int64_t           *pllMax
);
```

[Parameters]

Parameter		Description
hCam	[in]	Camera-Handle of target camera.
hNode	[in]	Node handle of target node.
pllMax	[out]	A pointer to a variable that receives the maximum valid value .

[Return value]

Returns result status. Refer to 5.8 Status code.

[Remarks]

This function remains for backward compatibility.
When creating a new application, use of GenApi_GetIntMax() is recommended.

5.6.11.3.3. Nd_GetIntInc

This function reports the Increment value of the specified node.
This function assumes that target node is IInteger type node.

[Syntax]

```
CAM_API_STATUS  Nd_GetIntInc (
  CAM_HANDLE        hCam,
  CAM_NODE_HANDLE   hNode,
  int64_t           *pllInc
);
```

[Parameters]

Parameter		Description
hCam	[in]	Camera-Handle of target camera.
hNode	[in]	Node handle of target node.
pllInc	[out]	A pointer to a variable that receives the Increment value .

[Return value]

Returns result status. Refer to 5.8 Status code.

[Remarks]

This function remains for backward compatibility.
When creating a new application, use of GenApi_GetIntInc() is recommended.

5.6.11.3.4. Nd_GetIntValue

This function reports current value of the specified node.
This function assumes that target node is IInteger type node.

[Syntax]

```
CAM_API_STATUS  Nd_GetIntValue (
  CAM_HANDLE        hCam,
  CAM_NODE_HANDLE   hNode,
  int64_t           *pllValue,
  bool8_t           bVerify = false,
  bool8_t           bIgnoreCache = false,
);
```

[Parameters]

Parameter		Description
hCam	[in]	Camera-Handle of target camera.
hNode	[in]	Node handle of target node.
pllValue	[out]	A pointer to a variable that receives current value.
bVerify	[in]	Specify true to check access mode and valid value range. Specify false to simply read register value. This argument is optional. If nothing is specified to this argument, false will be selected.
bIgnoreCache	[in]	Specify true to read camera register value neglecting cached data. Specify false to use cached data if it is available. This argument is optional. If nothing is specified to this argument, false will be selected.

[Return value]

Returns result status. Refer to 5.8 Status code.

[Remarks]

This function remains for backward compatibility.
When creating a new application, use of GenApi_GetIntValue() is recommended.

5.6.11.3.5. Nd_SetIntValue

This function writes new value to the node specified by the argument node handle.
This function assumes that target node is IInteger type node.

[Syntax]

```
CAM_API_STATUS  Nd_SetIntValue (
  CAM_HANDLE        hCam,
  CAM_NODE_HANDLE   hNode,
  int64_t           llValue,
  bool8_t           bVerify = true
);
```

[Parameters]

Parameter		Description
hCam	[in]	Camera-Handle of target camera.
hNode	[in]	Node handle of target node.
llValue	[in]	New value.
bVerify	[in]	Specify true to check access mode and valid value range before writing data. Specify false to skip checking. This argument is oprional. If nothing is specified to this argument, true will be selected.

[Return value]

Returns result status. Refer to 5.8 Status code.

[Remarks]

This function remains for backward compatibility.
When creating a new application, use of GenApi_SetIntValue() is recommended.

5.6.11.4. IFloat node functions

5.6.11.4.1. Nd_GetFloatMin

This function reports the minimum valid value of the specified node.
This function assumes that target node is IFloat type node.

[Syntax]

```
CAM_API_STATUS  Nd_GetFloatMin (
  CAM_HANDLE        hCam,
  CAM_NODE_HANDLE   hNode,
  float64_t         *pdMin
);
```

[Parameters]

Parameter		Description
hCam	[in]	Camera-Handle of target camera.
hNode	[in]	Node handle of target node.
pdMin	[out]	A pointer to a variable that receives the minimum valid value.

[Return value]

Returns result status. Refer to 5.8 Status code.

[Remarks]

This function remains for backward compatibility.
When creating a new application, use of GenApi_GetFloatMin() is recommended.

5.6.11.4.2. Nd_GetFloatMax

This function reports the maximum valid value of the specified node.
This function assumes that target node is IFloat type node.

[Syntax]

```
CAM_API_STATUS  Nd_GetFloatMax (
  CAM_HANDLE        hCam,
  CAM_NODE_HANDLE   hNode,
  float64_t         *pdMax
);
```

[Parameters]

Parameter		Description
hCam	[in]	Camera-Handle of target camera.
hNode	[in]	Node handle of target node whose maximum valid value is to be reported.
dMax	[out]	A pointer to a variable that receives the maximum valid value.

[Return value]

Returns result status. Refer to 5.8 Status code.

[Remarks]

This function remains for backward compatibility.
When creating a new application, use of GenApi_GetFloatMax() is recommended.

5.6.11.4.3. Nd_GetFloatHasInc

This function reports whether the specified node has valid Increment value or not.
This function assumes that target node is IFloat type node.

[Syntax]

```
CAM_API_STATUS  Nd_GetFloatHasInc (
  CAM_HANDLE        hCam,
  CAM_NODE_HANDLE   hNode,
  bool8_t           *pbInc
);
```

[Parameters]

Parameter		Description
hCam	[in]	Camera-Handle of target camera.
hNode	[in]	Node handle of target node.
pbInc	[out]	A pointer to a variable that receives flag which describes that the node has valid Increment value or not. If true, valid Increment value exists, otherwise valid Increment value does not exist.

[Return value]

Returns result status. Refer to 5.8 Status code.

[Remarks]

This function remains for backward compatibility.
When creating a new application, use of GenApi_GetFloatHasInc() is recommended.

5.6.11.4.4. Nd_GetFloatInc

This function reports the Increment value of the specified node.
This function assumes that target node is IFloat type node.

[Syntax]

```
CAM_API_STATUS  Nd_GetFloatInc (
  CAM_HANDLE        hCam,
  CAM_NODE_HANDLE   hNode,
  float64_t         *pdInc
);
```

[Parameters]

Parameter		Description
hCam	[in]	Camera-Handle of target camera.
hNode	[in]	Node handle of target node.
pdInc	[out]	A pointer to a variable that receives the Increment value.

[Return value]

Returns result status. Refer to 5.8 Status code.

[Remarks]

This function remains for backward compatibility.
When creating a new application, use of GenApi_GetFloatInc() is recommended.

5.6.11.4.5. Nd_GetFloatDisplayNotation

This function reports the display notation of the specified node.
This function assumes that target node is IFloat type node.

[Syntax]

```
CAM_API_STATUS  Nd_GetFloatDisplayNotation (
  CAM_HANDLE               hCam,
  CAM_NODE_HANDLE          hNode,
  TC_NODE_DISPLAY_NOTATION *peDisplayNotation
);
```

[Parameters]

Parameter		Description
hCam	[in]	Camera-Handle of target camera.
hNode	[in]	Node handle of target node.
peDisplayNotation	[out]	A pointer to a variable that receives display notation.

[Return value]

Returns result status. Refer to 5.8 Status code.

[Remarks]

This function remains for backward compatibility.
When creating a new application, use of GenApi_GetFloatDisplayNotation() is recommended.

5.6.11.4.6. Nd_GetFloatDisplayPrecision

This function reports the recommended display precision value of the specified node.
This function assumes that target node is IFloat type node.

[Syntax]

