

# TeliCamAPI for Linux

## ライブラリマニュアル

Version 1.0.0 (2016/01/26)

# 東芝テリー株式会社

改善の為予告なく変更することがありますので、最新の取扱説明書にて機能をご確認ください

---

# 目次

1. はじめに .....	4
2. 構成 .....	5
3. システム要件 .....	6
4. ライブラリ .....	7
4.1. 使用方法 .....	8
4.1.1. TeliCamAPI の初期化と終了 .....	8
4.1.2. カメラの検索 .....	8
4.1.3. カメラ オープン/クローズ .....	8
4.1.4. カメラの制御、情報取得 .....	8
4.1.5. ストリームコントロール .....	9
4.1.6. カメライベント（メッセージ）コントロール .....	18
4.1.7. カメラのアクセスモード（制御チャネル特権） .....	24
4.1.8. ハートビート処理 .....	24
4.2. SDK のインストール .....	25
4.3. SDK のアンインストール .....	25
4.4. 開発環境の設定 .....	25
4.5. パフォーマンスのチューニング .....	26
5. ライブラリ関数 .....	27
5.1. システム関数 .....	27
5.1.1. Sys_Initialize .....	27
5.1.2. Sys_Terminate .....	28
5.1.3. Sys_GetInformation .....	29
5.1.4. Sys_GetNumOfCameras .....	31
5.1.5. Sys_CreateSignal .....	32
5.1.6. Sys_CloseSignal .....	33
5.1.7. Sys_WaitForSignal .....	34
5.1.8. Sys_ResetSignal .....	35
5.2. カメラ関数 .....	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 .....	52
5.2.8. Cam_GetHeartbeat .....	54
5.2.9. Cam_SetHeartbeat .....	55
5.3. カメラストリーム関数 .....	56
5.3.1. 高水準関数 .....	56
5.3.2. 低水準関数 .....	75
5.3.3. 共通関数 .....	88
5.4. カメライベント（メッセージ）通知関数 .....	92
5.4.1. 高水準関数 .....	92
5.4.2. 低水準関数 .....	99
5.4.3. 共通関数 .....	109
5.5. カメラ制御関数 .....	110
5.5.1. ImageFormatControl .....	111
5.5.2. Scalable .....	113
5.5.3. Binning .....	129
5.5.4. Decimation .....	135
5.5.5. Reverse .....	141
5.5.6. PixelFormat .....	145
5.5.7. TestPattern .....	147
5.5.8. AcquisitionControl .....	149
5.5.9. ImageBuffer .....	159
5.5.10. TriggerControl .....	163

5.5.11. ExposureTime .....	177
5.5.12. DigitalLoControl .....	183
5.5.13. TimerConrtol .....	191
5.5.14. Gain .....	199
5.5.15. BlackLevel .....	204
5.5.16. Gamma .....	207
5.5.17. WhiteBalance .....	210
5.5.18. Hue .....	215
5.5.19. Saturation .....	218
5.5.20. Sharpness .....	221
5.5.21. ColorCorrectionMatrix .....	224
5.5.22. LUTControl .....	228
5.5.23. UserSetControl .....	232
5.5.24. SequentialShutterControl .....	235
5.5.25. UserDefinedName (DeviceUserID) .....	244
5.6. GenlCam 関数 .....	246
5.6.1. Node 系関数 .....	246
5.6.2. Node 系 ICategory 型 関数 .....	259
5.6.3. Node 系 IInteger 型 関数 .....	262
5.6.4. Node 系 IFloat 型 関数 .....	268
5.6.5. Node 系 IBoolean 型 関数 .....	278
5.6.6. Node 系 IEnumeration 型 関数 .....	281
5.6.7. Node 系 IEnumEntry 型 関数 .....	290
5.6.8. Node 系 ICommand 型 関数 .....	292
5.6.9. Node 系 IString 型 関数 .....	294
5.7. ユーティリティ関数 .....	297
5.7.1. 画像フォーマット変換関数 .....	297
5.7.2. その他ユーティリティ .....	304
5.8. その他の関数 .....	308
5.8.1. GetCamIndexFromCamHandle .....	308
5.8.2. GetCamTypeFromCamHandle .....	309
5.8.3. GetCamTLParamsLocked .....	310
5.8.4. Misc_GetLastGenlCamError .....	311
5.9. ステータスコード .....	312
6. サンプル ソースコード .....	314
6.1. コンソール サンプル .....	315
6.2. Qt サンプル .....	315
7. その他 .....	316
7.1. 改定履歴 .....	316
7.2. 免責事項 .....	316
7.3. ライセンス .....	316
7.4. お問い合わせ .....	317

