

TeliCamAPI

Library manual

Version 2.0.3(2017/09/05)

TOSHIBA TELI CORPORATION

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

Contents

1. Introduction	9
2. Configuration.....	10
3. Operation Environment	11
4. Library	12
4.1. Usage	13
4.1.1. Initialize and terminate API	13
4.1.2. Enumerate cameras	13
4.1.3. Open and close camera	13
4.1.4. Control camera and acquire information on camera	13
4.1.5. Streaming control	15
4.1.5.1. Acquiring image data using High-Level API functions	15
4.1.5.2. Acquiring image data using Low-Level API functions	20
4.1.6. CameraEvent control.....	24
4.1.6.1. CameraEvent notification using High-Level API functions	24
4.1.6.2. Event notification using Low-Level API functions	26
4.1.7. Camera Access mode (Control Channel Privilege).....	29
4.1.8. Heartbeat process	29
4.2. SDK Installation	30
4.3. SDK Uninstallation	30
4.4. Set up development environment	30
5. Library functions.....	31
5.1. System functions.....	31
5.1.1. Sys_Initialize.....	31
5.1.2. Sys_Terminate.....	32
5.1.3. Sys_GetInformation.....	33
5.1.4. Sys_GetNumOfCameras.....	35
5.2. Camera functions	36
5.2.1. Cam_GetInformation	36
5.2.2. Cam_Open	41
5.2.3. Cam_OpenFromInfo.....	44
5.2.4. Cam_Close.....	47
5.2.5. Cam_ReadReg.....	48
5.2.6. Cam_WriteReg	50
5.2.7. Cam_ResetPort.....	51
5.2.8. Cam_GetHeartbeat	53
5.2.9. Cam_SetHeartbeat.....	54
5.3. Camera streaming functions	55
5.3.1. High-Level API functions	55
5.3.1.1. Strm_OpenSimple.....	55
5.3.1.2. Strm_ReadCurrentImage.....	60
5.3.1.3. Strm_GetCurrentBufferIndex	62
5.3.1.4. Strm_LockBuffer	65
5.3.1.5. Strm_UnlockBuffer	66
5.3.1.6. Strm_SetCallbackImageAcquired	67
5.3.1.7. Strm_SetCallbackImageError	70
5.3.1.8. Strm_SetCallbackBufferBusy.....	71
5.3.2. Low-level API functions	72
5.3.2.1. Strm_Open.....	72
5.3.2.2. Strm_CreateRequest	77

5.3.2.3.	Strm_ReleaseRequest.....	79
5.3.2.4.	Strm_EnqueueRequest.....	80
5.3.2.5.	Strm_DequeueRequest	81
5.3.2.6.	Strm_FlushWaitQueue.....	82
5.3.2.7.	Strm_GetStrmReqInfo.....	83
5.3.3.	Common functions	85
5.3.3.1.	Strm_Close	85
5.3.3.2.	Strm_Start	86
5.3.3.3.	Strm_Stop	88
5.3.3.4.	Strm_Abort.....	89
5.4.	Camera event notification functions.....	90
5.4.1.	High-level API functions	90
5.4.1.1.	Evt_OpenSimple.....	90
5.4.1.2.	Evt_Activate.....	94
5.4.1.3.	Evt_Deactivate.....	97
5.4.2.	Low-Level API functions	98
5.4.2.1.	Evt_Open.....	98
5.4.2.2.	Evt_CreateRequest	103
5.4.2.3.	Evt_ReleaseRequest.....	104
5.4.2.4.	Evt_EnqueueRequest.....	105
5.4.2.5.	Evt_DequeueRequest.....	106
5.4.2.6.	Evt_FlushWaitQueue	108
5.4.3.	Common functions	109
5.4.3.1.	Evt_Close	109
5.5.	Controlling camera feature functions	110
5.5.1.	ImageFormatControl	111
5.5.1.1.	GetCamImageFormatSelector.....	111
5.5.1.2.	SetCamImageFormatSelector	112
5.5.2.	Scalable.....	113
5.5.2.1.	GetCamSensorWidth.....	113
5.5.2.2.	GetCamSensorHeight.....	114
5.5.2.3.	GetCamRoi	115
5.5.2.4.	SetCamRoi	116
5.5.2.5.	GetCamWidthMinMax.....	117
5.5.2.6.	GetCamWidth	118
5.5.2.7.	SetCamWidth.....	119
5.5.2.8.	GetCamHeightMinMax	120
5.5.2.9.	GetCamHeight	121
5.5.2.10.	SetCamHeight	122
5.5.2.11.	GetCamOffsetXMinMax.....	123
5.5.2.12.	GetCamOffsetX	124
5.5.2.13.	SetCamOffsetX.....	125
5.5.2.14.	GetCamOffsetYMinMax.....	126
5.5.2.15.	GetCamOffsetY	127
5.5.2.16.	SetCamOffsetY.....	128
5.5.3.	Binning.....	129
5.5.3.1.	GetCamBinningHorizontalMinMax	129
5.5.3.2.	GetCamBinningHorizontal	130
5.5.3.3.	SetCamBinningHorizontal.....	131
5.5.3.4.	GetCamBinningVerticalMinMax.....	132

5.5.3.5.	GetCamBinningVertical.....	133
5.5.3.6.	SetCamBinningVertical.....	134
5.5.4.	Decimation.....	135
5.5.4.1.	GetCamDecimationHorizontalMinMax.....	135
5.5.4.2.	GetCamDecimationHorizontal.....	136
5.5.4.3.	SetCamDecimationHorizontal.....	137
5.5.4.4.	GetCamDecimationVerticalMinMax.....	138
5.5.4.5.	GetCamDecimationVertical.....	139
5.5.4.6.	SetCamDecimationVertical.....	140
5.5.5.	Reverse.....	141
5.5.5.1.	GetCamReverseX.....	141
5.5.5.2.	SetCamReverseX.....	142
5.5.5.3.	GetCamReverseY.....	143
5.5.5.4.	SetCamReverseY.....	144
5.5.6.	PixelFormat.....	145
5.5.6.1.	GetCamPixelFormat.....	145
5.5.6.2.	SetCamPixelFormat.....	147
5.5.7.	TestPattern.....	148
5.5.7.1.	GetCamTestPattern.....	148
5.5.7.2.	SetCamTestPattern.....	149
5.5.8.	AcquisitionControl.....	150
5.5.8.1.	GetCamStreamPayloadSize.....	150
5.5.8.2.	GetStreamEnable.....	151
5.5.8.3.	GetCamAcquisitionFrameCountMinMax.....	152
5.5.8.4.	GetCamAcquisitionFrameCount.....	153
5.5.8.5.	SetCamAcquisitionFrameCount.....	154
5.5.8.6.	GetCamAcquisitionFrameRateControl.....	155
5.5.8.7.	SetCamAcquisitionFrameRateControl.....	156
5.5.8.8.	GetCamAcquisitionFrameRateMinMax.....	157
5.5.8.9.	GetCamAcquisitionFrameRate.....	158
5.5.8.10.	SetCamAcquisitionFrameRate.....	159
5.5.9.	ImageBuffer.....	160
5.5.9.1.	GetCamImageBufferMode.....	160
5.5.9.2.	SetCamImageBufferMode.....	161
5.5.9.3.	GetCamImageBufferFrameCount.....	162
5.5.9.4.	ExecuteCamImageBufferRead.....	163
5.5.10.	TriggerControl.....	164
5.5.10.1.	GetCamTriggerMode.....	164
5.5.10.2.	SetCamTriggerMode.....	165
5.5.10.3.	GetCamTriggerSequence.....	166
5.5.10.4.	SetCamTriggerSequence.....	168
5.5.10.5.	GetCamTriggerSource.....	169
5.5.10.6.	SetCamTriggerSource.....	170
5.5.10.7.	GetCamTriggerAdditionalParameterMinMax.....	171
5.5.10.8.	GetCamTriggerAdditionalParameter.....	172
5.5.10.9.	SetCamTriggerAdditionalParameter.....	173
5.5.10.10.	GetCamTriggerDelayMinMax.....	174
5.5.10.11.	GetCamTriggerDelay.....	175
5.5.10.12.	SetCamTriggerDelay.....	176
5.5.10.13.	ExecuteCamSoftwareTrigger.....	177

5.5.11. ExposureTime	178
5.5.11.1. GetCamExposureTimeControl.....	178
5.5.11.2. SetCamExposureTimeControl	180
5.5.11.3. GetCamExposureTimeMinMax.....	181
5.5.11.4. GetCamExposureTime	182
5.5.11.5. SetCamExposureTime.....	183
5.5.12. DigitalIoControl	184
5.5.12.1. GetCamLineModeAll.....	184
5.5.12.2. SetCamLineModeAll	185
5.5.12.3. GetCamLineInverterAll	186
5.5.12.4. SetCamLineInverterAll.....	187
5.5.12.5. GetCamLineStatusAll.....	188
5.5.12.6. GetCamUserOutputValueAll	189
5.5.12.7. SetCamUserOutputValueAll	190
5.5.12.8. GetCamLineSource	191
5.5.12.9. SetCamLineSource.....	193
5.5.13. TimerConrtol.....	194
5.5.13.1. GetCamTimerDurationMinMax	194
5.5.13.2. GetCamTimerDuration.....	195
5.5.13.3. SetCamTimerDuration	196
5.5.13.4. GetCamTimerDelayMinMax	197
5.5.13.5. GetCamTimerDelay	198
5.5.13.6. SetCamTimerDelay.....	199
5.5.13.7. GetCamTimerTriggerSource.....	200
5.5.13.8. SetCamTimerTriggerSource	201
5.5.14. Gain	202
5.5.14.1. GetCamGainMinMax	202
5.5.14.2. GetCamGain	203
5.5.14.3. SetCamGain	204
5.5.14.4. GetCamGainAuto	205
5.5.14.5. SetCamGainAuto	206
5.5.15. BlackLevel	207
5.5.15.1. GetCamBlackLevelMinMax	207
5.5.15.2. GetCamBlackLevel	208
5.5.15.3. SetCamBlackLevel	209
5.5.16. Gamma.....	210
5.5.16.1. GetCamGammaMinMax	210
5.5.16.2. GetCamGamma.....	211
5.5.16.3. SetCamGamma	212
5.5.17. WhiteBalance	213
5.5.17.1. GetCamBalanceRatioMinMax	213
5.5.17.2. GetCamBalanceRatio	215
5.5.17.3. SetCamBalanceRatio	216
5.5.17.4. GetCamBalanceWhiteAuto.....	217
5.5.17.5. SetCamBalanceWhiteAuto	218
5.5.18. Hue.....	219
5.5.18.1. GetCamHueMinMax	219
5.5.18.2. GetCamHue.....	220
5.5.18.3. SetCamHue	221
5.5.19. Saturation	222

5.5.19.1.	GetCamSaturationMinMax	222
5.5.19.2.	GetCamSaturation	223
5.5.19.3.	SetCamSaturation	224
5.5.20.	Sharpness	225
5.5.20.1.	GetCamSharpnessMinMax.....	225
5.5.20.2.	GetCamSharpness	226
5.5.20.3.	SetCamSharpness.....	227
5.5.21.	ColorCorrectionMatrix	228
5.5.21.1.	GetCamColorCorrectionMatrixMinMax.....	229
5.5.21.2.	GetCamColorCorrectionMatrix	230
5.5.21.3.	SetCamColorCorrectionMatrix.....	231
5.5.22.	LUT	232
5.5.22.1.	GetCamLutEnable	232
5.5.22.2.	SetCamLutEnable.....	233
5.5.22.3.	GetCamLutValue	234
5.5.22.4.	SetCamLutValue	235
5.5.23.	UserSetControl	236
5.5.23.1.	ExecuteCamUserSetLoad	236
5.5.23.2.	ExecuteCamUserSetSave	238
5.5.23.3.	ExecuteCamUserSetQuickSave.....	239
5.5.23.4.	GetCamUserSetDefault	240
5.5.23.5.	SetCamUserSetDefault	241
5.5.23.6.	ExecuteCamUserSetSaveAndSetDefault.....	242
5.5.24.	SequentialShutter	243
5.5.24.1.	GetCamSequentialShutterEnable	243
5.5.24.2.	SetCamSequentialShutterEnable.....	244
5.5.24.3.	GetCamSequentialShutterTerminateAt.....	245
5.5.24.4.	GetCamSequentialShutterTerminateAt	246
5.5.24.5.	SetCamSequentialShutterTerminateAt	247
5.5.24.6.	GetCamSequentialShutterIndexMinMax	248
5.5.24.7.	GetCamSequentialShutterEntryMinMax.....	249
5.5.24.8.	GetCamSequentialShutterEntry	250
5.5.24.9.	SetCamSequentialShutterEntry.....	251
5.5.25.	UserDefinedName (DeviceUserID)	252
5.5.25.1.	GetCamUserDefinedName.....	252
5.5.25.2.	SetCamUserDefinedName	253
5.5.26.	Chunk	254
5.5.26.1.	GetCamChunkModeActive	254
5.5.26.2.	SetCamChunkModeActive.....	255
5.5.27.	Miscellaneous.....	256
5.5.27.1.	GetCamIndexFromHandle.....	256
5.5.27.2.	GetCamTypeFromCamHandle	257
5.5.27.3.	GetCamTLParamsLocked	258
5.5.27.4.	Misc_GetLastGenlCamError	259
5.6.	GenlCam functions	260
5.6.1.	Inode functions.....	260
5.6.1.1.	Nd_GetNode	260
5.6.1.2.	Nd_GetType.....	264
5.6.1.3.	Nd_GetName.....	265
5.6.1.4.	Nd_GetAccessMode.....	266

5.6.1.5.	Nd_GetVisibility	267
5.6.1.6.	Nd_GetCachingMode	268
5.6.1.7.	Nd_GetDescription	269
5.6.1.8.	Nd_GetToolTip	270
5.6.1.9.	Nd_GetRepresentation	271
5.6.1.10.	Nd_GetUnit	272
5.6.2.	ICategory node functions	273
5.6.2.1.	Nd_GetNumOfFeatures	273
5.6.2.2.	Nd_GetFeatureByIndex	275
5.6.3.	IInteger node functions	276
5.6.3.1.	Nd_GetIntMin	276
5.6.3.2.	Nd_GetIntMax	278
5.6.3.3.	Nd_GetIntInc	279
5.6.3.4.	Nd_GetIntValue	280
5.6.3.5.	Nd_SetIntValue	281
5.6.4.	IFloat node functions	282
5.6.4.1.	Nd_GetFloatMin	282
5.6.4.2.	Nd_GetFloatMax	285
5.6.4.3.	Nd_GetFloatHasInc	286
5.6.4.4.	Nd_GetFloatInc	287
5.6.4.5.	Nd_GetFloatDisplayNotation	288
5.6.4.6.	Nd_GetFloatDisplayPrecision	289
5.6.4.7.	Nd_GetFloatValue	290
5.6.4.8.	Nd_SetFloatValue	291
5.6.5.	IBoolean node functions	292
5.6.5.1.	Nd_GetBoolValue	292
5.6.5.2.	Nd_SetBoolValue	294
5.6.6.	IEnumeration node functions	295
5.6.6.1.	Nd_GetNumOfEnumEntries	295
5.6.6.2.	Nd_GetEnumIntValue	298
5.6.6.3.	Nd_SetEnumIntValue	300
5.6.6.4.	Nd_GetEnumStrValue	301
5.6.6.5.	Nd_SetEnumStrValue	303
5.6.6.6.	Nd_GetEnumEntryByIndex	304
5.6.7.	IEnumEntry node functions	305
5.6.7.1.	Nd_GetEnumEntryIntValue	305
5.6.7.2.	Nd_GetEnumEntryStrValue	306
5.6.8.	ICommand node functions	307
5.6.8.1.	Nd_CmdExecute	307
5.6.8.2.	Nd_GetCmdIsDone	309
5.6.9.	IString node functions	310
5.6.9.1.	Nd_GetStrValue	310
5.6.9.2.	Nd_SetStrValue	312
5.6.10.	Chunk functions	313
5.6.10.1.	Chunk_AttachBuffer	313
5.7.	Utility functions	315
5.7.1.	Image format converter	315
5.7.1.1.	PrepareLUT	315
5.7.1.2.	Conv*ToBGRA	316
5.7.1.3.	Conv*ToBGR	318

5.7.1.4. ConvImage	320
5.7.2. Others	322
5.7.2.1. BitPerPixel	322
5.7.2.2. DataDepth	322
5.7.2.3. IsMonochromic	323
5.7.2.4. IsPixelBayer	323
5.7.2.5. SaveBmp*	324
5.7.2.6. ReverseImg	325
5.8. Status code	326
6. Sample source code	328
6.1. GrabStreamSimple*	329
6.1.1. Operation	330
6.2. GrabEvent	332
6.2.1. Operation	332
6.3. MultiCamera*	333
6.3.1. Operation	334
6.3.1.1. Main window	334
6.3.1.2. Parameter window	335
7. Others	336
7.1. Revision History	336
7.2. Disclaimer	337
7.3. License	337
7.4. Inquiry	337

1. Introduction

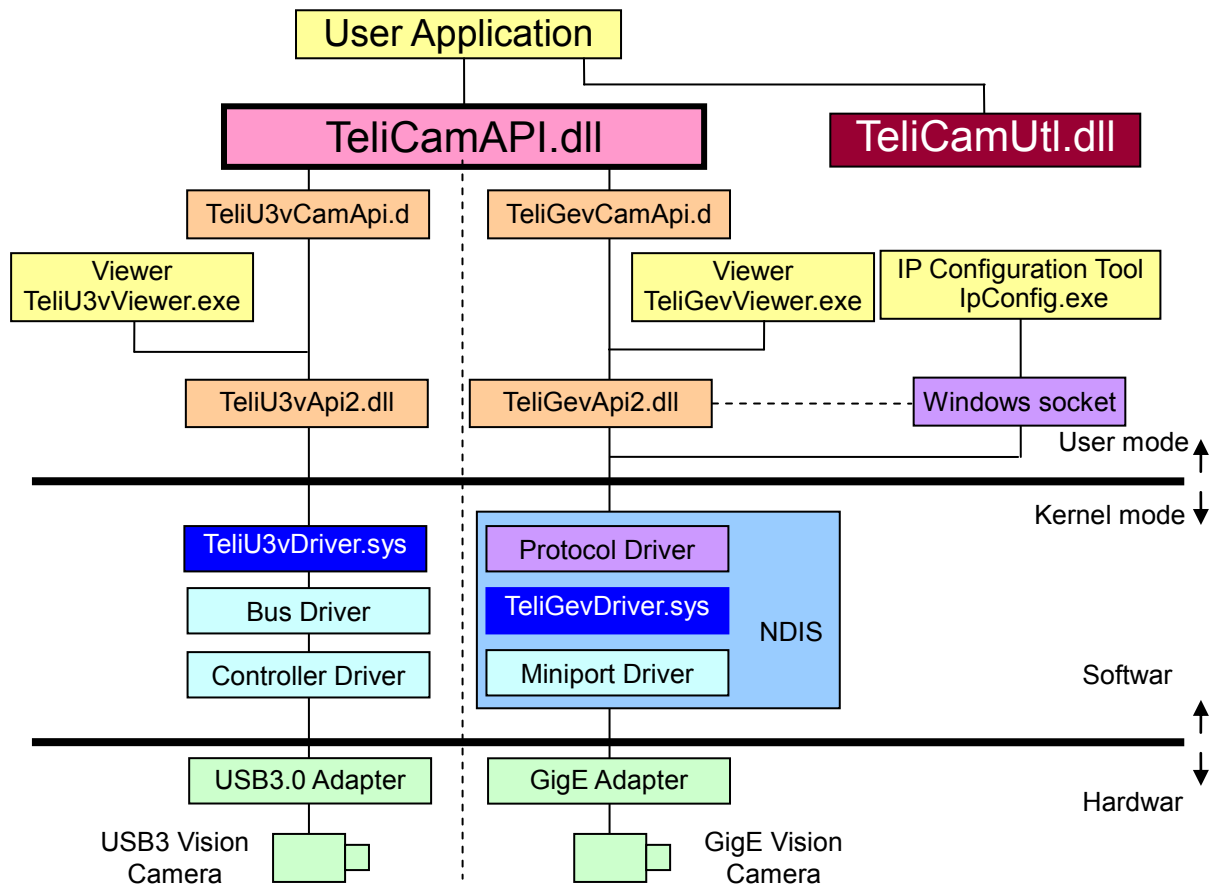
TeliCamSDK is a software development kit to control Toshiba Teli USB3 and GigE Vision digital camera series from PCs.

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

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.

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

3. Operation Environment

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	<ul style="list-style-type: none">● Windows 7 32/64-bit● Windows 8.1 32/64-bit● Windows 10 32/64-bit
Recommended PC Specifications	<ul style="list-style-type: none">● CPU : Intel Core2 2.40GHz or above recommended● Memory : 2Gbytes or above● Graphics : VRAM with 256 Mbytes or above mounted
Recommended USB3.0 Adapter	<ul style="list-style-type: none">● USB3.0 adapter with USB3.0 host controller from Renesas Electronics *1
Recommended Network Adapter	<ul style="list-style-type: none">● Gigabit Ethernet adapter supporting Jumbo Frame (Jumbo Packet) (9014 bytes or above) (Intel PRO/1000 series, etc.) *2
Required Software	<ul style="list-style-type: none">● 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	<ul style="list-style-type: none">● Toshiba Teli USB3 Vision digital camera● Toshiba Teli Gig-E Vision digital camera

Notes

*1: Necessary in application that uses USB3 Vision cameras.

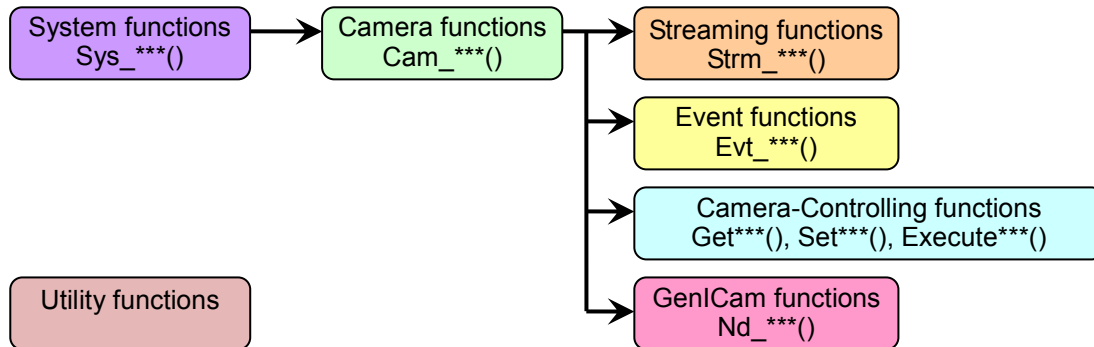
*2: Necessary in application that uses GigE Vision cameras.

*3: Necessary in development environment even if GenApi module is disabled in user application.

*4: Necessary in TeliU3vViewer and TeliGevViewer.

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.

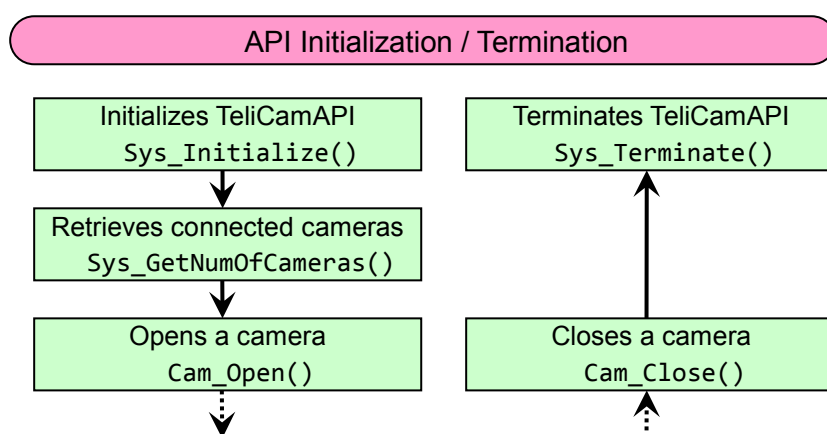
4.1. Usage

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 GenlCam 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.

TeliCamSDK provides “XMLFeatures.h” which contains declarations of frequently used feature names and values.

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.

User application can utilize “InfoNode.h” in sample code “MultiCamera” or “MultiCameraPrimitive”, which contains declaration of classes for acquiring various information associating with specified feature name(for example, value, maximum, minimum, access mode and so on). These classes also provide method for setting parameters to MFC form controls (CScrollBar, CEdit, and CComboBox).

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.

TeliCamSDK provides register map for BU series camera “RegisterMap_BU.h” and register map for BG series camera “RegisterMap_BG_Type1.h”.

In USB3 Vision camera case, user application can utilize “IIDC2.h” in sample code “GrabStreamSimpleIIDC2BU”, which contains declaration of structures corresponding to FeatureCSRs (Feature Control and Status Register) in IIDC2 register map. “IIDC2 Digital Camera Control Specification” defines IIDC2 register map.

User application can read a whole FeatureCSR in a single call for “Cam_ReadReg()”, get information of the feature (implemented, Active, Writable, etc.), get actual value (calculated using Value, Mult, Div, Min, Max register values). These structures also provide method for setting parameters to MFC form controls (CScrollBar, CEdit, and CComboBox).

For detailed information on IIDC2 register map, please refer to <http://jia.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 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 object specified on opening stream interface.

TeliCamAPI will notify reception of image by setting signal state to event object specified on opening 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.

➤ Using callback function.

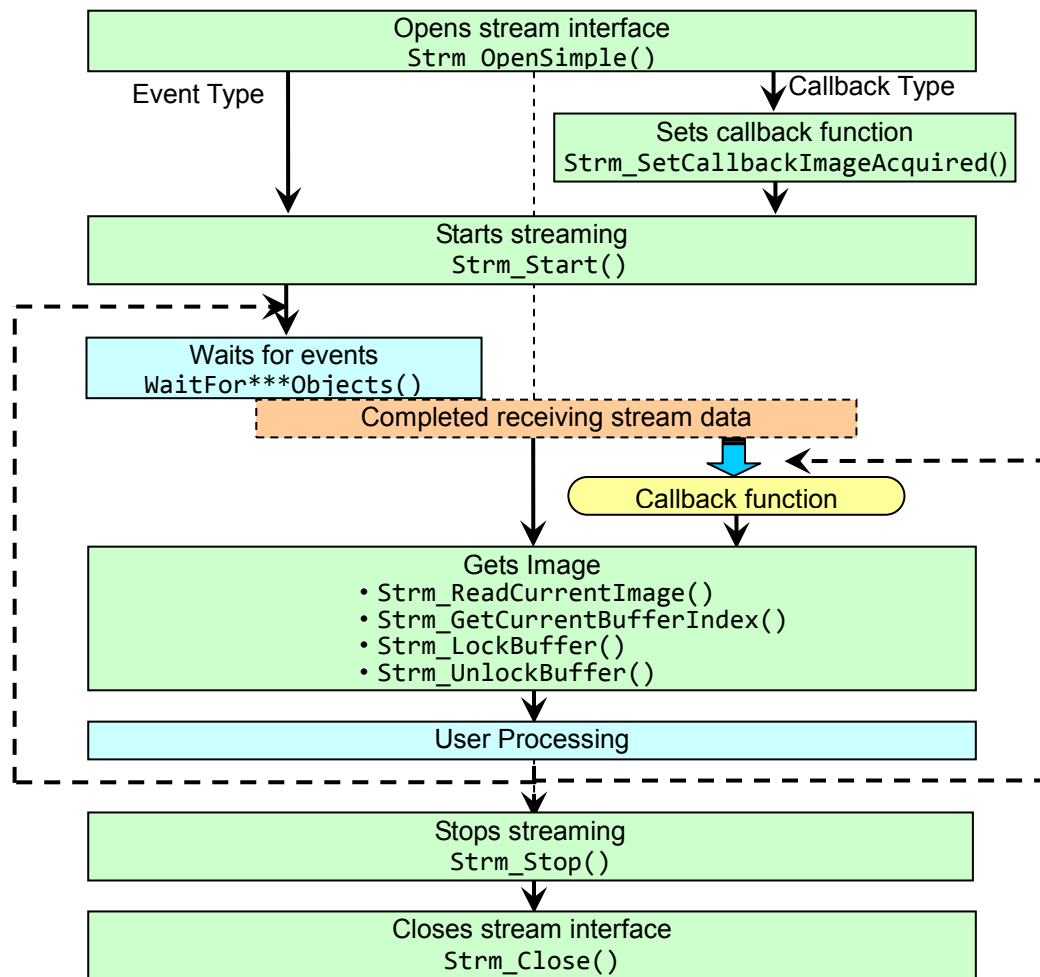
TeliCamAPI will notify reception of image by calling callback function registered with `Strm_SetCallbackImageAcquired()`.

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 stream interface

Call "Strm_OpenSimple()" to open a stream interface. "Strm_OpenSimple()" will return a Stream-Handle for the created 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.

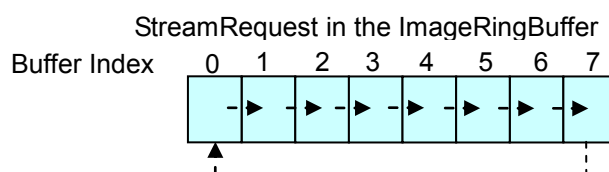
TeliCamAPI will cause a buffer busy error and discard the new stream data, if the callback function keeps using the StreamRequest for writing just received image. Making processing time of callback function as short as possible is recommended.

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 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.

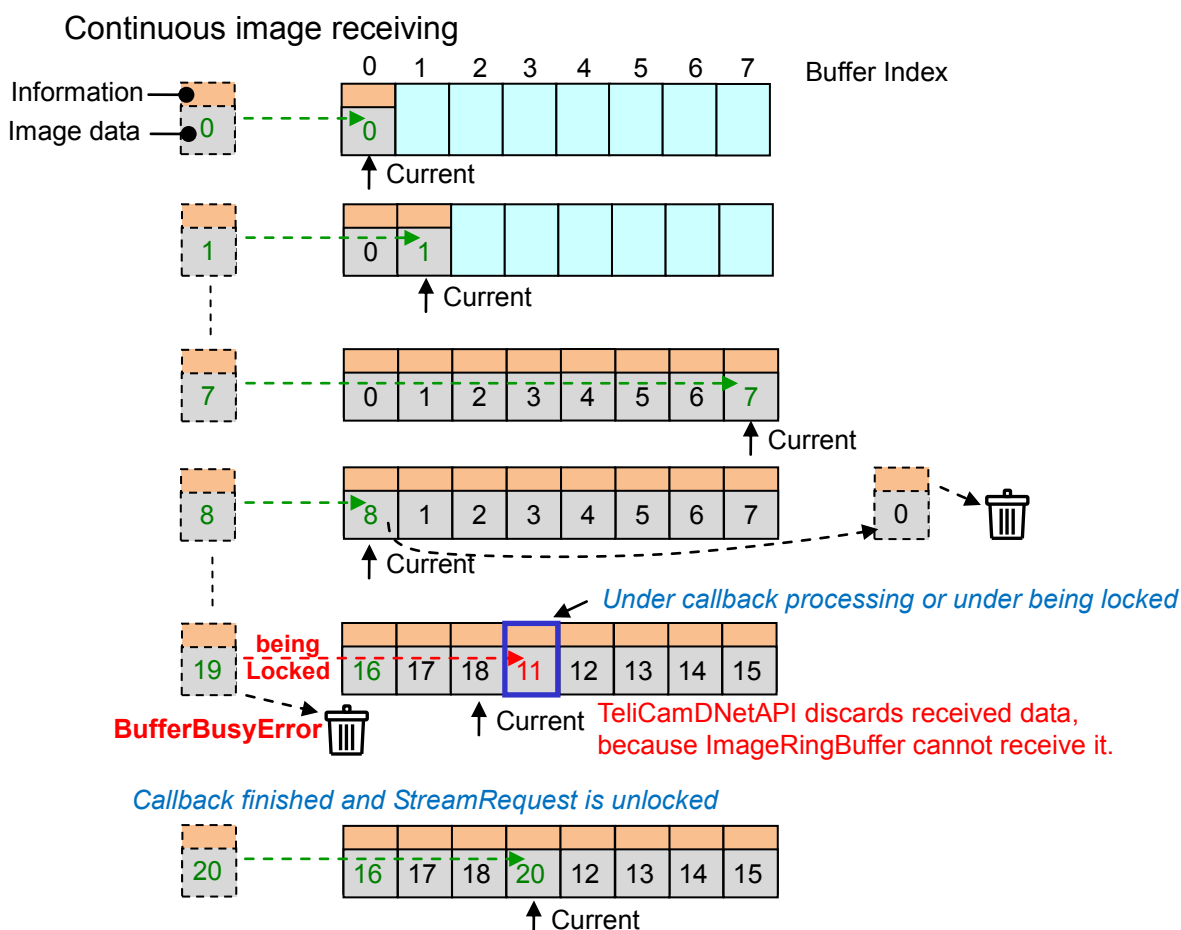
Unlock the locked buffer using “Strm_UnlockBuffer()” as soon as finished accessing to (copying) the image data. Otherwise, buffer busy error may occur.

➤ 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 AcquisitionFrameCount 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 stream interface

Call “Strm_Close()” to terminate use of the stream interface.