```
CAM_API_STATUS  Nd_GetFloatDisplayPrecision (
  CAM_HANDLE       hCam,
  CAM_NODE_HANDLE  hNode,
  int64_t          *pllPrecision
);
```

[Parameters]

Parameter		Description
hCam	[in]	Camera-Handle of target camera.
hNode	[in]	Node handle of target node.
pllPrecision	[out]	A pointer to a variable that receives the recommended display precision value.

[Return value]

Returns result status. Refer to 5.8 Status code.

[Remarks]

This function remains for backward compatibility.
When creating a new application, use of GenApi_GetFloatDisplayPrecision() is recommended.

5.6.11.4.7. Nd_GetFloatValue

This function reports current value of the specified node.
This function assumes that target node is IFloat type node.

[Syntax]

```
CAM_API_STATUS  Nd_GetFloatValue (
  CAM_HANDLE        hCam,
  CAM_NODE_HANDLE   hNode,
  float64_t         *pdValue,
  bool8_t           bVerify = false,
  bool8_t           bIgnoreCache = false,
);
```

[Parameters]

Parameter		Description
hCam	[in]	Camera-Handle of target camera.
hNode	[in]	Node handle of target node.
pdValue	[out]	A pointer to a variable that receives current value.
bVerify	[in]	Specify true to check access mode and valid value range. Specify false to simply read register value. This argument is optional. If nothing is specified to this argument, false will be selected.
bIgnoreCache	[in]	Specify true to read camera register value neglecting cached data. Specify false to use cached data, if it is available. This argument is optional. If nothing is specified to this argument, false will be selected.

[Return value]

Returns result status. Refer to 5.8 Status code.

[Remarks]

This function remains for backward compatibility.
When creating a new application, use of GenApi_GetFloatValue() is recommended.

5.6.11.4.8. Nd_SetFloatValue

This function writes new value to the node specified by the argument node handle.
This function assumes that target node is IFloat type node.

[Syntax]

```
CAM_API_STATUS  Nd_SetFloatValue (
  CAM_HANDLE        hCam,
  CAM_NODE_HANDLE   hNode,
  float64_t         dValue,
  bool8_t           bVerify = true
);
```

[Parameters]

Parameter		Description
hCam	[in]	Camera-Handle of target camera.
hNode	[in]	Node handle of target node.
dValue	[in]	New value.
bVerify	[in]	Specify true to check access mode and valid value range before writing data. Specify false to skip checking. This argument is oprional. If nothing is specified to this argument, true will be selected.

[Return value]

Returns result status. Refer to 5.8 Status code.

[Remarks]

This function remains for backward compatibility.
When creating a new application, use of GenApi_SetFloatValue() is recommended.

5.6.11.5. IBoolean node functions

5.6.11.5.1. Nd_GetBoolValue

This function reports current value of the specified node.
This function assumes that target node is IBoolean type node.

[Syntax]

```
CAM_API_STATUS  Nd_GetBoolValue (
  CAM_HANDLE        hCam,
  CAM_NODE_HANDLE   hNode,
  bool8_t           *pbValue,
  bool8_t           bVerify = false,
  bool8_t           bIgnoreCache = false
);
```

[Parameters]

Parameter		Description
hCam	[in]	Camera-Handle of target camera.
hNode	[in]	Node handle of target node.
pbValue	[out]	A pointer to a variable that receives current value.
bVerify	[in]	Specify true to check access mode and valid value range. Specify false to simply read register value. This argument is optional. If nothing is specified to this argument, false will be selected.
bIgnoreCache	[in]	Specify true to read camera register value neglecting cached data. Specify false to use cached data if it is available, This argument is optional. If nothing is specified to this argument, false will be selected.

[Return value]

Returns result status. Refer to 5.8 Status code.

[Remarks]

This function remains for backward compatibility.
When creating a new application, use of GenApi_GetBoolValue() is recommended.

5.6.11.5.2. Nd_SetBoolValue

This function writes new value to the node specified by the argument node handle.
This function assumes that target node is IBoolean type node.

[Syntax]

```
CAM_API_STATUS  Nd_SetBoolValue (
  CAM_HANDLE        hCam,
  CAM_NODE_HANDLE   hNode,
  bool8_t           bValue,
  bool8_t           bVerify = true
);
```

[Parameters]

Parameter		Description
hCam	[in]	Camera-Handle of target camera.
hNode	[in]	Node handle of target node.
bValue	[in]	New value.
bVerify	[in]	Specify true to check access mode and valid value range before writing data. Specify false to skip checking. This argument is oprional. If nothing is specified to this argument, true will be selected.

[Return value]

Returns result status. Refer to 5.8 Status code.

[Remarks]

This function is available only for IBoolean type node (`TC_NODE_TYPE_BOOLEAN`).

This function will return error status if node type of target node is not IBoolean type.

Including TeliCamAPI.h is required.

[Example]

This function remains for backward compatibility.
When creating a new application, use of GenApi_SetBoolValue() is recommended.

5.6.11.6. IEnumeration node functions

5.6.11.6.1. Nd_GetEnumIntValue

This function reports current value of the target enumeration node as an integer value.
This function assumes that target node is IEnumeration type node.

[Syntax]

```
CAM_API_STATUS  Nd_GetEnumIntValue (
  CAM_HANDLE        hCam,
  CAM_NODE_HANDLE   hNode,
  int64_t           *pllValue,
  bool8_t           bVerify = false,
  bool8_t           bIgnoreCache = false,
);
```

[Parameters]

Parameter		Description
hCam	[in]	Camera-Handle of target camera.
hNode	[in]	Node handle of target node whose current value is to be reported.
pllValue	[out]	A pointer to a variable that receives current value.
bVerify	[in]	Specify true to check access mode and validity of the value. Specify false to simply read register value. This argument is optional. If nothing is specified to this argument, false will be selected.
bIgnoreCache	[in]	Specify true to read camera register value neglecting cached data. Specify false to use cached data if it is available, This argument is optional. If nothing is specified to this argument, false will be selected.

[Return value]

Returns result status. Refer to 5.8 Status code.

[Remarks]

This function remains for backward compatibility.
When creating a new application, use of GenApi_GetEnumIntValue() is recommended.

5.6.11.6.2. Nd_SetEnumIntValue

This function writes new value expressed as integer value to the target enumeration node.
This function assumes that target node is IEnumeration type node.

[Syntax]

```
CAM_API_STATUS  Nd_SetEnumIntValue (
  CAM_HANDLE          hCam,
  CAM_NODE_HANDLE     hNode,
  int64_t             llValue,
  bool8_t             bVerify = true
  );
```

[Parameters]

Parameter		Description
hCam	[in]	Camera-Handle of target camera.
hNode	[in]	Node handle of target node.
llValue	[in]	New value expressed in integer value.
bVerify	[in]	Specify true to check access mode and validity of the value before writing data. Specify false to skip checking. This argument is oprional. If nothing is specified to this argument, true will be selected.

[Return value]

Returns result status. Refer to 5.8 Status code.

[Remarks]

This function remains for backward compatibility.
When creating a new application, use of GenApi_SetEnumIntValue() is recommended.

5.6.11.6.3. Nd_GetEnumStrValue

This function reports current value of the target enumeration node as an string value.
This function assumes that target node is IEnumeration type node.

[Syntax]