---

# 1. はじめに

TeliCamSDK は、東芝テリー製 USB3 および GigE Vision デジタルカメラシリーズを PC から制御するためのソフトウェア開発キットです。

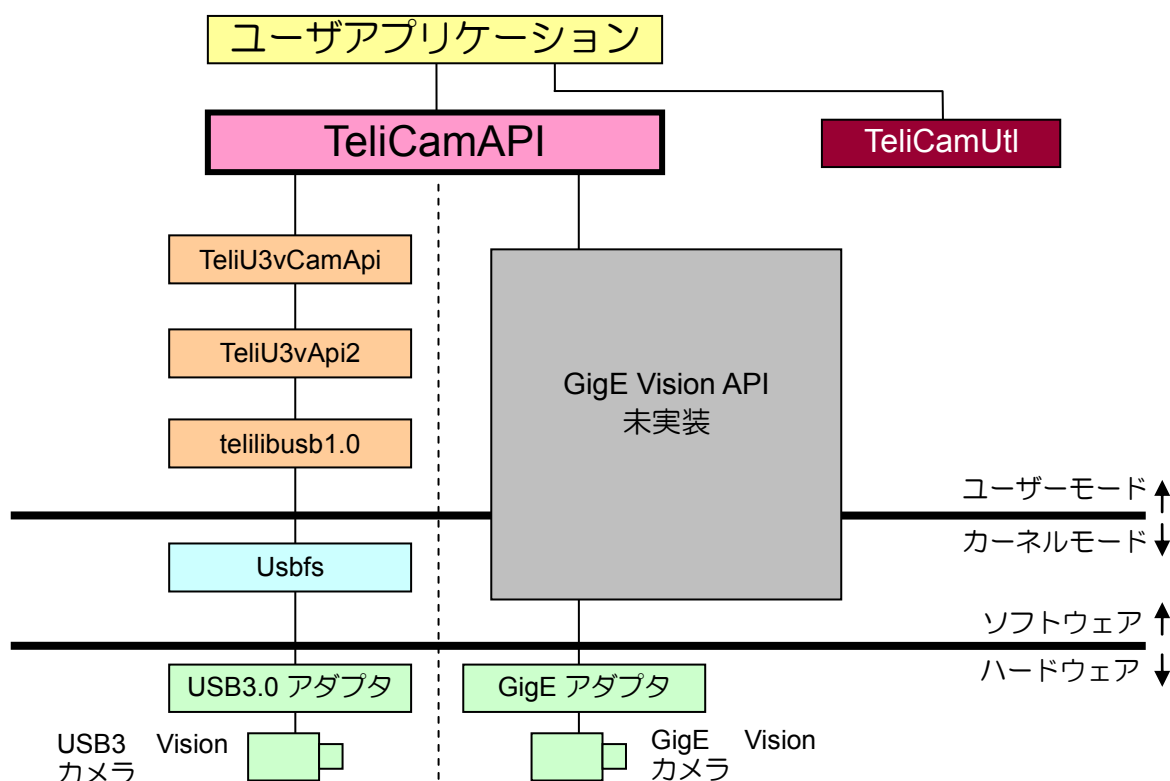
本ドキュメントは、TeliCamSDK パッケージの C++用プログラミングインタフェースである TeliCamAPI の使用方法について記載しています。 TeliCamAPI を使用することにより、カメラのインターフェースを意識することなく簡単にアプリケーションを作成することができます。