4.1.5.2. Acquiring image data using Low-Level API functions

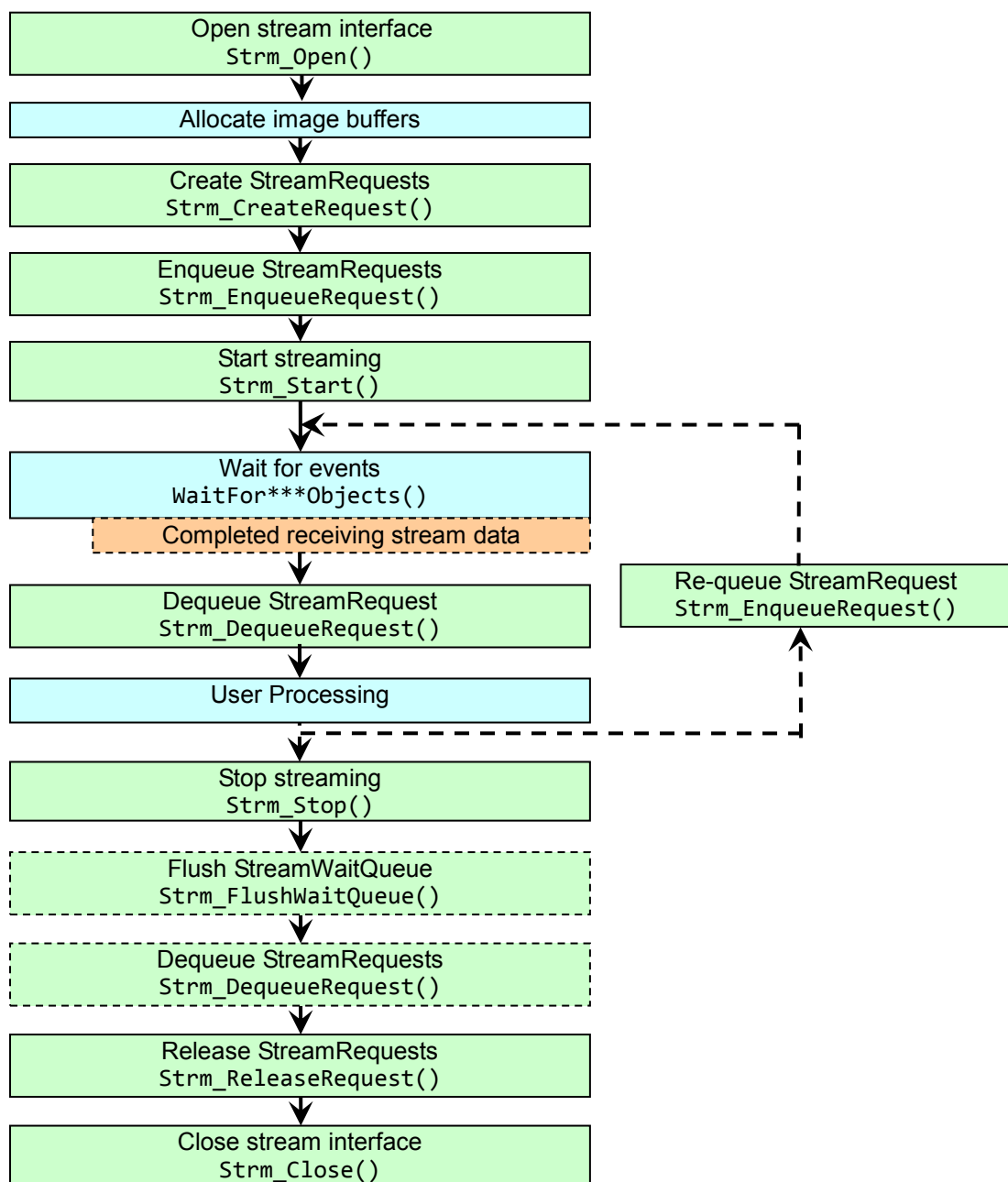
“Low-Level” means that these functions belong to layer near transport layer.

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 stream interface

Call "Strm_Open()" to open a stream interface and to register event object for image acquired event. Stream-Handle of the opened 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.

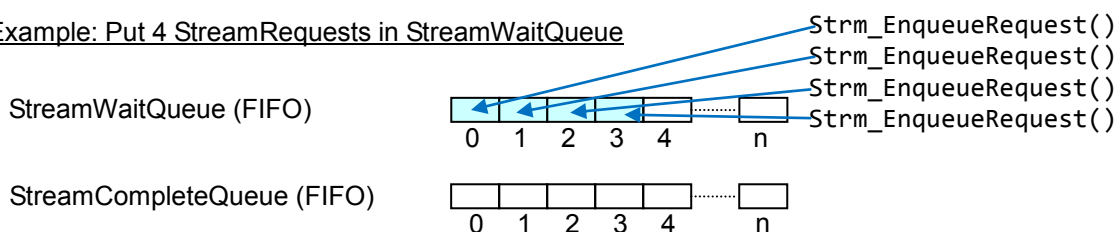
Release StreamRequests and create new StreamRequests including image buffer of new image size, when Image size is changed.

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).

Example: Put 4 StreamRequests in StreamWaitQueue



4.1.5.2.5. Start streaming

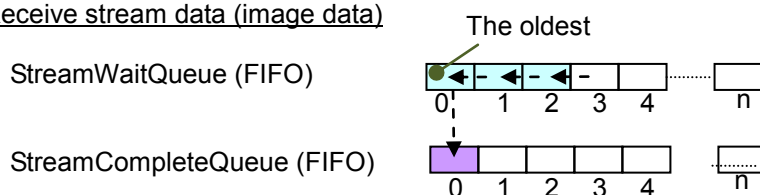
Call "Strm_Start()" to instruct a camera to start acquiring images and start streaming.

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 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.

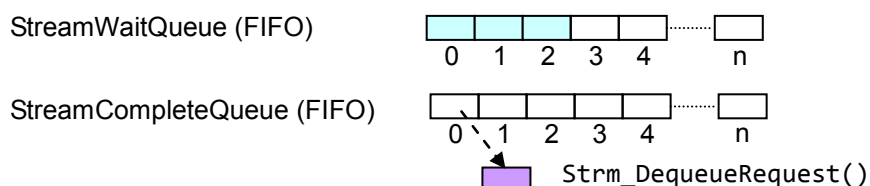
Receive stream data (image data)



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.

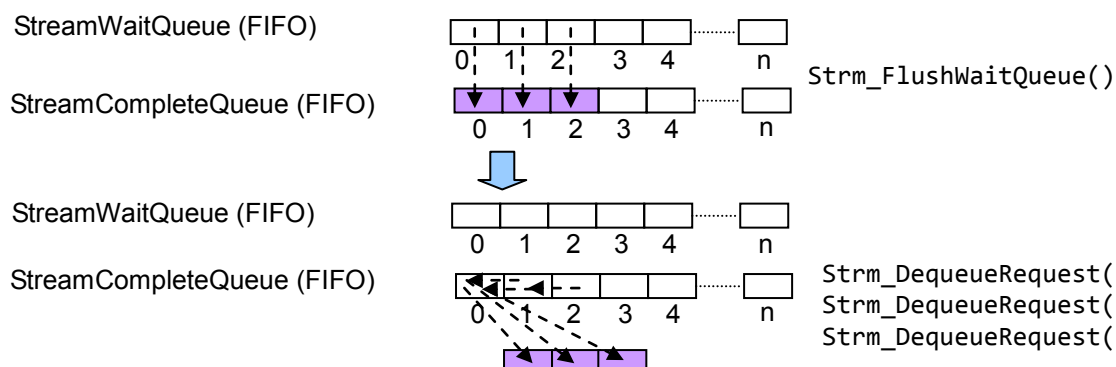
4.1.5.2.9. Finish streaming of TeliCamAPI (PC)

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 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 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.

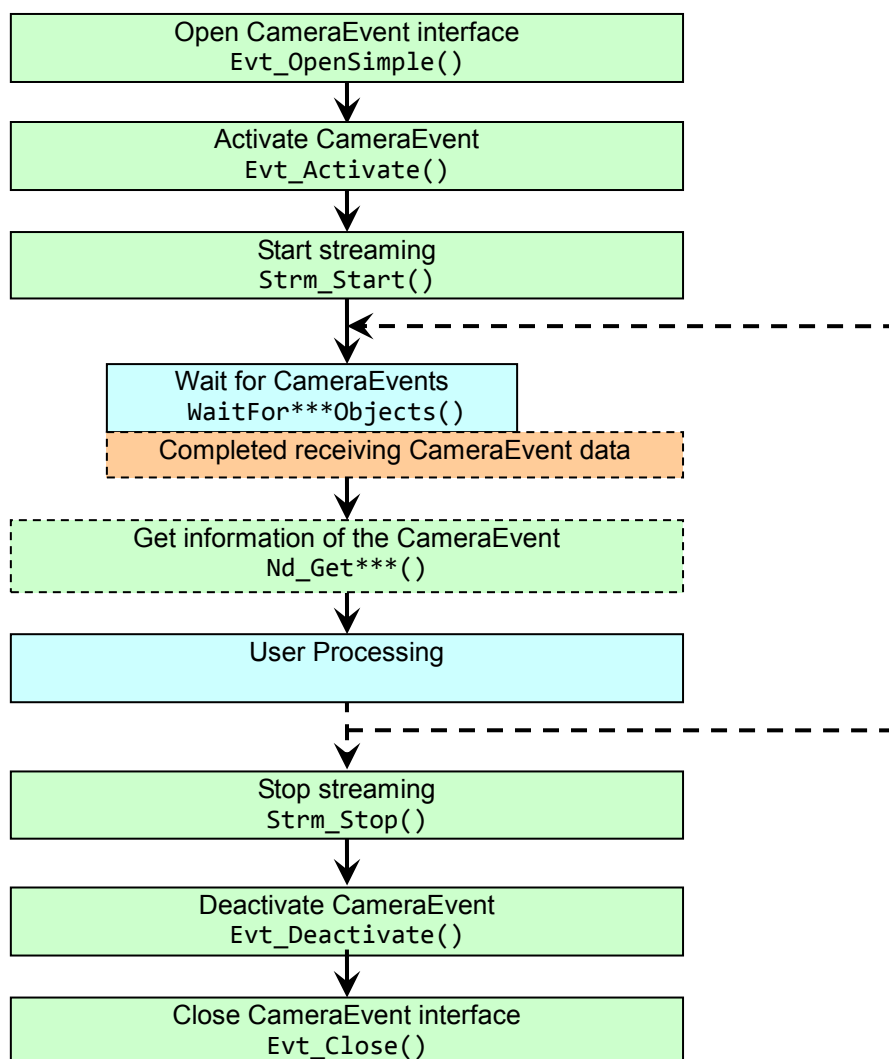
4.1.6.1. CameraEvent notification using High-Level API functions

“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 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 object which will be set signaled on reception of the CameraEvent. User application should create the event 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 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 ("Nd_GetNode()", "Nd_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.

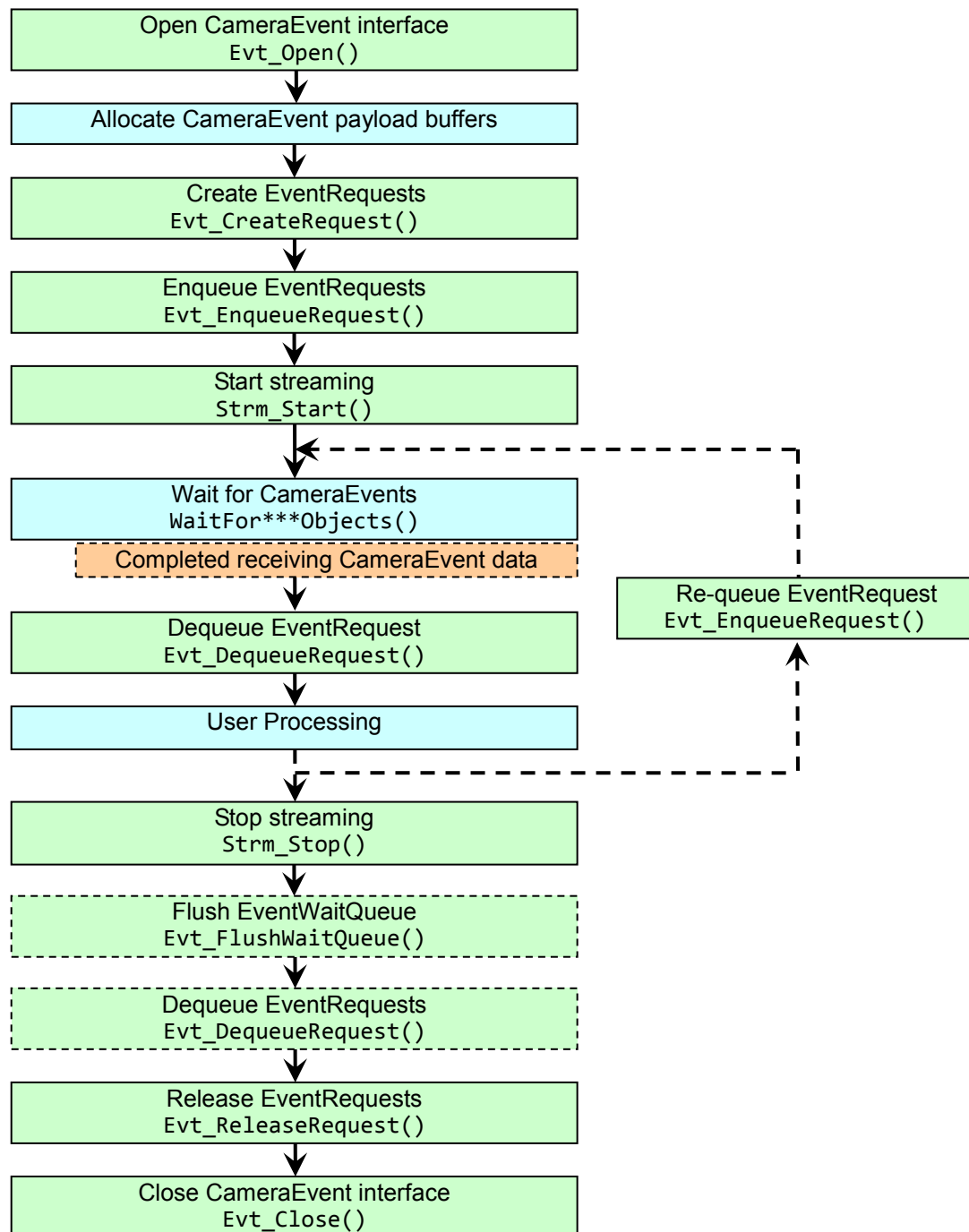
4.1.6.1.7. Close CameraEvent interface

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 object for CameraEvent. Event-Handle of the opened 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

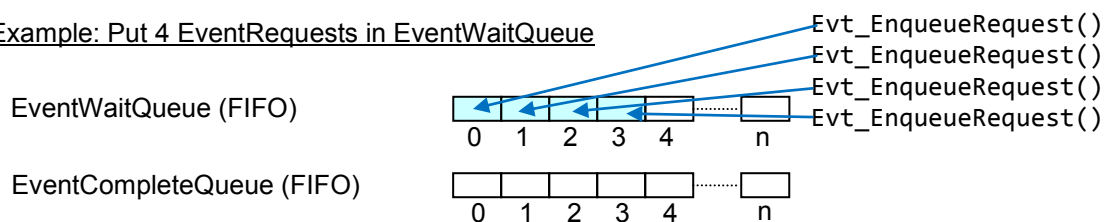
Call "Evt_CreateRequest()" to create an EventRequest for receiving a CameraEvent data with the allocated payload buffer as a member of the created EventRequest.

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.

Example: Put 4 EventRequests in EventWaitQueue



4.1.6.2.5. Start streaming

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

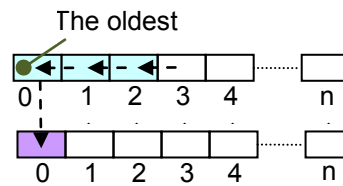
4.1.6.2.6. Receive CameraEvent data

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 object for notifying reception of the CameraEvent signaled.

Receive 1 CameraEvent data)

EventWaitQueue (FIFO)

EventCompleteQueue (FIFO)



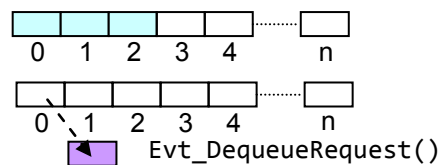
4.1.6.2.7. Extract EventRequest from EventCompleteQueue

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

EventWaitQueue (FIFO)

EventCompleteQueue (FIFO)



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)

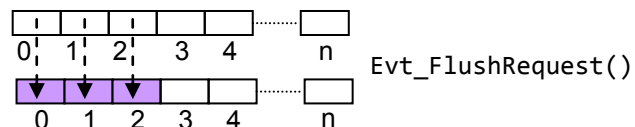
Call “Evt_FlushWaitQueue()” to move EventRequests in EventWaitQueue to EventCompleteQueue, which will stop the CameraEvent data receiving operation of TeliCamAPI.

Call “Evt_DequeueRequest()” until EventCompleteQueue becomes empty to prepare for closing CameraEvent interface.

Stop receiving CameraEvent data operation of TeliCamAPI (PC)

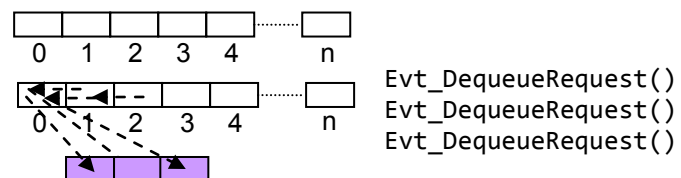
EventWaitQueue (FIFO)

EventCompleteQueue (FIFO)



EventWaitQueue (FIFO)

EventCompleteQueue (FIFO)



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 PDF document "[Start-up Guide Eng](#)" in "Documents" folder under the TeliCamSDK folder to install TeliCamSDK.

4.3. SDK Uninstallation

Refer to PDF document "[Start-up Guide Eng](#)" in "Documents" folder under the TeliCamSDK folder to uninstall TeliCamSDK.

4.4. Set up development environment

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

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
<i>eCamType</i> [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
<i>psSysInfo</i> [in]	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 extension(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 extension(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
<i>puiNum</i>	[out]	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](#).

5.2. 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
<i>hCam</i>	[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 .
<i>uiCamIdx</i>	[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.
<i>psCamInfo</i>	[out]	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
<i>eCamType</i>	[out]	Interface type of camera.
<i>szManufacturer</i>	[out]	Name of manufacturer.
<i>szModelName</i>	[out]	Camera model name.
<i>szSerialNumber</i>	[out]	Serial number of the camera.
<i>szUserDefinedName</i>	[out]	Name that user defined.
<i>sU3vCamInfo</i>	[out]	Structure which contains information of USB 3 vision camera.
<i>sGevCamInfo</i>	[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 key 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 Addresss IP configuration scheme. 0: Not supported, Other than 0: Supported.
cSupportIP_DHCP	[out]	Shows if the camera supports DHCP IP configuration scheme. 0: Not supported, Other than 0: Supported.
cSupportIP_Persistent	[out]	Shows if the camera supports Persistent-IP IP configuration scheme. 0: Not supported, Other than 0: Supported.
cCurrentIP_LLA	[out]	Shows if Link Local Addresss 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(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](#)).

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]

```
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  
);
```

[Parameters]

Parameter	Description
<i>uiCamIdx</i> [in]	Index of the camera. Index value should be up to number of detected cameras minus 1, from zero. Number of detected camera can be retrieved using "Sys_GetNumOfCameras()".
<i>phCam</i> [out]	Pointer to a variable that receives Camera-Handle assigned to the opening camera.
<i>hRmv</i> [in]	Handle of event object for notifying disconnection of the camera. This event object will also be set signaled, on Heartbeat timeout in GigE Vision camera. "CreateEvent()" of Win32 API is usually used for creating event object. Specify NULL or do not specify argument, if notification of disconnection event is not necessary.
<i>bUseGenICam</i> [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 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.

Parameter	Description
<i>pvXml</i> [in]	<p>Pointer to camera description data (Xml data).</p> <p>Camera description data will be loaded from data buffer that the argument points.</p> <p>If NULL is specified or this argument is not specified, data will be loaded from camera.</p> <p>This parameter will be ignored when bUseGenICam is false,.</p>
<i>eAccessMode</i> [in]	<p>Access mode(Control Channel Privilege) of the GigE Vision camera.</p> <p>TeliCamAPI will try to acquire the specified privilage.</p> <p>If no value is specified, CAM_ACCESS_MODE_CONTROL will be used as argument value.</p> <p>This parameter will be ignored when type of camera is not GigE Vision.</p>

[CAM_ACCESS_MODE Enumeration]

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

[Return value]

Returns result status. Refer to [5.8 Status code](#).

[Remarks]

In USB3 Vision camera case, this method 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 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 a few minutes for loading and preparing data.

When false is specified to [bUseGenICam](#) parameter, processing time of opening camera will be shortened, but all APIs in section [5.4.1 High-level API functions](#), [5.6 GenICam functions](#) and most APIs in section [5.5 Controlling camera feature functions](#) will become unavailable. (Functions provided in TeliCamAPI for Camera event (message) function of GigE Vision camera 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.

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]

```
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  
);
```

[Parameters]

Parameter	Description
<i>pszSerialNo</i> [in]	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.
<i>pszModelName</i> [in]	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.
<i>pszUserDefinedName</i> [in]	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.
<i>phCam</i> [out]	Pointer to a variable that receives Camera-Handle assigned to the opening camera.
<i>hRmv</i> [in]	Handle of an event object for notifying disconnection of the camera. This event object will also be set signaled, on Heartbeat timeout in GigE Vision camera. “CreateEvent()” of Win32 API is usually used for creating event objects. Specify NULL or do not specify this argument, if notification of disconnection event is not necessary.

Parameter	Description
<i>bUseGenICam</i> [in]	<p>Flag for enabling GenICam function.</p> <p>If true, TeliCamAPI will enable GenICam function after loading camera description file (Xml file).</p> <p>If false, TeliCamAPI does not load camera description file (Xml file) and GenICam function will be disabled.</p> <p>All APIs in section 5.4.1, 5.6 and most APIs in section 5.5 use GenICam function inside them.</p> <p>We recommend users to specify true to this parameter, except there is solid reason to disable GenICam function.</p> <p>If this parameter is not specified, true will be used as parameter value.</p>
<i>pvXml</i> [in]	<p>Pointer to a camera description data (Xml data).</p> <p>If NULL is specified or nothing are specified, camera description data will be loaded from the camera.</p> <p>If value other than NULL is specified, camera description data will be loaded from data buffer that the argument points.</p> <p>This parameter will be ignored when bUseGenICam is false.</p>
<i>eAccessMode</i> [in]	<p>Access mode(Control Channel Privilege) of the GigE Vision camera.</p> <p>TeliCamAPI will try to acquire the specified privilege.</p> <p>If no value is specified, CAM_ACCESS_MODE_CONTROL will be used as argument value.</p> <p>This parameter will be ignored when type of camera is not GigE Vision.</p>

[Return value]

Returns result status. Refer to [5.8 Status code](#).

[Remarks]

In USB3 Vision camera case, this method clears “StreamEnable” node (SIControl1 register) and “EventEnable” node (EIControl1 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 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 a few minutes for loading and preparing data.

When false is specified to [bUseGenICam](#) parameter processing time of opening camera will be shortened, but all APIs in section [5.4.1 High-level API functions](#), [5.6 GenICam functions](#), and most APIs in section [5.5 Controlling camera feature functions](#) will become unavailable. (Functions provided in TeliCamAPI for Camera event (message) function of GigE Vision camera 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
<i>hCam</i> [in]	Camera-Handle of the opened camera.

[Return value]

Returns result status. Refer to [5.8 Status code](#).

[Remarks]

Calling this method during the other thread is using the camera may cause error in the other thread.
Call this method 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
<i>hCam</i>	[in]	Camera-Handle of target camera.
<i>ullAdrs</i>	[in]	Start addres of registers to read.
<i>uiSizeQuadlet</i>	[in]	Size of data to read in Quadlet. Quadlet is 4 bytes in size. Specifying 1 means reading 4 bytes.
<i>pvData</i>	[out]	Pointer to a variable 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;
CAM_INFO	sCamInfo;
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;

// Get information of a camera.
uiStatus = Cam_GetInformation(hCam, NULL, &sCamInfo);
if (uiStatus != CAM_API_STS_SUCCESS)
    return -1;

// Set Width address.
if (sCamInfo.eCamType == CAM_TYPE_U3V) {
    ullAddress = 0x00202098; // 0x100000(IIDC Address)
                           // + 0x102000(OffsetCategoryBlocck2)
                           // + 0x98(Width Offset)
} else if (sCamInfo.eCamType == CAM_TYPE_GEV) {
    ullAddress = 0x0000A804;
} else {
    return -1;
}

// 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
<i>hCam</i>	[in]	Camera-Handle of target camera.
<i>ullAdrs</i>	[in]	Start address of registers to write data.
<i>uiSizeQuadlet</i>	[in]	Size of data to write, in Quadlet. Quadlet is 4 bytes in size. Specifying 1 means writing 4 bytes.
<i>pvData</i>	[in]	Pointer to a variable 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
<i>hCam</i> [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 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++
<pre>CAM_API_STATUS uiStatus; uint32_t uiNum, uiNumOld, uiRet, i; HANDLE hRemoveEvt = NULL; CAM_HANDLE hCam = NULL; __try { // 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))</pre>

```

        return -1;
    uiNumOld = uiNum;

    // Create event for detecting camera removing.
    hRemoveEvt = CreateEvent(NULL, TRUE, FALSE, NULL);

    // 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");

    // Reset port.
    uiStatus = Cam_ResetPort(hCam);
    if (uiStatus != CAM_API_STS_SUCCESS)
        return -1;

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

    // Wait remove-event that is signaled.
    uiRet = WaitForSingleObject(hRemoveEvt, 2000);
    if (uiRet != WAIT_OBJECT_0)
        return -1;

    // 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)
            return -1;

        if (uiNum == uiNumOld)
            break;

        Sleep(100);        // For sample
    }

    if (i == 100)
        return -1;        // 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)
        return -1;

    printf("Cam_Open was successful.(2nd)¥n");
}
finally
{
    // Close camera.
    if (hCam != NULL)
    {
        uiStatus = Cam_Close(hCam);
        if (uiStatus != CAM_API_STS_SUCCESS)
            printf("Cam_Close error! (0x%x)", uiStatus);
    }

    // Close event.
    if (hRemoveEvt != NULL)
        CloseHandle(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
<i>hCam</i>	[in]	Camera-Handle of target camera.
<i>pbEnable</i>	[out]	Pointer to a variable that receives current Heartbeat enable state. If true, Heartbeat process is enabled, otherwise disabled.
<i>puiHbTimeout</i>	[out]	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
<i>hCam</i>	[in]	Camera-Handle of target camera.
<i>bEnable</i>	[in]	Flag for enabling Heartbeat process. If true, Heartbeat process is enabled. There are cameras which will not accept false value.
<i>uiHbTimeout</i>	[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 is available 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 camera 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](#).

5.3. 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.

Even though multiple applications can open the same camera simultaneously, user application cannot open a stream that the other application is using.

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 stream interface for receiving images from camera. ImageRingBuffer consists of structure CAM_STRM_REQUEST_INFO.

[Syntax]

```
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  
);
```

[Parameters]

Parameter		Description
<i>hCam</i>	[in]	Camera-Handle of target camera.
<i>phStrm</i>	[out]	Pointer to a variable that receives Stream-Handle of the new stream.
<i>puiMaxPayloadSize</i>	[out]	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.
<i>hCmpEvt</i>	[in]	Handle of an event object for notifying that at least one image has been received from the camera and stored in ImageRingBuffer. This argument is optional. "CreateEvent()" of Win32 API is usually used for creating event objects. Specify NULL or do not specify argument, if notification of imgsge aquired event is not necessary.

Parameter	Description
<i>uiApiBufferCount</i> [in]	<p>Number of image buffer (StreamRequests) to create in ImageRingBuffer.</p> <p>This argument is optional.</p> <p>Valid value range is up to 128 from 1.</p> <p>If 0 is specified or this argument is not specified, default number DEFAULT_API_BUFFER_CNT (8) is used as this parameter.</p>
<i>uiMaxPacketSize</i> [in]	<p>The maximum packet size that camera driver can receive, in bytes. This argument is optional.</p> <p>If 0 is specified, or value is not specified, the following default value will be used.</p> <p>USB3 Vision camera : 65536 bytes GigE Vision camera : 1500 bytes</p> <p>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 <i>uiMaxPacketSize</i>.</p> <p>Note that <i>uiMaxPacketSize</i> 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).</p> <p>For example, when Jumbo-Frame size is 9014, which usually includes Ethernet header size, specify 9000 instead of 9014.</p> <p>Specify 0 or do not specify value for USB3 Vision camera.</p>

[Return value]

Returns result status. Refer to [5.8 Status code](#).

[Remarks]

If user application calls this method for a CameraStream that the other application is using, this method will return CamApiStatus.AlreadyOpened.

This function creates ImageRingBuffer inside TeliCamAPI and open 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 transferred 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 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 object.

Note that [hCmpEvt](#) event 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, uiRet, i;
HANDLE              hStrmCmpEvt = NULL;
CAM_HANDLE          hCam = NULL;
CAM_STRM_HANDLE     hStrm = NULL;
void*               pvPayloadBuf = NULL;
CAM_IMAGE_INFO      sImageInfo;

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

__try
{
    // 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;

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

// Create completion event object for stream.
hStrmCmpEvt = CreateEvent(NULL, FALSE, FALSE, NULL);
if (hStrmCmpEvt == NULL)
    return -1;

// Open stream interface.
uiStatus = Strm_OpenSimple(hCam, &hStrm, &uiPyldSize, hStrmCmpEvt);
if (uiStatus != CAM_API_STS_SUCCESS)
    return -1;

// Allocate buffer for receiving image data.
pvPayloadBuf = (void *)VirtualAlloc(NULL,
                                     uiPyldSize,
                                     MEM_RESERVE | MEM_COMMIT,
                                     PAGE_EXECUTE_READWRITE);

if (pvPayloadBuf == NULL)
    return -1;

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

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

// Start stream.
uiStatus = Strm_Start(hStrm);
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 image completion event.
uiRet = WaitForSingleObject(hStrmCmpEvt, 2000);
if (uiRet != WAIT_OBJECT_0)
    return -1;

// Get current image.
uiStatus = Strm_ReadCurrentImage(hStrm, pvPayloadBuf, &uiPyldSize, &sImageInfo);
if (uiStatus != CAM_API_STS_SUCCESS)
    return -1;

// 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);
}
__finally
{
    if (hStrm != 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 != 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)
        VirtualFree(pvPayloadBuf, 0, MEM_RELEASE);

    // Close completion event object for stream.
    if (hStrmCmpEvt != NULL)
        CloseHandle(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
<i>hStrm</i>	[in]	Stream-Handle of target stream interface.
<i>pvBuf</i>	[out]	Pointer to a memory block that receives current image.
<i>puiSize</i>	[in, out]	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.
<i>psImageInfo</i>	[out]	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]	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
<i>hStrm</i>	[in]	Stream-Handle of target stream interface.
<i>puiBufferIndex</i>	[out]	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++
<pre>CAM_API_STATUS uiStatus; uint32_t uiNum, uiPyldSize, uiBufferIndex, uiAve, uiRet, i; HANDLE hStrmCmpEvt = NULL; CAM_HANDLE hCam = NULL; CAM_STRM_HANDLE hStrm = NULL; void* pvPayloadBuf = NULL; CAM_IMAGE_INFO sImageInfo; // Initialize system. uiStatus = Sys_Initialize(); if (uiStatus != CAM_API_STS_SUCCESS) return -1; __try { // 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; // Create completion event object for stream. hStrmCmpEvt = CreateEvent(NULL, FALSE, FALSE, NULL);</pre>

```

if (hStrmCmpEvt == NULL)
    return -1;

// Open stream interface.
uiStatus = Strm_OpenSimple(hCam, &hStrm, &uiPyldSize, hStrmCmpEvt);
if (uiStatus != CAM_API_STS_SUCCESS)
    return -1;

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

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

// Start stream.
uiStatus = Strm_Start(hStrm);
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 image completion event.
uiRet = WaitForSingleObject(hStrmCmpEvt, 2000);
if (uiRet != WAIT_OBJECT_0)
    return -1;

// Get current ring buffer index.
uiRet = Strm_GetCurrentBufferIndex(hStrm, &uiBufferIndex);
if (uiRet != WAIT_OBJECT_0)
    return -1;

// Lock the ring buffer pointer.
uiStatus = Strm_LockBuffer(hStrm, uiBufferIndex, &sImageInfo);
if (uiStatus != CAM_API_STS_SUCCESS)
    return -1;

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)
    return -1;
}
__finally
{
    if (hStrm != 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 != 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 != NULL)
        CloseHandle(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
<i>hStrm</i>	[in]	Stream-Handle of target stream interface.
<i>uiBufferIndex</i>	[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.
<i>psImageInfo</i>	[out]	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
<i>hStrm</i>	[in]	Stream-Handle of target stream interface.
<i>uiBufferIndex</i>	[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
<i>hStrm</i>	[in]	Stream-Handle of target stream interface.
<i>pvContext</i>	[in]	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.
<i>func</i>	[in]	Pointer to a callback function.

[Parameters of Callback function]

Parameter		Description
<i>hRcvCam</i>	[out]	Camera-Handle of the camera.
<i>hRcvStrm</i>	[out]	Stream-Handle of the stream.
<i>psImageInfo</i>	[out]	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 .
<i>uiBufferIndex</i>	[out]	Index of StreamRequest in the ImageRingBuffer.
<i>pvRcvContext</i>	[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, NULL);
}
```

```

    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;

        Sleep(50); // For sample
    }

    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
<i>hStrm</i>	[in]	Stream-Handle of target stream interface.
<i>pvContext</i>	[in]	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.
<i>func</i>	[in]	Pointer to a callback function.

[Parameters of Callback function]

Parameter		Description
<i>hRcvCam</i>	[out]	Camera-Handle of the camera.
<i>hRcvStrm</i>	[out]	Stream-Handle of the stream.
<i>uiErrorStatus</i>	[out]	Error code of the error in receiving stream.
<i>uiBufferIndex</i>	[out]	Index of StreamRequest in the ImageRingBuffer.
<i>pvRcvContext</i>	[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
<i>hStrm</i>	[in]	Stream-Handle of target stream interface.
<i>pvContext</i>	[in]	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.
<i>func</i>	[in]	Pointer to a callback function.

[Parameters of Callback function]

Parameter		Description
<i>hRcvCam</i>	[out]	Camera-Handle of camera.
<i>hRcvStrm</i>	[out]	Stream-Handle of the stream.
<i>uiBufferIndex</i>	[out]	Index of StreamRequest that the discarded image was to be stored.
<i>pvRcvContext</i>	[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 stream interface for receiving images from camera.

[Syntax]

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

[Parameters]

Parameter	Description
<i>hCam</i> [in]	Camera-Handle of target camera.
<i>hCmpEvt</i> [in]	Handle of an event object for notifying that at least one image has been received from the camera and stored in CompleteQueue. This argument is optional. “CreateEvent()” of Win32 API is usually used for creating event objects. Specify NULL or do not specify this argument if notification of image aquired event is not necessary.
<i>puiMaxPayloadSize</i> [in, out]	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.
<i>phStrm</i> [out]	Pointer to a variable that receives Stream-Handle assigned to the opening stream.

Parameter	Description
<i>uiMaxPacketSize</i> [in]	<p>The maximum packet size in bytes that the camera driver can receive.</p> <p>This argument is optional.</p> <p>If 0 is specified, or value is not specified, the following default value will be used.</p> <p>USB3 Vision camera : 65536 bytes GigE Vision camera : 1500 bytes</p> <p>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.</p> <p>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).</p> <p>For example, when Jumbo-Frame size is 9014, which usually includes Ethernet header size, specify 9000 instead of 9014.</p> <p>Specify 0 or do not specify value for USB3 Vision camera.</p>

[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 StreamRequest, 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 method for a CameraStream that the other application is using, this method will return CamApiStatus.AlreadyOpened.