```
CAM_API_STATUS  Nd_GetEnumStrValue (
  CAM_HANDLE       hCam,
  CAM_NODE_HANDLE  hNode,
  char             *pszBuf,
  uint32_t         *puiSize,
  bool8_t          bVerify = false,
  bool8_t          bIgnoreCache = false
);
```

[Parameters]

Parameter		Description
hCam	[in]	Camera-Handle of target camera.
hNode	[in]	Node handle of target node whose current value is to be reported.
pszBuf	[out]	A pointer to a variable that receives string value of the target node. If NULL is specified to this argument, this function will writes buffer size necessary for writing string value of the node to variable pointed by puiSize, without reporting string value.
puiSize	[in, out]	A pointer to a variable that contains size of buffer pointed by `pszBuf`. If NULL is specified to `pszBuf`, this function will writes buffer size necessary for writing string value of the node to variable pointed by this argument.
bVerify	[in]	Specify true to check access mode and validity of the value. Specify false to simply read register value. This argument is optional. If nothing is specified to this argument, false will be selected.
bIgnoreCache	[in]	Specify true to read camera register value ignoring cached data. Specify false to use cached data, if it is available, This argument is optional. If nothing is specified to this argument, false will be selected.

[Return value]

Returns result status. Refer to 5.8 Status code.

[Remarks]

This function remains for backward compatibility.
When creating a new application, use of GenApi_GetEnumStrValue() is recommended.

5.6.11.6.4. Nd_SetEnumStrValue

This function writes new value expressed as string to the target enumeration node.
This function assumes that target node is IEnumeration type node.

[Syntax]

```
CAM_API_STATUS  Nd_SetEnumStrValue (
  CAM_HANDLE        hCam,
  CAM_NODE_HANDLE   hNode,
  const char        *pszBuf,
  bool8_t           bVerify = true
);
```

[Parameters]

Parameter		Description
hCam	[in]	Camera-Handle of target camera.
hNode	[in]	Node handle of target node.
pszBuf	[in]	A pointer to a new string value.
bVerify	[in]	Specify true to check access mode and validity of the value before writing data. Specify false to skip checking. This argument is oprional. If nothing is specified to this argument, true will be selected.

[Return value]

Returns result status. Refer to 5.8 Status code.

[Remarks]

This function remains for backward compatibility.
When creating a new application, use of GenApi_SetEnumStrValue() is recommended.

5.6.11.6.5. Nd_GetNumOfEnumEntries

This function reports number of entries of the IEnumeration node specified by the argument node handle.
This function assumes that target node is IEnumeration type node.

[Syntax]

```
CAM_API_STATUS  Nd_GetNumOfEnumEntries (
  CAM_HANDLE        hCam,
  CAM_NODE_HANDLE   hNode,
  uint32_t          *puiNum
);
```

[Parameters]

Parameter		Description
hCam	[in]	Camera-Handle of target camera.
hNode	[in]	Node handle of target iEnumeration node.
puiNum	[out]	A pointer to a variable that receives the number of the entries.

[Return value]

Returns result status. Refer to 5.8 Status code.

[Remarks]

This function remains for backward compatibility.

When creating a new application, use of GenApi_GetNumOfEnumEntries() is recommended.

5.6.11.6.6. Nd_GetEnumEntryByIndex

This function reports node handle of enumeration entry node specified by index of the entry.
This function assumes that target node is IEnumeration type node.

[Syntax]

```
CAM_API_STATUS  Nd_GetEnumEntryByIndex (
  CAM_HANDLE        hCam,
  CAM_NODE_HANDLE   hNode,
  uint32_t          uiEnumIdx,
  CAM_NODE_HANDLE   *phEnumEntryNode
);
```

[Parameters]

Parameter		Description
hCam	[in]	Camera-Handle of target camera.
hNode	[in]	Node handle of target IEnumeration node.
uiEnumIdx	[in]	Index in entry list described in target node.
phEnumEntryNode	[out]	A pointer to a variable that receives node handle of the enumeration entry node.

[Return value]

Returns result status. Refer to 5.8 Status code.

[Remarks]

This function remains for backward compatibility.

5.6.11.7. IEnumEntry node functions

5.6.11.7.1. Nd_GetEnumEntryIntValue

This function reports value of enumeration entry node specified by argument entry node handle, as integer value.
This function assumes that target node is IEnumEntry type node.

[Syntax]

```
CAM_API_STATUS  Nd_GetEnumEntryIntValue (
  CAM_HANDLE        hCam,
  CAM_NODE_HANDLE   hNode,
  int64_t           *pllValue
);
```

[Parameters]

Parameter		Description
hCam	[in]	Camera-Handle of target camera.
hNode	[in]	Node handle of target enumeration entry node.
pllValue	[out]	A pointer to a variable that receives integer value of the target enumeration entry node.

[Return value]

Returns result status. Refer to 5.8 Status code.

[Remarks]

This function remains for backward compatibility.
When creating a new application, use of GenApi_GetEnumEntryIntValue() is recommended.

5.6.11.7.2. Nd_GetEnumEntryStrValue

This function reports value of enumeration entry node specified by argument entry node handle, as string value.
This function assumes that target node is IEnumEntry type node.

[Syntax]

```
CAM_API_STATUS  Nd_GetEnumEntryStrValue (
  CAM_HANDLE        hCam,
  CAM_NODE_HANDLE   hNode,
  char              *pszBuf,
  uint32_t          *puiSize
);
```

[Parameters]

Parameter		Description
hCam	[in]	Camera-Handle of target camera.
hNode	[in]	Node handle of target enumeration entry node.
pszBuf	[out]	A pointer to a variable that receives string value of the target enumeration entry node. If NULL is specified to this argument , this function will writes buffer size necessary for writing string value of the entry node to variable pointed by puiSize, without reporting string value.
puiSize	[in, out]	A pointer to a variable that contains size of buffer pointed by pszBuf. If NULL is specified to pszBuf, this function will writes buffer size necessary for writing string value of the entry node to variable pointed by this argument.

[Return value]

Returns result status. Refer to 5.8 Status code.

[Remarks]

This function remains for backward compatibility.
When creating a new application, use of GenApi_GetEnumEntryStrValue() is recommended.

5.6.11.8. ICommand node functions

5.6.11.8.1. Nd_CmdExecute

This function issues the command of ICommand type node.
This function assumes that target node is ICommand type node.

[Syntax]

```
CAM_API_STATUS  Nd_CmdExecute (
  CAM_HANDLE        hCam,
  CAM_NODE_HANDLE   hNode,
  bool8_t           bVerify = true
);
```

[Parameters]

Parameter		Description
hCam	[in]	Camera-Handle of target camera.
hNode	[in]	Node handle of target command node.
bVerify	[in]	Specify true to check access mode. Specify false to skip checking. This argument is oprional. If nothing is specified to this argument, true will be selected.

[Return value]

Returns result status. Refer to 5.8 Status code.

[Remarks]

This function remains for backward compatibility.
When creating a new application, use of GenApi_CmdExecute() is recommended.

5.6.11.8.2. Nd_GetCmdIsDone

This function reports whether the command has finished or not.
This function assumes that target node is ICommand type node.

[Syntax]

```
CAM_API_STATUS  Nd_GetCmdIsDone (
  CAM_HANDLE        hCam,
  CAM_NODE_HANDLE   hNode,
  bool8_t           *pbDone,
  bool8_t           bVerify = false
);
```

[Parameters]

Parameter		Description
hCam	[in]	Camera-Handle of target camera.
hNode	[in]	Node handle of target command node.
pbDone	[out]	A pointer to a variable that receives current execution status . True means that the command has finished. False means that the command has not finished.
bVerify	[in]	Specify true to check access mode. Specify false to skip checking. This argument is oprional. If nothing is specified to this argument, false will be selected.

[Return value]

Returns result status. Refer to 5.8 Status code.

[Remarks]

This function remains for backward compatibility.
When creating a new application, use of GenApi_GetDmdIsDone() is recommended.

5.6.11.9. IString node functions

5.6.11.9.1. Nd_GetStrValue

This function reports string value of the node specified by the argument node handle.
This function assumes that target node is IString node.

[Syntax]

```
CAM_API_STATUS  Nd_GetStrValue (
  CAM_HANDLE        hCam,
  CAM_NODE_HANDLE   hNode,
  char              *pszBuf,
  uint32_t          *puiSize,
  bool8_t           bVerify = false,
  bool8_t           bIgnoreCache = false,
);
```

[Parameters]

Parameter		Description
hCam	[in]	Camera-Handle of target camera.
hNode	[in]	Node handle of target node.
pszBuf	[out]	A pointer to a variable that receives string value of target node. If NULL is specified to this argument , this function will writes buffer size necessary for writing string value of the node to variable pointed by puiSize, without reporting string value.
puiSize	[in, out]	A pointer to a variable that contains size of buffer pointed by `pszBuf`. If NULL is specified to `pszBuf`, this function will writes buffer size necessary for writing string value of the node to variable pointed by this argument.
bVerify	[in]	Specify true to check access mode and validity of the value. Specify false to simply read register value. This argument is optional. If nothing is specified to this argument, false will be selected.
bIgnoreCache	[in]	Specify true to read camera register value neglecting cached data. Specify false to use cached data if it is available. This argument is optional. If nothing is specified to this argument, false will be selected.

[Return value]

Returns result status. Refer to 5.8 Status code.

[Remarks]

This function remains for backward compatibility.
When creating a new application, use of GenApi_GetStrValue() is recommended.

5.6.11.9.2. Nd_SetStrValue

This function writes new string value to the node specifid by the argument node handle.
This function assumes that target node is IString type node.

[Syntax]

```
CAM_API_STATUS  Nd_SetStrValue (
  CAM_HANDLE        hCam,
  CAM_NODE_HANDLE   hNode,
  const char        *pszBuf,
  bool8_t           bVerify = true
);
```

[Parameters]

Parameter		Description
hCam	[in]	Camera-Handle of target camera.
hNode	[in]	Node handle of target node.
pszBuf	[in]	A pointer to a new string value.
bVerify	[in]	Specify true to check access mode and validity of the value. Specify false to skip checking. This argument is optional. If nothing is specified to this argument, false will be selected.

[Return value]

Returns result status. Refer to 5.8 Status code.

[Remarks]

This function remains for backward compatibility.
When creating a new application, use of GenApi_SetStrValue() is recommended.

5.6.11.10. Chunk functions

5.6.11.10.1. Chunk_AttachBuffer

Attach the buffer to the GenICam chunk adapter.

[Syntax]

```
CAM_API_STATUS  Chunk_AttachBuffer (
  CAM_STRM_HANDLE   hStrm,
  void              *pvPayloadBuf,
  uint32_t          uiPayloadSize
);
```

[Parameters]

Parameter		Description
hStrm	[in]	Stream-Handle of the target stream interface.
pvPayloadBuf	[in]	A pointer to the buffer storing the received payload data.
uiPayloadSize	[in]	The size of the received payload data.

[Return value]

Returns result status. Refer to 5.8 Status code.

[Remarks]

This function remains for backward compatibility.
When creating a new application, use of GenApi_ChunkAttachBuffer() is recommended.

5.6.11.11. Others

5.6.11.11.1. Misc_GetLastGenICamError

This function reports detail error information about CAM_API_STS_GENICAM_ERR.

[Syntax]

```
CAM_API_STATUS  Misc_GetLastGenICamError (
  CAM_GENICAM_ERR_MSG      *peErrMsg
);
```

[Parameters]

Parameter		Description
peErrMsg	[out]	A pointer to a variable that receives detail error information.

[Return value]

Returns result status. Refer to 5.8 Status code.

[Remarks]

This function remains for backward compatibility.
When creating a new application, use of GenApi_GetLastError() is recommended.

5.7. Utility functions

5.7.1. Image format converter

5.7.1.1. PrepareLUT

This function sets data to lookup tables used in image format converter to make them available.

[Syntax]

```
CAM_API_STATUS  PrepareLUT (void);
```

[Return value]

Returns result status. Refer to 5.8 Status code.

[Remarks]

User application should call this function once, before calling image format converter.
Including TeliCamUtl.h is required.

5.7.1.2. Conv*ToBGRA

These functions convert source image data to BGRA format data. This utility library provides BGRA converters for 23 kinds of pixel formats source image data.

BGRA format is the format that data of a pixel consists of four 8bit components (B: blue, G: green, R: red, A: alpha (transparency)). B data is located at the smallest address and A data is located at the largest address. Data layout of BGRA format is same as image data portion of 32bit ARGB format Bitmap object. Value 0xFF is set to transparency component (A) in this function.

[Syntax]

```
CAM_API_STATUS  Conv*ToBGRA (
  void              *pvDstBGRA,
  void              *pvSrc,
  uint32_t          uiWidth,
  uint32_t          uiHeight
);
```

[Functions]

There are 23 BGRA converters in this library, which have same syntax.

Functions	Source PixelFormat	PixelFormat ID
ConvMono8ToBGRA	Mono8	0x01080001
ConvMono10ToBGRA	Mono10	0x01100003
ConvMono12ToBGRA	Mono12	0x01100005
ConvMono16ToBGRA	Mono16	0x01100007
ConvByrGR8ToBGRA	BayerGR8	0x01080008
ConvByrRG8ToBGRA	BayerRG8	0x01080009
ConvByrGB8ToBGRA	BayerGB8	0x0108000A
ConvByrBG8ToBGRA	BayerBG8	0x0108000B
ConvByrGR10ToBGRA	BayerGR10	0x0110000C
ConvByrRG10ToBGRA	BayerRG10	0x0110000D
ConvByrGB10ToBGRA	BayerGB10	0x0110000E
ConvByrBG10ToBGRA	BayerBG10	0x0110000F
ConvByrGR12ToBGRA	BayerGR12	0x01100010
ConvByrRG12ToBGRA	BayerRG12	0x01100011
ConvByrGB12ToBGRA	BayerGB12	0x01100012
ConvByrBG12ToBGRA	BayerBG12	0x01100013
ConvRGB8PToBGRA	RGB8 (RGB8Packed)	0x02180014
ConvBGR8PToBGRA	BGR8 (BGR8Packed)	0x02180015
ConvBGR10PToBGRA	BGR10 (BGR10Packed)	0x02300019
ConvBGR12PToBGRA	BGR12 (BGR12Packed)	0x0230001B
ConvYUV411PToBGRA	YUV411_8_UYYVYY ((YUV411Packed)	0x020C001E
ConvYUV422PToBGRA	YUV422_8_UYVY (YUV422Packed)	0x0210001F
ConvYUV444PToBGRA	YUV8_UYV (YUV444Packed)	0x02180020

[Parameters]

Parameter		Description
puiDstBGRA	[out]	A pointer to destination image data that receives converted BGRA data. Memory should be allocated to this beforehand.
pvSrc	[in]	A pointer to source image data.
uiWidth	[in]	Width of source image data in pixel. Width must be multiple of 4.
uiHeight	[in]	Height of source image data in pixel.

[Return value]

Returns result status. Refer to 5.8 Status code.

[Remarks]

Including TeliCamUtl.h is required.

5.7.1.3. Conv*ToBGR

These functions convert source image data to BGR format data. This utility library provides BGR converters for 23 kinds of pixel formats source image data.

BGR format is the format that data of a pixel consists of three 8bit components (B: blue, G: green, R: red). B data is located at the smallest address and R data is located at the largest address. Data layout of BGR format is same as image data portion of 24bit RGB format Bitmap object.

[Syntax]