TeliCamAPI will make event 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++	
<pre>const int STRM_REQUEST_NUM = 5; CAM_API_STATUS uiStatus; uint32_t uiNum, uiPyldSize, uiRcvSize, uiRet, i; HANDLE hStrmCmpEvt = NULL; CAM_HANDLE hCam = NULL; CAM_STRM_HANDLE hStrm = NULL; void* pvPayloadBuf = NULL;</pre>	

```

CAM_STRM_REQUEST_HANDLE hStrmReq[STRM_REQUEST_NUM];
void*                    pvRcvPayloadBuf = NULL;
CAM_STRM_REQUEST_HANDLE hRcvStrmReq = NULL;

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

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

__try
{
    // 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;

    // Create completion event object for stream.
    hStrmCmpEvt = CreateEvent(NULL, FALSE, FALSE, NULL);
    if (hStrmCmpEvt == NULL)
        return -1;

    // 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)
        return -1;

    // Allocate buffer for receiving image data.
    pvPayloadBuf = (void *)VirtualAlloc(NULL,
                                         uiPyldSize * STRM_REQUEST_NUM,
                                         MEM_RESERVE | MEM_COMMIT,
                                         PAGE_EXECUTE_READWRITE);

    if (pvPayloadBuf == NULL)
        return -1;

    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)
            return -1;

        // Enqueue a StreamRequest to stream wait queue.
        uiStatus = Strm_EnqueueRequest(hStrm, hStrmReq[i]);
        if (uiStatus != CAM_API_STS_SUCCESS)
            return -1;
    }

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

    // Set TriggerSource software.

```

```

        uiStatus = SetCamTriggerSource(hCam, CAM_TRIGGER_SOFTWARE);
        if (uiStatus != CAM_API_STS_SUCCESS)
            return -1;

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

        for (i=0; i<10; i++) {
            // Send Software Trigger command.
            uiStatus = ExecuteCamSoftwareTrigger(hCam);
            if (uiStatus != CAM_API_STS_SUCCESS)
                return -1;

            // Wait for receiving image completion event.
            uiRet = WaitForSingleObject(hStrmCmpEvt, 2000);
            if (uiRet != WAIT_OBJECT_0)
                return -1;

            // Retrieve a StreamRequest from stream complete queue.
            uiStatus = Strm_DequeueRequest(
                hStrm,
                &hRcvStrmReq,
                &pvRcvPayloadBuf,
                &uiRcvSize);
            if (uiStatus != CAM_API_STS_SUCCESS)
                return -1;

            printf("Receive stream.(%d) : 0x%08X\n", i, 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)
                return -1;
        }
    }
__finally
{
    if (hStrm != 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 + 1); i++) { // + 1 : For sample
            // 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] != 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 != 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)
    VirtualFree(pvPayloadBuf, 0, MEM_RELEASE);

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

// Terminate system.
Sys_Terminate();
}

```

5.3.2.2. Strm_CreateRequest

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

This function creates 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 StreamRequest structure from the StreamCompleteQueue.

[Syntax]

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

[Parameters]

Parameter		Description
<i>hStrm</i>	[in]	Stream-Handle of target stream interface.
<i>pvPayloadBuf</i>	[in]	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 <i>uiPayloadSize</i> argument.
<i>uiPayloadSize</i>	[in]	Payload size (image buffer size) of an image in bytes. Specify value returned from "Strm_Open()" as puiMaxPayloadSize , in usual case.
<i>phStrmRequest</i>	[out]	Pointer to a variable 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 StreamCompleteQueue 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 method. .

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
<i>hStrm</i>	[in]	Stream-Handle of target stream interface.
<i>hStrmRequest</i>	[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
<i>hStrm</i>	[in]	Stream-Handle of target stream interface.
<i>hStrmRequest</i>	[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
<i>hStrm</i>	[in]	Stream-Handle of target stream interface.
<i>phStrmRequest</i>	[out]	Pointer to a variable that receives handle of the retrieved StreamRequest structure.
<i>ppvPayloadBuf</i>	[out]	Pointer to a variable that receives pointer to image data in the retrieved StreamRequest.
<i>puiPayloadSize</i>	[out]	Pointer to a variable that receives 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, TeliCamSPI will return CAM_API_STS_EMPTY_COMPLETE_QUEUE as status 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
<i>hStrm</i> [in]	Stream-Handle of 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,

Call this function to make the StreamWaitQueue empty if StreamRequest structure may be left in the StreamWaitQueue, and call “Strm_DequeueRequest()” until StreamCompleteQueue becomes empty, before closing streaming interface using “Strm_Close()”.

When StreamRequest structure retrieved by “Strm_DequeueRequest()” was moved to the StreamCompleteQueue using this function, TeliCamAPI will return CAM_API_STS_FLUSH_REQUESTED as result status of “Strm_DequeueRequest()”.

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
<i>hStrm</i>	[in]	Stream-Handle of target stream interface.
<i>hStrmRequest</i>	[in]	Handle of StreamRequest for retrieving its information.
<i>psStrmReqInfo</i>	[out]	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]	Pointer to the Leader data of the stream.
pvPayload	[out]	Pointer to the Payload data of the stream.
pvTrailer	[out]	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]	Pointer to the Leader data of the stream.
pvPayload	[out]	Pointer to the Payload data of the stream.
pvTrailer	[out]	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 stream interface for receiving images from camera.

[Syntax]

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

[Parameters]

Parameter	Description
<i>hStrm</i> [in]	Stream-Handle of target stream interface.

[Return value]

Returns result status. Refer to [5.8 Status code](#).

[Remarks]

When streaming interfaces are opened 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
<i>hStrm</i>	[in]	Stream-Handle of target stream interface.
<i>eAcqMode</i>	[in]	Image acquisition mode. If this parameter is not specified, continuous acquisition mode (CAM_ACQ_MODE_CONTINUOUS) will be used.

[CAM_ACQ_MODE_TYPE Enumeration]

Member	Description
<i>CAM_ACQ_MODE_CONTINUOUS</i>	The camera will acquire images and send them as stream continuously until "Strm_Stop()" is called.
<i>CAM_ACQ_MODE_MULTI_FRAME</i>	The camera will acquire images and send them as stream continuously until a number of images specified by "SetCamAcquisitionFrameCount()" are acquired. Calling "Strm_Stop()" is required even if streaming was stopped after the specified number of images were acquired.
<i>CAM_ACQ_MODE_IMAGE_BUFFER_READ</i>	The camera will acquire images and save 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 "

SetCamImageBufferMode()” before calling “Strm_Start()” with CAM_ACQ_MODE_IMAGE_BUFFER_READ mode, when the user application uses ImageBuffer mode. .

TeliCamAPI will write 1 to TLPParamsLocked register when this function is called. TLPParamsLocked 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
<i>hStrm</i> [in]	Stream-Handle of 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.

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
<i>hStrm</i> [in]	Stream-Handle of 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.

[Example]

Refer to [example](#) of [5.3.1.1 Strm_OpenSimple](#).

5.4. Camera event notification functions

TeliCamAPI provides two types of CameraEvent notification functions, High-level API and low-level API functions. Refer to [4.1.6 CameraEvent control](#).

Even though multiple applications can open the same camera simultaneously, user application cannot open a CameraEvent that the other application is using.

5.4.1. High-level API functions

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

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 event interface for receiving CameraEvent (message), creates EventRequest structures, and creates the EventRingBuffer for CameraEvent 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
<i>hCam</i>	[in]	Camera-Handle of target camera.
<i>phEvt</i>	[out]	Pointer to a variable that receives Event-Handle assigned to the opening event interface.
<i>uiApiBufferCount</i>	[in]	Size (number of EventRequest) of the EventRingBuffer 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]

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

Use GenApi functions to get detail information of a CameraEvent, when the event object is signaled. Refer to the following sample code.

If user application calls this method for a CameraEvent interface that the other application is using, this method will return CamApiStatus.AlreadyOpened.

Activating plural CameraEvent notifications to a camera will make response for CameraEvent notifications worse. In this case, TeliCamAPI may silently discard incoming EventRequest when no EventRequests are left for receiving the incoming CameraEvent data, if processing cost of user application EventHandler is heavy or CPU power is not enough.

TeliCamAPI will inform no error even if it discarded the incoming EventRequest.

We recommend users to refrain from activating plural camera event notifications to a camera while acquiring images continuously.

Including TeliCamAPI.h is required.

[Example]

```
C++

CAM_API_STATUS    uiStatus;
uint32_t          uiNum, uiPyldSize, uiRet;
int64_t           llVal;
CAM_HANDLE        hCam = NULL;
CAM_STRM_HANDLE   hStrm = NULL;
CAM_EVT_HANDLE    hEvent = NULL;
HANDLE            hStrmCmpEvt = NULL;
HANDLE            hFrmTrgEvt = NULL;
void*             pvPayloadBuf = NULL;
CAM_NODE_HANDLE   hNode = NULL;

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

__try
{
    // 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;

#ifdef _DEBUG
    // For Gig-E 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)
        return -1;

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

    // Open event interface.
```

```

        uiStatus = Evt_OpenSimple(hCam, &hEvent);
        if (uiStatus != CAM_API_STS_SUCCESS)
            return -1;

        // Create FrameTrigger event handle.
        hFrmTrgEvt = CreateEvent(NULL, FALSE, FALSE, NULL);
        if (hFrmTrgEvt == NULL)
            return -1;

        // Activate FrameTrigger event.
        uiStatus = Evt_Activate(hEvent, "FrameTrigger", hFrmTrgEvt);
        if (uiStatus != CAM_API_STS_SUCCESS)
            return -1;

        // Create completion event object for stream.
        hStrmCmpEvt = CreateEvent(NULL, FALSE, FALSE, NULL);
        if (hStrmCmpEvt == NULL)
            return -1;

        // Open stream interface.
        uiStatus = Strm_OpenSimple(hCam, &hStrm, &uiPyldSize, hStrmCmpEvt);
        if (uiStatus != CAM_API_STS_SUCCESS)
            return -1;

        // Allocate buffer for receiving image data.
        pvPayloadBuf = (void *)VirtualAlloc(NULL,
                                            uiPyldSize,
                                            MEM_RESERVE | MEM_COMMIT,
                                            PAGE_EXECUTE_READWRITE);

        if (pvPayloadBuf == NULL)
            return -1;

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

        for (uint32_t i = 0; i < 5; i++) {
            // Send Software Trigger command.
            uiStatus = ExecuteCamSoftwareTrigger(hCam);
            if (uiStatus != CAM_API_STS_SUCCESS)
                return -1;

            // Wait FrameTrigger event signaled.
            uiRet = WaitForSingleObject(hFrmTrgEvt, 2000);
            if (uiRet != WAIT_OBJECT_0)
                return -1;
            printf("Receive hEventCmpEvt %d : ", i);

            // Get Timestamp.
            uiStatus = Nd_GetNode(hCam, "EventFrameTriggerTimestamp", &hNode);
            if (uiStatus != CAM_API_STS_SUCCESS)
                return -1;

            uiStatus = Nd_GetIntValue(hCam, hNode, &llVal);
            if (uiStatus != CAM_API_STS_SUCCESS)
                return -1;
            printf("Timestamp%d = %I64u¥n", i, (uint64_t)llVal);

            // Wait for receiving image completion event.
            uiRet = WaitForSingleObject(hStrmCmpEvt, 2000);
            if (uiRet != WAIT_OBJECT_0)
                return -1;
        }

        return 0;
    }
__finally
{
    // Stop stream.

```

```

uiStatus = Strm_Stop(hStrm);
if (uiStatus != CAM_API_STS_SUCCESS)
    return -1;

if (hEvent != 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 != NULL) {
    uiStatus = Strm_Close(hStrm);
    if (uiStatus != CAM_API_STS_SUCCESS)
        printf("Strm_Close error! (0x%x)\n", uiStatus);
}

// Close camera.
if (hCam != 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 != NULL)
    CloseHandle(hStrmCmpEvt);

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

// Terminate system.
Sys_Terminate();
}

```

5.4.1.2. Evt_Activate

This function activates specified CameraEvent.

[Syntax]

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

[Parameters]

Parameter		Description
<i>hEvt</i>	[in]	Handle of target event interface.
<i>pszEvtName</i>	[in]	Pointer to a character array that contains node name (register name) of the activating CameraEvent.
<i>hCmpEvt</i>	[in]	Handle of an event object for notifying reception of CameraEvent from the camera. “CreateEvent()” of Win32 API is usually used for creating event objects. Specify NULL, if notification of the event is not necessary.

[Return value]

Returns result status. Refer to [5.8 Status code](#).

[Remarks]

This function is available when the event interface is opened using “Evt_OpenSimple()”.

User application can get information of CameraEvent that was activated using this function, when the event object [hCmpEvt](#) is signaled.

The event object [hCmpEvt](#) will be signaled on receiving CameraEvent notification from the camera.

The value specified to [pszEvtName](#) will be set to EventSelector node to activate notification of the CameraEvent. 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.

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

EVENT_NAME	Description	Node name of event information
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
AL Converged	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 (“Nd_GetNode()”).

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

User application has to reset [hCmpEvt](#) event object using “ResetEvent()”, otherwise TeliCamAPI cannot manage camera event in High-Level API mode.

For example, accessing to “EventFrameTrigger” node using “Nd_GetNode()” will make the [hCmpEvt](#) event object signaled if “FrameTrigger” event has been activated.

There are various restrictions in using camera event. 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 CcameraEvents.

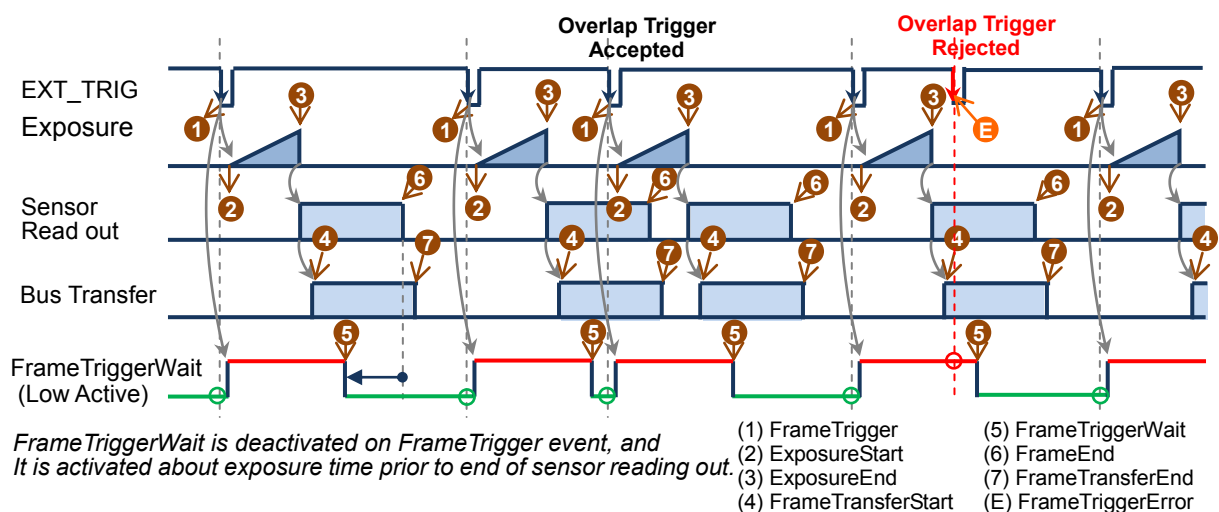


Figure 1 External Trigger mode

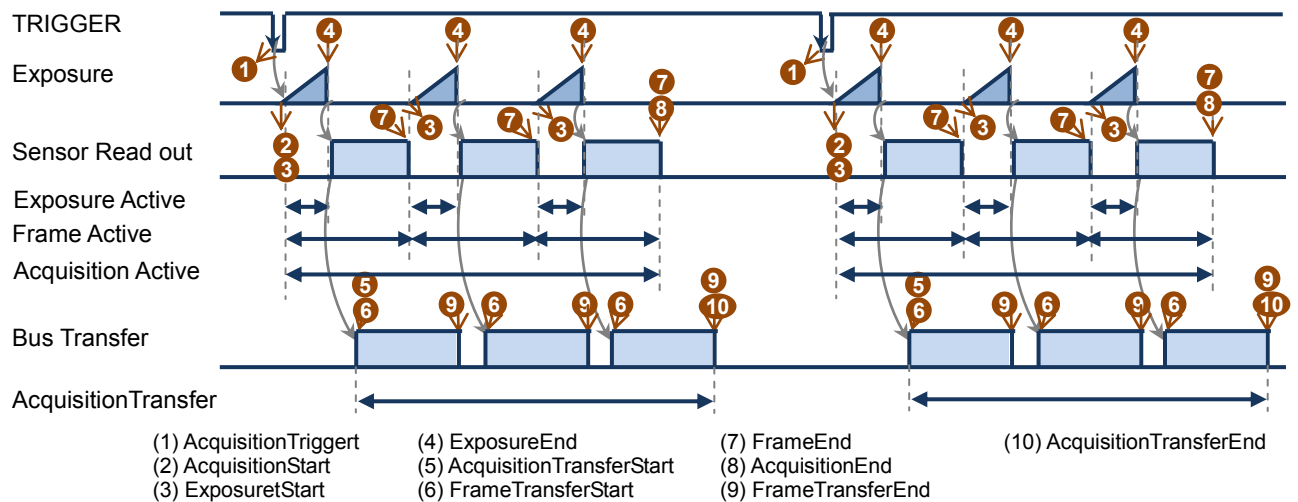


Figure 2 Bulk Trigger mode, 3 frames per a trigger

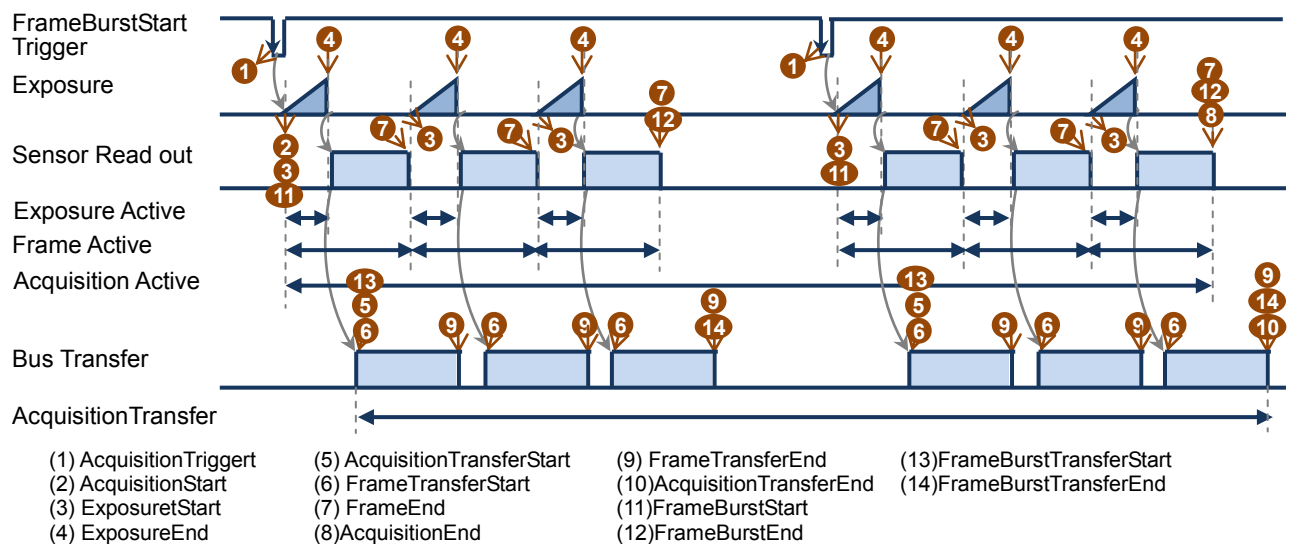


Figure 3 Burst Signals

[Example]

Refer to [example](#) of [5.4.1.1 Evt. OpenSimple](#).

5.4.1.3. Evt_Deactivate

This function deactivates specified camera event.

[Syntax]

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

[Parameters]

Parameter		Description
<i>hEvt</i>	[in]	Handle of target event interface.
<i>pszEvtName</i>	[in]	Pointer to a character array that contains node name (register name) of the deactivating camera event.

[Return value]

Returns result status. Refer to [5.8 Status code](#).

[Remarks]

This function is available when the event interface is opened using “Evt_OpenSimple()”.

The camera event that activated using “Evt_OpenSimple()” 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](#).

5.4.2. Low-Level API functions

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

Low-Level event functions of the current version TeliCamAPI are not available for GigE Vision camera.

5.4.2.1. Evt_Open

This function opens event interface for receiving CameraEvent (message).

[Syntax]

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

[Parameters]

Parameter		Description
<i>hCam</i>	[in]	Camera-Handle of target camera.
<i>phEvt</i>	[out]	Pointer to a variable that receives Event-Handle assigned to the opening event interface.
<i>puiMaxPayloadSize</i>	[in, out]	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.
<i>hCmpEvt</i>	[in]	Handle of an event object for notifying reception of CameraEvent from the camera. “CreateEvent()” of Win32 API is usually used for creating event object. Specify NULL, if notification of the camera event is not necessary.

[Return value]

Returns result status. Refer to [5.8 Status code](#).

[Remarks]

If user application calls this method for a CameraEvent interface that the other application is using, this method will return CamApiStatus.AlreadyOpened.

Note that if plural camera events are activated, the event 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++

```
// Only USB Vision Camera.
CAM_API_STATUS      uiStatus;
uint32_t            uiNum, uiPyldSize, uiRet, i;
CAM_HANDLE          hCam = NULL;
CAM_STRM_HANDLE     hStrm = NULL;
CAM_EVT_HANDLE      hEvent = NULL;
HANDLE              hStrmCmpEvt = NULL;
HANDLE              hEventCmpEvt = NULL;
void*               pvPayloadBuf = NULL;
void*               pvEvtPayloadBuf = NULL;
uint32_t            uiEvtMaxPayloadSize = 0;
CAM_NODE_HANDLE     hNode = NULL;
uint32_t            uiData;
uint64_t            ullAddress;
CAM_EVT_REQUEST_HANDLE hEvtRequest = NULL;
CAM_EVT_REQUEST_HANDLE hRcvEvtRequest = NULL;
EVT_REQUEST_INFO    sEventReqInfo;

// 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
{
#ifdef _DEBUG
    // For Gig-E 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)
        return -1;

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

    // Set event mode.
    {
        // Set TransferEvent register.
        ullAddress = 0x0021F230; // TransferEvent
        uiData = 0x00000000; // Start=OFF, End=OFF
        uiStatus = Cam_WriteReg(hCam, ullAddress, 1, (void*)&uiData);
        if (uiStatus != CAM_API_STS_SUCCESS)
            return -1;

        // Set ExposureEvent register.
        ullAddress = 0x0021F250; // ExposureEvent
        uiData = 0x00000001; // Start=ON, End=OFF
        uiStatus = Cam_WriteReg(hCam, ullAddress, 1, (void*)&uiData);
        if (uiStatus != CAM_API_STS_SUCCESS)
            return -1;
    }

    // Start & open event.
}
```

```

{
    // Create completion event for event.
    hEventCmpEvt = CreateEvent(NULL, FALSE, FALSE, NULL);
    if (hEventCmpEvt == NULL)
        return -1;

    // Open event interface.
    uiStatus = Evt_Open(hCam, &hEvent, &uiEvtMaxPayloadSize, hEventCmpEvt);
    if (uiStatus != CAM_API_STS_SUCCESS)
        return -1;

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

    // Allocate buffer for receiving event data.
    pvEvtPayloadBuf = (void *)VirtualAlloc(NULL,
                                           uiEvtMaxPayloadSize,
                                           MEM_RESERVE | MEM_COMMIT,
                                           PAGE_EXECUTE_READWRITE);

    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.
hStrmCmpEvt = CreateEvent(NULL, FALSE, FALSE, NULL);
if (hStrmCmpEvt == NULL)
    return -1;

// Open stream interface.
uiStatus = Strm_OpenSimple(hCam, &hStrm, &uiPyldSize, hStrmCmpEvt);
if (uiStatus != CAM_API_STS_SUCCESS)
    return -1;

// Allocate buffer for receiving image data.
pvPayloadBuf = (void *)VirtualAlloc(NULL,
                                     uiPyldSize,
                                     MEM_RESERVE | MEM_COMMIT,
                                     PAGE_EXECUTE_READWRITE);

if (pvPayloadBuf == NULL)
    return -1;

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

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.
    uiRet = WaitForSingleObject(hEventCmpEvt, 2000);
    if (uiRet != WAIT_OBJECT_0)
        return -1;
}

```

```

    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;

    printf(" request_id = %d, event_id = 0x%04x\n",
        sEventReqInfo.sU3vInfo.ushRequestId, sEventReqInfo.sU3vInfo.ushEventId);
    printf(" Timestamp = %I64u, pPayload = 0x%I64X\n",
        sEventReqInfo.sU3vInfo.u11Timestamp,
        (uint64_t)sEventReqInfo.sU3vInfo.pvPayload);

    // Wait for receiving image completion event.
    uiRet = WaitForSingleObject(hStrmCmpEvt, 2000);
    if (uiRet != WAIT_OBJECT_0)
        return -1;
}

// Stop stream.
uiStatus = Strm_Stop(hStrm);
if (uiStatus != CAM_API_STS_SUCCESS)
    return -1;

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

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

    // Close camera.
    if (hCam != 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 != NULL)
        CloseHandle(hStrmCmpEvt);

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

    // Release buffer for receiving event data.

```

```
if (pvEvtPayloadBuf != NULL)
    VirtualFree(pvEvtPayloadBuf, 0, MEM_RELEASE);

// Release buffer for receiving image data.
if (pvPayloadBuf != NULL)
    VirtualFree(pvPayloadBuf, 0, MEM_RELEASE);

// Terminate system.
Sys_Terminate();

printf("Completion.¥n");
}
```

5.4.2.2. Evt_CreateRequest

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

[Syntax]

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

[Parameters]

Parameter		Description
<i>hEvt</i>	[in]	Event-Handle of target event interface.
<i>pvPayloadBuf</i>	[in]	Pointer to a data buffer that receives the CameraEvent data. User application must prepare CameraEvent data buffer, whose size is at least value specified in uiPayloadSize argument.
<i>uiPayloadSize</i>	[in]	Payload size (event data size) of one CameraEvent data in bytes. Specify puiMaxPayloadSize argument value returned from Evt_Open(), in usual case.
<i>phEvtRequest</i>	[out]	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 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 CameraEvent data, TeliCamAPI will write received CameraEvent data to the oldest EventRequest in the EventWaitQueue, move the EventRequest filled with CameraEvent data to EventCompleteQueue in TeliCamAPI, and set [hCmpEvt event](#) signaled. User application can take out EventRequest from the EventCompleteQueue using “Evt_DequeueRequest()” to get the received CameraEvent data.

Releasing CameraEvent data buffer being used in EventRequest 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 EventCompleteQueue,
3. Take out all EventRequests from the EventCompleteQueue using “Evt_DequeueRequest()”.
4. Release EventRequests using “Evt_ReleaseRequest()”.
5. Release CameraEvent data buffer.

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
<i>hEvt</i>	[in]	Event-Handle of target event interface.
<i>hEvtRequest</i>	[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 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 CameraEvent data from the camera.

[Syntax]

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

[Parameters]

Parameter		Description
<i>hEvt</i>	[in]	Event-Handle of target event interface.
<i>hEvtRequest</i>	[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 event interface was opened using “Evt_OpenSimple()”, this function does not work correctly.

On receiving CameraEvent, TeliCamAPI will take out an EventRequest from the EventWaitQueue and start writing received CameraEvent data to event data block in the EventRequest.

On completion of receiving CameraEvent data, TeliCamAPI will put the EventRequest into the EventCompleteQueue inside TeliCamAPI.

It is necessary to keep plural EventRequest structures in the EventWaitQueue by appending EventRequests, whose event data will not be used any more, to the EventWaitQueue periodically, to receive all CameraEvent data without dropping out.

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
<i>hEvt</i>	[in]	Event-Handle of target event interface.
<i>phEvtRequest</i>	[out]	Pointer to a variable that receives handle of the retrieved EventRequest.
<i>psEvtReqInfo</i>	[out]	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 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 status 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
<i>hEvt</i> [in]	Event-Handle of 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 event interface was opened using "Evt_OpenSimple()", this function does not work correctly.

Before closing 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 streaming 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
<i>hEvt</i> [in]	Event-Handle of target event interface.

[Return value]

Returns result status. Refer to [5.8 Status code](#).

[Remarks]

When event interfaces are opened using “Evt_OpenSimple()” or “Evt_Open()”, never forget to close them using this function after finished using them.

Never close 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](#).

5.5. 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.

In GigE Vision camera case, all functions in this section use GenICam GenApi library for controlling camera. In USB3 Vision camera, most functions in this section access camera register directly, which will realize the higher performance. The functions that use GenICam GenApi function 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 camera does not support the feature.

Please check instruction manual of the camera to confirm the availability of the feature.

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

Note that Enum values used in functions of this section 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 function group performs control of image format of the camera. (Format0 ~ Format2)
The function of image format control depends on the camera or camera's version.
Refer to instruction manual of the camera about image format control.

5.5.1.1. GetCamImageFormatSelector

This function reports image format of the camera.

[Syntax]

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

[Parameters]

Parameter	Description
<i>hCam</i> [in]	Camera-Handle of target camera.
<i>peFormat</i> [out]	Pointer to a variable that receives current image format value.

[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.

Refer to TeliCamAPI.h about CAM_IMAGE_FORMAT_SELECTOR_TYPE.

Including TeliCamAPI.h is required.

5.5.1.2. SetCamImageFormatSelector

This function writes new value to image format selector node.

[Syntax]

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

[Parameters]

Parameter		Description
<i>hCam</i>	[in]	Camera-Handle of target camera.
<i>eFormat</i>	[in]	New image format value.

[Return value]

Returns result status. Refer to [5.8 Status code](#).

[Remarks]

Refer to TeliCamAPI.h about CAM_IMAGE_FORMAT_SELECTOR_TYPE.

Settings can not be changed during image stream data output.

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

Including TeliCamAPI.h is required.

5.5.2. Scalable

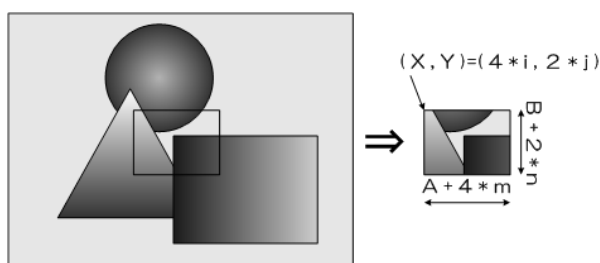
This function group performs control of scalable mode of the camera.

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



For more details about scalable mode, refer to instruction manual of the camera.

5.5.2.1. GetCamSensorWidth

This function reports effective width of the sensor in pixels,

[Syntax]

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

[Parameters]

Parameter	Description
<i>hCam</i> [in]	Camera-Handle of target camera.
<i>puiSensorWidth</i> [out]	Pointer to a variable that receives 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

This function reports effective height of the sensor in pixels,

[Syntax]

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

[Parameters]

Parameter		Description
<i>hCam</i>	[in]	Camera-Handle of target camera.
<i>puiSensorHeight</i>	[out]	Pointer to a variable that receives 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

This function reports ROI (Region of Interest) 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
<i>hCam</i>	[in]	Camera-Handle of target camera.
<i>puiWidth</i>	[out]	Pointer to a variable that receives current image width, in pixels.
<i>puiHeight</i>	[out]	Pointer to a variable that receives current image height ,in pixels.
<i>puiOffsetX</i>	[out]	Pointer to a variable that receives current horizontal offset from the origin to the region of interest, in pixels.
<i>puiOffsetY</i>	[out]	Pointer to a variable that receives 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

This function writes new ROI (Region of Interest) setting to 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
<i>hCam</i>	[in]	Camera-Handle of target camera.
<i>uiWidth</i>	[in]	New image width in pixels.
<i>uiHeight</i>	[in]	New image height in pixels.
<i>uiOffsetX</i>	[in]	New horizontal offset in pixels from the origin to the region of interest.
<i>uiOffsetY</i>	[in]	New vertical offset in pixels from the origin to the region of interest.

[Return value]

Returns result status. Refer to [5.8 Status code](#).

[Remarks]

Refrain from using this function during image streaming is active.

Writing data to ROI registers during image streaming is active is not acceptable.

Including TeliCamAPI.h is required.

5.5.2.5. GetCamWidthMinMax

This function reports the maximum, the minimum, and pitch of width that the camera accepts.

[Syntax]

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

[Parameters]

Parameter		Description
<i>hCam</i>	[in]	Camera-Handle of target camera.
<i>puiWidthMin</i>	[out]	Pointer to a variable that receives the minimum image width in pixels.
<i>puiWidthMax</i>	[out]	Pointer to a variable that receives the maximum image width in pixels.
<i>puiWidthInc</i>	[out]	Pointer to a variable that receives the pitch of 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

This function reports the width of image data in pixels.

[Syntax]

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

[Parameters]

Parameter		Description
<i>hCam</i>	[in]	Camera-Handle of target camera.
<i>puiWidth</i>	[out]	Pointer to a variable that receives 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

This function writes new image width in pixels to the camera.

[Syntax]

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

[Parameters]

Parameter		Description
<i>hCam</i>	[in]	Camera-Handle of target camera.
<i>uiWidth</i>	[in]	New image width in pixels.

[Return value]

Returns result status. Refer to [5.8 Status code](#).

[Remarks]

Refrain from using this function during image streaming is active.

Writing data to width register during image streaming is active is not acceptable.

Including TeliCamAPI.h is required.

5.5.2.8. GetCamHeightMinMax

This function reports the maximum, the minimum, and pitch of height that the camera accepts.

[Syntax]

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

[Parameters]

Parameter		Description
<i>hCam</i>	[in]	Camera-Handle of target camera.
<i>puiHeightMin</i>	[out]	Pointer to a variable that receives the minimum image height, in pixels.
<i>puiHeightMax</i>	[out]	Pointer to a variable that receives the maximum image height, in pixels.
<i>puiHeightInc</i>	[out]	Pointer to a variable that receives the pitch of 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

This function reports the height of image data in pixels.

[Syntax]

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

[Parameters]

Parameter		Description
<i>hCam</i>	[in]	Camera-Handle of target camera.
<i>puiHeight</i>	[out]	Pointer to a variable that receives 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

This function writes new image height in pixels to the camera.

[Syntax]

```
CAM_API_STATUS SetCamHeight (  
    CAM_HANDLE      hCam,  
    uint32_t        uiHeight  
);
```

[Parameters]

Parameter		Description
<i>hCam</i>	[in]	Camera-Handle of target camera..
<i>uiHeight</i>	[in]	New image height, in pixels.