```
CAM_API_STATUS  Conv*ToBGR (
  void              *pvDstBGR,
  void              *pvSrc,
  uint32_t          uiWidth,
  uint32_t          uiHeight
);
```

[Functions]

There are 23 BGR converters in this library, which have same syntax.

Functions	Source PixelFormat	PixelFormat ID
ConvMono8ToBGR	Mono8	0x01080001
ConvMono10ToBGR	Mono10	0x01100003
ConvMono12ToBGR	Mono12	0x01100005
ConvMono16ToBGR	Mono16	0x01100007
ConvByrGR8ToBGR	BayerGR8	0x01080008
ConvByrRG8ToBGR	BayerRG8	0x01080009
ConvByrGB8ToBGR	BayerGB8	0x0108000A
ConvByrBG8ToBGR	BayerBG8	0x0108000B
ConvByrGR10ToBGR	BayerGR10	0x0110000C
ConvByrRG10ToBGR	BayerRG10	0x0110000D
ConvByrGB10ToBGR	BayerGB10	0x0110000E
ConvByrBG10ToBGR	BayerBG10	0x0110000F
ConvByrGR12ToBGR	BayerGR12	0x01100010
ConvByrRG12ToBGR	BayerRG12	0x01100011
ConvByrGB12ToBGR	BayerGB12	0x01100012
ConvByrBG12ToBGR	BayerBG12	0x01100013
ConvRGB8PToBGR	RGB8 (RGB8Packed)	0x02180014
ConvBGR8PToBGR	BGR8 (BGR8Packed)	0x02180015
ConvBGR10PToBGR	BGR10 (BGR10Packed)	0x02300019
ConvBGR12PToBGR	BGR12 (BGR12Packed)	0x0230001B
ConvYUV411PToBGR	YUV411_8_UYYVYY ((YUV411Packed)	0x020C001E
ConvYUV422PToBGR	YUV422_8_UYVY (YUV422Packed)	0x0210001F
ConvYUV444PToBGR	YUV8_UYV (YUV444Packed)	0x02180020

[Parameters]

Parameter		Description
puiDstBGR	[out]	A pointer to destination image data that receives converted BGR data. Memory should be allocated to this beforehand.
pvSrc	[in]	A pointer to source image data.
uiWidth	[in]	Width of source image data in pixel. Width must be multiple of 4.
uiHeight	[in]	Height of source image data in pixel.

[Return value]

Returns result status. Refer to 5.8 Status code.

[Remarks]

Including TeliCamUtl.h is required.

5.7.1.4. ConvImage

This function converts various type of source image data to BGR or BGRA format data. This utility uses image converter functions in section 5.7.1.2 and 5.7.1.3.

[Syntax]

```
CAM_API_STATUS ConvImage (
  DST_FORMAT        eDstFormat,
  CAM_PIXEL_FORMAT  uiSrcPixelFormat,
  bool8_t           bBayreConversion,
  void              *pvDst,
  void              *pvSrc,
  uint32_t          uiWidth,
  uint32_t          uiHeight
);
```

[Parameters]

Parameter		Description
eDstFormat	[in]	Output image format.
uiSrcPixelFormat	[in]	PixelFormat of source image data.
bBayerConversion	[in]	Flag for converting Bayer type source image regarding color filter on each pixel or regardless of color filter. If false is specified, image data is treates as grayscale image data. If source PixelFormat is not Bayer type, this value is ignored.
pvDst	[out]	A pointer to destination image data that receives Converted image data. Memory should be allocated to this beforehand.
pvSrc	[in]	A pointer to source image data.
uiWidth	[in]	Width of source image data in pixel. Width must be multiple of 4.
uiHeight	[in]	Height of source image data in pixel.

[*DST_FORMAT* Enumeration]

Member	Description
DST_FMT_BGRA32	Convert to BGRA format (32 bits) data.
DST_FMT_BGR24	Convert to BGR format (24 bits) data.

[Return value]

Returns result status. Refer to 5.8 Status code.

[Remarks]

Including TeliCamUtl.h is required.

[Example]

C++

```
  CAM_API_STATUS    uiStatus;
  uint8_t               *pucImgBGR;
  void                  *pvImgSrc;
  uint32_t              uiWidth;
  uint32_t              uiHeight;
  CAM_PIXEL_FORMAT      uiPixelFormat;

  // Initialize lookup table
  uiStatus = PrepareLUT();
  if (uiStatus != CAM_API_STS_SUCCESS)
    return -1;

  // Get Soure image and set image size and PixelFormat
  uiStatus = GetCamWidth(m_hCam, &uiWidth);
  if (uiStatus != CAM_API_STS_SUCCESS)
    return -1;

  uiStatus = GetCamHeight(m_hCam, &uiHeight);
  if (uiStatus != CAM_API_STS_SUCCESS)
    return -1;

  uiStatus = GetCamPixelFormat(m_hCam, &uiPixelFormat);
  if (uiStatus != CAM_API_STS_SUCCESS)
    return -1;

  // TODO: add your handling code here


  // Allocate memory to BGR buffer (uiWidth should be multiple of 4)
  pucImgBGR = new uint8_t[uiWidth * uiHeight * 3];
  if (pucImgBGR == NULL)
    return -1;

// Convert source image to BGR
uiStatus = ConvImage(DST_FMT_BGR24, uiPixelFormat, true,
             (void *)pucImgBGR, (void *)pvImgSrc, uiWidth, uiHeight);

if (uiStatus != CAM_API_STS_SUCCESS)
    return -1;

  // TODO: add your handling code here
```

5.7.2. Others

5.7.2.1. BitPerPixel

This function returns bit per pixel data of the argument PixelFormat.
When pixel data consists of multiple components, summation of all components bit count will be returned.

[Syntax]

```
uint8_t  BitPerPixel (
  CAM_PIXEL_FORMAT      uiPixelFormat
);
```

[Parameters]

Parameter		Description
uiPixelFormat	[in]	PixelFormat.

[Return value]

Returns bit per pixel value. For example, 24 will be returned when uiPixelFormat is 'RGB8'.

[Remarks]

Refer to TeliCamPxlFmt.h about `CAM_PIXEL_FORMAT`.

Including TeliCamUtl.h is required.

5.7.2.2. DataDepth

This function returns data depth of the argument PixelFormat, in bit count.
When pixel data consists of multiple components, depth of a component will be returned.

[Syntax]

```
uint8_t  DataDepth (
  CAM_PIXEL_FORMAT      uiPixelFormat
);
```

[Parameters]

Parameter		Description
uiPixelFormat	[in]	PixelFormat.

[Return value]

Data depth value in bit count will be returned. For example, 8 will be returned when uiPixelFormat is 'RGB8'.

[Remarks]

Refer to TeliCamAPI.h about `CAM_PIXEL_FORMAT`.

Including TeliCamUtl.h is required.

5.7.2.3. IsMonochromic

This function returns whether the argument PixelFormat is monochromic or not.
Monochromic format is a format that the most significant byte of PixelFormat is “01”.
Mono8, Mono10, Mono12, Mono16, and Bayer formats are Monochromic format.

[Syntax]

```
bool8_t  IsMonochromic (
  CAM_PIXEL_FORMAT      uiPixelFormat
);
```

[Parameters]

Parameter		Description
uiPixelFormat	[in]	PixelFormat.

[Return value]

Returns true if PixelFormat is Mono8、 Mono10、 Mono12、 Mono16 or Bayer format type, otherwise returns false.

[Remarks]

Refer to TeliCamAPI.h about CAM_PIXEL_FORMAT.

Including TeliCamUtl.h is required.

5.7.2.4. IsPixelBayer

This function returns whether the argument PixelFormat is Bayer format or not.

[Syntax]