[Return value]

Returns result status. Refer to [5.8 Status code](#).

[Remarks]

Refrain from using this function during image streaming is active.

Writing data to height register during image streaming is active is not acceptable.

Including TeliCamAPI.h is required.

5.5.2.11. GetCamOffsetXMinMax

This function reports the maximum, minimum, and pitch value of horizontal image offset in pixels that the camera accepts.

[Syntax]

```
CAM_API_STATUS GetCamOffsetXMinMax (  
    CAM_HANDLE      hCam,  
    uint32_t         *puiOffsetXMin,  
    uint32_t         *puiOffsetXMax,  
    uint32_t         *puiOffsetXInc  
);
```

[Parameters]

Parameter		Description
<i>hCam</i>	[in]	Camera-Handle of target camera.
<i>puiOffsetXMin</i>	[out]	Pointer to a variable that receives the minimum horizontal image offset, in pixels.
<i>puiOffsetXMax</i>	[out]	Pointer to a variable that receives the maximum horizontal image offset, in pixels.
<i>puiOffsetXInc</i>	[out]	Pointer to a variable that receives the pitch 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

This function reports the horizontal offset of image data in pixels.

[Syntax]

```
CAM_API_STATUS GetCamOffsetX (  
    CAM_HANDLE      hCam,  
    uint32_t        *puiOffsetX  
);
```

[Parameters]

Parameter		Description
<i>hCam</i>	[in]	Camera-Handle of target camera.
<i>puiOffsetX</i>	[out]	Pointer to a variable that receives 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

This function writes new horizontal offset of image data in pixels to the camera.

[Syntax]

```
CAM_API_STATUS SetCamOffsetX (  
    CAM_HANDLE      hCam,  
    uint32_t        uiOffsetX  
);
```

[Parameters]

Parameter		Description
<i>hCam</i>	[in]	Camera-Handle of target camera.
<i>uiOffsetX</i>	[in]	New horizontal image offset, in pixels.

[Return value]

Returns result status. Refer to [5.8 Status code](#).

[Remarks]

Refrain from using this function during image streaming is active.

Writing data to OffsetX register during image streaming is active is not acceptable.

Including TeliCamAPI.h is required.

5.5.2.14. GetCamOffsetYMinMax

This function reports the maximum, minimum, and pitch value of vertical image offset in pixels that the camera accepts.

[Syntax]

```
CAM_API_STATUS GetCamOffsetYMinMax (  
    CAM_HANDLE      hCam,  
    uint32_t         *puiOffsetYMin,  
    uint32_t         *puiOffsetYMax,  
    uint32_t         *puiOffsetYInc  
);
```

[Parameters]

Parameter		Description
<i>hCam</i>	[in]	Camera-Handle of target camera.
<i>puiOffsetYMin</i>	[out]	Pointer to a variable that receives the minimum vertical image offset, in pixels.
<i>puiOffsetYMax</i>	[out]	Pointer to a variable that receives the maximum vertical image offset, in pixels.
<i>puiOffsetYInc</i>	[out]	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

This function reports the vertical offset of image data in pixels.

[Syntax]

```
CAM_API_STATUS GetCamOffsetY (  
    CAM_HANDLE      hCam,  
    uint32_t        *puiOffsetY  
);
```

[Parameters]

Parameter		Description
<i>hCam</i>	[in]	Camera-Handle of target camera.
<i>puiOffsetY</i>	[out]	Pointer to a variable that receives 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

This function writes new vertical offset of image data in pixels to the camera.

[Syntax]

```
CAM_API_STATUS SetCamOffsetY (  
    CAM_HANDLE      hCam,  
    uint32_t         puiOffsetY  
);
```

[Parameters]

Parameter		Description
<i>hCam</i>	[in]	Camera-Handle of target camera.
<i>puiOffsetY</i>	[in]	New vertical image offset, in pixels.

[Return value]

Returns result status. Refer to [5.8 Status code](#).

[Remarks]

Refrain from using this function during image streaming is active.

Writing data to OffsetY register during image streaming is active is not acceptable.

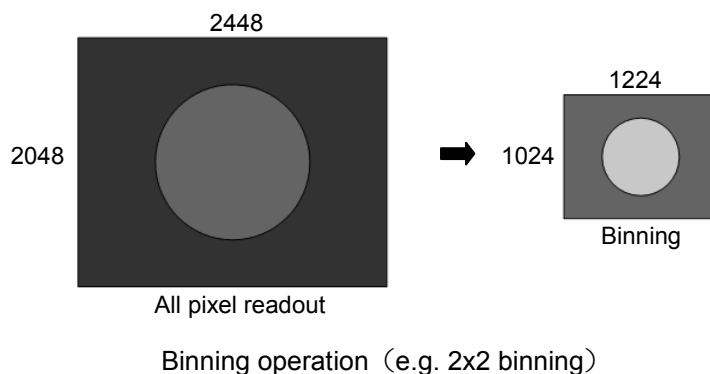
Including TeliCamAPI.h is required.

5.5.3. Binning

This function group performs control of binning feature of the camera.

Binning feature adds the neighboring pixels together.

Binning feature can increase the sensitivity of the image, make frame rate faster, and decrease interface bandwidth occupation.



For more details about binning feature, refer to instruction manual of the camera.

5.5.3.1. GetCamBinningHorizontalMinMax

This function reports the maximum, the minimum value of horizontal binning setting that the camera accepts.

[Syntax]

```
CAM_API_STATUS GetCamBinningHorizontalMinMax (  
    CAM_HANDLE      hCam,  
    uint32_t         *puiMin  
    uint32_t         *puiMax  
);
```

[Parameters]

Parameter		Description
<i>hCam</i>	[in]	Camera-Handle of target camera.
<i>puiMin</i>	[out]	Pointer to a variable that receives the minimum horizontal binning setting.
<i>puiMax</i>	[out]	Pointer to a variable that receives the maximum horizontal binning setting.

[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, refer to the operation manual of the camera.

Including TeliCamAPI.h is required.

5.5.3.2. GetCamBinningHorizontal

This function reports the BinningHorizontal register value.

[Syntax]

```
CAM_API_STATUS GetCamBinningHorizontal (  
    CAM_HANDLE      hCam,  
    uint32_t         *puiValue  
);
```

[Parameters]

Parameter	Description
<i>hCam</i> [in]	Camera-Handle of target camera.
<i>puiValue</i> [out]	Pointer to a variable that receives current BinningHorizontal register value.

[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, refer to the operation manual of the camera.

Including TeliCamAPI.h is required.

5.5.3.3. SetCamBinningHorizontal

This function writes new horizontal binning setting to the camera.

[Syntax]

```
CAM_API_STATUS SetCamBinningHorizontal (  
    CAM_HANDLE      hCam,  
    uint32_t        uiValue  
);
```

[Parameters]

Parameter		Description
<i>hCam</i>	[in]	Camera-Handle of target camera.
<i>uiValue</i>	[in]	New horizontal binning value.

[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.

Refrain from using this function during image streaming is active.

Writing data to BinningHorizontal register during image streaming is active is not acceptable.

Conditions that can get and set the binning parameters depend on the camera.

For details, refer to the operation manual of the camera.

Including TeliCamAPI.h is required.

5.5.3.4. GetCamBinningVerticalMinMax

This function reports the maximum, the minimum value of vertical binning setting that the camera accepts.

[Syntax]

```
CAM_API_STATUS GetCamBinningVerticalMinMax (  
    CAM_HANDLE      hCam,  
    uint32_t        *puiMin  
    uint32_t        *puiMax  
);
```

[Parameters]

Parameter		Description
<i>hCam</i>	[in]	Camera-Handle of target camera.
<i>puiMin</i>	[out]	Pointer to a variable that receives the minimum vertical binning setting.
<i>puiMax</i>	[out]	Pointer to a variable that receives the maximum vertical binning setting.

[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, refer to the operation manual of the camera.

Including TeliCamAPI.h is required.

5.5.3.5. GetCamBinningVertical

This function reports the BinningVertical register value.

[Syntax]

```
CAM_API_STATUS GetCamBinningVertical (  
    CAM_HANDLE      hCam,  
    uint32_t        *puiValue  
);
```

[Parameters]

Parameter		Description
<i>hCam</i>	[in]	Camera-Handle of target camera.
<i>puiValue</i>	[out]	Pointer to a variable that receives current BinningVertical register value.

[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, refer to the operation manual of the camera.

Including TeliCamAPI.h is required.

5.5.3.6. SetCamBinningVertical

This function writes new vertical binning setting to the camera.

[Syntax]

```
CAM_API_STATUS SetCamBinningVertical (  
    CAM_HANDLE      hCam,  
    uint32_t        uiValue  
);
```

[Parameters]

Parameter		Description
<i>hCam</i>	[in]	Camera-Handle of target camera.
<i>uiValue</i>	[in]	New vertical binning value.

[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.

Refrain from using this function during image streaming is active.

Writing data to BinningVertical register during image streaming is active is not acceptable.

Conditions that can get and set the binning parameters depend on the camera.

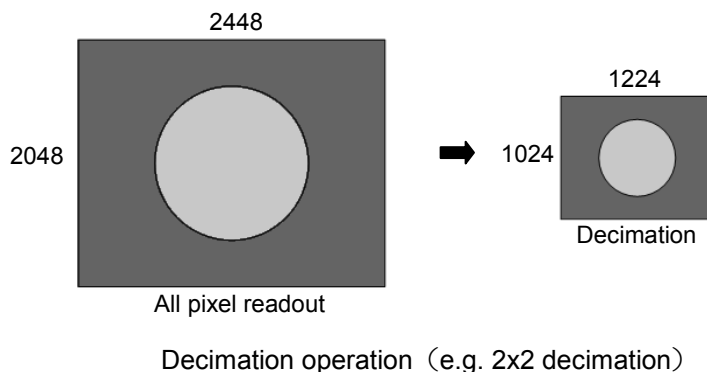
For details, refer to the operation manual of the camera.

Including TeliCamAPI.h is required.

5.5.4. Decimation

This function group performs control of decimation feature of the camera.

Decimation feature reads out all effective areas at high speed by skipping pixels and lines. Decimation feature can make frame rate faster, and decrease interface bandwidth occupation.



For more details about decimation feature, refer to instruction manual of the camera.

5.5.4.1. GetCamDecimationHorizontalMinMax

This function reports the maximum, the minimum value of horizontal decimation setting that the camera accepts.

[Syntax]

```
CAM_API_STATUS GetCamDecimationHorizontalMinMax (  
    CAM_HANDLE      hCam,  
    uint32_t         *puiMin  
    uint32_t         *puiMax  
);
```

[Parameters]

Parameter	Description
<i>hCam</i> [in]	Camera-Handle of target camera.
<i>puiMin</i> [out]	Pointer to a variable that receives the minimum horizontal decimation setting.
<i>puiMax</i> [out]	Pointer to a variable that receives the maximum horizontal decimation setting.

[Return value]

Returns result status. Refer to

[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, refer to the operation manual of the camera.

Including TeliCamAPI.h is required.

5.5.4.2. GetCamDecimationHorizontal

This function reports the DecimationHorizontal register value.

[Syntax]

```
CAM_API_STATUS GetCamDecimationHorizontal (  
    CAM_HANDLE      hCam,  
    uint32_t        *puiValue  
);
```

[Parameters]

Parameter		Description
<i>hCam</i>	[in]	Camera-Handle of target camera.
<i>puiValue</i>	[out]	Pointer to a variable that receives current value of DecimationHorizontal register.

[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, refer to the operation manual of the camera.

Including TeliCamAPI.h is required.

5.5.4.3. SetCamDecimationHorizontal

This function writes new horizontal decimation setting to the camera.

[Syntax]

```
CAM_API_STATUS SetCamDecimationHorizontal (  
    CAM_HANDLE      hCam,  
    uint32_t        uiValue  
);
```

[Parameters]

Parameter		Description
<i>hCam</i>	[in]	Camera-Handle of target camera.
<i>uiValue</i>	[in]	New horizontal decimation value.

[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.

Refrain from using this function during image streaming is active.

Writing data to DecimationHorizontal register during image streaming is active is not acceptable.

Conditions that can get and set the decimation parameters depend on the camera.

For details, refer to the operation manual of the camera.

Including TeliCamAPI.h is required.

5.5.4.4. GetCamDecimationVerticalMinMax

This function reports the maximum, the minimum value of vertical decimation setting that the camera accepts.

[Syntax]

```
CAM_API_STATUS GetCamDecimationVerticalMinMax (  
    CAM_HANDLE      hCam,  
    uint32_t        *puiMin  
    uint32_t        *puiMax  
);
```

[Parameters]

Parameter	Description
<i>hCam</i> [in]	Camera-Handle of target camera.
<i>puiMin</i> [out]	Pointer to a variable that receives the minimum vertical decimation setting.
<i>puiMax</i> [out]	Pointer to a variable that receives the maximum vertical decimation setting.

[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, refer to the operation manual of the camera.

Including TeliCamAPI.h is required.

5.5.4.5. GetCamDecimationVertical

This function reports the DecimationVertical register value.

[Syntax]

```
CAM_API_STATUS GetCamDecimationVertical (  
    CAM_HANDLE      hCam,  
    uint32_t        *puiValue  
);
```

[Parameters]

Parameter		Description
<i>hCam</i>	[in]	Camera-Handle of target camera.
<i>puiValue</i>	[out]	Pointer to a variable that receives current DecimationVertical register value.

[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, refer to the operation manual of the camera.

Including TeliCamAPI.h is required.

5.5.4.6. SetCamDecimationVertical

This function writes new vertical decimation setting to the camera.

[Syntax]

```
CAM_API_STATUS SetCamDecimationVertical (  
    CAM_HANDLE      hCam,  
    uint32_t        uiValue  
);
```

[Parameters]

Parameter		Description
<i>hCam</i>	[in]	Camera-Handle of target camera.
<i>uiValue</i>	[in]	New vertical decimation value.

[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.

Refrain from using this function during image streaming is active.

Writing data to DecimationVertical register during image streaming is active is not acceptable.

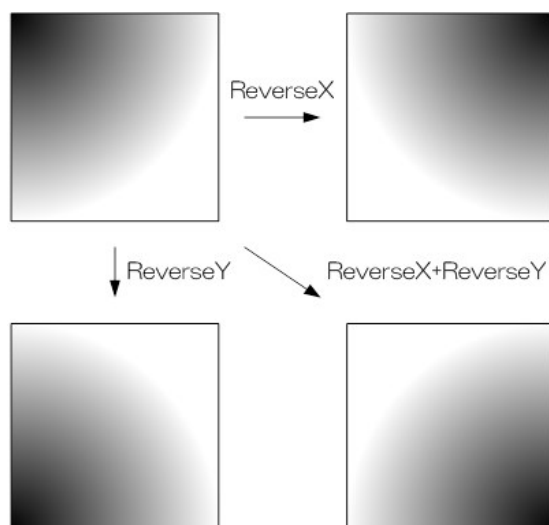
Conditions that can get and set the decimation parameters depend on the camera.

For details, refer to the operation manual of the camera.

Including TeliCamAPI.h is required.

5.5.5. Reverse

This function group performs control of image reversing (mirroring) function of the camera.



Refer to instruction manual of the camera about Reverse.

5.5.5.1. GetCamReverseX

This function reports the ReverseX register value.

[Syntax]

```
CAM_API_STATUS GetCamReverseX (  
    CAM_HANDLE      hCam,  
    bool8_t         *pbValue  
);
```

[Parameters]

Parameter		Description
<i>hCam</i>	[in]	Camera-Handle of target camera.
<i>pbValue</i>	[out]	Pointer to a variable that receives current ReverseX register value. If the value is true, image will be reversed horizontally. if the value is false, reversing horizontal is disabled.

[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

This function writes new horizontal reversing setting to the camera.

[Syntax]

```
CAM_API_STATUS SetCamReverseX (  
    CAM_HANDLE      hCam,  
    bool18_t        bValue  
);
```

[Parameters]

Parameter		Description
<i>hCam</i>	[in]	Camera-Handle of target camera.
<i>bValue</i>	[in]	New horizontal reversing value. If the value is true, image will be reversed horizontally. if the value is false, reversing horizontal is disabled.

[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.

Refrain from using this function during image streaming is active.

Writing data to ReverseX register during image streaming is active is not acceptable.

Including TeliCamAPI.h is required.

5.5.5.3. GetCamReverseY

This function reports the ReverseY register value.

[Syntax]

```
CAM_API_STATUS GetCamReverseY (  
    CAM_HANDLE      hCam,  
    bool18_t        *pbValue  
);
```

[Parameters]

Parameter		Description
<i>hCam</i>	[in]	Camera-Handle of target camera.
<i>pbValue</i>	[out]	Pointer to a variable that receives current ReverseY register value. If the value is true, image will be reversed vertically. if the value is false, reversing vertical is disabled.

[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

This function writes new vertical reversing setting to the camera.

[Syntax]

```
CAM_API_STATUS SetCamReverseY (  
    CAM_HANDLE      hCam,  
    bool18_t        bValue  
);
```

[Parameters]

Parameter		Description
<i>hCam</i>	[in]	Camera-Handle of target camera.
<i>bValue</i>	[in]	New vertical reversing value. If the value is true, image will be reversed vertically. if the value is false, reversing vertical is disabled.

[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.

Refrain from using this function during image streaming is active.

Writing data to ReverseY register during image streaming is active is not acceptable.

Including TeliCamAPI.h is required.

5.5.6. PixelFormat

This function group performs control of PixelFormat of image to be acquired.

Refer to instruction manual of the camera or section 6.5 of document “How to Use GenApi with TeliGevSDK and TeliU3vSDK” downloadable from our home page about PixelFormat.

5.5.6.1. GetCamPixelFormat

This function reports cuurrent PixelFormat setting.

[Syntax]

```
CAM_API_STATUS GetCamPixelFormat (  
    CAM_HANDLE          hCam,  
    CAM_PIXEL_FORMAT    *puiPixelFormat  
);
```

[Parameters]

Parameter		Description
<i>hCam</i>	[in]	Camera-Handle of target camera.
<i>puiPixelFormat</i>	[out]	Pointer to a variable that receives current PixelFormat value.

[CAM_PIXEL_FORMAT Enumeration]

Member	Description
<i>PXL_FMT_Unknown</i>	Unknown format
<i>PXL_FMT_Mono8</i>	Mono8 format
<i>PXL_FMT_Mono10</i>	Mono10 format
<i>PXL_FMT_Mono12</i>	Mono12 format
<i>PXL_FMT_Mono16</i>	Mono16 format
<i>PXL_FMT_BayerGR8</i>	BayerGR8 format
<i>PXL_FMT_BayerGR10</i>	BayerGR10 format
<i>PXL_FMT_BayerGR12</i>	BayerGR12 format
<i>PXL_FMT_BayerRG8</i>	BayerRG8 format
<i>PXL_FMT_BayerRG10</i>	BayerRG10 format
<i>PXL_FMT_BayerRG12</i>	BayerRG12 format
<i>PXL_FMT_BayerGB8</i>	BayerGB8 format
<i>PXL_FMT_BayerGB10</i>	BayerGB10 format
<i>PXL_FMT_BayerGB12</i>	BayerGB12 format
<i>PXL_FMT_BayerBG8</i>	BayerBG8 format
<i>PXL_FMT_BayerBG10</i>	BayerBG10 format
<i>PXL_FMT_BayerBG12</i>	BayerBG12 format
<i>PXL_FMT_RGB8</i>	RGB8 format
<i>PXL_FMT_BGR8</i>	BGR8 format
<i>PXL_FMT_BGR10</i>	BGR10 format
<i>PXL_FMT_BGR12</i>	BGR12 format
<i>PXL_FMT_YUV411_8</i>	YUV411_8 format
<i>PXL_FMT_YUV422_8</i>	YUV422_8 format
<i>PXL_FMT_YUV8</i>	YUV8 format

[Return value]

Returns result status. Refer to [5.8 Status code](#).

[Remarks]

Refer to TeliCamAPI.h about CAM_PIXEL_FORMAT.

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

Including TeliCamAPI.h is required.

5.5.6.2. SetCamPixelFormat

This function writes new PixelFormat setting to the camera.

[Syntax]

```
CAM_API_STATUS SetCamPixelFormat (  
    CAM_HANDLE          hCam,  
    CAM\_PIXEL\_FORMAT     uiPixelFormat  
);
```

[Parameters]

Parameter		Description
<i>hCam</i>	[in]	Camera-Handle of target camera.
<i>uiPixelFormat</i>	[in]	New PixelFormat value.

[Return value]

Returns result status. Refer to [5.8 Status code](#).

[Remarks]

Set PixelFormat register or PixelCoding register and PixelSize register.
(The registers to be accessed is different depending on the camera.)

Refer to TeliCamAPI.h about CAM_PIXEL_FORMAT.

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

Refrain from using this function during image streaming is active.

PixelFormat register or PixelCoding and PixelSize registers are protected from writing value during streaming is active.

Including TeliCamAPI.h is required.

5.5.7. TestPattern

This function group performs control of test pattern generation function of the camera.
Refer to instruction manual of the camera about test pattern (test image).

5.5.7.1. GetCamTestPattern

This function reports the current test pattern (test image) setting.

[Syntax]

```
CAM_API_STATUS GetCamTestPattern (  
    CAM_HANDLE          hCam,  
    CAM_TEST_PATTERN_TYPE *peTestPattern  
);
```

[Parameters]

Parameter		Description
<i>hCam</i>	[in]	Camera-Handle of target camera.
<i>peTestPattern</i>	[out]	Pointer to a variable that receives current setting of test pattern (test image) .

[CAM_TEST_PATTERN_TYPE Enumeration]

Member	Description
<i>CAM_TEST_PATTERN_OFF</i>	OFF
<i>CAM_TEST_PATTERN_BLACK</i>	Black
<i>CAM_TEST_PATTERN_WHITE</i>	White
<i>CAM_TEST_PATTERN_GREY_A</i>	GreyA
<i>CAM_TEST_PATTERN_GREY_B</i>	GreyB
<i>CAM_TEST_PATTERN_GREY_HORIZONTAL_RAMP</i>	GreyHorizontalRamp
<i>CAM_TEST_PATTERN_GREY_SCALE</i>	GreyScale
<i>CAM_TEST_PATTERN_COLOR_BAR</i>	ColorBar
<i>CAM_TEST_PATTERN_GREY_VERTICAL_RAMP</i>	GreyVerticalramp

[Return value]

Returns result status. Refer to [5.8 Status code](#).

[Remarks]

Refer to TeliCamAPI.h about CAM_TEST_PATTERN_TYPE.

BU series camera, BG series camera and GiantDragon series camera have similar test patterns inside them, but integer value for setting a similar test pattern may be different. String value for setting a similar test pattern may be also different. This function converts register or node value of the camera to CAM_TEST_PATTERN_TYPE value to avoid confusion.

This function will return error status if “TestPattern” or “TestImageSelector” register or node is not implemented in the camera.

Including TeliCamAPI.h is required.

5.5.7.2. SetCamTestPattern

This function writes new test pattern (test image) setting to the camera.

[Syntax]

```
CAM_API_STATUS SetCamTestPattern (  
    CAM_HANDLE          hCam,  
    CAM_TEST_PATTERN_TYPE eTestPattern  
);
```

[Parameters]

Parameter		Description
<i>hCam</i>	[in]	Camera-Handle of target camera.
<i>eTestPattern</i>	[in]	New test pattern (test image) value.

[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.

Writing data to “TestPattern” or “TestImageSelector” register during image streaming is active is not acceptable.

Including TeliCamAPI.h is required.

5.5.8. AcquisitionControl

This function group performs control of image acquisition of the camera.
Refer to instruction manual of the camera about image acquisition.

5.5.8.1. GetCamStreamPayloadSize

This function reports current payload size setting. Payload size is usually same as image size.

[Syntax]

```
CAM_API_STATUS GetCamStreamPayloadSize (  
    CAM_HANDLE      hCam,  
    uint32_t        *puiPayloadSize  
);
```

[Parameters]

Parameter		Description
<i>hCam</i>	[in]	Camera-Handle of target camera.
<i>puiPayloadSize</i>	[out]	Pointer to a variable that receives 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

This function reports the status of image streaming.

[Syntax]

```
CAM_API_STATUS GetStreamEnable (  
    CAM_HANDLE      hCam,  
    bool18_t        *pbEnable  
);
```

[Parameters]

Parameter		Description
<i>hCam</i>	[in]	Camera-Handle of target camera.
<i>pbEnable</i>	[out]	Pointer to a variable that receives current status of image streaming. if the value is true, image streaming is active, otherwise image streaming is inactive.

[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

This function reports the maximum and the minimum value that the camera accepts as Acquisition Frame Count value for Multi Frame mode. AcquisitionFrameCount register value is used as transfer image count in ImageBufferRead mode, too.

[Syntax]

```
CAM_API_STATUS GetCamAcquisitionFrameCountMinMax (  
    CAM_HANDLE      hCam,  
    uint32_t         *puiAcqFrameCountMin,  
    uint32_t         *puiAcqFrameCountMax  
);
```

[Parameters]

Parameter		Description
<i>hCam</i>	[in]	Camera-Handle of target camera.
<i>puiAcqFrameCountMin</i>	[out]	Pointer to a variable that receives the minimum frame count value.
<i>puiAcqFrameCountMax</i>	[out]	Pointer to a variable that receives the maximum frame count value.

[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

This function reports current Acquisition Frame Count setting.

Camera will stop acquiring image when Acquisition Frame Count value frames of image has been acquired.

This value is also used in ExecuteCamImageBufferRead().

Camera will send Acquisition Frame Count value frames of image from the image buffer in the camera on each calling ExecuteCamImageBufferRead().

[Syntax]

```
CAM_API_STATUS GetCamAcquisitionFrameCount (
    CAM_HANDLE      hCam,
    uint32_t         *puiAcqFrameCount
);
```

[Parameters]

Parameter	Description
<i>hCam</i> [in]	Camera-Handle of target camera.
<i>puiAcqFrameCount</i> [out]	Pointer to a variable that receives current acquisition frame count value .

[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

This function writes new AcquisitionFrameCount value to the camera.

Camera will stop acquiring image when AcquisitionFrameCount value frames of image has been acquired.

This value is also used in ExecuteCamImageBufferRead().

Camera will send Acquisition Frame Count value frames of image from the image buffer in the camera on each calling ExecuteCamImageBufferRead().

[Syntax]

```
CAM_API_STATUS SetCamAcquisitionFrameCount (  
    CAM_HANDLE      hCam,  
    uint32_t         uiAcqFrameCount  
);
```

[Parameters]

Parameter		Description
<i>hCam</i>	[in]	Camera-Handle of target camera.
<i>uiAcqFrameCount</i>	[in]	New acquisition frame count value.

[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.

Refrain from using this function during image streaming is active.

Writing data to AcquisitionFrameCount register during image streaming is active is not acceptable.

Including TeliCamAPI.h is required.

5.5.8.6. GetCamAcquisitionFrameRateControl

This function reports AcquisitionFrameRateControl register value.

[Syntax]

```
CAM_API_STATUS GetCamAcquisitionFrameRateControl (  
    CAM_HANDLE          hCam,  
    CAM_ACQ_FRAME_RATE_CTRL_TYPE *peFrameRateControl  
);
```

[Parameters]

Parameter	Description
<i>hCam</i> [in]	Camera-Handle of target camera.
<i>peFrameRateControl</i> [out]	Pointer to a variable that receives current AcquisitionFrameRateControl register value.

[CAM_ACQ_FRAME_RATE_CTRL_TYPE Enumeration]

Member	Description
<i>CAM_ACQ_FRAME_RATE_CTRL_NO_SPECIFY</i>	The camera will decide frame rate using parameters other than AcquisitionFrameRate register and the decided frame rate value will be read out from AcquisitionFrameRate register.
<i>CAM_ACQ_FRAME_RATE_CTRL_MANUAL</i>	The camera will control frame rate to realize AcquisitionFrameRate register value.

[Return value]

Returns result status. Refer to [5.8 Status code](#).

[Remarks]

Refer to TeliCamAPI.h about CAM_ACQ_FRAME_RATE_CTRL_TYPE.

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

Including TeliCamAPI.h is required.

5.5.8.7. SetCamAcquisitionFrameRateControl

This function writes new AcquisitionFrameRateControl setting to the camera.

[Syntax]

```
CAM_API_STATUS SetCamAcquisitionFrameRateControl (  
    CAM_HANDLE          hCam,  
    CAM_ACQ_FRAME_RATE_CTRL_TYPE eFrameRateControl  
);
```

[Parameters]

Parameter		Description
<i>hCam</i>	[in]	Camera-Handle of target camera.
<i>eFrameRateControl</i>	[in]	New AcquisitionFrameRateControl setting.

[Return value]

Returns result status. Refer to [5.8 Status code](#).

[Remarks]

Refer to TeliCamAPI.h about CAM_ACQ_FRAME_RATE_CTRL_TYPE.

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

Including TeliCamAPI.h is required.

5.5.8.8. GetCamAcquisitionFrameRateMinMax

This function reports the maximum and the minimum value that the camera accepts as AcquisitionFrameRate value.

[Syntax]

```
CAM_API_STATUS GetCamAcquisitionFrameRateMinMax (  
    CAM_HANDLE      hCam,  
    float64_t        *pdAcqFrameRateMin,  
    float64_t        *pdAcqFrameRateMax  
);
```

[Parameters]

Parameter		Description
<i>hCam</i>	[in]	Camera-Handle of target camera.
<i>pdAcqFrameRateMin</i>	[out]	Pointer to a variable that receives the minimum acquisition frame rate value.
<i>pdAcqFrameRateMax</i>	[out]	Pointer to a variable that receives the maximum acquisition frame rate value.

[Return value]

Returns result status. Refer to [5.8 Status code](#).

[Remarks]

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

Including TeliCamAPI.h is required.

5.5.8.9. GetCamAcquisitionFrameRate

This function reports current AcquisitionFrameRate value.

[Syntax]

```
CAM_API_STATUS GetCamAcquisitionFrameRate (  
    CAM_HANDLE      hCam,  
    float64_t       *pdAcqFrameRate  
);
```

[Parameters]

Parameter		Description
<i>hCam</i>	[in]	Camera-Handle of target camera.
<i>pdAcqFrameRate</i>	[out]	Pointer to a variable that receives current AcquisitionFrameRate value.

[Return value]

Returns result status. Refer to [5.8 Status code](#).

[Remarks]

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

Including TeliCamAPI.h is required.

5.5.8.10. SetCamAcquisitionFrameRate

This function writes new frame rate value to the camera.

[Syntax]

```
CAM_API_STATUS SetCamAcquisitionFrameRate (  
    CAM_HANDLE      hCam,  
    float64_t       dAcqFrameRate  
);
```

[Parameters]

Parameter		Description
<i>hCam</i>	[in]	Camera-Handle of target camera.
<i>dAcqFrameRate</i>	[in]	New frame rate value.

[Return value]

Returns result status. Refer to [5.8 Status code](#).

[Remarks]

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

If AcquisitionFrameRateControl register value is CAM_ACQ_FRAME_RATE_CTRL_NO_SPECIFY, this function may fail to write data, or the camera will overwrite AcquisitionFrameRateControl with other value immediately.

There are two types of cameras. Writing data to AcquisitionFrameRate register during image streaming will result in failure in the first group, and it will result in success in the second group.

Refer to instruction manual of the camera about AcquisitionFrameRate.

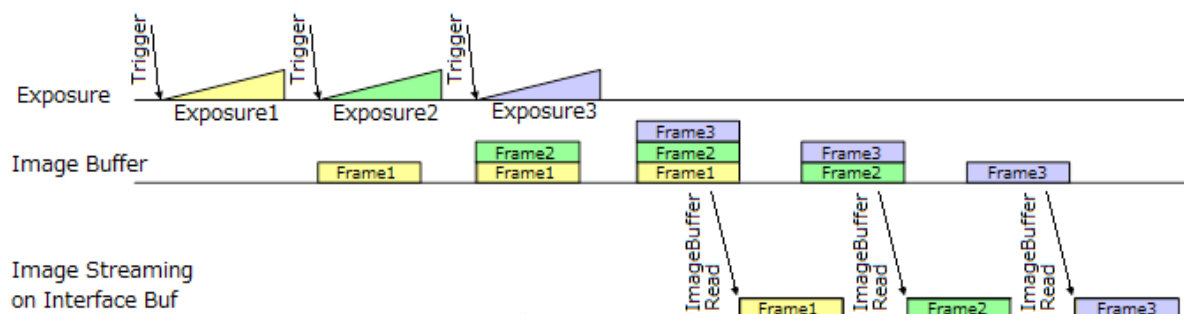
Including TeliCamAPI.h is required.

5.5.9. ImageBuffer

This function 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 more details about image buffer function, refer to instruction manual of the camera.

5.5.9.1. GetCamImageBufferMode

This function reports current image buffer mode of the camera.

[Syntax]

```
CAM_API_STATUS GetCamImageBufferMode (  
    CAM_HANDLE          hCam,  
    CAM_IMAGE_BUFFER_MODE_TYPE *peMode  
);
```

[Parameters]

Parameter	Description
<i>hCam</i> [in]	Camera-Handle of target camera.
<i>peMode</i> [out]	Pointer to a variable that receives current image buffer mode of the camera.

[CAM_IMAGE_BUFFER_MODE_TYPE Enumeration]

Member	Description
CAM_IMAGE_BUFFER_MODE_OFF	Image Buffer Mode = OFF
CAM_IMAGE_BUFFER_MODE_ON	Image Buffer Mode = ON

[Return value]

Returns result status. Refer to [5.8 Status code](#).

[Remarks]

Refer to TeliCamAPI.h about CAM_IMAGE_BUFFER_MODE_TYPE.

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

Refer to instruction manual of the camera about how to utilize image buffer in the camera.

Including TeliCamAPI.h is required.

5.5.9.2. SetCamImageBufferMode

This function writes new camera image buffer mode setting to the camera.

[Syntax]

```
CAM_API_STATUS SetCamImageBufferMode (  
    CAM_HANDLE          hCam,  
    CAM\_IMAGE\_BUFFER\_MODE\_TYPE eMode  
);
```

[Parameters]

Parameter		Description
<i>hCam</i>	[in]	Camera-Handle of target camera.
<i>eMode</i>	[in]	New camera image buffer mode setting.

[Return value]

Returns result status. Refer to [5.8 Status code](#).

[Remarks]

Refer to TeliCamAPI.h about CAM_IMAGE_BUFFER_MODE_TYPE.

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

Refer to instruction manual of the camera about how to utilize image buffer in the camera.

Including TeliCamAPI.h is required.

5.5.9.3. GetCamImageBufferFrameCount

This function reports number of images stored in camera image buffer.

[Syntax]

```
CAM_API_STATUS GetCamImageBufferFrameCount (  
    CAM_HANDLE      hCam,  
    uint32_t        *puiCount  
);
```

[Parameters]

Parameter		Description
<i>hCam</i>	[in]	Camera-Handle of target camera.
<i>puiCount</i>	[out]	Pointer to a variable that receives current number of images stored in camera image buffer.

[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 instruction manual of the camera about how to utilize image buffer in the camera.

Including TeliCamAPI.h is required.

5.5.9.4. ExecuteCamImageBufferRead

This function sends ImageBufferRead command to the camera for starting reading out images in the camera image buffer.

[Syntax]

```
CAM_API_STATUS ExecuteCamImageBufferRead (  
    CAM_HANDLE      hCam  
);
```

[Parameters]

Parameter	Description
<i>hCam</i> [in]	Camera-Handle of target camera.

[Return value]

Returns result status. Refer to [5.8 Status code](#).

[Remarks]

Camera will send specified frame count images for a single ExecuteCamImageBufferRead() call. User application can specify frame count using SetCamAcquisitionFrameCount().

User application can use camera streaming functions in section [5.3.1](#) or [5.3.2](#) to get images in the camera image buffer.

If images were not stored enough 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 image buffer in the camera.

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

Refer to instruction manual of the camera about how to utilize image buffer in the camera.

Including TeliCamAPI.h is required.

5.5.10.TriggerControl

This function 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 more details about trigger control, refer to instruction manual of the camera.

5.5.10.1. GetCamTriggerMode

This function reports current TriggerMode setting.

[Syntax]

```
CAM_API_STATUS GetCamTriggerMode (  
    CAM_HANDLE      hCam,  
    bool18_t        *pbValue  
);
```

[Parameters]

Parameter		Description
<i>hCam</i>	[in]	Camera-Handle of target camera.
<i>pbValue</i>	[out]	Pointer to a variable that receives current TriggerMode setting. If true, Trigger function is active, otherwise Trigger function is inactive.

[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 TriggerMode 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 TriggerMode 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.

Including TeliCamAPI.h is required.

5.5.10.2. SetCamTriggerMode

This function writes new TriggerMode value to the camera.

[Syntax]

```
CAM_API_STATUS SetCamTriggerMode (  
    CAM_HANDLE      hCam,  
    bool18_t        bValue  
);
```

[Parameters]

Parameter		Description
<i>hCam</i>	[in]	Camera-Handle of target camera.
<i>bValue</i>	[in]	New TriggerMode value, If true is specified, Trigger function will be activated, otherwise Trigger function will be inactivated.

[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.

There are two types of cameras. Writing data to TriggerMode register during image streaming will result in failure in the first group, and it will result in success in the second group.

Including TeliCamAPI.h is required.

5.5.10.3. GetCamTriggerSequence

This function reports current trigger sequence setting.

[Syntax]

```
CAM_API_STATUS GetCamTriggerSequence (  
    CAM_HANDLE          hCam,  
    CAM_TRIGGER_SEQUENCE_TYPE *peTriggerSequence  
);
```

[Parameters]

Parameter	Description
<i>hCam</i> [in]	Camera-Handle of target camera.
<i>peTriggerSequence</i> [out]	Pointer to a variable that receives current trigger sequence setting.

[CAM_TRIGGER_SEQUENCE_TYPE Enumeration]

Member	Description
<i>CAM_TRIGGER_SEQUENCE0</i>	Edge mode. The exposure time is determined by Exposure Time setting. USB3 Vision Camera and some GigE Vision Camera : TriggerSequence register = TriggerSequence0 Some GigE Vision Camera : ExposureMode register = Timed TriggerSelector register = FrameStart
<i>CAM_TRIGGER_SEQUENCE1</i>	Level mode. The exposure time is determined by the pulse width of the trigger signal. USB3 Vision Camera and some GigE Vision Camera : TriggerSequence register = TriggerSequence1 Some GigE Vision Camera : ExposureMode register = TriggerWidth TriggerSelector register = FrameStart
<i>CAM_TRIGGER_SEQUENCE6</i>	Bulk (FrameBurst) mode. Camera exposes and transfers multiple frames by a single trigger. USB3 Vision Camera and some GigE Vision Camera : TriggerSequence register = TriggerSequence6 Some GigE Vision Camera : ExposureMode register = Timed TriggerSelector register = FrameBurstStart

[Return value]

Returns result status. Refer to [5.8 Status code](#).

[Remarks]

Refer to TeliCamAPI.h about CAM_TRIGGER_SEQUENCE_TYPE.

The registers are different depending on the camera.

For more details about above registers, refer to instruction manual of the camera.

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

Including TeliCamAPI.h is required.

5.5.10.4. SetCamTriggerSequence

This function writes new trigger sequence setting to the camera.

[Syntax]

```
CAM_API_STATUS SetCamTriggerSequence (  
    CAM_HANDLE          hCam,  
    CAM\_TRIGGER\_SEQUENCE\_TYPE eTriggerSequence  
);
```

[Parameters]

Parameter		Description
<i>hCam</i>	[in]	Camera-Handle of target camera.
<i>eTriggerSequence</i>	[in]	New trigger sequence setting value.

[Return value]

Returns result status. Refer to [5.8 Status code](#).

[Remarks]

Refer to [5.5.10.3 GetCamTriggerSequence](#) and TeliCamAPI.h about CAM_TRIGGER_SEQUENCE_TYPE.

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

Including TeliCamAPI.h is required.

5.5.10.5. GetCamTriggerSource

This function reports current trigger source setting.

[Syntax]

```
CAM_API_STATUS GetCamTriggerSource (  
    CAM_HANDLE          hCam,  
    CAM_TRIGGER_SOURCE_TYPE *peTriggerSource  
);
```

[Parameters]

Parameter		Description
<i>hCam</i>	[in]	Camera-Handle of target camera.
<i>peTriggerSource</i>	[out]	Pointer to a variable that receives current trigger source setting.

[CAM_TRIGGER_SOURCE_TYPE Enumeration]

Member	Description
<i>CAM_TRIGGER_LINE0</i>	Hardware Trigger Line0
<i>CAM_TRIGGER_LINE1</i>	Hardware Trigger Line1
<i>CAM_TRIGGER_LINE2</i>	Hardware Trigger Line2
<i>CAM_TRIGGER_SOFTWARE</i>	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

This function writes new trigger source setting to the camera.

[Syntax]

```
CAM_API_STATUS SetCamTriggerSource (  
    CAM_HANDLE          hCam,  
    CAM\_TRIGGER\_SOURCE\_TYPE eTriggerSource  
);
```

[Parameters]

Parameter		Description
<i>hCam</i>	[in]	Camera-Handle of target camera.
<i>eTriggerSource</i>	[in]	New trigger source setting value.

[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

This function reports the maximum, the minimum value for TriggerAdditionalParameter or AcquisitionBurstFrameCount register.

Those registers are used as “number of frames acquired by a single trigger signal” in Bulk (FrameBurst) mode (TriggerSequence6)

[Syntax]

```
CAM_API_STATUS GetCamTriggerAdditionalParameter (  
    CAM_HANDLE      hCam,  
    uint32_t         *puiAdditionalParameterMin,  
    uint32_t         *puiAdditionalParameterMax  
);
```

[Parameters]

Parameter		Description
<i>hCam</i>	[in]	Camera-Handle of target camera.
<i>puiAdditionalParameterMin</i>	[out]	Pointer to a variable that receives the minimum value for TriggerAdditionalParameter or AcquisitionBurstFrameCount.
<i>puiAdditionalParameterMax</i>	[out]	Pointer to a variable that receives the maximum value for TriggerAdditionalParameter or AcquisitionBurstFrameCount.

[Return value]

Returns result status. Refer to [5.8 Status code](#).

[Remarks]

In BU series camera and some BG series camera, TriggerAdditionalParameter register is used as “number of frames acquired by a single trigger signal” in Bulk mode.

In some BG series camera, AcquisitionBurstFrameCount register is used as “number of frames acquired by a single trigger signal” in FrameBurst mode.

For more details about above registers, refer to instruction manual of the camera.

This function will return error status if those registers or nodes are not implemented in the camera.

Including TeliCamAPI.h is required.

5.5.10.8. GetCamTriggerAdditionalParameter

This function reports current TriggerAdditionalParameter or AcquisitionBurstFrameCount register value.

Those registers are used as “number of frames aquired by a single trigger signal” in Bulk (FrameBurst) mode (TriggerSequence6)

[Syntax]

```
CAM_API_STATUS GetCamTriggerAdditionalParameter (  
    CAM_HANDLE      hCam,  
    uint32_t         *puiAdditionalParameter  
);
```

[Parameters]

Parameter		Description
<i>hCam</i>	[in]	Camera-Handle of target camera.
<i>puiAdditionalParameter</i>	[out]	Pointer to a variable that receives current TriggerAdditionalParameter or AcquisitionBurstFrameCount register value.

[Return value]

Returns result status. Refer to [5.8 Status code](#).

[Remarks]

In BU series camera and some BG series camera, TriggerAdditionalParameter register is used as “number of frames aquired by a single trigger signal” in Bulk mode.

In some BG series camera, AcquisitionBurstFrameCount register is used as “number of frames aquired by a single trigger signal” in FrameBurst mode.

For more details about above registers, refer to instruction manual of the camera.

This function will return error status if those registers or nodes are not implemented in the camera.

Including TeliCamAPI.h is required.

5.5.10.9. SetCamTriggerAdditionalParameter

This function writes new TriggerAdditionalParameter or AcquisitionBurstFrameCount value to the camera.

Those registers are used as “number of frames acquired by a single trigger signal” in Bulk (FrameBurst) mode (TriggerSequence6)

[Syntax]

```
CAM_API_STATUS SetCamTriggerAdditionalParameter (  
    CAM_HANDLE      hCam,  
    uint32_t         uiAdditionalParameter  
);
```

[Parameters]

Parameter		Description
<i>hCam</i>	[in]	Camera-Handle of target camera.
<i>uiAdditionalParameter</i>	[in]	New TriggerAdditionalParameter or AcquisitionBurstFrameCount value.

[Return value]

Returns result status. Refer to [5.8 Status code](#).

[Remarks]

In BU series camera and some BG series camera, TriggerAdditionalParameter register is used as “number of frames acquired by a single trigger signal” in Bulk mode.

In some BG series camera, AcquisitionBurstFrameCount register is used as “number of frames acquired by a single trigger signal” in FrameBurst mode.

For more details about above registers, refer to instruction manual of the camera.

This function will return error status if those registers or nodes are not implemented in the camera.

Including TeliCamAPI.h is required.

5.5.10.10. GetCamTriggerDelayMinMax

This function reports the maximum, the minimum value for TriggerDeley register.
TriggerDelay register controls delay time from detection of trigger signal until starting exposure.

[Syntax]

```
CAM_API_STATUS GetCamTriggerDelayMinMax (  
    CAM_HANDLE      hCam,  
    float64_t        *pdDeaLyUsMin,  
    float64_t        *pdDeaLyUsMax  
);
```

[Parameters]

Parameter		Description
<i>hCam</i>	[in]	Camera-Handle of target camera.
<i>pdDeaLyUsMin</i>	[out]	Pointer to a variable that receives the minimum TriggerDelay value, in microseconds.
<i>pdDeaLyUsMax</i>	[out]	Pointer to a variable that receives the maximum TriggerDelay value, 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

This function reports current TriggerDeley register value.

TriggerDelay register controls delay time from detection of trigger signal until starting exposure.

[Syntax]

```
CAM_API_STATUS GetCamTriggerDelay (  
    CAM_HANDLE      hCam,  
    float64_t        *pdDeaLyUs  
);
```

[Parameters]

Parameter		Description
<i>hCam</i>	[in]	Camera-Handle of target camera.
<i>pdDeaLyUs</i>	[out]	Pointer to a variable that receives current TriggerDelay setting, 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

This function writes new TriggerDeley setting to the camera.

TriggerDelay register controls delay time from detection of trigger signal until starting exposure.

[Syntax]

```
CAM_API_STATUS SetCamTriggerDelay (  
    CAM_HANDLE      hCam,  
    float64_t        dDeaLyUs  
);
```

[Parameters]

Parameter		Description
<i>hCam</i>	[in]	Camera-Handle of target camera.
<i>dDeaLyUs</i>	[in]	New delay time 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

This function sends a command for issuing software trigger signal to the camera.

[Syntax]

```
CAM_API_STATUS ExecuteCamSoftwareTrigger (  
    CAM_HANDLE    hCam  
);
```

[Parameters]

Parameter	Description
<i>hCam</i> [in]	Camera-Handle of target camera.

[Return value]

Returns result status. Refer to [5.8 Status code](#).

[Remarks]

When Software trigger command is not acceptable for the camera due to its settings, the camera may return CAM_API_STS_SUCCESS to "ExecuteCamSoftwareTrigger()" command with doing nothing.

Check settings of the camera before using this function.

Including TeliCamAPI.h is required.

5.5.11.ExposureTime

This function group performs control of exposure.

Refer to instruction manual of the camera about exposure control.

5.5.11.1. GetCamExposureTimeControl

This function reports current ExposureTimeControl setting.

ExposureTimeControl controls mode for controlling exposure time, for example automatic, manual, and so on.

[Syntax]

```
CAM_API_STATUS GetCamExposureTimeControl (  
    CAM_HANDLE          hCam,  
    CAM_EXPOSURE_TIME_CONTROL_TYPE *peExpControl  
);
```

[Parameters]

Parameter	Description
<i>hCam</i> [in]	Camera-Handle of target camera.
<i>peExpControl</i> [out]	Pointer to a variable that receives current ExposureTimeControl setting.

[CAM_EXPOSURE_TIME_CONTROL_TYPE Enumeration]

Member	Description
<i>CAM_EXPOSURE_TIME_CONTROL_AUTO</i>	The camera will control exposure time automatically. USB3 Vision Camera : ExposureTimeControl register = Auto GigE Vision Camera : ExposureAuto register = Continuous
<i>CAM_EXPOSURE_TIME_CONTROL_MANUAL</i>	The camera will determine exposure time using ExposureTime register value. USB3 Vision Camera : ExposureTimeControl register = Manula GigE Vision Camera : ExposureAuto register = Off
<i>CAM_EXPOSURE_TIME_CONTROL_NO_SPECIFY</i>	The camera will determine exposure time using registers other than ExposureTime register. USB3 Vision Camera : ExposureTimeControl register = NoSpecify GigE Vision Camera : Not Available

[Return value]

Returns result status. Refer to [5.8 Status code](#).

[Remarks]

For example, when TriggerMode is On and TriggerSequence is TriggerSequence1, TeliCamAPI will

control exposure time using trigger pulse width. In BU series camera , ExposureTimeControl register value will be CAM_EXPOSURE_TIME_CONTROL_NO_SPECIFY.

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

Including TeliCamAPI.h is required.

5.5.11.2. SetCamExposureTimeControl

This function writes new ExposureTimeControl setting to the camera.

ExposureTimeControl controls mode for controlling exposure time, for example automatic, manual, and so on.

[Syntax]

```
CAM_API_STATUS SetCamExposureTimeControl (  
    CAM_HANDLE          hCam,  
    CAM_EXPOSURE_TIME_CONTROL_TYPE eExpControl  
);
```

[Parameters]

Parameter	Description
<i>hCam</i> [in]	Camera-Handle of target camera.
<i>eExpControl</i> [in]	New ExposureTimeControl setting.

[Return value]

Returns result status. Refer to [5.8 Status code](#).

[Remarks]

For example, when TriggerMode is On and TriggerSequence is TriggerSequence1, the camera will control exposure time using trigger pulse width. In BU series camera, ExposureTimeControl register value will be CAM_EXPOSURE_TIME_CONTROL_NO_SPECIFY.

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

Including TeliCamAPI.h is required

5.5.11.3. GetCamExposureTimeMinMax

This function reports the maximum and the minimum value that the camera accepts as exposure time when ExposureTimeControl register value is CAM_EXPOSURE_TIME_CONTROL_MANUAL.

[Syntax]

```
CAM_API_STATUS GetCamExposureTime (  
    CAM_HANDLE      hCam,  
    float64_t        *pdExpTimeUsMin,  
    float64_t        *pdExpTimeUsMax,  
);
```

[Parameters]

Parameter		Description
<i>hCam</i>	[in]	Camera-Handle of target camera.
<i>pdExpTimeUsMin</i>	[out]	Pointer to a variable that receives the minimum exposure time value, in micro seconds.
<i>pdExpTimeUsMax</i>	[out]	Pointer to a variable that receives the maximum exposure time value, in micro seconds.

[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

This function reports current exposure time value used when ExposureTimeControl register value is CAM_EXPOSURE_TIME_CONTROL_MANUAL.

[Syntax]

```
CAM_API_STATUS GetCamExposureTime (  
    CAM_HANDLE      hCam,  
    float64_t        *pdExpTimeUs  
);
```

[Parameters]

Parameter		Description
<i>hCam</i>	[in]	Camera-Handle of target camera.
<i>pdExpTimeUs</i>	[out]	Pointer to a variable that receives current exposure time value, in micro seconds.

[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

This function writes new exposure time setting to the camera, which will be used when ExposureTimeControl register value is CAM_EXPOSURE_TIME_CONTROL_MANUAL.

[Syntax]

```
CAM_API_STATUS SetCamExposureTime (  
    CAM_HANDLE      hCam,  
    float64_t       dExpTimeUs  
);
```

[Parameters]

Parameter		Description
<i>hCam</i>	[in]	Camera-Handle of target camera.
<i>dExpTimeUs</i>	[in]	New exposure time setting, in micro seconds.

[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.

5.5.12.DigitalIoControl

This function group performs control of digital I/O ports.

Refer to instruction manual of the camera about digital I/O port.

5.5.12.1. GetCamLineModeAll

This function reports information of all I/O lines in the camera whether the line is input line or output line.

[Syntax]

```
CAM_API_STATUS GetCamLineModeAll (  
    CAM_HANDLE      hCam,  
    uint32_t         *puiValue  
);
```

[Parameters]

Parameter	Description
<i>hCam</i> [in]	Camera-Handle of target camera.
<i>puiValue</i> [out]	Pointer to a variable that receives current information of all I/O lines in the camera. 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 <i>puiValue</i> 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, or LineSelector and LineMode, register (or node) is not implemented in the camera.

In USB3 Vision camera case, TeliCamAPI will return LineModeAll register value as is.

In GigE Vision camera case, TeliCamAPI will return the I/O line information value created with LineSelector and LineMode registers.

Refer to instruction manual of the camera about registers mentioned.

Including TeliCamAPI.h is required.

5.5.12.2. SetCamLineModeAll

This function writes new input / output setting of all I/O lines to the camera.

[Syntax]

```
CAM_API_STATUS SetCamLineModeAll (  
    CAM_HANDLE      hCam,  
    uint32_t         uiValue  
);
```

[Parameters]

Parameter		Description
<i>hCam</i>	[in]	Camera-Handle of target camera.
<i>uiValue</i>	[in]	New input / output setting of all I/O lines of the camera. Each bit corresponds to ech 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 <i>uiValue</i> 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, or LineSelector and LineMode, register (or node) is not implemented in the camera.

In USB3 Vision camera case, TeliCamAPI will write LineModeAll register value as is.

In GigE Vision camera case, TeliCamAPI will write LineSelector and LineMode registers.

Refer to instruction manual of the camera about registers mentioned.

Including TeliCamAPI.h is required.

5.5.12.3. GetCamLineInverterAll

This function reports polarity of all I/O lines in 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]	<p>Pointer to a variable that receives current polarity of all I/O lines in the camera.</p> <p>Each bit corresponds to each I/O line. (Line0 : Bit0 , Line1 : Bit1 , Line2 : Bit2 , .)</p> <p>If bit value is 0, the line is not inverted, If bit value is 1, the line is inverted.</p> <p>For example, If puiValue is 0x06, Line0 : not inverted, Line1 : inverted, Line2 : inverted , .)</p>

[Return value]

Returns result status. Refer to [5.8 Status code](#).

[Remarks]

This function will return error status if LineInverterAll, or LineSelector and LineInverter, register (or node) is not implemented in the camera.

In USB3 Vision camera case, TeliCamAPI will return LineInverterAll register value as is.

In GigE Vision camera case, TeliCamAPI will return the I/O line inverter information value created with LineSelector and LineInverter registers.

Refer to instruction manual of the camera about registers mentioned.

Including TeliCamAPI.h is required.

5.5.12.4. SetCamLineInverterAll

This function writes new polarity setting of all I/O lines to the camera.

[Syntax]

```
CAM_API_STATUS SetCamLineInverterAll (  
    CAM_HANDLE      hCam,  
    uint32_t        uiValue  
);
```

[Parameters]

Parameter		Description
<i>hCam</i>	[in]	Camera-Handle of target camera.
<i>uiValue</i>	[in]	New polarity setting of all I/O lines of the camera. 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 <i>uiValue</i> 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, or LineSelector and LineInverter, register (or node) is not implemented in the camera.

In USB3 Vision camera case, TeliCamAPI will write [uiValue](#) to LineInverterAll register, as is.

In GigE Vision camera case, TeliCamAPI will write settings specified by [uiValue](#) to LineInverter registers using LineSelector register (node).

Refer to instruction manual of the camera about registers mentioned.

Including TeliCamAPI.h is required.

5.5.12.5. GetCamLineStatusAll

This function reports status of all I/O lines in the camera.

[Syntax]

```
CAM_API_STATUS GetCamLineStatusAll (  
    CAM_HANDLE      hCam,  
    uint32_t         *puiValue  
);
```

[Parameters]

Parameter	Description
<i>hCam</i> [in]	Camera-Handle of target camera.
<i>puiValue</i> [out]	Pointer to a variable that receives current status of all I/O lines in the camera. Each bit corresponds to ech I/O line. (Line0 : Bit0 , Line1 : Bit1 , Line2 : Bit2 , .)

[Return value]

Returns result status. Refer to [5.8 Status code](#).

[Remarks]

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

Including TeliCamAPI.h is required.

5.5.12.6. GetCamUserOutputValueAll

This function reports output value of all user output lines in the camera.

[Syntax]

```
CAM_API_STATUS GetCamUserOutputValueAll (  
    CAM_HANDLE      hCam,  
    uint32_t         *puiValue  
);
```

[Parameters]

Parameter		Description
<i>hCam</i>	[in]	Camera-Handle of target camera.
<i>puiValue</i>	[out]	Pointer to a variable that receives current value of all output lines in the camera. Each bit corresponds to ech I/O line. (Line0 : Bit0 , Line1 : Bit1 , Line2 : Bit2 , . .)

[Return value]

Returns result status. Refer to [5.8 Status code](#).

[Remarks]

“User output line” is an output line of the camera whose data source is specified as UserOutput (CAM_LINE_SOURCE_USER_OUTPUT).

User application can get or set data source of output lines using “GetCamLineSource()” or “SetCamLineSource()”.

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

Including TeliCamAPI.h is required.

5.5.12.7. SetCamUserOutputValueAll

This function writes new values of all user output lines to the camera.

[Syntax]

```
CAM_API_STATUS SetCamUserOutputValueAll (  
    CAM_HANDLE      hCam,  
    uint32_t         uiValue  
);
```

[Parameters]

Parameter		Description
<i>hCam</i>	[in]	Camera-Handle of target camera.
<i>uiValue</i>	[in]	New values of all user output lines to the camera. Each bit corresponds to each I/O line. (Line0 : Bit0 , Line1 : Bit1 , Line2 : Bit2 , .)

[Return value]

Returns result status. Refer to [5.8 Status code](#).

[Remarks]

“User output line” is an output line of the camera whose data source is specified as UserOutput (CAM_LINE_SOURCE_USER_OUTPUT).

User application can get or set data source of output lines using “GetCamLineSource()” or “SetCamLineSource()”.

The value of output lines, whose data source is not UserOutput, will not be changed by this function.

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

Including TeliCamAPI.h is required.

5.5.12.8. GetCamLineSource

This function reports data source of the specified output line.

[Syntax]

```
CAM_API_STATUS GetCamLineSource (  
    CAM_HANDLE          hCam,  
    CAM_LINE_SELECTOR_TYPE eLineSelector,  
    CAM_LINE_SOURCE_TYPE *peLineSource  
);
```

[Parameters]

Parameter		Description
<i>hCam</i>	[in]	Camera-Handle of target camera.
<i>eLineSelector</i>	[in]	Target output line.
<i>peLineSource</i>	[out]	Pointer to a variable that receives current data source of the specified output line .

[CAM_LINE_SELECTOR_TYPE Enumeration]

Member	Description
<i>CAM_LINE_SELECTOR_LINE0</i>	Line0
<i>CAM_LINE_SELECTOR_LINE1</i>	Line1
<i>CAM_LINE_SELECTOR_LINE2</i>	Line2

[CAM_LINE_SOURCE_TYPE Enumeration]

Member	Description
<i>CAM_LINE_SOURCE_OFF</i>	No output.
<i>CAM_LINE_SOURCE_VD</i>	Internal VD sync signal.(Only for GigE Vision Camer)
<i>CAM_LINE_SOURCE_USER_OUTPUT</i>	Outputs the value set in "UserOutputValue".
<i>CAM_LINE_SOURCE_TIMER0_ACTIIVE</i>	This signal can be used as strobe control signal. The delay time and pulse width of this signal are configurable.
<i>CAM_LINE_SOURCE_ACQUISITION_ACTIVE</i>	This signal indicates that it is in the AcquisitionStart state.
<i>CAM_LINE_SOURCE_FRAME_TRIGGER_WAIT</i>	Indicating waiting a Random Trigger Shutter. An External trigger is input during this period, exposure starts immediately.
<i>CAM_LINE_SOURCE_FRAME_ACTIVE</i>	Period from exposure start to CMOS transfer completion.
<i>CAM_LINE_SOURCE_FRAME_TRANSFER_ACTIVE</i>	Period of transferring image data on bus.
<i>CAM_LINE_SOURCE_EXPOSURE_ACTIVE</i>	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 or LineSource register (or node) is not implemented in the camera.

Including TeliCamAPI.h is required.

5.5.12.9. SetCamLineSource

This function write new data source of specified output line to the camera.

[Syntax]

```
CAM_API_STATUS SetCamLineSource (  
    CAM_HANDLE          hCam,  
    CAM_LINE_SELECTOR_TYPE eLineSelector,  
    CAM_LINE_SOURCE_TYPE  eLineSource  
);
```

[Parameters]

Parameter		Description
<i>hCam</i>	[in]	Camera-Handle of target camera.
<i>eLineSelector</i>	[in]	Target output line.
<i>eLineSource</i>	[in]	New data source.

[Return value]

Returns result status. Refer to [5.8 Status code](#).

[Remarks]

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

Refer to instruction manual of the camera about data source of the output line, for checking the choice available in the target camera.

Including TeliCamAPI.h is required.

5.5.13.TimerControl

This function 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.

Refer to instruction manual of the camera about timer.

5.5.13.1. GetCamTimerDurationMinMax

This function reports the maximum and the minimum value of timer duration setting (width of Timer0Active signal) for Timer0Active signal.

[Syntax]

```
CAM_API_STATUS GetCamTimerDurationMinMax (  
    CAM_HANDLE      hCam,  
    float64_t       *pdTimerDurationMin,  
    float64_t       *pdTimerDurationMax  
);
```

[Parameters]

Parameter		Description
<i>hCam</i>	[in]	Camera-Handle of target camera.
<i>pdTimerDurationMin</i>	[out]	Pointer to a variable that receives the minimum timer duration value, in microseconds.
<i>pdTimerDurationMax</i>	[out]	Pointer to a variable that receives the maximum timer duration value, in microseconds.

[Return value]

Returns result status. Refer to [5.8 Status code](#).

[Remarks]

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

Including TeliCamAPI.h is required.

5.5.13.2. GetCamTimerDuration

This function reports current timer duration value (width of Timer0Active signal) for Timer0Active signal.

[Syntax]

```
CAM_API_STATUS GetCamTimerDuration (  
    CAM_HANDLE      hCam,  
    float64_t       *pdTimerDuration  
);
```

[Parameters]

Parameter		Description
<i>hCam</i>	[in]	Camera-Handle of target camera.
<i>pdTimerDuration</i>	[out]	Pointer to a variable that receives current timer duration value in microseconds.

[Return value]

Returns result status. Refer to [5.8 Status code](#).

[Remarks]

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

Including TeliCamAPI.h is required.

5.5.13.3. SetCamTimerDuration

This function writes new timer duration setting (width of Timer0Active signal) for Timer0Active signal to the camera.

[Syntax]

```
CAM_API_STATUS SetCamTimerDuration (  
    CAM_HANDLE      hCam,  
    float64_t        dTimerDuration  
);
```

[Parameters]

Parameter		Description
<i>hCam</i>	[in]	Camera-Handle of target camera.
<i>dTimerDuration</i>	[in]	New timer duration setting (width of Timer0Active signal) in microseconds.

[Return value]

Returns result status. Refer to [5.8 Status code](#).

[Remarks]

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

Including TeliCamAPI.h is required.

5.5.13.4. GetCamTimerDelayMinMax

This function reports the maximum and the minimum value of timer delay setting for Timer0Active signal.

[Syntax]

```
CAM_API_STATUS GetCamTimerDelayMinMax (  
    CAM_HANDLE      hCam,  
    float64_t       *pdTimerDelayMin,  
    float64_t       *pdTimerDelayMax  
);
```

[Parameters]

Parameter		Description
<i>hCam</i>	[in]	Camera-Handle of target camera.
<i>pdTimerDelayMin</i>	[out]	Pointer to a variable that receives the minimum timer delay value, in microseconds.
<i>pdTimerDelayMax</i>	[out]	Pointer to a variable that receives the maximum timer delay value, 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.

5.5.13.5. GetCamTimerDelay

This function reports current timer delay value for Timer0Active signal.

[Syntax]

```
CAM_API_STATUS GetCamTimerDelay (  
    CAM_HANDLE      hCam,  
    float64_t       *pdTimerDelay  
);
```

[Parameters]

Parameter		Description
<i>hCam</i>	[in]	Camera-Handle of target camera.
<i>pdTimerDelay</i>	[out]	Pointer to a variable that receives current timer delay value, 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.

5.5.13.6. SetCamTimerDelay

This function writes new timer delay setting for Timer0Active signal to the camera.

[Syntax]

```
CAM_API_STATUS SetCamTimerDelay (  
    CAM_HANDLE      hCam,  
    float64_t        dTimerDelay  
);
```

[Parameters]

Parameter		Description
<i>hCam</i>	[in]	Camera-Handle of target camera.
<i>dTimerDelay</i>	[in]	New timer delay setting (width 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.

5.5.13.7. GetCamTimerTriggerSource

This function reports current trigger source signal setting for Timer0Active signal.

[Syntax]

```
CAM_API_STATUS GetCamTimerTriggerSource (  
    CAM_HANDLE          hCam,  
    CAM_TIMER_TRIGGER_SOURCE_TYPE *peTimerTriggerSource  
);
```

[Parameters]

Parameter		Description
<i>hCam</i>	[in]	Camera-Handle of target camera.
<i>peTimerTriggerSource</i>	[out]	Pointer to a variable that receives current trigger source signal setting.

[CAM_TIMER_TRIGGER_SOURCE_TYPE Enumeration]

Member	Description
<i>CAM_TIMER_TRIGGER_SOURCE_OFF</i>	DisablesTimer0Active signal.
<i>CAM_TIMER_TRIGGER_SOURCE_LINE0_ACTIVE</i>	Starts when Line0 is active.
<i>CAM_TIMER_TRIGGER_SOURCE_FRAME_TRIGGER</i>	Starts with the reception of the Frame Start Trigger.
<i>CAM_TIMER_TRIGGER_SOURCE_EXPOSURE_START</i>	Starts with the reception of the Exposure Start.
<i>CAM_TIMER_TRIGGER_SOURCE_EXPOSURE_END</i>	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.

5.5.13.8. SetCamTimerTriggerSource

This function writes new trigger source signal setting for Timer0Active signal to the camera.

[Syntax]

```
CAM_API_STATUS SetCamTimerTriggerSource (  
    CAM_HANDLE          hCam,  
    CAM\_TIMER\_TRIGGER\_SOURCE\_TYPE eTimerTriggerSource  
);
```

[Parameters]

Parameter		Description
<i>hCam</i>	[in]	Camera-Handle of target camera.
<i>eTimerTriggerSource</i>	[in]	New trigger source signal setting.

[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.

Refer to instruction manual of the camera about trigger source of Timer0Active signal, for checking the choice available in the target camera.

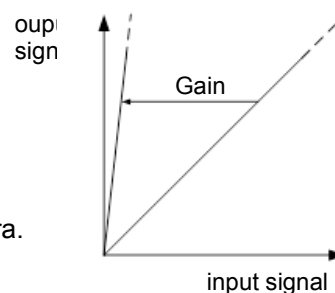
Including TeliCamAPI.h is required.

5.5.14.Gain

This function group performs control of Gain.

This section describes Gain feature. This control adjusts an amplification factor applied to the output signal.

Gain feature adjusts manual gain. GainAuto feature adjusts gain automatically.



For more details about Gain, refer to instruction manual of the camera.

5.5.14.1. GetCamGainMinMax

This function reports the maximum and the minimum value that the camera accepts as Gain value.

[Syntax]

```
CAM_API_STATUS GetCamGainMinMax (  
    CAM_HANDLE      hCam,  
    float64_t       *pdGainMin,  
    float64_t       *pdGainMax  
);
```

[Parameters]

Parameter		Description
<i>hCam</i>	[in]	Camera-Handle of target camera.
<i>pdGainMin</i>	[out]	Pointer to a variable that receives the minimum Gain value in dB.
<i>pdGainMax</i>	[out]	Pointer to a variable that receives the maximum Gain value in dB.

[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.

5.5.14.2. GetCamGain

This function reports current Gain setting.

[Syntax]

```
CAM_API_STATUS GetCamGain (  
    CAM_HANDLE      hCam,  
    float64_t        *pdGain  
);
```

[Parameters]

Parameter		Description
<i>hCam</i>	[in]	Camera-Handle of target camera.
<i>pdGain</i>	[out]	Pointer to a variable that receives current Gain value in dB.

[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.

5.5.14.3. SetCamGain

This function writes new Gain setting to the camera.

[Syntax]

```
CAM_API_STATUS SetCamGain (  
    CAM_HANDLE      hCam,  
    float64_t       dGain  
);
```

[Parameters]

Parameter		Description
<i>hCam</i>	[in]	Camera-Handle of target camera.
<i>dGain</i>	[in]	New Gain setting in dB.

[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.

5.5.14.4. GetCamGainAuto

This function reports current AGC setting of the camera.

[Syntax]

```
CAM_API_STATUS GetCamGainAuto (  
    CAM_HANDLE      hCam,  
    CAM_GAIN_AUTO_TYPE *peGainAuto  
);
```

[Parameters]

Parameter		Description
<i>hCam</i>	[in]	Camera-Handle of target camera.
<i>peGainAuto</i>	[out]	Pointer to a variable that receives current AGC setting.