```
bool8_t  IsBayer (
  CAM_PIXEL_FORMAT      uiPixelFormat
);
```

[Parameters]

Parameter		Description
uiPixelFormat	[in]	PixelFormat.

[Return value]

Returns true if PixelFormat is Bayer type format, otherwise returns false.

[Remarks]

Refer to TeliCamAPI.h about CAM_PIXEL_FORMAT.

Including TeliCamUtl.h is required.

5.7.2.5. SaveBmp*

This function saves argument image data as a Bitmap file.
Three Functions are available, saving as 32bit ARGB Bitmap, 24bit RGB Bitmap, or 8bit indexed Bitmap (for monochrome image).
If argument path file already exists, this function will overwrite the existing file with new Bitmap file.

[Syntax]

```
CAM_API_STATUS SaveBmp* (
  void          *pvTgt,
  uint32_t      uiWidth,
  uint32_t      uiHeight,
  const char    *pszPath
);
```

[Parameters]

Functions	Bitmap Pixel Format	Remarks
SaveBmpARGB	Format32bppArgb	
SaveBmpRGB	Format24bppRgb	
SaveBmpMono	Format8bppIndexed	Uses color pallet for monochrome image.

[Parameters]

Parameter		Description
pvTgt	[in]	A pointer to target image data.
uiWidth	[in]	Width of target image in pixels.
uiHeight	[in]	Height of target image in pixels.
pszPath	[in]	File path for saving the created bitmap. Folder of the path should be accessible as writable folder.

[Return value]

Returns result status. Refer to 5.8 Status code.

[Remarks]

Including TeliCamUtl.h is required. T

5.7.2.6. ReverseImg

This function creates image reversed horizontally and / or vertically.

When source image is Bayer type, image data will be reversed as monochrome image data, which means that color filter layout of the reversed image will be different from that of source image.

For example, when BayerBG image is horizontally reversed, BayerGB image will be created as result image. When BayerBG image is vertically reversed, BayerGR image will be created as result image.

This function can handle the following PixelFormat images.
PXL_FMT_Mono8, PXL_FMT_Mono10, PXL_FMT_Mono12
PXL_FMT_BayerGR8, PXL_FMT_BayerRG8, PXL_FMT_BayerGB8, PXL_FMT_BayerBG8,
PXL_FMT_BayerGR10, PXL_FMT_BayerRG10, PXL_FMT_BayerGB10, PXL_FMT_BayerBG10,
PXL_FMT_YUV411_8, PXL_FMT_YUV422_8,
PXL_FMT_RGB8, PXL_FMT_BGR8

[Syntax]

```
CAM_API_STATUS ReverseImg (
  void              *pvDst,
  void              *pvSrc,
  CAM_PIXEL_FORMAT  uiPixelFormat,
  uint32_t          uiWidth,
  uint32_t          uiHeight,
  bool8_t           bRevX,
  bool8_t           bRevY
);
```

[Parameters]

Parameter		Description
pvDst	[in]	A pointer to destination image data that receives reversed image.
pvSrc	[in]	A pointer to source image data.
uiPixelFormat	[in]	PixelFormat of source image.
uiWidth	[in]	Width of source image in pixel.
uiHeight	[in]	Height of source image in pixel.
bRevX	[in]	Flag for reversing horizontally. If true, horizontally.reversed image will be created.
bRevY	[in]	Flag for reversing vertically. If true, vertically.reversed image will be created.

[Return value]

Returns result status. Refer to 5.8 Status code.

[Remarks]

Refer to TeliCamAPI.h about CAM_PIXEL_FORMAT.

Including TeliCamUtl.h is required.

5.8. Status code

Most functions in TeliCamAPI returns status code in the following table.

CAM_API_STATUS	Value	Description
CAM_API_STS_SUCCESS	0x00000000	It Succeeded
CAM_API_STS_NOT_INITIALIZED	0x00000001	The initialization for API has never been performed.
CAM_API_STS_ALREADY_INITIALIZED	0x00000002	The initialization for API has already been performed.
CAM_API_STS_NOT_FOUND	0x00000003	API detected no cameras.
CAM_API_STS_ALREADY_OPENED	0x00000004	Camera or the other object has been already opened,
CAM_API_STS_ALREADY_ACTIVATED	0x00000005	Camera or the other object has been already activated,
CAM_API_STS_INVALID_CAMERA_INDEX	0x00000006	The specified camera index is invalid.
CAM_API_STS_INVALID_CAMERA_HANDLE	0x00000007	The specified camera handle is invalid.
CAM_API_STS_INVALID_NODE_HANDLE	0x00000008	The specified node handle is invalid.
CAM_API_STS_INVALID_STREAM_HANDLE	0x00000009	The specified stream handle is invalid.
CAM_API_STS_INVALID_REQUEST_HANDLE	0x0000000A	The specified request handle is invalid.
CAM_API_STS_INVALID_EVENT_HANDLE	0x0000000B	The specified event handle is invalid.
CAM_API_STS_INVALID_PARAMETER	0x0000000D	The specified parameter is invalid.
CAM_API_STS_BUFFER_TOO_SMALL	0x0000000E	The specified buffer is too small.
CAM_API_STS_NO_MEMORY	0x0000000F	To allocate internal buffer of API is unsuccessfu
CAM_API_STS_MEMORY_NO_ACCESS	0x00000010	To access to the specified buffer is unsuccessful.
CAM_API_STS_NOT_IMPLEMENTED	0x00000011	Feature is not implemented in the camera or API. This status may be also returned when wrong register name or node name is used,
CAM_API_STS_TIMEOUT	0x00000012	The timeout occurred.
CAM_API_STS_CAMERA_NOT_RESPONDING	0x00000013	The specified camera does not send response. The cable which has connected the camra may have separated. Please check the connection,
CAM_API_STS_EMPTY_COMPLETE_QUEUE	0x00000014	The request in the Complete Queue is empty.
CAM_API_STS_NOT_READY	0x00000015	The target is not ready state.
CAM_API_STS_ACCESS_MODE_SET_ERR	0x00000016	Api failed to set access mode of the camera.
CAM_API_STS_IO_DEVICE_ERROR	0x00000020	Controller caused an I/O error.
CAM_API_STS_LOGICAL_PARAM_ERROR	0x00000021	Passed parameters have logical error(s).
CAM_API_STS_XML_LOAD_ERR	0x00000101	Api failed to load XML file (camera description file).
CAM_API_STS_GENICAM_ERR	0x00000102	Error occurred in GenApi.
CAM_API_STS_DLL_LOAD_ERR	0x00000103	Api failed to load shared object files.
CAM_API_STS_NO_SYSTEM_RESOURCES	0x000005AA	Insufficient system resources exist to complete the requested service.
CAM_API_STS_INVALID_ADDRESS	0x00000801	The specified addres is invalid.
CAM_API_STS_WRITE_PROTECT	0x00000802	The register is protected from writing data.
CAM_API_STS_BAD_ALIGNMENT	0x00000803	The address is not aligned to specified boundary.
CAM_API_STS_ACCESS_DENIED	0x00000804	Accessing data was denied. User application may not have access privilege.
CAM_API_STS_BUSY	0x00000805	The camera is in busy state. Try again after a while.
CAM_API_STS_NOT_READABLE	0x00000806	The target is not readable.
CAM_API_STS_NOT_WRITABLE	0x00000807	The target is not writable.

CAM_API_STATUS	Value	Description
CAM_API_STS_NOT_AVAILABLE	0x00000808	The function or parameter is not available. The controlling camera feature functions that use GenICam GenApi function are not available when GenApi module was disabled on opening the camera.
CAM_API_STS_VERIFY_ERR	0x00000809	A verify error occurred when data was written to the camera registers.
CAM_API_STS_REQUEST_TIMEOUT	0x00001001	Timeout occurred in requesting stream or event.
CAM_API_STS_RESEND_TIMEOUT	0x00001002	The StreamRequests coud not be completed in time after the first packet was received. (only in GigE Vision camera)
CAM_API_STS_RESPONSE_TIMEOUT	0x00001003	The next packet could not be received in time after somepacket was received. (Only in GigE Vision camera)
CAM_API_STS_BUFFER_FULL	0x00001004	The received data size exceeded maximum size specified.
CAM_API_STS_UNEXPECTED_BUFFER_SIZE	0x00001005	The data size actually received was different from the sizedescribed in the trailer.
CAM_API_STS_UNEXPECTED_NUMBER	0x00001006	The received packet count exceeded the maximum.
CAM_API_STS_PACKET_STATUS_ERROR	0x00001007	The packet returned error status.
CAM_API_STS_RESEND_NOT_IMPLEMENTED	0x00001008	Resend command is not implemented in the camera.
CAM_API_STS_PACKET_UNAVAILABLE	0x00001009	The packet is unavailable.
CAM_API_STS_MISSING_PACKETS	0x0000100A	A leader of the next block has been received before the completion of a frame data. Packets may be lost.
CAM_API_STS_FLUSH_REQUESTED	0x0000100B	The requestwas flushed by the user.
CAM_API_STS_TOO_MANY_PACKET_MISSING	0x0000100C	The loss pf packet exceeded the specified value (default:20) In GigE Vision camera case, band width of the network may not be enough for sending images in current settings. Enabling Jumbo-Packet or restrictimg frame rate will be required.
CAM_API_STS_FLUSHED_BY_D0EXIT	0x0000100D	The request was flushed due to a change of the power state.
CAM_API_STS_FLUSHED_BY_CAMERA_REMOVE	0x0000100E	The request was flushed due to a disconnection wth the camera.
CAM_API_STS_DRIVER_LOAD_ERR	0x0000100F	Api failed to load Driver.
CAM_API_STS_MAPPING_ERROR	0x00001010	Mapping user buffer to system-space virtual address is failed. It may be caused by low system resources.
CAM_API_STS_FILE_OPEN_ERROR	0x00002001	Api failed to open file,
CAM_API_STS_FILE_WRITE_ERROR	0x00002002	Api failed to warite data in file
CAM_API_STS_FILE_READ_ERROR	0x00002003	Api failed to read data from file.
CAM_API_STS_FILE_NOT_FOUND	0x00002004	The specified file was not found.
CAM_API_STS_INVALID_PARAMETER_FROM_CAM	0x00008002	Invalid parameter error returned from camera.
CAM_API_STS_SI_PAYLOAD_SIZE_NOT_ALIGNED	0x0000A003	The value written to the SI streaming size registers is not aligned to Payload Size Alignment value of the SI Info register.
CAM_API_STS_DATA_DISCARDED	0x0000A100	Some data in the block has been discarded.
CAM_API_STS_DATA_OVERRUN	0x0000A101	The camere cannot send all data because the data

CAM_API_STATUS	Value	Description
		does not fit within the programmed SIRM register settings.
CAM_API_STS_UNSUCCESSFUL	0xFFFFFFFF	The other error occurred.

6. Sample source code

The knowledge about GenICam, GigE Vision, USB3 Vision, and IIDC2 register map will be useful to understand sample source code. The detail or latest information of these standards can be obtained from home page of these standards.

GenICam	http://www.emva.org/cms/index.php?idcat=47&lang=1
GigE Vision	http://www.visiononline.org/vision-standards-details.cfm?type=5
USB3 Vision	http://www.visiononline.org/vision-standards-details.cfm?type=11
IIDC2	http://jiia.org/standard_dl/ngcp-wg/

6.1. Sample source coude under Windows

TeliCamSDK provides 9 sample source code projects in the following table for C++ user's reference. Samples for .NET are described in the TeliCamDNet API Library Manual. More sample sources will be provided in our home page subsequently.

Sample name	Language	UI	Function	Remarks
Camera_Information	VC2005	CUI	Display of camera information.	Use API in section 5.1, section 5.2.
Camera_ControllingFunction	VC2005	CUI	Acquisition and setting of parameters.	Use API in section 5.1, section 5.2, section 5.5
Stream_FreerunCallback	VC2005	CUI	Pixel values in the image shown. (Continuous capture)	Use API in section 5.1, section 5.2, section 5.3
Stream_FreerunLockBuffer	VC2005	CUI		Use API in section 5.1, section 5.2,
Stream_SWTrgReadCurrentImage	VC2005	CUI	Pixel values in the image shown.	Use API in section 5.1, section 5.2, section 5.3
Stream_Lowlevel	VC2005	CUI	Pixel values in the image shown. (Continuous capture)	Use API in section 5.1, section 5.2, section 5.3
CameraEvent	VC2005	CUI	Get "FrameTrigger" event.	Use API in section 5.1, section 5.2, section 5.4
MultiCamera	VC2005	GUI	Draw images of up to 4 cameras in a form.	Use API in section 5.1, section 5.2, section 5.3 section 5.4, section 5.5, section 5.5
MultiCameraPrimitive	VC2005	GUI	Draw images of up to 4 cameras in a form.	Use API in section 5.1, section 5.2, section 5.3 section 5.4, section 5.5, section 5.5

These sample source code projects are installed in "Samples" folder in the folder that TeliCamSDK is installed. Administrator privilege is necessary to write or modify data in that folder, in OS newer than Windows 7. Please copy sample source code project to user writable folder such as "My documents" folder, before compiling it.

6.2. Sample source coude under Linux

TeliCamSDK for Linux provides sample applications in the following table for user's reference.

Sample name	UI	Function
Camera_Information	CUI	Display of camera information.
Camera_ControllingFunction	CUI	Acquisition and setting of parameters.
Stream_FreerunCallback	CUI	Continuous capture of images using the Callback function.
Stream_FreerunLockBuffer	CUI	Continuous capture of images using the LockBuffer function.