[CAM_GAIN_AUTO_TYPE Enumeration]

Member	Description
<i>CAM_GAIN_AUTO_OFF</i>	Manual Gain Control (MANUAL)
<i>CAM_GAIN_AUTO_AUTO</i>	Auto Gain Control (AGC)

[Return value]

Returns result status. Refer to [5.8 Status code](#).

[Remarks]

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

Including TeliCamAPI.h is required.

5.5.14.5. SetCamGainAuto

This function writes new AGC setting to the camera.

[Syntax]

```
CAM_API_STATUS SetCamGainAuto (  
    CAM_HANDLE      hCam,  
    CAM\_GAIN\_AUTO\_TYPE eGainAuto  
);
```

[Parameters]

Parameter		Description
<i>hCam</i>	[in]	Camera-Handle of target camera.
<i>eGainAuto</i>	[in]	New AGC setting.

[Return value]

Returns result status. Refer to [5.8 Status code](#).

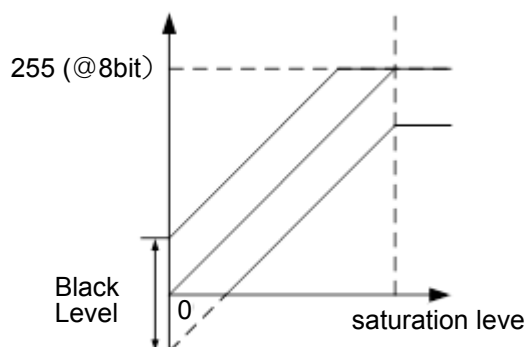
[Remarks]

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

Including TeliCamAPI.h is required.

5.5.15.BlackLevel

This function group performs control of Black Level control.



For more details about Black Level control, refer to instruction manual of the camera.

5.5.15.1. GetCamBlackLevelMinMax

This function reports the maximum and the minimum value of Black Level setting that the camera accepts.

[Syntax]

```
CAM_API_STATUS GetCamBlackLevelMinMax (  
    CAM_HANDLE      hCam,  
    float64_t       *pdBlackLevelMin,  
    float64_t       *pdBlackLevelMax  
);
```

[Parameters]

Parameter		Description
<i>hCam</i>	[in]	Camera-Handle of target camera. d.
<i>pdBlackLevelMin</i>	[out]	Pointer to a variable that receives the minimum Black Level setting value, in %.
<i>pdBlackLevelMax</i>	[out]	Pointer to a variable that receives the maximum Black Level setting value, 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.

5.5.15.2. GetCamBlackLevel

This function reports current Black Level setting.

[Syntax]

```
CAM_API_STATUS GetCamBlackLevel (  
    CAM_HANDLE      hCam,  
    float64_t       *pdBlackLevel  
);
```

[Parameters]

Parameter		Description
<i>hCam</i>	[in]	Camera-Handle of target camera.
<i>pdBlackLevel</i>	[out]	Pointer to a variable that receives current Black Level setting.

[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.

5.5.15.3. SetCamBlackLevel

This function writes new Black Level setting to the camera.

[Syntax]

```
CAM_API_STATUS SetCamBlackLevel (  
    CAM_HANDLE      hCam,  
    float64_t       dBlackLevel  
);
```

[Parameters]

Parameter		Description
<i>hCam</i>	[in]	Camera-Handle of target camera.
<i>dBlackLevel</i>	[in]	New Black Level setting in %.

[Return value]

Returns result status. Refer to [5.8 Status code](#).

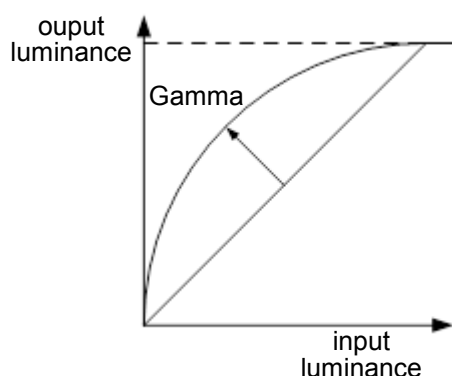
[Remarks]

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. Gamma

This function group performs control of Gamma correction.



For more details about Gamma, refer to instruction manual of the camera.

5.5.16.1. GetCamGammaMinMax

This function reports the maximum and the minimum value of Gamma correction setting that the camera accepts.

[Syntax]

```
CAM_API_STATUS GetCamGammaMinMax (  
    CAM_HANDLE      hCam,  
    float64_t        *pdGammaMin,  
    float64_t        *pdGammaMax  
);
```

[Parameters]

Parameter		Description
<i>hCam</i>	[in]	Camera-Handle of target camera.
<i>pdGammaMin</i>	[out]	Pointer to a variable that receives the minimum Gamma parameter value.
<i>pdGammaMax</i>	[out]	Pointer to a variable that receives the maximum Gamma parameter value.

[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.

5.5.16.2. GetCamGamma

This function reports current Gamma correction parameter of the camera.

[Syntax]

```
CAM_API_STATUS GetCamGamma (  
    CAM_HANDLE      hCam,  
    float64_t       *pdGamma  
);
```

[Parameters]

Parameter		Description
<i>hCam</i>	[in]	Camera-Handle of target camera.
<i>pdGamma</i>	[out]	Pointer to a variable that receives current Gamma correction parameter.

[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.

5.5.16.3. SetCamGamma

This function writes new Gamma correction setting to the camera.

[Syntax]

```
CAM_API_STATUS SetCamGamma (  
    CAM_HANDLE      hCam,  
    float64_t       dGamma  
);
```

[Parameters]

Parameter		Description
<i>hCam</i>	[in]	Camera-Handle of target camera.
<i>dGamma</i>	[in]	New Gamma correction setting.

[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.

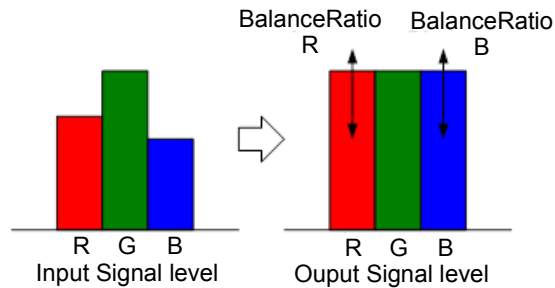
Including TeliCamAPI.h is required.

5.5.17.WhiteBalance

This function group performs control of White balance control.

These functions are 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 more details about White balance, refer to instruction manual of the camera.

5.5.17.1. GetCamBalanceRatioMinMax

This function reports the maximum and the minimum relative gain value of specified color component that the camera accepts.

[Syntax]

```
CAM_API_STATUS GetCamBalanceRatioMinMax (  
    CAM_HANDLE          hCam,  
    CAM_BALANCE_RATIO_SELECTOR_TYPE eSelector,  
    float64_t           *pdBalanceRatioMin,  
    float64_t           *pdBalanceRatioMax  
);
```

[CAM_BALANCE_RATIO_SELECTOR_TYPE Enumeration]

Member	Description
CAM_BALANCE_RATIO_SELECTOR_RED	BalanceRatio = Red Gain
CAM_BALANCE_RATIO_SELECTOR_BLUE	BalanceRatio = Blue Gain

[Parameters]

Parameter	Description
<i>hCam</i> [in]	Camera-Handle of target camera.
<i>eSelector</i> [in]	Target color component (R or B).
<i>pdBalanceRatioMin</i> [out]	Pointer to a variable that receives the minimum relative gain value.
<i>pdBalanceRatioMax</i> [out]	Pointer to a variable that receives the maximum relative gain value.

[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 or BalanceRatio register (or node) is not implemented in the camera.

Including TeliCamAPI.h is required.

5.5.17.2. GetCamBalanceRatio

This function reports current relative gain value of the specified color component.

[Syntax]

```
CAM_API_STATUS GetCamBalanceRatio (  
    CAM_HANDLE          hCam,  
    CAM_BALANCE_RATIO_SELECTOR_TYPE eSelector,  
    float64_t           *pdBalanceRatio  
);
```

[Parameters]

Parameter		Description
<i>hCam</i>	[in]	Camera-Handle of target camera.
<i>eSelector</i>	[in]	Target color component (R or B).
<i>pdBalanceRatio</i>	[out]	Pointer to a variable that receives current relative gain value of the specified color component.

[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 or BalanceRatio register (or node) is not implemented in the camera.

Including TeliCamAPI.h is required.

5.5.17.3. SetCamBalanceRatio

This function writes new relative gain value of the specified color component to the camera.

[Syntax]

```
CAM_API_STATUS SetCamBalanceRatio (  
    CAM_HANDLE          hCam,  
    CAM_BALANCE_RATIO_SELECTOR_TYPE eSelector,  
    float64_t           dBalanceRatio  
);
```

[Parameters]

Parameter		Description
<i>hCam</i>	[in]	Camera-Handle of target camera.
<i>eSelector</i>	[in]	Target color component (R or B).
<i>dBalanceRatio</i>	[in]	New relative gain value of the specified color component.

[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 or BalanceRatio register (or node) is not implemented in the camera.

Including TeliCamAPI.h is required.

5.5.17.4. GetCamBalanceWhiteAuto

This function reports current automatic white balance setting of the camera.

[Syntax]

```
CAM_API_STATUS GetCamBalanceWhiteAuto (  
    CAM_HANDLE          hCam,  
    CAM_BALANCE_WHITE_AUTO_TYPE *peBalanceWhiteAuto  
);
```

[Parameters]

Parameter	Description
<i>hCam</i> [in]	Camera-Handle of target camera.
<i>peBalanceWhiteAuto</i> [out]	Pointer to a variable that receives current automatic white balance setting.

[CAM_BALANCE_WHITE_AUTO_TYPE Enumeration]

Member	Description
<i>CAM_BALANCE_WHITE_AUTO_OFF</i>	No operation.
<i>CAM_BALANCE_WHITE_AUTO_CONTINUOUS</i>	Execute auto white balance continuously.
<i>CAM_BALANCE_WHITE_AUTO_ONCE</i>	Execute auto white balance once.

[Return value]

Returns result status. Refer to [5.8 Status code](#).

[Remarks]

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

Including TeliCamAPI.h is required.

5.5.17.5. SetCamBalanceWhiteAuto

This function writes new automatic white balance setting to the camera.

[Syntax]

```
CAM_API_STATUS SetCamBalanceWhiteAuto (  
    CAM_HANDLE          hCam,  
    CAM_BALANCE_WHITE_AUTO_TYPE eBalanceWhiteAuto  
);
```

[Parameters]

Parameter		Description
<i>hCam</i>	[in]	Camera-Handle of target camera.
<i>BalanceWhiteAuto</i>	[in]	New automatic white balance setting.

[Return value]

Returns result status. Refer to [5.8 Status code](#).

[Remarks]

Refer to TeliCamAPI.h about CAM_BALANCE_WHITE_AUTO_TYPE.

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

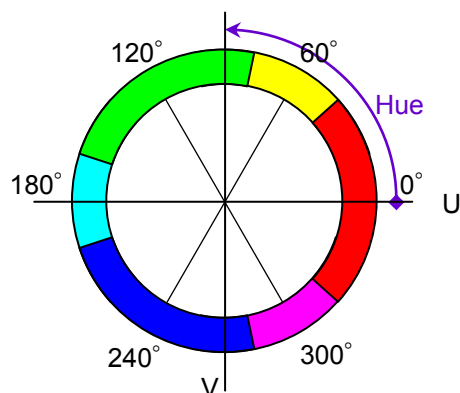
Including TeliCamAPI.h is required.

5.5.18.Hue

This function group performs control of Hue

These functions are available only in color model cameras.

Refer to instruction manual of the camera about Hue.



5.5.18.1. GetCamHueMinMax

This function reports the maximum and the minimum value that the camera accepts as Hue value.

[Syntax]

```
CAM_API_STATUS GetCamHueMinMax (  
    CAM_HANDLE      hCam,  
    float64_t       *pdHueMin,  
    float64_t       *pdHueMax  
);
```

[Parameters]

Parameter		Description
<i>hCam</i>	[in]	Camera-Handle of target camera.
<i>pdHueMin</i>	[out]	Pointer to a variable that receives the minimum Hue value (chroma phase) in degrees.
<i>pdHueMax</i>	[out]	Pointer to a variable that receives the maximum Hue value (chroma phase) in degrees.

[Return value]

Returns result status. Refer to [5.8 Status code](#).

[Remarks]

Initial factory setting of Hue is 0.

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

Including TeliCamAPI.h is required.

5.5.18.2. GetCamHue

This function reports current Hue setting of the camera.

[Syntax]

```
CAM_API_STATUS GetCamHue (  
    CAM_HANDLE      hCam,  
    float64_t       *pdHue,  
);
```

[Parameters]

Parameter		Description
<i>hCam</i>	[in]	Camera-Handle of target camera.
<i>pdHue</i>	[out]	Pointer to a variable that receives current Hue setting (chroma phase) in degrees.

[Return value]

Returns result status. Refer to [5.8 Status code](#).

[Remarks]

Initial factory setting of Hue is 0.

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

Including TeliCamAPI.h is required.

5.5.18.3. SetCamHue

This function writes new Hue setting to the camera.

[Syntax]

```
CAM_API_STATUS GetCamHue (  
    CAM_HANDLE      hCam,  
    float64_t       dHue,  
);
```

[Parameters]

Parameter		Description
<i>hCam</i>	[in]	Camera-Handle of target camera.
<i>dHue</i>	[in]	New Hue setting (chroma phase) in degrees.

[Return value]

Returns result status. Refer to [5.8 Status code](#).

[Remarks]

Initial factory setting of Hue is 0.

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. Saturation

This function group performs control of Saturation.
These functions are available only in color model cameras.
Refer to instruction manual of the camera about Saturation

5.5.19.1. GetCamSaturationMinMax

This function reports the maximum and the minimum value of Saturation that the camera accepts.

[Syntax]

```
CAM_API_STATUS GetCamSaturationMinMax (  
    CAM_HANDLE      hCam,  
    float64_t        *pdSaturationMin,  
    float64_t        *pdSaturationMax  
);
```

[Parameters]

Parameter		Description
<i>hCam</i>	[in]	Camera-Handle of target camera.
<i>pdSaturationMin</i>	[out]	Pointer to a variable that receives the minimum Saturation value.
<i>pdSaturationMax</i>	[out]	Pointer to a variable that receives the maximum Saturation value.

[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.

Including TeliCamAPI.h is required.

5.5.19.2. GetCamSaturation

This function reports the current Saturation setting of the camera.

[Syntax]

```
CAM_API_STATUS GetCamSaturation (  
    CAM_HANDLE      hCam,  
    float64_t       *pdSaturation  
);
```

[Parameters]

Parameter		Description
<i>hCam</i>	[in]	Camera-Handle of target camera.
<i>pdSaturation</i>	[out]	Pointer to a variable that receives current Saturation value.

[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.

Including TeliCamAPI.h is required.

5.5.19.3. SetCamSaturation

This function writes new Saturation setting to the camera.

[Syntax]

```
CAM_API_STATUS SetCamSaturation (  
    CAM_HANDLE      hCam,  
    float64_t       dSaturation  
);
```

[Parameters]

Parameter		Description
<i>hCam</i>	[in]	Camera-Handle of target camera.
<i>dSaturation</i>	[in]	New Saturation setting.

[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.

Including TeliCamAPI.h is required.

5.5.20. Sharpness

This function group performs control of Sharpness.

Refer to instruction manual of the camera about Sharpness

5.5.20.1. GetCamSharpnessMinMax

This function reports the maximum and the minimum value of Sharpness that the camera accepts.

[Syntax]

```
CAM_API_STATUS GetCamSharpnessMinMax (  
    CAM_HANDLE      hCam,  
    uint32_t         *puiSharpnessMin,  
    uint32_t         *puiSharpnessMax  
);
```

[Parameters]

Parameter		Description
<i>hCam</i>	[in]	Camera-Handle of target camera.
<i>puiSharpnessMin</i>	[out]	Pointer to a variable that receives the minimum Sharpness parameter.
<i>puiSharpnessMax</i>	[out]	Pointer to a variable that receives the maximum Sharpness parameter.

[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.

5.5.20.2. GetCamSharpness

This function reports the current Sharpness setting of the camera.

[Syntax]

```
CAM_API_STATUS GetCamSharpness (  
    CAM_HANDLE      hCam,  
    uint32_t        *puiSharpness  
);
```

[Parameters]

Parameter		Description
<i>hCam</i>	[in]	Camera-Handle of target camera.
<i>puiSharpness</i>	[out]	Pointer to a variable that receives current Sharpness parameter.

[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.

5.5.20.3. SetCamSharpness

This function writes new Sharpness setting to the camera.

[Syntax]

```
CAM_API_STATUS SetCamSharpness (  
    CAM_HANDLE      hCam,  
    uint32_t        uiSharpness  
);
```

[Parameters]

Parameter		Description
<i>hCam</i>	[in]	Camera-Handle of target camera.
<i>uiSharpness</i>	[in]	New Sharpness setting.

[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. ColorCorrectionMatrix

This function group performs control of Color Correction Matrix for correcting pixel color value.

These functions are available only in color model cameras.

Refer to instruction manual of the camera about ColorCorrectionMatrix.

The relationship between original pixel data (R, G, and B) and corrected pixel data (R', G', and B') are represented in the following formula.

$$\begin{pmatrix} R' \\ G' \\ B' \end{pmatrix} = \begin{pmatrix} 1 & -mask_rg & -mask_rb \\ -mask_gr & 1 & -mask_gb \\ -mask_br & -mask_bg & 1 \end{pmatrix} \begin{pmatrix} R & (G-R) & (B-R) \\ (R-G) & G & (B-G) \\ (R-B) & (G-B) & B \end{pmatrix}$$
$$R' = (1 - mask_rg - mask_rb) \times R + mask_rg \times G + mask_rb \times B$$
$$G' = mask_gr \times R + (1 - mask_gr - mask_gb) \times G + mask_gb \times B$$
$$B' = mask_br \times R + mask_bg \times G + (1 - mask_br - mask_bg) \times B$$

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

5.5.21.1. GetCamColorCorrectionMatrixMinMax

This function reports the maximum and the minimum coefficient of specified Color Correction Matrix element.

[Syntax]

```
CAM_API_STATUS GetCamColorCorrectionMatrixMinMax (  
    CAM_HANDLE          hCam,  
    CAM_COLOR_CORRECTION_MATRIX_TYPE eType,  
    float64_t           *pdMatrixMin,  
    float64_t           *pdMatrixMax  
);
```

[Parameters]

Parameter		Description
<i>hCam</i>	[in]	Camera-Handle of target camera.
<i>eType</i>	[in]	Target Color Collection Matrix element.
<i>pdMatrixMin</i>	[out]	Pointer to a variable that receives the minimum coefficient value.
<i>pdMatrixMax</i>	[out]	Pointer to a variable that receives the maximum coefficient value.

[CAM_COLOR_CORRECTION_MATRIX_TYPE Enumeration]

Member	Description
<i>CAM_COLOR_CORRECTION_MATRIX_RG</i>	SelectorI = R , SelectorJ = G
<i>CAM_COLOR_CORRECTION_MATRIX_RB</i>	SelectorI = R , SelectorJ = B
<i>CAM_COLOR_CORRECTION_MATRIX_GR</i>	SelectorI = G , SelectorJ = R
<i>CAM_COLOR_CORRECTION_MATRIX_GB</i>	SelectorI = G , SelectorJ = B
<i>CAM_COLOR_CORRECTION_MATRIX_BR</i>	SelectorI = B , SelectorJ = R
<i>CAM_COLOR_CORRECTION_MATRIX_BG</i>	SelectorI = B , SelectorJ = G

[Return value]

Returns result status. Refer to [5.8 Status code](#).

[Remarks]

This function will return error status if ColorCorrectionMatrix, ColorCorrectionMatrixSelectorI, and ColorCorrectionMatrixSelectorJ registers or nodes are not implemented in the camera.

Including TeliCamAPI.h is required.

5.5.21.2. GetCamColorCorrectionMatrix

This function reports a current coefficient of specified Color Correction Matrix element.

[Syntax]

```
CAM_API_STATUS GetCamColorCorrectionMatrix (  
    CAM_HANDLE          hCam,  
    CAM_COLOR_CORRECTION_MATRIX_TYPE eType,  
    float64_t           *pdMatrix  
);
```

[Parameters]

Parameter		Description
<i>hCam</i>	[in]	Camera-Handle of target camera.
<i>eType</i>	[in]	Target Color Correction Matrix element.
<i>pdMatrix</i>	[out]	Pointer to a variable that receives current coefficient value of the Color Correction Matrix element.

[Return value]

Returns result status. Refer to [5.8 Status code](#).

[Remarks]

This function will return error status if ColorCorrectionMatrix, ColorCorrectionMatrixSelectorI, and ColorCorrectionMatrixSelectorJ registers or nodes are not implemented in the camera.

Including TeliCamAPI.h is required.

5.5.21.3. SetCamColorCorrectionMatrix

This function writes a new coefficient value to the specified Color Correction Matrix element.

[Syntax]

```
CAM_API_STATUS SetCamColorCorrectionMatrix (  
    CAM_HANDLE          hCam,  
    CAM_COLOR_CORRECTION_MATRIX_TYPE eType,  
    float64_t           dMatrix  
);
```

[Parameters]

Parameter		Description
<i>hCam</i>	[in]	Camera-Handle of target camera.
<i>eType</i>	[in]	Target Color Correction Matrix element.
<i>dMatrix</i>	[in]	New coefficient value.

[Return value]

Returns result status. Refer to [5.8 Status code](#).

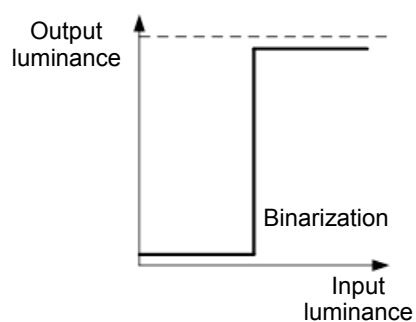
[Remarks]

This function will return error status if ColorCorrectionMatrix, ColorCorrectionMatrixSelectorI, and ColorCorrectionMatrixSelectorJ registers or nodes are not implemented in the camera.

Including TeliCamAPI.h is required.

5.5.22.LUT

This function group performs control of LUT (Look up table) feature for correcting pixel value.
Refer to instruction manual of the camera about LUT.



example of LUT setting

5.5.22.1. GetCamLutEnable

This function reports current status (enabled /disabled) of LUT in the camera,

[Syntax]

```
CAM_API_STATUS GetCamLutEnable (  
    CAM_HANDLE      hCam,  
    bool18_t        *pbEnable  
);
```

[Parameters]

Parameter		Description
<i>hCam</i>	[in]	Camera-Handle of target camera.
<i>pbEnable</i>	[out]	Pointer to a variable that receives current status of LUT. If the value is true, LUT is enabled, otherwise LUT is disabled.

[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.

5.5.22.2. SetCamLutEnable

This function enables or disables LUT feature of the camera,

[Syntax]

```
CAM_API_STATUS SetCamLutEnable (  
    CAM_HANDLE      hCam,  
    bool8_t         bEnable  
);
```

[Parameters]

Parameter		Description
<i>hCam</i>	[in]	Camera-Handle of target camera.
<i>bEnable</i>	[in]	Specify true for enabling LUT in the camera. Specify false for disabling LUT in the camera.

[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.

5.5.22.3. GetCamLutValue

This function reports value of specified element in LUT (output value of LUT feature).

[Syntax]

```
CAM_API_STATUS GetCamLutValue (  
    CAM_HANDLE      hCam,  
    uint32_t         uiLutIndex,  
    uint32_t         *puiLutValue  
);
```

[Parameters]

Parameter		Description
<i>hCam</i>	[in]	Camera-Handle of target camera.
<i>uiLutIndex</i>	[in]	Index of target element in the LUT. (Input value of LUT feature.) Index should be from 0 up to 1023, or from 0 up to 4095.
<i>puiLutValue</i>	[out]	Pointer to a variable that receives current value of the element. (Output value of LUT feature.)

[Return value]

Returns result status. Refer to [5.8 Status code](#).

[Remarks]

LUTIndex register and LUTValue or LUTEntry register is necessary for this function.

This function will return error status if the necessary register or node are not implemented in the camera.

Range of input and output values of LUT, depends on the camera.

Including TeliCamAPI.h is required.

5.5.22.4. SetCamLutValue

This function writes new data to specified element of LUT.

[Syntax]

```
CAM_API_STATUS SetCamLutValue (  
    CAM_HANDLE      hCam,  
    uint32_t         uiLutIndex,  
    uint32_t         uiLutValue  
);
```

[Parameters]

Parameter		Description
<i>hCam</i>	[in]	Camera-Handle of target camera.
<i>uiLutIndex</i>	[in]	Index of target element in the LUT. (Input value of LUT feature.) Index should be from 0 up to 1023, or from 0 up to 4095.
<i>puiLutValue</i>	[in]	New element value. (Output value of LUT feature.)

[Return value]

Returns result status. Refer to [5.8 Status code](#).

[Remarks]

LUTIndex register and LUTValue or LUTEntry register is necessary for this function.

This function will return error status if the necessary register or node are not implemented in the camera.

Range of input and output values of LUT, depends on the camera.

Including TeliCamAPI.h is required.

5.5.23. UserSetControl

This function group performs control of UserSet.
Refer to instruction manual of the camera about UserSet

User application can save current settings of the camera to non-volatile user memory, load saved settings to camera registers using this control.

5.5.23.1. ExecuteCamUserSetLoad

This function 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
<i>hCam</i>	[in]	Camera-Handle of target camera.
<i>eSelector</i>	[in]	Channel of user memory to load settings.

[CAM_USER_SET_SELECTOR_TYPE Enumeration]

Member	Description
<i>CAM_USER_SET_SELECTOR_DEFAULT</i>	Initial factory setting.
<i>CAM_USER_SET_SELECTOR_USER_SET1</i>	Memory channel 1 for user setting.
<i>CAM_USER_SET_SELECTOR_USER_SET2</i>	Memory channel 2 for user setting.
<i>CAM_USER_SET_SELECTOR_USER_SET3</i>	Memory channel 3 for user setting.
<i>CAM_USER_SET_SELECTOR_USER_SET4</i>	Memory channel 4 for user setting.
<i>CAM_USER_SET_SELECTOR_USER_SET5</i>	Memory channel 5 for user setting.
<i>CAM_USER_SET_SELECTOR_USER_SET6</i>	Memory channel 6 for user setting.
<i>CAM_USER_SET_SELECTOR_USER_SET7</i>	Memory channel 7 for user setting.
<i>CAM_USER_SET_SELECTOR_USER_SET8</i>	Memory channel 8 for user setting.
<i>CAM_USER_SET_SELECTOR_USER_SET9</i>	Memory channel 9 for user setting.
<i>CAM_USER_SET_SELECTOR_USER_SET10</i>	Memory channel 10 for user setting.
<i>CAM_USER_SET_SELECTOR_USER_SET11</i>	Memory channel 11 for user setting.
<i>CAM_USER_SET_SELECTOR_USER_SET12</i>	Memory channel 12 for user setting.
<i>CAM_USER_SET_SELECTOR_USER_SET13</i>	Memory channel 13 for user setting.
<i>CAM_USER_SET_SELECTOR_USER_SET14</i>	Memory channel 14 for user setting.
<i>CAM_USER_SET_SELECTOR_USER_SET15</i>	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 or UserSetLoad register (or node) is 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.

5.5.23.2. ExecuteCamUserSetSave

This function saves 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
<i>hCam</i>	[in]	Camera-Handle of target camera.
<i>eSelector</i>	[in]	Channel of user memory to save settings.

[Return value]

Returns result status. Refer to [5.8 Status code](#).

[Remarks]

This function will return error status if UserSetSelector or UserSetSave register (or node) is 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.

5.5.23.3. ExecuteCamUserSetQuickSave

This function 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
<i>hCam</i>	[in]	Camera-Handle of target camera.
<i>eSelector</i>	[in]	Channel of user memory to save settings.

[Return value]

Returns result status. Refer to [5.8 Status code](#).

[Remarks]

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

UserSetQuickSave can reduce the overhed 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.

5.5.23.4. GetCamUserSetDefault

This function reports a user setting 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
<i>hCam</i>	[in]	Camera-Handle of target camera.
<i>peSelector</i>	[out]	Pointer to a variable that receives current value of the channel of user setting.

[Return value]

Returns result status. Refer to [5.8 Status code](#).

[Remarks]

This function will return error status if UserSetDefault is not implemented in the camera.

Including TeliCamAPI.h is required.

5.5.23.5. SetCamUserSetDefault

This function writes new user setting 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
<i>hCam</i>	[in]	Camera-Handle of target camera.
<i>eSelector</i>	[in]	new user setting channel.

[Return value]

Returns result status. Refer to [5.8 Status code](#).

[Remarks]

This function will return error status if UserSetDefault 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 camera's firmware version.) For cameras that are not saved in non-volatile memory, use [ExecuteCamUserSetSaveAndSetDefault\(\)](#) function.

Refer to instruction manual of the camera about UserSet

Including TeliCamAPI.h is required.

5.5.23.6. ExecuteCamUserSetSaveAndSetDefault

This function saves current parameters to specified channel of non-volatile memory (user memory) in the camera, and set the specified channel as a channel loaded on initialization of the camera.

[Syntax]

```
CAM_API_STATUS ExecuteCamUserSetSaveAndSetDefault (  
    CAM_HANDLE          hCam,  
    CAM_USER_SET_SELECTOR_TYPE eSelector  
);
```

[Parameters]

Parameter		Description
<i>hCam</i>	[in]	Camera-Handle of target camera.
<i>eSelector</i>	[in]	Channel of user memory to save settings. This channel is set to a default channel, from which register data will be loaded on initialization of the camera. If CAM_USER_SET_SELECTOR_DEFAULT is specified, the channel (initial factory setting channel) is set as initial load channel without saving current register data to non-volatile memory.

[Return value]

Returns result status. Refer to [5.8 Status code](#).

[Remarks]

This function will return error status if UserSetSelector or UserSetLoad register (or node) is 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 channel 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.

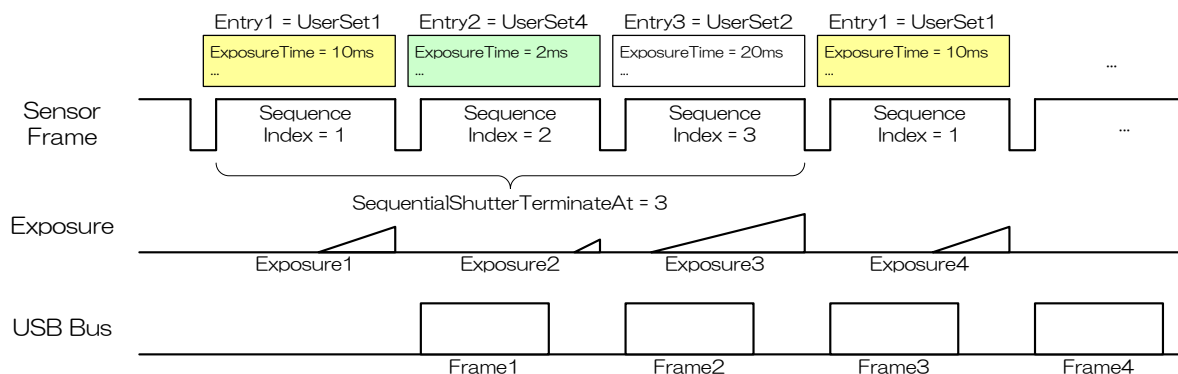
Including TeliCamAPI.h is required.

5.5.24.SequentialShutter

This function group performs control of Sequential shutter.

Refer to instruction manual of the camera about Sequential shutter.

Sequential Shutter is a feature that switches acquisition parameters saved in UserSet memory sequentially.



5.5.24.1. GetCamSequentialShutterEnable

This function reports current status (enabled /disabled) of Sequential Shutter mode of the camera,

[Syntax]

```
CAM_API_STATUS GetCamSequentialShutterEnable (  
    CAM_HANDLE      hCam,  
    bool18_t        *pbEnable  
);
```

[Parameters]

Parameter	Description
<i>hCam</i> [in]	Camera-Handle of target camera.
<i>pbEnable</i> [out]	Pointer to a variable that receives current status of Sequential Shutter mode. If the value is true, it is enabled, otherwise it is disabled.

[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.

5.5.24.2. SetCamSequentialShutterEnable

This function enables /disables Sequential Shutter mode to the camera.

[Syntax]

```
CAM_API_STATUS SetCamSequentialShutterEnable (  
    CAM_HANDLE      hCam,  
    bool18_t        bEnable  
);
```

[Parameters]

Parameter		Description
<i>hCam</i>	[in]	Camera-Handle of target camera.
<i>bEnable</i>	[in]	Specify true to enable Sequential Shutter. Specify false to disable Sequential Shutter.

[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.

5.5.24.3. GetCamSequentialShutterTerminateAtMinMax

This function reports the maximum and the minimum sequence length of Sequential Shutter mode.

[Syntax]

```
CAM_API_STATUS GetCamSequentialShutterTerminateAtMinMax (  
    CAM_HANDLE      hCam,  
    uint32_t         *puiMin,  
    uint32_t         *puiMax  
);
```

[Parameters]

Parameter		Description
<i>hCam</i>	[in]	Camera-Handle of target camera.
<i>puiMin</i>	[out]	Pointer to a variable that receives the minimum sequence length.
<i>puiMax</i>	[out]	Pointer to a variable that receives the maximum sequence length.

[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.

5.5.24.4. GetCamSequentialShutterTerminateAt

This function reports current sequence length of Sequential Shutter mode.
Refer to figure in 5.5.24 SequentialShutter about SequentialShutterTerminateAt,

[Syntax]

```
CAM_API_STATUS GetCamSequentialShutterTerminateAt (  
    CAM_HANDLE      hCam,  
    uint32_t        *puiValue  
);
```

[Parameters]

Parameter		Description
<i>hCam</i>	[in]	Camera-Handle of target camera.
<i>puiValue</i>	[out]	Pointer to a variable that receives current sequence length of Sequential shutter mode.

[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.

5.5.24.5. SetCamSequentialShutterTerminateAt

This function writes new sequence length of Sequential Shutter mode to the camera.
Refer to figure in 5.5.24 SequentialShutter about SequentialShutterTerminateAt.,

[Syntax]

```
CAM_API_STATUS SetCamSequentialShutterTerminateAt (  
    CAM_HANDLE      hCam,  
    uint32_t        uiValue  
);
```

[Parameters]

Parameter		Description
<i>hCam</i>	[in]	Camera-Handle of target camera.
<i>uiValue</i>	[in]	New sequence length of Sequential Shutter mode

[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.