Stream_SWTrgReadCurrentImage	CUI	Image capture of software trigger using ReadCurrentImage function.
Stream_LowLevel	CUI	Continuous capture of images using the low level stream function.
CameraEvent	CUI	Get "FrameTrigger" event.
MultiCamera	GUI	Draw images of up to 4 cameras.

Sample applications are located in the following directory.

$HOME/TeliCamSDK/samples

To compile and run applications using TeliCamSDK, you must set the environment variables.

TELICAMSDK=/opt/TeliCamSDK
export TELICAMSDK

export
LD_LIBRARY_PATH=$TELICAMSDK/lib:$TELICAMSDK/genicam/bin/Linux64_x64:$LD_LIBRARY_PATH

This can be set by the shell script.

source /opt/TeliCamSDK/set_env.sh

6.2.1. Console sample

Follow the instructions below to compile console samples :

1. Open a terminal window.
2. Move the sample directory.

   cd $HOME/TeliCamSDK/samples/CPP/ConsoleSamples
3. Compile console projects.

   make

If successful, a binary file will be generated in each project directory.
When you run the script in each project directory, you can run the application.

For example:

cd ./GrabStream_FreerunUsingCallback
sh ./execute_GrabStream_FreerunUsingCallback.sh

6.2.2. Qt sample

To compile, you need to install Qt.

Follow the instructions below to compile Qt samples :

1. Open a terminal window.
2. Move the sample directory.

   For example:

   cd $HOME/TeliCamSDK/samples/CPP/QtSamples/Qt5/MultiCamera
3. Set environments, and run Qt Creator.

   sh ./set_qt_env.sh

7. Others

7.1. Disclaimer

The disclaimer of this Software is described in another “License Agreement TeliCamSDK Eng.pdf”.
Make sure to read this Agreement carefully before using it.
Refer to the following folder.

Windows version : [TeliCamSDK installation folder]/Licenses
Linux version : /opt/TeliCamSDK/licenses

7.2. License

TeliCamSDK consists of multiple, independent software components. Each software component is copyrighted by a third party. TeliCamSDK uses software components that are distributed as freeware under a third-party end-user license agreement or copyright notice (hereinafter referred to as a “EULA”).

Some EULAs require that the source code of the applicable component be disclosed as the condition for distributing the software component in executable format. You can check the software components subject to such EULA requirements. For more information, please contact our inquiries described in section 7.4.

Toshiba Teli corporation provides a warranty for TeliCamSDK under conditions set forth by Toshiba Teli corporation. (See the following documents.

Windows version : “License Agreement TeliCamSDK for Eng.txt”,
“License Agreement TeliCamSDK for Sample Eng.txt”
Linux version : “License Agreement TeliCamSDK for Linux Eng.txt”,
“License Agreement TeliCamSDK for Linux Sample Eng.txt”)

However, some of the software components distributed under an EULA are made available for use by the user on the assumption that they are not copyrighted or warranted by a third party. These software components are licensed to the user free of charge and therefore not covered by any warranty within the scope of the applicable laws. These software components are not subject to any copyrights or other third-party rights and are provided in “as is” condition without any warranty, whether express or implied. “Warranty” here includes, but not limited to, an implied warranty for marketability
or fitness for specific uses. All risks associated with the quality or performance of these software components are assumed by the user.

EULAs are included in the following directory:

Windows version : [TeliCamSDK installation folder]/Licenses
Linux version : /opt/TeliCamSDK/licenses

Toshiba Teli corporation shall not be liable whatsoever for any cost of repair or correction or other incidental expense incurred in connection with a defect found in any of these software components. Unless specified under the applicable laws or in a written agreement, a party who changes or redistributes the software with consent from the copyright holders or based on the aforementioned licenses shall not be held liable whatsoever for any loss arising from the use of or inability to use such software components. The same applies even when the copyright holders or relevant third parties have been informed of the possibility of such loss. “Loss” here includes normal, special, incidental and indirect loss (including, but not limited to, the loss of data or its accuracy; loss incurred by the user or any third party; and interface incompatibility with other software). Please read each EULA for details on the

use conditions and items that must be observed regarding these software components.

The table below lists the software components using in TeliCamSDK, which are subject to EULAs.
The user should read the applicable EULAs carefully before using these software components.

Windows version

Project name	Project license
GenICam GenApi	GenICam License

Linux version

Project name	Project license
gcc libgcc	GPLv3.txt and gcc-exception.txt (GPLv3 with GCC Runtime Library Exception)
gcc libstdc++	GPLv3.txt and gcc-exception.txt (GPLv3 with GCC Runtime Library Exception)
glibc	LGPLv2.1
libteliusb (libusb)	LGPLv2.1
GenICam	GenICam license
Qt	LGPLv2.1 and Digia Qt LGPL Exception version 1.1

GenICam GenApi uses the following third party software.

Project name	Project license
MathParser	LGPLv2.1
Log4Cpp	LGPLv2.1
CppUnit	LGPLv2.1
CLSerAll	NI license
xs3p	DSTC license
xxhash	xxhash license
XSLTProc	MIT license
XSDe	Proprietary

TeliCamSDK redistributes the binaries of LGPL-applied software, and for these source code only, you have the right to obtain, modify and redistribute it in accordance with the LGPL provisions.
To the customer who wants the source code, we write to the media (CD - ROM etc.) and send it by post. Customers must pay for actual expenses such as shipping fee. If you want, please contact our inquiries described in section 7.4. We distribute source code only for open source software that you have right to obtain. (Source code of TeliCamSDK is not included.) Please understand beforehand that we can not answer questions about the content of the source code etc.

Microsoft, Windows, Windows XP, Windows Vista, Windows 7, Windows 8.1, Windows 10 and Visual C++ are the trademark or the registered trademark of Microsoft Corporation.
GigE Vision™ and USB3 Vision™ are camera interface standard defined by AIA (Automated Imaging Association).
GenICam™ is the trademark or the registered trademark of EMVA (European Machine Vision association).
Furthermore, the trade name used in this document is the trademark or the registered trademark of each company.

7.3. Revision History

Date	Version	Description
2014/09/11	1.0.0	Created the initial version
2014/10/17	1.0.1	● Added the status code. ●Changed the name of function. (Cam_PortReset → Cam_ResetPort)
2014/11/14	1.0.2	● Changed default uiApiBufferCount parameter value of Strm_OpenSimple to 8. (Previous default value was 5.) ● Appended section 5.5.27 UserDefiedName(DeviceUserID), corresponding to supplement of these functions to TeliCamAPI.
2015/02/25	1.0.3	● Replaced body of setion 2. Configuration. ● Changed description in section 5.2.2, 5.2.3, 5.3, 5.3.1.1, 5.3.2.1, 5.4, 5.4.1.1, 5.4.2.1 due to the following specification change. ✓ Opening a camera that the other application is using is made possible. ✓ Installation of GenApi becomes not necessary when false is specified to *bUseGenICam* in Cam_Open() or Cam_OpenFromInfo() ● Appended unit of argument value to puiHbTimeout in section 5.2.8. Cam_GetHeartbeat. ● Appended [Example] in section 5.2.9. Cam_SetHeartbeat. ● Amended namespace of XML features in section 5.6.1. INode functions. ● Inserted section 5.6.3. ICategory node function, and moved “Nd_GetNumOfFeatures” and “Nd_GetFeatureByIndex” to the inserted section. ● Inserted section 5.6.7. IEnumEntry node function, and moved “Nd_GetEnumEntryIntValue” and “Nd_GetEnumEntryStrValue” to the inserted section. ● Inserted CAM_API_STS_NOT_READABLE, CAM_API_STS_NOT_WRITABLE, and CAM_API_STS_NOT_AVAILABLE in section 5.8 Status code.
2015/06/12	1.0.4	● Added the Windows 8.1 description
2015/10/21	1.0.5	● Added the camera connection possibility number description
2016/07/12	2.0.0	● Added the Windows 10 description ● Deleted the Windows XP & Windows Vista description ● Supported the FrameBurst function. ● Added new function. (`SetCamLineModeAll()`)
2016/12/16	2.0.1	● Added functions related to Chunk feature. ● Added functions related to UserSetControl.
2017/06/13	2.0.2	● Added Strm_Abort() function. ● Added some Status code. ● Modified the description of each item.
2017/09/05	2.0.3	● Modified setting range of argument uiApiBufferCount of Strm_OpenSimple(). (Minimum value : 3 → 1 , Maximum value : 30 → 128)
2018/01/25	2.0.4	● Added description of new controlling camera feature functions. ● Added description of new GenICam functions. ● Added description of new status codes.
2018/06/29	2.0.5	● Added multicast description. ● Added the single frame mode description. ● Added description of new functions. (Cam_GetMulticast() , Cam_SetMultica

		st() , ExecuteCamAcquisitionStart()) ● Added description of new status codes. ● Modified the “6. Sample source code”.
2019/02/07	2.0.6	● Modified the “3. Operation Environment”. ● Modified explanation of Evt_Activate(), Evt_Deactivate(). ● Added description of new status codes.
2020/01/08	3.0.0	● Integrate Windows and Linux version library manual. ● Added functions corresponding to HighFramerateMode and ShortExposure Mode of the camera.

7.4. Inquiry

If you need help with TeliCamSDK, GigE Vision camera, USB3 Vision camera, please visit the following website :

https://secure.toshiba-teli.co.jp/ttfa/web/faq/top.html

If you still can not solve the problem, please contact “inquiries” on the following web site :

https://www.toshiba-teli.co.jp/en/support/contact/industrial.htm