5.5.24.6. GetCamSequentialShutterIndexMinMax

This function reports the maximum and the minimum sequence index for accessing shutter definition data.

[Syntax]

```
CAM_API_STATUS GetCamSequentialShutterIndexMinMax (  
    CAM_HANDLE      hCam,  
    uint32_t        *puiMin,  
    uint32_t        *puiMax  
);
```

[Parameters]

Parameter		Description
<i>hCam</i>	[in]	Camera-Handle of target camera.
<i>puiMin</i>	[out]	Pointer to a variable that receives the minimum sequence index value.
<i>puiMax</i>	[out]	Pointer to a variable that receives the maximum sequence index value.

[Return value]

Returns result status. Refer to [5.8 Status code](#).

[Remarks]

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

Including TeliCamAPI.h is required.

5.5.24.7. GetCamSequentialShutterEntryMinMax

This function reports the maximum and the minimum UserSet channel number available in each sequence of Sequential Shutter mode.

Register set data in UserSet is used for switching parameters of image acquisition

[Syntax]

```
CAM_API_STATUS GetCamSequentialShutterEntryMinMax (  
    CAM_HANDLE      hCam,  
    uint32_t        *puiMin,  
    uint32_t        *puiMax  
);
```

[Parameters]

Parameter		Description
<i>hCam</i>	[in]	Camera-Handle of target camera.
<i>puiMin</i>	[out]	Pointer to a variable that receives the minimum Userset index value.
<i>puiMax</i>	[out]	Pointer to a variable that receives the maximum Userset index value.

[Return value]

Returns result status. Refer to [5.8 Status code](#).

[Remarks]

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

Including TeliCamAPI.h is required.

5.5.24.8. GetCamSequentialShutterEntry

This function reports current UserSet channel number registered in the specified sequence index of Sequential Shutter Entries.

[Syntax]

```
CAM_API_STATUS GetCamSequentialShutterEntry (  
    CAM_HANDLE      hCam,  
    uint32_t        uiIndex,  
    uint32_t        *puiEntry  
);
```

[Parameters]

Parameter		Description
<i>hCam</i>	[in]	Camera-Handle of target camera.
<i>uiIndex</i>	[in]	Target sequence index.
<i>puiEntry</i>	[out]	Pointer to a variable that receives current UserSet channel number registered to specified sequence index.

[Return value]

Returns result status. Refer to [5.8 Status code](#).

[Remarks]

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

Including TeliCamAPI.h is required.

5.5.24.9. SetCamSequentialShutterEntry

This function writes new UserSet channel number to the specified sequence index of Sequential Shutter Entries.

[Syntax]

```
CAM_API_STATUS SetCamSequentialShutterEntry (  
    CAM_HANDLE      hCam,  
    uint32_t         uiIndex,  
    uint32_t         uiEntry  
);
```

[Parameters]

Parameter		Description
<i>hCam</i>	[in]	Camera-Handle of target camera.
<i>uiIndex</i>	[in]	Target sequence index.
<i>uiEntry</i>	[in]	UserSet channelnumber.

[Return value]

Returns result status. Refer to [5.8 Status code](#).

[Remarks]

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

Including TeliCamAPI.h is required.

5.5.25. UserDefinedName (DeviceUserID)

This function 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,

Refer to instruction manual of the camera about "UserDefinedName" or "DeviceUserID" register.

5.5.25.1. GetCamUserDefinedName

This function reports user defined name of the camera.

[Syntax]

```
CAM_API_STATUS GetCamUserDefinedName (  
    CAM_HANDLE      hCam,  
    char             *pszName,  
    uint32_t         *puiSize  
);
```

[Parameters]

Parameter	Description
<i>hCam</i> [in]	Camera-Handle of target camera.
<i>pszName</i> [out]	Pointer to a buffer that receives user defined name. If NULL is specified to pszName, the necessary buffer size will be written to the variable pointed by puiSize.
<i>puiSize</i> [in,out]	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 either "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 writes new user defined name, the old user defined name may remain until "Sys_GetNumOfCameras()" is called.

Including TeliCamAPI.h is required.

5.5.25.2. SetCamUserDefinedName

This function writes new user defined name to the camera.

[Syntax]

```
CAM_API_STATUS SetCamUserDefinedName (  
    CAM_HANDLE      hCam,  
    char            *pszName,  
);
```

[Parameters]

Parameter	Description
<i>hCam</i> [in]	Camera-Handle of target camera.
<i>pszName</i> [in]	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 either "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 within length if user defined name.

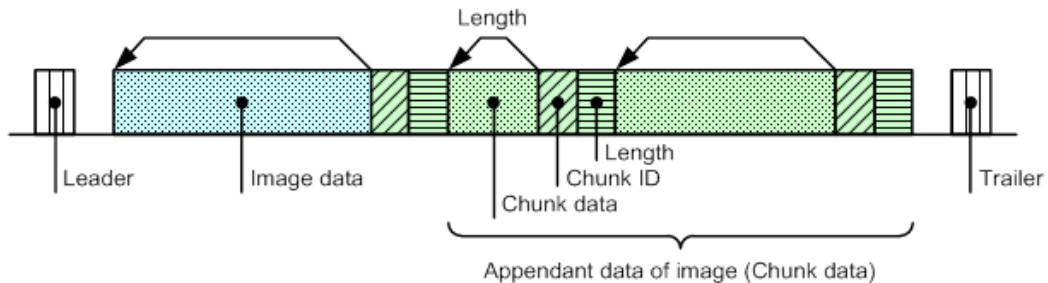
Including TeliCamAPI.h is required.

5.5.26.Chunk

This function group performs control of Chunk feature.

Chunk is a function to add various information to payload data.

When the chunk output is valid, chunk data is added to the payload data after the image data.



For detailed information on chunk, please 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's chunk port.

For details, see the description of the [Chunk_AttachBuffer \(\)](#) function.

5.5.26.1. GetCamChunkModeActive

This function reports current status (enabled /disabled) of chunk data output mode of the camera.

[Syntax]

```
CAM_API_STATUS GetCamChunkModeActive (  
    CAM_HANDLE      hCam,  
    bool18_t        *pbValue  
);
```

[Parameters]

Parameter	Description
<i>hCam</i> [in]	Camera-Handle of target camera.
<i>pbValue</i> [out]	Pointer to a variable that receives current status of chunk data output mode. If the value is true, it is enabled, otherwise it is disabled.

[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.

5.5.26.2. SetCamChunkModeActive

This function enables /disables chunk data output mode to the camera.

[Syntax]

```
CAM_API_STATUS SetCamChunkModeActive (  
    CAM_HANDLE      hCam,  
    bool18_t        bValue  
);
```

[Parameters]

Parameter		Description
<i>hCam</i>	[in]	Camera-Handle of target camera.
<i>bValue</i>	[in]	Specify true to enable chunk data output. Specify false to disable chunk data output.

[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.

5.5.27. Miscellaneous

5.5.27.1. 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
<i>hCam</i>	[in]	Camera-Handle of target camera.
<i>puiCamIndex</i>	[out]	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.

5.5.27.2. 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
<i>hCam</i>	[in]	Camera-Handle of target camera.
<i>peType</i>	[out]	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.

5.5.27.3. 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 TeliCamAPI is called, TeliCamAPI write 1 to TLParamsLocked.

When streaming stop function of TeliCamAPI is called, TeliCamAPI writes 0 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
<i>hCam</i> [in]	Camera-Handle of target camera.
<i>puiValue</i> [out]	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.

5.5.27.4. 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
<i>peErrMsg</i> [out]	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
<i>pszDescription</i> [out]	Error idescription.
<i>pszSpurceFileName</i> [out]	The file name in which the error occurred.
<i>uiSourceLine</i> [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 , 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. 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. 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 pNode.

[Syntax]

```
CAM_API_STATUS Nd_GetNode (  
    CAM_HANDLE      hCam,  
    const char      *pszName,  
    CAM_NODE_HANDLE *pNode  
);
```

[Parameters]

Parameter	Description
<i>hCam</i> [in]	Camera-Handle of target camera.
<i>pszName</i> [in]	NULL terminated node name.
<i>pNode</i> [out]	Pointer to a variable that receives node handle of the retrived node.

[Return value]

Returns result status. Refer to [5.8 Status code](#).

[Remarks]

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.

Including TeliCamAPI.h is required

[Example]

```
C++

CAM_API_STATUS      uiStatus;
uint32_t            uiNum, uiSize;
CAM_HANDLE          hCam = NULL;
CAM_NODE_HANDLE     hNode = 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 node handle.
    uiStatus = Nd_GetNode(hCam, "Gain", &hNode);
    if (uiStatus != CAM_API_STS_SUCCESS)
        return -1;
    printf("hNode = 0x%I64x\n", hNode);

    // Get type.
    uiStatus = Nd_GetType(hCam, hNode, &eNodeType);
    if (uiStatus != CAM_API_STS_SUCCESS)
        return -1;
    printf("Type = %d\n", (uint32_t)eNodeType);

    // Get access mode.
    uiStatus = Nd_GetAccessMode(hCam, hNode, &eNodeAccessMode);
    if (uiStatus != CAM_API_STS_SUCCESS)
        return -1;
    printf("AccessMode = %d\n", (uint32_t)eNodeAccessMode);

    // Get visibility.
    uiStatus = Nd_GetVisibility(hCam, hNode, &eNodeVisibility);
    if (uiStatus != CAM_API_STS_SUCCESS)
        return -1;
    printf("Visibility = %d\n", (uint32_t)eNodeVisibility);

    // Get cachingMode.
    uiStatus = Nd_GetCachingMode(hCam, hNode, &eNodeCachingMode);
    if (uiStatus != CAM_API_STS_SUCCESS)
        return -1;
    printf("CachingMode = %d\n", (uint32_t)eNodeCachingMode);

    // Get description.
    uiSize = 0;
    uiStatus = Nd_GetDescription(hCam, hNode, 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 = Nd_GetDescription(hCam, hNode, 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 = Nd_GetToolTip(hCam, hNode, 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 = Nd_GetToolTip(hCam, hNode, 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 = Nd_GetRepresentation(hCam, hNode, &nodeRepresentation);
if (uiStatus != CAM_API_STS_SUCCESS)
    return -1;
printf("Representation = %d\n", (uint32_t)nodeRepresentation);

// Get unit.
uiSize = 0;
uiStatus = Nd_GetUnit(hCam, hNode, 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 = Nd_GetUnit(hCam, hNode, 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 != 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. 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
<i>hCam</i> [in]	Camera-Handle of target camera.
<i>hNode</i> [in]	Node handle of target node.
<i>peNodeType</i> [out]	Pointer to a variable that receives node type of the specified node.

[*TC_NODE_TYPE* Enumeration]

Member	Description
<i>TC_NODE_TYPE_VALUE</i>	IValue interface
<i>TC_NODE_TYPE_BASE</i>	IBase interface
<i>TC_NODE_TYPE_INTEGER</i>	IInteger interface
<i>TC_NODE_TYPE_BOOLEAN</i>	IBoolean interface
<i>TC_NODE_TYPE_COMMAND</i>	ICommand interface
<i>TC_NODE_TYPE_FLOAT</i>	IFloat interface
<i>TC_NODE_TYPE_STRING</i>	IString interface
<i>TC_NODE_TYPE_REGISTER</i>	IRegister interface
<i>TC_NODE_TYPE_CATEGORY</i>	ICategory interface
<i>TC_NODE_TYPE_ENUMERATION</i>	IEnumeration interface
<i>TC_NODE_TYPE_ENUM_ENTRY</i>	IEnumEntry interface
<i>TC_NODE_TYPE_PORT</i>	IPort interface
<i>TC_NODE_TYPE_UNKNOWN</i>	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]

Refer to [example](#) of [5.6.1.1 Nd_GetNode](#).

5.6.1.3. 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
<i>hCam</i>	[in]	Camera-Handle of target camera.
<i>hNode</i>	[in]	Node handle of target node.
<i>pszNode</i>	[out]	Pointer to a variable that receives node name of the node.

[CAM_NODE_NAME structure]

```
typedef struct _UNI_NODE_NAME  
{  
    char    szNodeName[NODE_NAME_LENGTH_MAX];  
} UNI_NODE_NAME, *PUNI_NODE_NAME;
```

Member		Description
<i>szNodeName</i>	[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 Nd_GetNumOfFeatures](#).

5.6.1.4. Nd_GetAccessMode

This function reports access mode of the node specified by the argument node handle.

[Syntax]

```
CAM_API_STATUS Nd_GetName (  
    CAM_HANDLE          Cam,  
    CAM_NODE_HANDLE     hNode,  
    TC\_NODE\_ACCESS\_MODE *peAccessMode  
);
```

[Parameters]

Parameter		Description
<i>hCam</i>	[in]	Camera-Handle of target camera.
<i>hNode</i>	[in]	Node handle of target node.
<i>peAccessMode</i>	[out]	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]

[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](#).

Returns result status. Refer to [5.8 Status code](#).

[Remarks]

Including TeliCamAPI.h is required.

[Example]

Refer to [example](#) of [5.6.1.1 Nd_GetNode](#).

5.6.1.5. Nd_GetVisibility

This function reports visibility of the node specified by the argument node handle.

[Syntax]

```
CAM_API_STATUS Nd_GetVisibility (  
    CAM_HANDLE          Cam,  
    CAM_NODE_HANDLE     hNode,  
    TC\_NODE\_VISIBILITY *peVisibility  
);
```

[Parameters]

Parameter		Description
<i>hCam</i>	[in]	Camera-Handle of target camera.
<i>hNode</i>	[in]	Node handle of target node.
<i>peVisibility</i>	[out]	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]

[TC_NODE_VISIBILITY](#) is declared in TeliCamNode.h.

Enum type “EVisibility” declared in Types.h of GenApi reference implementation corresponds to [TC_NODE_VISIBILITY](#).

Returns result status. Refer to [5.8 Status code](#).

[Remarks]

Including TeliCamAPI.h is required.

[Example]

Refer to [example](#) of [5.6.1.1 Nd_GetNode](#).

5.6.1.6. Nd_GetCachingMode

This function reports caching mode of the node specified by the argument node handle

[Syntax]

```
CAM_API_STATUS Nd_GetCachingMode (  
    CAM_HANDLE          Cam,  
    CAM_NODE_HANDLE     hNode,  
    TC\_NODE\_CACHING\_MODE *peCachingMode  
);
```

[Parameters]

Parameter		Description
<i>hCam</i>	[in]	Camera-Handle of target camera.
<i>hNode</i>	[in]	Node handle of target node.
<i>peCachingMode</i>	[out]	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]

[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](#).

Returns result status. Refer to [5.8 Status code](#).

[Remarks]

Including TeliCamAPI.h is required.

[Example]

Refer to [example](#) of [5.6.1.1 Nd_GetNode](#).

5.6.1.7. Nd_GetDescription

This function reports description of the node specified by the argument node handle.

[Syntax]

```
CAM_API_STATUS Nd_GetDescription (  
    CAM_HANDLE          Cam,  
    CAM_NODE_HANDLE     hNode,  
    char                *pszBuf,  
    uint32_t             *puiSize  
);
```

[Parameters]

Parameter		Description
<i>hCam</i>	[in]	Camera-Handle of target camera.
<i>hNode</i>	[in]	Node handle of target node.
<i>pszBuf</i>	[out]	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 <i>puiSize</i> , without reporting description string.
<i>puiSize</i>	[in, out]	Pointer to a variable that contains size of buffer pointed by <i>pszBuf</i> If NULL is specified to <i>pszBuf</i> , 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 Nd_GetNode](#).

5.6.1.8. Nd_GetToolTip

This function reports tool tip of the node specified by the argument node handle.

[Syntax]

```
CAM_API_STATUS Nd_GetToolTip (  
    CAM_HANDLE          Cam,  
    CAM_NODE_HANDLE     hNode,  
    char                *pszBuf,  
    uint32_t             *puiSize  
);
```

[Parameters]

Parameter		Description
<i>hCam</i>	[in]	Camera-Handle of target camera.
<i>hNode</i>	[in]	Node handle of target node.
<i>pszBuf</i>	[out]	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 <i>puiSize</i> , without reporting tool tip string.
<i>puiSize</i>	[in, out]	Pointer to a variable that contains size of buffer pointed by <i>pszBuf</i> . If NULL is specified to <i>pszBuf</i> , 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 Nd_GetNode](#).

5.6.1.9. Nd_GetRepresentation

This function reports recommended representation of the node specified by the argument node handle.

[Syntax]

```
CAM_API_STATUS Nd_GetRepresentation (  
    CAM_HANDLE          Cam,  
    CAM_NODE_HANDLE     hNode,  
    TC\_NODE\_REPRESENTATION *peRepresentation  
);
```

[Parameters]

Parameter		Description
<i>hCam</i>	[in]	Camera-Handle of target camera.
<i>hNode</i>	[in]	Node handle of target node.
<i>peRepresentation</i>	[out]	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 Nd_GetNode](#).

5.6.1.10. 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
<i>hCam</i> [in]	Camera-Handle of target camera.
<i>hNode</i> [in]	Node handle of target node.
<i>pszBuf</i> [out]	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 <i>puiSize</i> , without reporting unit name string.
<i>puiSize</i> [in, out]	Pointer to variable that contains size of buffer pointed by <i>pszBuf</i> . If NULL is specified to <i>pszBuf</i> , 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 Nd_GetNode](#).

5.6.2. ICategory node functions

5.6.2.1. 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
<i>hCam</i>	[in]	Camera-Handle of target camera.
<i>hNode</i>	[in]	Node handle of target node.
<i>puiNum</i>	[out]	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 = NULL;  
CAM_NODE_HANDLE      hNode = NULL;  
CAM_NODE_HANDLE      hCatNode = NULL;  
CAM_NODE_NAME        sNodeName;  
  
// 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
{
#ifdef _DEBUG
    // For Gig-E Vision camera debug.
    Cam_SetHeartbeat(hCam, false, 0);
#endif

    // Get node handle.
    uiStatus = Nd_GetNode(hCam, "DeviceControl", &hNode);
    if (uiStatus != CAM_API_STS_SUCCESS)
        return -1;
    printf("hNode = %I64x¥n", hNode);

    // Get num of features.
    uiStatus = Nd_GetNumOfFeatures(hCam, hNode, &uiFeatureNum);
    if (uiStatus != CAM_API_STS_SUCCESS)
        return -1;
    printf("Num of features = %d¥n", uiFeatureNum);

    for (i = 0; i < uiFeatureNum; i++) {
        // Get feature by index.
        uiStatus = Nd_GetFeatureByIndex(hCam, hNode, i, &hCatNode);
        if (uiStatus != CAM_API_STS_SUCCESS)
            return -1;

        // Get node name.
        uiStatus = Nd_GetName(hCam, hCatNode, &sNodeName);
        if (uiStatus != CAM_API_STS_SUCCESS)
            return -1;
        printf("No.%d : %s¥n", i, sNodeName.szNodeName);
    }

    return 0;
}
__finally
{
    // Close camera.
    if (hCam != 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. 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
<i>hCam</i>	[in]	Camera-Handle of target camera.
<i>hNode</i>	[in]	Node handle of the target category node.
<i>uiNodeIndex</i>	[in]	Index in feature list of the category node.
<i>phNode</i>	[out]	Pointer to a variable that receives node handle of a feature node, whose index is same as index specified in the argument .

[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]

Refer to [example](#) of [5.6.2.1 Nd_GetNumOfFeatures](#).

5.6.3. Integer node functions

5.6.3.1. Nd_GetIntMin

This function reports the minimum valid value of the specified node.

This function assumes that target node is Integer type node.

[Syntax]

```
CAM_API_STATUS Nd_GetIntMin (  
    CAM_HANDLE      hCam,  
    CAM_NODE_HANDLE hNode,  
    int64_t          *pLLMin  
);
```

[Parameters]

Parameter		Description
<i>hCam</i>	[in]	Camera-Handle of target camera.
<i>hNode</i>	[in]	Node handle of target node.
<i>pLLMin</i>	[out]	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 Integer type node (TC_NODE_TYPE_INTEGER).

This function will return error status if node type of target node is not Integer type.

Including TeliCamAPI.h is required.

[Example]

```
C++  
  
CAM_API_STATUS   uiStatus;  
uint32_t         uiNum;  
CAM_HANDLE       hCam = NULL;  
CAM_NODE_HANDLE  hNode = 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 node handle.
uiStatus = Nd_GetNode(hCam, "Width", &hNode);
if (uiStatus != CAM_API_STS_SUCCESS)
    return -1;
printf("hNode = %I64x¥n", hNode);

// Get minimum value.
uiStatus = Nd_GetIntMin(hCam, hNode, &llMin);
if (uiStatus != CAM_API_STS_SUCCESS)
    return -1;
printf("Width Min   : %d¥n", (uint32_t)llMin);

// Get maximum value.
uiStatus = Nd_GetIntMax(hCam, hNode, &llMax);
if (uiStatus != CAM_API_STS_SUCCESS)
    return -1;
printf("          Max   : %d¥n", (uint32_t)llMax);

// Get increment value.
uiStatus = Nd_GetIntInc(hCam, hNode, &llInc);
if (uiStatus != CAM_API_STS_SUCCESS)
    return -1;
printf("          Inc   : %d¥n", (uint32_t)llInc);

// Get value.
uiStatus = Nd_GetIntValue(hCam, hNode, &llRdValue);
if (uiStatus != CAM_API_STS_SUCCESS)
    return -1;
printf("          Before Value : %d¥n", (uint32_t)llRdValue);

// Set value.
llWrValue = llMin + llInc;
uiStatus = Nd_SetIntValue(hCam, hNode, llWrValue);
if (uiStatus != CAM_API_STS_SUCCESS)
    return -1;

// Get value.
uiStatus = Nd_GetIntValue(hCam, hNode, &llRdValue);
if (uiStatus != CAM_API_STS_SUCCESS)
    return -1;
printf("          After Value : %d¥n", (uint32_t)llRdValue);

return 0;
}
__finally
{
    // Close camera.
    if (hCam != 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. Nd_GetIntMax

This function reports the maximum valid value of the specified node.
This function assumes that target node is Integer type node.

[Syntax]

```
CAM_API_STATUS Nd_GetIntMax (  
    CAM_HANDLE      hCam,  
    CAM_NODE_HANDLE hNode,  
    int64_t          *pLLMax  
);
```

[Parameters]

Parameter		Description
<i>hCam</i>	[in]	Camera-Handle of target camera.
<i>hNode</i>	[in]	Node handle of target node.
<i>pLLMax</i>	[out]	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 Integer type node (TC_NODE_TYPE_INTEGER).

This function will return error status if node type of target node is not Integer type.

Including TeliCamAPI.h is required.

[Example]

Refer to [example](#) of [5.6.3.1 Nd_GetIntMin](#).

5.6.3.3. Nd_GetIntInc

This function reports the Increment value of the specified node.
This function assumes that target node is Integer type node.

[Syntax]

```
CAM_API_STATUS Nd_GetIntInc (  
    CAM_HANDLE      hCam,  
    CAM_NODE_HANDLE hNode,  
    int64_t          *pLLInc  
);
```

[Parameters]

Parameter		Description
<i>hCam</i>	[in]	Camera-Handle of target camera.
<i>hNode</i>	[in]	Node handle of target node.
<i>pLLInc</i>	[out]	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 Integer type node (TC_NODE_TYPE_INTEGER).

This function will return error status if node type of target node is not Integer type.

Including TeliCamAPI.h is required.

[Example]

Refer to [example](#) of [5.6.3.1 Nd_GetIntMin](#).

5.6.3.4. Nd_GetIntValue

This function reports current value of the specified node.

This function assumes that target node is Integer 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
<i>hCam</i>	[in]	Camera-Handle of target camera.
<i>hNode</i>	[in]	Node handle of target node.
<i>pLLValue</i>	[out]	Pointer to a variable that receives current value.
<i>bVerify</i>	[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.
<i>bIgnoreCache</i>	[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 Integer type node (TC_NODE_TYPE_INTEGER).

This function will return error status if node type of target node is not Integer type.

Including TeliCamAPI.h is required.

[Example]

Refer to [example](#) of [5.6.3.1 Nd_GetIntMin](#).

5.6.3.5. Nd_SetIntValue

This function writes new value to the node specified by the argument node handle.
This function assumes that target node is Integer 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
<i>hCam</i>	[in]	Camera-Handle of target camera.
<i>hNode</i>	[in]	Node handle of target node.
<i>LLValue</i>	[in]	New value.
<i>bVerify</i>	[in]	Specify true to check access mode and valid value range before writing data. Specify false to skip checking. This argument is optional. 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 Integer type node (TC_NODE_TYPE_INTEGER).

This function will return error status if node type of target node is not Integer type.

Including TeliCamAPI.h is required.

[Example]

Refer to [example](#) of [5.6.3.1 Nd_GetIntMin](#).

5.6.4. IFloat node functions

5.6.4.1. 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
<i>hCam</i>	[in]	Camera-Handle of target camera.
<i>hNode</i>	[in]	Node handle of target node.
<i>pdMin</i>	[out]	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++	
<pre>CAM_API_STATUS uiStatus; uint32_t uiNum; CAM_HANDLE hCam = NULL; CAM_NODE_HANDLE hNode = 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;</pre>	

```

__try
{
    // Get node handle.
    uiStatus = Nd_GetNode(hCam, "Gain", &hNode);
    if (uiStatus != CAM_API_STS_SUCCESS)
        return -1;
    printf("hNode = %I64x¥n", hNode);

    // Get minimum value.
    uiStatus = Nd_GetFloatMin(hCam, hNode, &dMin);
    if (uiStatus != CAM_API_STS_SUCCESS)
        return -1;
    printf("Width Min   : %f¥n", dMin);

    // Get maximum value.
    uiStatus = Nd_GetFloatMax(hCam, hNode, &dMax);
    if (uiStatus != CAM_API_STS_SUCCESS)
        return -1;
    printf("          Max    : %f¥n", dMax);

    // Confirm whether the float node has a constant increment.
    uiStatus = Nd_GetFloatHasInc(hCam, hNode, &bInc);
    if (uiStatus != CAM_API_STS_SUCCESS)
        return -1;

    if (bInc) {
        // Get increment value.
        uiStatus = Nd_GetFloatInc(hCam, hNode, &dInc);
        if (uiStatus != CAM_API_STS_SUCCESS)
            return -1;
        printf("          Inc    : %f¥n", dInc);
    }

    // Get display notation.
    uiStatus = Nd_GetFloatDisplayNotation(hCam, hNode, &eDisplayNotation);
    if (uiStatus != CAM_API_STS_SUCCESS)
        return -1;
    printf("      Display Notation : %d¥n", eDisplayNotation);

    // Get display precision.
    uiStatus = Nd_GetFloatDisplayPrecision(hCam, hNode, &llPrecision);
    if (uiStatus != CAM_API_STS_SUCCESS)
        return -1;
    printf("      Display Precision : %d¥n", (uint32_t)llPrecision);

    // Get value.
    uiStatus = Nd_GetFloatValue(hCam, hNode, &dRdValue);
    if (uiStatus != CAM_API_STS_SUCCESS)
        return -1;
    printf("      Before Value : %f¥n", dRdValue);

    // Set value.
    dWrValue = (dMin + dMax) / 2.0f;
    uiStatus = Nd_SetFloatValue(hCam, hNode, dWrValue);
    if (uiStatus != CAM_API_STS_SUCCESS)
        return -1;

    // Get value.
    uiStatus = Nd_GetFloatValue(hCam, hNode, &dRdValue);
    if (uiStatus != CAM_API_STS_SUCCESS)
        return -1;
    printf("      After Value : %f¥n", dRdValue);

    return 0;
}
__finally
{
    // Close camera.
    if (hCam != 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. 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
<i>hCam</i>	[in]	Camera-Handle of target camera.
<i>hNode</i>	[in]	Node handle of target node whose maximum valid value is to be reported.
<i>dMax</i>	[out]	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 Nd_GetFloatMin](#).

5.6.4.3. 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,  
    bool18_t        *pbInc  
);
```

[Parameters]

Parameter		Description
<i>hCam</i>	[in]	Camera-Handle of target camera.
<i>hNode</i>	[in]	Node handle of target node.
<i>pbInc</i>	[out]	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 Nd_GetFloatMin](#).

5.6.4.4. 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
<i>hCam</i>	[in]	Camera-Handle of target camera.
<i>hNode</i>	[in]	Node handle of target node.
<i>pdInc</i>	[out]	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 Nd_GetFloatMin](#).

5.6.4.5. 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
<i>hCam</i>	[in]	Camera-Handle of target camera.
<i>hNode</i>	[in]	Node handle of target node.
<i>peDisplayNotation</i>	[out]	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 Nd_GetFloatMin](#).

5.6.4.6. 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
<i>hCam</i>	[in]	Camera-Handle of target camera.
<i>hNode</i>	[in]	Node handle of target node.
<i>pLLPrecision</i>	[out]	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 Nd_GetFloatMin](#).

5.6.4.7. 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
<i>hCam</i>	[in]	Camera-Handle of target camera.
<i>hNode</i>	[in]	Node handle of target node.
<i>pdValue</i>	[out]	Pointer to a variable that receives current value.
<i>bVerify</i>	[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.
<i>bIgnoreCache</i>	[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 Nd_GetFloatMin](#).

5.6.4.8. 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
<i>hCam</i>	[in]	Camera-Handle of target camera.
<i>hNode</i>	[in]	Node handle of target node.
<i>dValue</i>	[in]	New value.
<i>bVerify</i>	[in]	Specify true to check access mode and valid value range before writing data. Specify false to skip checking. This argument is optional. 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 Nd_GetFloatMin](#).

5.6.5. IBoolean node functions

5.6.5.1. Nd_GetBoolValue

This function reports the Increment 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
<i>hCam</i>	[in]	Camera-Handle of target camera.
<i>hNode</i>	[in]	Node handle of target node.
<i>pbValue</i>	[out]	Pointer to a variable that receives current value.
<i>bVerify</i>	[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.
<i>bIgnoreCache</i>	[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 = NULL;
CAM_NODE_HANDLE     hNode = NULL;
bool18_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 node handle.
    uiStatus = Nd_GetNode(hCam, "ReverseX", &hNode);
    if (uiStatus != CAM_API_STS_SUCCESS)
        return -1; // Not implemented, or any other error
    printf("hNode = %I64x\n", hNode);

    // Get value.
    uiStatus = Nd_GetBoolValue(hCam, hNode, &bRdValue);
    if (uiStatus != CAM_API_STS_SUCCESS)
        return -1;
    printf("    Before Value : %d\n", (uint32_t)bRdValue);

    // Set value.
    bWrValue = !bRdValue;
    uiStatus = Nd_SetBoolValue(hCam, hNode, bWrValue);
    if (uiStatus != CAM_API_STS_SUCCESS)
        return -1;

    // Get value.
    uiStatus = Nd_GetBoolValue(hCam, hNode, &bRdValue);
    if (uiStatus != CAM_API_STS_SUCCESS)
        return -1;
    printf("    After Value : %d\n", (uint32_t)bRdValue);

    return 0;
}
__finally
{
    // Close camera.
    if (hCam != 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. 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
<i>hCam</i> [in]	Camera-Handle of target camera.
<i>hNode</i> [in]	Node handle of target node.
<i>bValue</i> [in]	New value.
<i>bVerify</i> [in]	Specify true to check access mode and valid value range before writing data. Specify false to skip checking. This argument is optional. 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 Nd_GetBoolValue](#).

5.6.6. IEnumeration node functions

5.6.6.1. 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        *piNum  
);
```

[Parameters]

Parameter		Description
<i>hCam</i>	[in]	Camera-Handle of target camera.
<i>hNode</i>	[in]	Node handle of target IEnumeration node.
<i>piNum</i>	[out]	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++	
<pre>CAM_API_STATUS uiStatus; uint32_t uiNum, uiSize; CAM_HANDLE hCam = 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;</pre>	

```

// 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 = Nd_GetNode(hCam, "LineSelector", &hNode);
    if (uiStatus != CAM_API_STS_SUCCESS)
        return -1; // Not implemented, or any other error
    printf("hNode = %I64x\n", hNode);

    // Get num of entries.
    uiStatus = Nd_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 = Nd_GetEnumEntryByIndex(hCam, hNode, i, &hEnumEntryNode);
        if (uiStatus != CAM_API_STS_SUCCESS)
            return -1;

        // Get int value.
        uiStatus = Nd_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 = Nd_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 = Nd_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 != 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. 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
<i>hCam</i>	[in]	Camera-Handle of target camera.
<i>hNode</i>	[in]	Node handle of target node whose current value is to be reported.
<i>pLLValue</i>	[out]	Pointer to a variable that receives current value.
<i>bVerify</i>	[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.
<i>bIgnoreCache</i>	[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++
<pre>CAM_API_STATUS uiStatus; uint32_t uiNum; CAM_HANDLE hCam = NULL; CAM_NODE_HANDLE hNode = 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;</pre>

```

// 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
{
#ifdef _DEBUG
    // For Gig-E Vision camera debug.
    Cam_SetHeartbeat(hCam, false, 0);
#endif

    // Get node handle.
    uiStatus = Nd_GetNode(hCam, "TriggerSource", &hNode);
    if (uiStatus != CAM_API_STS_SUCCESS)
        return -1; // Not implemented, or any other error
    printf("hNode = %I64x¥n", hNode);

    // Set int value.
    llWrValue = 64; // 64 : Software
    uiStatus = Nd_SetEnumIntValue(hCam, hNode, llWrValue);
    if (uiStatus != CAM_API_STS_SUCCESS)
        return -1; // Not implemented, or any other error

    // Readback.
    uiStatus = Nd_GetEnumIntValue(hCam, hNode, &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 != NULL) {
        uiStatus = Cam_Close(hCam);
        if (uiStatus != CAM_API_STS_SUCCESS)
            printf("Cam_Close error! (0x%x)", uiStatus);
    }

    // Terminate system.
    Sys_Terminate();
}

```

5.6.6.3. 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
<i>hCam</i>	[in]	Camera-Handle of target camera.
<i>hNode</i>	[in]	Node handle of target node.
<i>llValue</i>	[in]	New value expressed in integer value.
<i>bVerify</i>	[in]	Specify true to check access mode and validity of the value before writing data. Specify false to skip checking. This argument is optional. 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 [5.6.6.2 Nd_GetEnumIntValue](#).

5.6.6.4. 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_GetEnumIntValue (  
    CAM_HANDLE      hCam,  
    CAM_NODE_HANDLE hNode,  
    char             *pszBuf,  
    uint32_t         *puiSize,  
    bool8_t          bVerify = false,  
    bool8_t          bIgnoreCache = false  
);
```

[Parameters]

Parameter		Description
<i>hCam</i>	[in]	Camera-Handle of target camera.
<i>hNode</i>	[in]	Node handle of target node whose current value is to be reported.
<i>pszBuf</i>	[out]	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 <i>puiSize</i> , without reporting string value.
<i>puiSize</i>	[in, out]	Pointer to variable that contains size of buffer pointed by <i>pszBuf</i> . If NULL is specified to <i>pszBuf</i> , this function will writes buffer size necessary for writing string value of the node to variable pointed by this argument.
<i>bVerify</i>	[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.
<i>bIgnoreCache</i>	[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 = NULL;

```

CAM_NODE_HANDLE    hNode = 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
{
#ifdef _DEBUG
    // For Gig-E Vision camera debug.
    Cam_SetHeartbeat(hCam, false, 0);
#endif

    // Get node handle.
    uiStatus = Nd_GetNode(hCam, "TriggerSource", &hNode);
    if (uiStatus != CAM_API_STS_SUCCESS)
        return -1; // Not implemented, or any other error

    // Read string.
    uiSize = 32;
    uiStatus = Nd_GetEnumStrValue(hCam, hNode, 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 = Nd_SetEnumStrValue(hCam, hNode, "Software");
    } else {
        uiStatus = Nd_SetEnumStrValue(hCam, hNode, "Line0");
    }
    if (uiStatus != CAM_API_STS_SUCCESS)
        return -1; // Not implemented, or any other error

    // Readback.
    uiSize = 32;
    uiStatus = Nd_GetEnumStrValue(hCam, hNode, 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 != NULL) {
        uiStatus = Cam_Close(hCam);
        if (uiStatus != CAM_API_STS_SUCCESS)
            printf("Cam_Close error! (0x%x)", uiStatus);
    }

    // Terminate system.
    Sys_Terminate();
}

```

5.6.6.5. 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
<i>hCam</i> [in]	Camera-Handle of target camera.
<i>hNode</i> [in]	Node handle of target node.
<i>pszBuf</i> [in]	Pointer to new string value.
<i>bVerify</i> [in]	Specify true to check access mode and validity of the value before writing data. Specify false to skip checking. This argument is optional. 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 Nd_GetNumOfEnumEntries](#).

5.6.6.6. 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
<i>hCam</i>	[in]	Camera-Handle of target camera.
<i>hNode</i>	[in]	Node handle of target IEnumeration node.
<i>uiEnumIdx</i>	[in]	Index in entry list described in target node.
<i>phEnumEntryNode</i>	[out]	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 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 Nd_GetNumOfEnumEntries](#).

5.6.7. IEnumEntry node functions

5.6.7.1. 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
<i>hCam</i>	[in]	Camera-Handle of target camera.
<i>hNode</i>	[in]	Node handle of target enumeration entry node.
<i>pLLValue</i>	[out]	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 IEnumEntry type node (TC_NODE_TYPE_ENUM_ENTRY).

This function will return error status if node type of target node is not IEnumEntry type.

Including TeliCamAPI.h is required.

[Example]

Refer to [example](#) of [5.6.6.1 Nd_GetNumOfEnumEntries](#).

5.6.7.2. 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
<i>hCam</i>	[in]	Camera-Handle of target camera.
<i>hNode</i>	[in]	Node handle of target enumeration entry node.
<i>pszBuf</i>	[out]	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 <i>puiSize</i> , without reporting string value.
<i>puiSize</i>	[in, out]	Pointer to a variable that contains size of buffer pointed by <i>pszBuf</i> . If NULL is specified to <i>pszBuf</i> , 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 IEnumEntry type node (TC_NODE_TYPE_ENUM_ENTRY).

This function will return error status if node type of target node is not IEnumEntry type.

Including TeliCamAPI.h is required.

[Example]

Refer to [example](#) of [5.6.6.1 Nd_GetNumOfEnumEntries](#).

5.6.8. ICommand node functions

5.6.8.1. 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,  
    bool18_t        bVerify = true  
);
```

[Parameters]

Parameter		Description
<i>hCam</i>	[in]	Camera-Handle of target camera.
<i>hNode</i>	[in]	Node handle of target command node.
<i>bVerify</i>	[in]	Specify true to check access mode. Specify false to skip checking. This argument is optional. 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++  
  
...  
  
// Get node handle.  
uiStatus = Nd_GetNode(hCam, "TriggerSoftware", &hNode);  
if (uiStatus != CAM_API_STS_SUCCESS)  
    return -1;  
  
// Execute command.  
uiStatus = Nd_CmdExecute(hCam, hNode);  
if (uiStatus != CAM_API_STS_SUCCESS)  
    return -1;  
  
while(1) {  
    // Confirm whether the execution has been accomplished.  
    uiStatus = Nd_GetCmdIsDone(hCam, hNode, &bDone);  
    if (uiStatus != CAM_API_STS_SUCCESS)  
        return -1;  
  
    if (bDone == true)
```

```
        break;  
    Sleep(0);  
}  
...
```

5.6.8.2. 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,  
    bool18_t        *pbDone,  
    bool18_t        bVerify = false  
);
```

[Parameters]

Parameter		Description
<i>hCam</i>	[in]	Camera-Handle of target camera.
<i>hNode</i>	[in]	Node handle of target command node.
<i>pbDone</i>	[out]	Pointer to a variable that receives current execution status . True means that the command has finished. False means that the command has not finished.
<i>bVerify</i>	[in]	Specify true to check access mode. Specify false to skip checking. This argument is optional. If nothing is specified to this argument, false will be selected.

[Return value]

R 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 [5.6.8.1 Nd_CmdExecute](#).

5.6.9. IString node functions

5.6.9.1. 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_SetStrValue (  
    CAM_HANDLE      hCam,  
    CAM_NODE_HANDLE hNode,  
    char             *pszBuf,  
    uint32_t         *puiSize,  
    bool8_t          bVerify = false,  
    bool8_t          bIgnoreCache = false,  
);
```

[Parameters]

Parameter		Description
<i>hCam</i>	[in]	Camera-Handle of target camera.
<i>hNode</i>	[in]	Node handle of target node.
<i>pszBuf</i>	[out]	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.
<i>puiSize</i>	[in, out]	Pointer to 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.
<i>bVerify</i>	[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.
<i>bIgnoreCache</i>	[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[32];
uint32_t      uiSize = 32;

// Get node handle.
uiStatus = Nd_GetNode(hCam, "UserDefinedName", &hNode);
if (uiStatus != CAM_API_STS_SUCCESS)
    return -1;

// Set string.
uiStatus = Nd_SetStrValue(hCam, hNode, "Test");
if (uiStatus != CAM_API_STS_SUCCESS)
    return -1;

// Get string.
uiStatus = Nd_GetStrValue(hCam, hNode, &szBuf[0], &uiSize);
if (uiStatus != CAM_API_STS_SUCCESS)
    return -1;
printf(" Nd_GetStrValue : %s\n", szBuf);

...

```

5.6.9.2. Nd_SetStrValue

This function writes new string value to the node specified 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
<i>hCam</i>	[in]	Camera-Handle of target camera.
<i>hNode</i>	[in]	Node handle of target node.
<i>pszBuf</i>	[in]	Pointer to new string value.
<i>bVerify</i>	[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 [5.6.9.1 Nd_GetStrValue](#).

5.6.10.Chunk functions

5.6.10.1. Chunk_AttachBuffer

Attach the buffer to the GenICam chunk port.

When acquiring chunk data from received payload data using GenICam, it is necessary to attach the buffer storing the payload data to GenICam chunk port.

[Syntax]

```
CAM_API_STATUS  Chunk_AttachBuffer (  
    CAM_STRM_HANDLE  hStrm,  
    void             *pvPayloadBuf,  
    uint32_t         uiPayloadSize  
);
```

[Parameters]

Parameter		Description
<i>hStrm</i>	[in]	Stream-Handle of target stream interface.
<i>pvPayloadBuf</i>	[in]	Pointer to the buffer storing the received payload data.
<i>uiPayloadSize</i>	[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++
<pre>... CAM_NODE_HANDLE hNode = NULL; 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 = Nd_GetNode(hCam, "ChunkFrameID", &hNode); if (uiStatus != CAM_API_STS_SUCCESS) return -1; uiStatus = Nd_GetIntValue(hCam, hNode, &lFrameID); if (uiStatus != CAM_API_STS_SUCCESS)</pre>

```
    return -1;

    // Get ChunkExposureTime
    uiStatus = Nd_GetNode(hCam, "ChunkExposureTime", &hNode);
    if (uiStatus != CAM_API_STS_SUCCESS)
        return -1;

    uiStatus = Nd_GetFloatValue (hCam, hNode, &dExposureTime);
    if (uiStatus != CAM_API_STS_SUCCESS)
        return -1;

    ...
```

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_UYVYY (YUV411Packed)	0x020C001E
ConvYUV422PToBGRA	YUV422_8_UYVY (YUV422Packed)	0x0210001F
ConvYUV444PToBGRA	YUV8_UYV (YUV444Packed)	0x02180020

[Parameters]

Parameter		Description
<i>puiDstBGRA</i>	[out]	Pointer to destination image data that receives converted BGRA data. Memory should be allocated to this beforehand.
<i>pvSrc</i>	[in]	Pointer to source image data.
<i>uiWidth</i>	[in]	Width of source image data in pixel. Width must be multiple of 4.
<i>uiHeight</i>	[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
<i>puiDstBGR</i>	[out]	Pointer to destination image data that receives converted BGR data. Memory should be allocated to this beforehand.
<i>pvSrc</i>	[in]	Pointer to source image data.
<i>uiWidth</i>	[in]	Width of source image data in pixel. Width must be multiple of 4.
<i>uiHeight</i>	[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,  
    bool18_t           bBayreConversion,  
    void               *pvDst,  
    void               *pvSrc,  
    uint32_t           uiWidth,  
    uint32_t           uiHeight  
);
```

[Parameters]

Parameter		Description
<i>eDstFormat</i>	[in]	Output image format.
<i>uiSrcPixelFormat</i>	[in]	PixelFormat of source image data.
<i>bBayerConversion</i>	[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 treated as grayscale image data. If source PixelFormat is not Bayer type, this value is ignored.
<i>pvDst</i>	[out]	Pointer to destination image data that receives Converted image data. Memory should be allocated to this beforehand.
<i>pvSrc</i>	[in]	Pointer to source image data.
<i>uiWidth</i>	[in]	Width of source image data in pixel. Width must be multiple of 4.
<i>uiHeight</i>	[in]	Height of source image data in pixel.

[*DST_FORMAT* Enumeration]

Member	Description
<i>DST_FMT_BGRA32</i>	Convert to BGRA format (32 bits) data.
<i>DST_FMT_BGR24</i>	Convert to BGR format (24 bits) data.

[Return value]

Returns result status. Refer to [5.8 Status code](#).

[Remarks]

Including TeliCamUtil.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 Source 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
<i>uiPixelFormat</i> [in]	PixelFormat.

[Return value]

Returns bit per pixel value. For example, 24 will be returned when *uiPixelFormat* is 'RGB8'.

[Remarks]

Refer to TeliCamPxIFmt.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
<i>uiPixelFormat</i> [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
<i>uiPixelFormat</i> [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 TeliCamUtil.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
<i>uiPixelFormat</i> [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 TeliCamUtil.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  
);
```

[Functions].

Functions	Bitmap Pixel Format	Remarks
SaveBmpARGB	Format32bppArgb	
SaveBmpRGB	Format24bppRgb	
SaveBmpMono	Format8bppIndexed	Uses color pallet for monochrome image.

[Parameters]

Parameter	Description
<i>pvTgt</i> [in]	Pointer to target image data.
<i>uiWidth</i> [in]	Width of target image in pixels.
<i>uiHeight</i> [in]	Height of target image in pixels.
<i>pszPath</i> [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 TeliCamUtil.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
<i>pvDst</i>	[in]	Pointer to destination image data that receives reversed image.
<i>pvSrc</i>	[in]	Pointer to source image data.
<i>uiPixelFormat</i>	[in]	PixelFormat of source image.
<i>uiWidth</i>	[in]	Width of source image in pixel.
<i>uiHeight</i>	[in]	Height of source image in pixel.
<i>bRevX</i>	[in]	Flag for reversing horizontally. If true, horizontally.reversed image will be created.
<i>bRevY</i>	[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 TeliCamUtil.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 unsuccessful.
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 camera 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_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_INVALID_ADDRESS	0x00000801	The specified address 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_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_REQUEST_TIMEOUT	0x00001001	Timeout occurred in requesting stream or event.

CAM_API_STATUS	Value	Description
CAM_API_STS_RESEND_TIMEOUT	0x00001002	The StreamRequests could 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 restricting 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_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_UNSUCCESSFUL	0xFFFFFFFF	The other error occurred.

6. Sample source code

TeliCamSDK provides 8 sample source code projects in the following table for user's reference. More sample sources will be provided in our home page subsequently.

Sample name	Language	UI	Function	Remarks
GrabStreamSimple	VC2005	CUI	Pixel values in the image shown.	Use API in section 5.5 .
GrabStreamSimpleViaNode	VC2005	CUI		Use API in section 5.6 .
GrabStreamSimpleRegBG	VC2005	CUI		Register direct access. (BG)
GrabStreamSimpleRegBU	VC2005	CUI		Register direct access. (BU)
GrabStreamSimpleIIDC2BU	VC2005	CUI		IIDC2 FeatureCSR access. (BU)
GrabEvent	VC2005	CUI	Get "FrameTrigger" event.	Use API in section 5.5 and 5.3.2 .
MultiCamera	VC2005	GUI	Draw images of up to 4 cameras in a form.	Use API in section 5.3.1 .
MultiCameraPrimitive	VC2005	GUI	Draw images of up to 4 cameras in a form.	Use API in section 5.3.2 .

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.

Do the following operation to use sample source code project with development environment newer than Visual studio 2005.

- Execute "Open project / solution" menu and select solution file (*.sln)" in the sample source code project. Source code for VC 2005 will be converted to the newer version 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. GrabStreamSimple*

TeliCamSDK provides 5 types of GrabStreamSimple sample projects, which provide same user interface.

Sample name	API	Target	Remarks
GrabStreamSimple	Section 5.5	General	User can design code without knowledge of GenICam.
GrabStreamSimpleViaNode	Section 5.6	General	This project controls camera via node of GenApi. If a camera feature is not supported by API in section 5.5, use API in section 5.6. Knowledge about GenICam is necessary for designing code.
GrabStreamSimpleRegBG	Section 5.2.5 and 5.2.6	Expert	This project accesses to camera registers directly without using GenICam. (For BG series) Knowledge of GigE Vision standard is necessary for designing code. Refer to this project if camera controlling timing is critical.
GrabStreamSimpleIIDC2BU		Expert	This project accesses to camera registers directly without using GenICam. (For BU series) Knowledge of USB3 Vision standard is necessary for designing code. Knowledge of IIDC2 register map is not necessary for designing code, by copying IIDC2.h to user project. Refer to this project if camera controlling timing is critical.
GrabStreamSimpleRegBU		Expert	This project accesses to camera registers directly without using GenICam. (For BU series) Knowledge of USB3 Vision standard and IIDC2 register map are necessary for designing code. Refer to this project if camera controlling timing is highly critical.

These sample projects contains sample code in the following table.

Contents	Method	file	
Initialize TeliCamAPI.	Initialize()	GrabStreamSimple*.cpp	*1
Set / get camera features.	SetFeature() SetSoftTriggerMode() SetImageFormat() SetExposureTime() SetGain() DispFeature()	GrabStreamSimple*.cpp	*1
Open image stream.	OpenStream()	GrabStreamSimple.cpp	*1
Start image acquisition.	StartAcquisition()	GrabStreamSimple.cpp	*1
Receive image.	ImageHandling()	GrabStreamSimple.cpp	

Contents	Method	file	
Stop image acquisition.	StopAcquisition()	GrabStreamSimple.cpp	*1
Close image stream.	CloseStream()	GrabStreamSimple.cpp	*1
Terminate TeliCamAPI.	Terminate()	GrabStreamSimple.cpp	
Feature name definition.	--	XmlFeatures.h (GrabStreamSimpleViaNode)	
Register map (BG).	--	RegisterMap_BG_Type1.h (GrabStreamRegBG)	
Register map (BU).	--	RegisterMap_BU.h (GrabStreamRegBU, GrabStreamIIDC2BU)	
IIDC2 FeatureCSR structure.	--	IIDC2.h (GrabStreamIIDC2BU)	
The others.	--	EnumStruct.h (GrabStreamIIDC2BU)	

*1: Codes in the methods are different from corresponding methods in other projects.

6.1.1. Operation

GrabStreamSimple is console application.

1) Start GrabStreamSimple.exe.

Command prompt window appears and explanation of the application will be shown.

```
GrabStreamSimple sample application for TeliCamSDK V1.0.0.
Copyrights (C) 2014 TOSHIBA TELI CORPORATION. All rights reserved.
Access camera feature registers using TeliCamApi default functions.
GenApi modules are used inside TeliCamApi.
Press any key to start...
```

2) Press any key once.

Version of dll in TeliCamSDK and information of detected cameras are shown.

```
Press any key to start...
```

```
Initialize() started!
<System information>
TeliU3vDriver.sys version : 1.4.0.1
TeliU3vApi.dll version : 1.3.1.201
TeliU3vCamApi.dll version : 1.3.1.201
TeliGevDriver.sys version : 1.3.0.1
TeliGevApi.dll version : 1.2.0.101
TeliGevCamApi.dll version : 1.2.0.101
TeliCamApi.dll version : 1.0.0.1
2 camera(s) found.
```

In this example, information of USB3 Vision camera BU130C and GigE Vision camera BG030 are shown.

```
<Camera0 information>
Type : USB3 Vision camera
Manufacturer : Toshiba-Teli
Model name : BU130C
Serial number : Sample2
User defined name : test
U3v family name : BU-Series
U3v device version : 2.0.0
U3v manufacturer information : SXGA COLOR
U3v adapter vendor ID : 0x104C
U3v adapter device ID : 0x8241
U3v Adapter default MaxPacketSize : 65536
```

It will take a few seconds to get information of GigE Vision camera.

```
<Camera1 information>
Type : GigE Vision camera
Manufacturer : Toshiba Teli Corporation
Model name : BG030
Serial number : 00000006
User defined name : #6
Gev display name : Toshiba Teli Corporation - BG030 - Rev. A
- 00000006 - [00-06-00-02-00-06]
Gev MAC address : 00-06-00-02-00-06
Gev support IP ILLA : 1
Gev support IP DHCP : 1
Gev Support IP Persistent-IP : 1
Gev current IP ILLA : 1
Gev current IP DHCP : 1
Gev current IP Persistent-IP : 1
Gev IP Address : 169.254.0.100
Gev subnet mask : 255.255.0.0
Gev default gateway : 0.0.0.0
Gev adapter MAC address : 00-26-18-6E-B8-F7
Gev adapter IP address : 169.254.122.99
Gev adapter subnet mask : 255.255.0.0
Gev adapter default gateway : 0.0.0.0
Gev adapter display name : Marvell Yukon 88E8001/8003/8010 PCI Gigabit Ethernet Controller - TeliGevDriver GigE Vision Miniport
Camera opened.
Initialize() successful.
```

It may take some minutes to get camera description file data, if there are cameras which have not been connected to the PC before.

After showing information of cameras, the first detected camera is selected as target camera.

Basic feature settings including software one-shot-trigger mode setting will be executed, and software one-shot-trigger acquisition operation will be started. ,
The selected camera will wait for receiving software-trigger command.

```
SetFeature() started!
SetFeature() successful.

SetTriggerMode() started!
SetTriggerMode() successful.

SetImage() started!
SetImage() successful.

OpenStream() started!
Payload size : 307200[byte]
OpenStream() successful.

StartAcquisition() started!
StartAcquisition() successful.
```

3) Process Loop started. Press '0', '1', or '2' key.

Exposure time, gain and frame-rate settings and key command guide will be shown.

```
ProcessLoop() started!

Exposure time : 12.0000[ms]
Gain          : 0.0000[db]
Frame rate    : 56.9458[fps]
-----
Press '0' key to issue "Software Trigger" and grab a frame.
Press '1' key to set exposure time
Press '2' key to set gain
Press the other key to quit application.
```

If '0' key is pressed, Software-Trigger command is sent to camera, a frame of image will be received, the first 8 pixel values in the image will be shown.

```
Received ImageAcquired event.
ImageBuffer : 06 06 00 07 05 01 07 06 ...
```

If '1' key is pressed, value range of exposure time will be shown.

```
SetExposureTime() started!
Exposure time min. : 0.0300, max. : 16000.0000[ms] >>>
```

Key in new exposure time value and press Enter. New value will be written to camera and current feature values and key guide will be shown again.

```
SetExposureTime() started!
Exposure time min. : 0.0300, max. : 16000.0000[ms] >>> 10
Current exposure time : 10.0000[ms].
SetExposureTime() finished!

Exposure time : 10.0000[ms]
Gain          : 0.0000[db]
Frame rate    : 56.9458[fps]
-----
Press '0' key to issue "Software Trigger" and grab a frame.
Press '1' key to set exposure time
Press '2' key to set gain
Press the other key to quit application.
```

If '2' key is pressed, value range of gain will be shown.

```
SetGain() started!
Gain min. : 0.0000, max. : 18.0218 [db] >>>
```

Key in new gain value and press Enter. New value will be written to camera and current feature values and key guide will be shown again.

```
SetGain() started!
Gain min. : 0.0000, max. : 18.0218 [db] >>> 0.1
Current gain : 0.1077[db].
SetGain() finished!

Exposure time : 10.0000[ms]
Gain          : 0.1077[db]
Frame rate    : 56.9458[fps]
-----
Press '0' key to issue "Software Trigger" and grab a frame.
Press '1' key to set exposure time
Press '2' key to set gain
Press the other key to quit application.
```

4) To terminate application, press keys other than '0', '1', and '2'.

Process loop will be finished and TeliCamAPI will be closed.

```
ProcessLoop() finished!

StopAcquisition() started!
StopAcquisition() successful.
CloseStream() successful.
```

Press any key to close command prompt.

```
Terminate() started!
Terminate() successful.

Press any key to exit...
```

6.2. GrabEvent

GrabEvent is sample project for showing code of acquiring FrameTrigger event. Base of this project is GrabStreamSimple sample project.

This sample uses stream API in section [5.3.2 Low-level API functions](#)

Note that GrabEvent is designed for cameras that support Mono8 PixelFormat. If a camera that does not support Mono8 format is used as target camera of GrabEvent, never forget to turn off power of the camera before starting another camera application.

Contents	Method	file	
Initialize TeliCamAPI.	Initialize()	GrabEvent.cpp	
Set / get camera features.	SetFeature() SetSoftTriggerMode() SetImageFormat() SetExposureTime() SetGain() DispFeature()	GrabEvent.cpp	
Open image stream.	OpenStream()	GrabEvent.cpp	*1
Open camera event.	OpenEvent()	GrabEvent.cpp	*2
Activate camera event.	ActivateEvent()	GrabEvent.cpp	Activate 'FrameTrigger' *2
Start image acquisition.	StartAcquisition()	GrabEvent.cpp	
Receive image.	ImageHandling()	GrabEvent.cpp	Receive 'FrameTrigger' *1
Stop image acquisition.	StopAcquisition()	GrabEvent.cpp	
Inactivate camera event.	InactivateEvent()	GrabEvent.cpp	*2
Close camera event.	CloseEvent()	GrabEvent.cpp	*2
Close image stream.	CloseStream()	GrabEvent.cpp	*1
Terminate TeliCamAPI.	Terminate()	GrabEvent.cpp	

*1: Codes are modified from that of GrabStreamSimple.

*2: This method does not exist in GrabStreamSimple.

6.2.1. Operation

Operation of GrabEvent is almost same as that of GrabStreamSimple.

When '0' key is pressed, GrabEvent will wait for 'FrameTrigger' event and show timestamp of the event, before receiving a frame of image

```
Exposure time : 10.0000[ms]
Gain          : 0.1077[db]
Frame rate    : 56.9458[fps]
-----
Press '0' key to issue "Software Trigger" and grab a frame.
Press '1' key to set exposure time
Press '2' key to set gain
Press the other key to quit application.
-----
Received FrameTrigger eventTimestamp = 4359602087944
Average picture level = 34.
Received FrameTrigger eventTimestamp = 4361509143080
Average picture level = 34.
Received FrameTrigger eventTimestamp = 4361828315400
Average picture level = 34.
```

6.3. MultiCamera*

TeliCamSDK provides 2 types of MultiCamera sample projects, which provide same user interface.

MultiCamera sample project uses high layer level API shown in section 5.5.

On the other hand, MultiCameraPrimitive uses low layer level API shown in section 5.6.

You will find that code of MultiCamera is simpler than MultiCameraPrimitive. We recommend you to use high layer level API, except your system requires special image acquisition sequence.

The following table shows codes contained in MultiCamera* project.

Contents	Method	file	
Initialize TeliCamAPI.	OnInitDialog()	MultiCamera*Dlg.cpp	
Enumerate connected cameras, open camera, and get information.	Initialize()	MultiCamera*Dlg.cpp	
Set / get camera features.	GetCameraParameter() SetFreeRunningMode()	MultiCamera*Dlg.cpp	
Open image stream.	OpenStream()	MultiCamera*Dlg.cpp	*1
Start image acquisition.	StartAcquisition()	MultiCamera*Dlg.cpp	*1
Receive image from TeliCamAPI.	CallBackImageAcquired()	MultiCameraDlg.cpp	
	ImgRcvThread()	MulltiCameraPrimitiveDlg.cpp	
Updating image in main dialog.	ImgUpdThread()	MultiCamera*Dlg.cpp	*1
	OnMessageLiveImage()		
Stop image acquisition.	StopAcquisition()	MultiCamera*Dlg.cpp	
Send SoftwareTrigger command.	SendSoftTrigger()	MultiCamera*Dlg.cpp	
Close image stream.	CloseStream()	MultiCamera*Dlg.cpp	*1
Close cameras.	CloseAllCamera()	MultiCamera*Dlg.cpp	
Terminate TeliCamAPI.	OnDestroy()	MultiCamera*Dlg.cpp	
Select image converter function. (convert camera raw image to BGRA format)	SelectBGRAConverter()	MultiCamera*Dlg.cpp	
Definition of original state value 'TriggerType'.	TriggerType.h	MultiCamera*Dlg.cpp	
Gather information of camera feature for setting GUI controls.	CInfoFloat::SetValue()	InfoNode.h	
	CInfoEnum::SetValue()		
Set GUI control data.	ShowCamParam()	ParamDialog.cpp	
	CInfoFloat::SetScrlBar_CEd it()	InfoNode.h	
	CInfoEnum::SetComboBox()		
Feature name definition.	--	XmlFeatures.h	

*1: Codes in the methods are different from corresponding methods in other project.

6.3.1. Operation

MultiCamera shows images acquired from up to 4 cameras.

Default 'TriggerType' setting is 'Internal (Free_Running)'. The other 'TriggerType' are also available using Parameter dialog. When 'Software' or 'Bulk_Trigger_Soft' is selected, use software trigger buttons for sending software-trigger command to all cameras or a camera.

6.3.1.1. Main window

- 1) Press [Update Camera List] button.

When MultiCamera application is started, the window shown in the right will appear.

Press [Update Camera List] button after checking that camera is properly connected to the PC.

Application will search cameras connected to the PC and show their information on the window.

It will take a few seconds to get the information of GigE Vision camera.

It may take some minutes to get camera description file data, if there are cameras which have not been connected to the PC before.

[Show Parameter] button and [Start Acquisition] button will be enabled when searching camera is completed.

- 2) Press [Start Acquisition] or [Show Parameter] button.

If [Start Acquisition] button is pressed, image acquisition of connected camera will start and the received images will be shown in main dialog. [Start Acquisition] button will be disabled and [Stop Acquisition] button will be enabled.

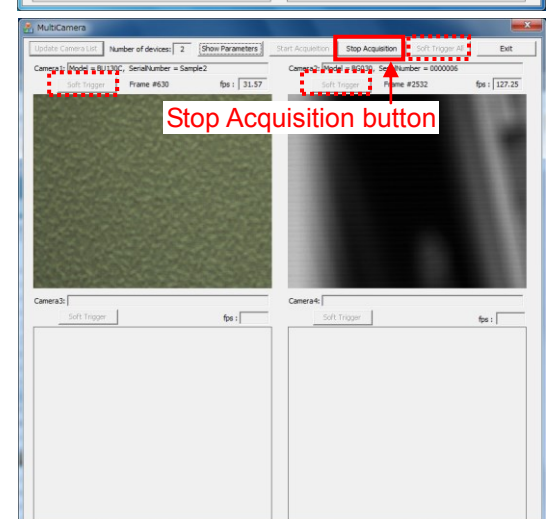
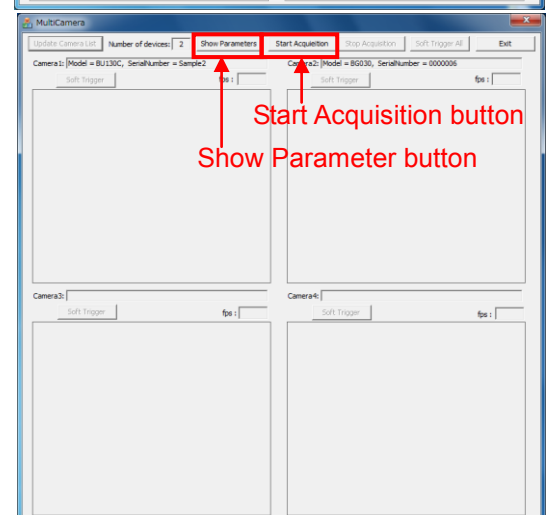
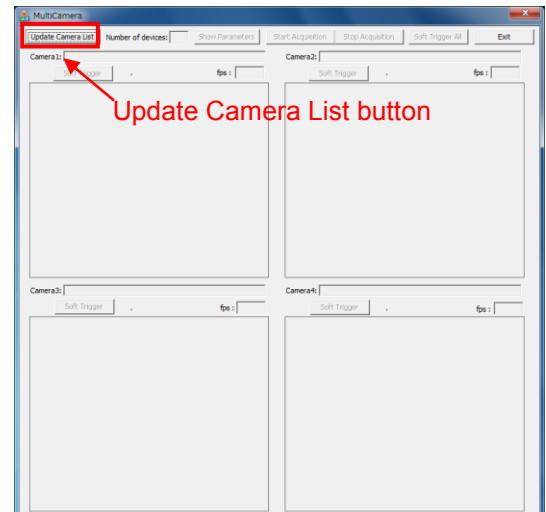
By pressing [Stop Acquisition] button, image acquisition will be stopped.

If at least one camera uses "Bulk_Trigger_Soft" or "Software" TriggerType, [Soft Trigger] button of the camera and [Soft Trigger All] button will be enabled.

If [Soft Trigger] button or [Soft Trigger All] button is pressed, SoftwareTrigger command will be sent to the camera.

MultiCamera shows Frame index sent from camera at the top of image. (BU series starts from 0, BG series starts from 1)

MultiCameraPrimitive shows Frame index counted in MultiCameraPrimitive application.



6.3.1.2. Parameter window

If [Show Parameter] button is pressed, Parameter window will appear. User can edit feature of the target camera using this window.

Scroll bar of Gain works as linear scale editor, and scroll bar of Exposure time works as logarithm scale editor.

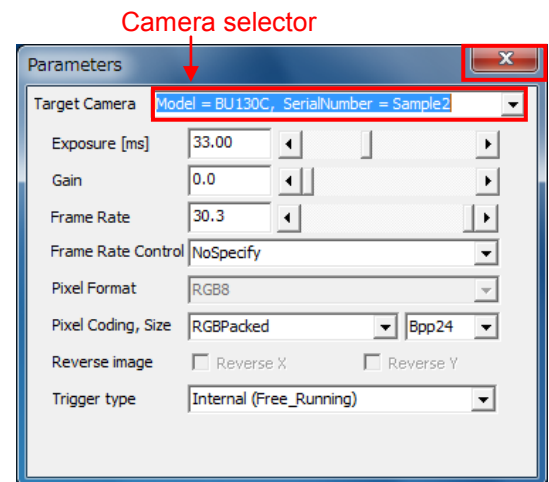
Target camera can be changed using Camera selector.

If USB3 Vision camera is selected, PixelFormat combo-box will be disabled, PixelCoding and PixelSize combo-box will be enabled.

If GigE Vision camera is selected, PixelFormat combo-box will be enabled, PixelCoding and PixelSize combo-box will be disabled, in this sample code.

(It is possible to write PixelFormat value to USB3 Vision camera using SetCamPixelFormat API, while it is not possible to write PixelFormat value to USB3 Vision camera using Nd_SetEnumIntValue API or Nd_SetEnumStrValue API.)

By pressing [x] button, Parameter window will be closed.



7. Others

7.1. Revision History

Date	Version	Description
2014/09/11	1.0.0	Created the initial version
2014/10/17	1.0.1	<ul style="list-style-type: none">● Added the status code.● Changed the name of function. (Cam_PortReset → Cam_ResetPort)
2014/11/14	1.0.2	<ul style="list-style-type: none">● Changed default uiApiBufferCount parameter value of Strm_OpenSimple to 8. (Previous default value was 5.)● Appended section 5.5.25 UserDefinedName(DeviceUserID), corresponding to supplement of these functions to TeliCamAPI.
2015/02/25	1.0.3	<ul style="list-style-type: none">● Replaced body of section 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.<ul style="list-style-type: none">✓ Opening a camera that the other application is using is made possible.✓ Installation of GenApi becomes not necessary when false is specified to <i>bUseGenICam</i> 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	<ul style="list-style-type: none">● Added the Windows 8.1 description
2015/10/21	1.0.5	<ul style="list-style-type: none">● Added the camera connection possibility number description
2016/07/12	2.0.0	<ul style="list-style-type: none">● Added the Windows 10 description● Deleted the Windows XP & Windows Vista description● Supports FrameBurst function.● Appended function (SetCamLineModeAll())
2016/12/16	2.0.1	<ul style="list-style-type: none">● Added functions related to Chunk feature● Added functions related to UserSetControl
2017/06/13	2.0.2	<ul style="list-style-type: none">● Added Strm_Abort() function.● Added some Status code.● Modified the description of each item.
2017/09/05	2.0.3	<ul style="list-style-type: none">● Modified setting range of argument uiApiBufferCount of Strm_OpenSimple(). (Minimum value : 3 → 1, Maximum value : 30 → 128)

7.2. 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 TeliCamSDK installation folder/Documents/License folder

7.3. License

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 a 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.

TeliCamSDK and Viewer use the library that the third party retains the copyright. Refer to TeliCamSDK installation folder/Documents/License folder for license.

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 the our customer support below :

Imaging Solution Engineering section
Imaging & Communication Solution Engineering Division
4-7-1, Asahigaoka, Hino,
Tokyo 191-0065, Japan
Mail : TELI-EXT-technical-support@toshiba-teli.co.jp