本ドキュメントの読者として、カメラを使用したシステムを構築するソフトウェア技術者を想定しています。

*現在の SDK のバージョンでは、USB3 Vision カメラしかサポートしておりません。*

## 2. 構成

TeliCamSDK のソフトウェア構成は以下の通りです。



	内 容
TeliCamApi (libTeliCamApi*.so)	ネイティブアプリケーション開発用関数ライブラリ
TeliU3vCamApi (libTeliU3vCamApi*.so)	USB3 Vision カメラ 用拡張関数ライブラリ
TeliU3vApi2 (libTeliU3vApi2*.so)	USB3 Vision カメラ用基本関数ライブラリ
TeliCamUtl (libTeliCamUtl*.so)	画像ハンドリングユーティリティ関数ライブラリ
teliusb1.0 (libteliusb1.0.so)	汎用 USB 関数ライブラリ

高水準 API の TeliCamAPI と TeliCamUtl を使用することにより、カメラインタフェースを意識することなく簡単にアプリケーションを作成することができます。

TeliCamAPI は、最大 64 台までカメラを認識することができます。

ただし、GigE Vision カメラ用ネットワークアダプタの認識枚数は最大 16 枚です。 また、1 アダプタあたりのカメラ認識台数は最大 16 台です。

なお、TeliCamAPI は、GigE Vision カメラのマルチキャストには対応しておりません。

### 3. システム要件

TeliCamApi for Linux は以下の OS で動作します。

Ubuntu 14.04 LTS amd64 ※1, ※2	For 64-bit Intel/AMD (x86_64)
Debian 8.1.0 amd64 ( with the GNOME desktop environment )	For 64-bit Intel/AMD (x86_64)

※1 Ubuntu 14.04 は、XHCI ドライバに関わる問題があります。

ストリームのスタート/ストップを連続的に繰り返すと、ストリームインターフェースの動作が停止することがあります。 この問題を回避するために、Strm\_Stop 関数の内部処理を Windows 版 TeliCamSDK とは変更して対応しています。(AcquisitionStop コマンドの代わりに AcquisitionAbort コマンドを実行)

**Ubuntu 14.04.1 またはそれ以上のバージョンを使用することを推奨します。** または、カーネルのバージョンアップにより問題を回避することができます。

※2 Ubuntu 14.04 は、サスペンドおよびハイバーネーション機能に問題があります。

サスペンドまたはハイバーネーション機能を実行すると、USB ポートが動作しなくなる場合があります。

**サスペンドおよびハイバーネーション機能を使用しないことを推奨します。**

TeliCamApi for Linux を使用するために、次の環境が必要です。

ただし、あらゆる PC 環境での動作を保障するものではありません。

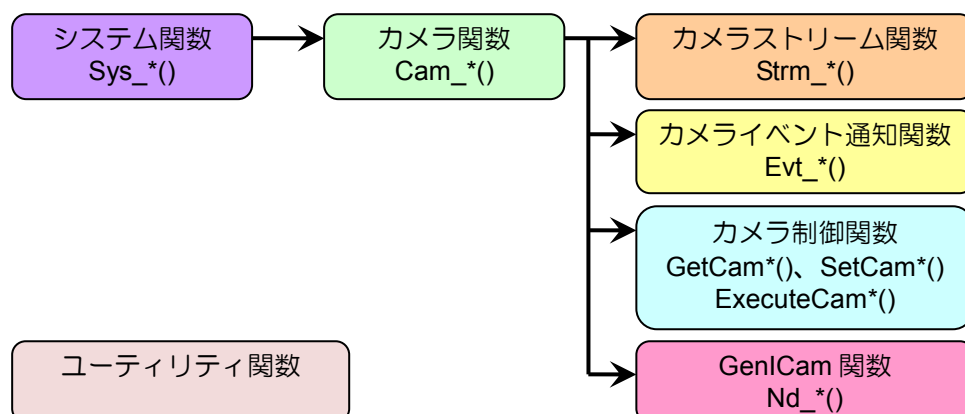
推奨 USB3.0 アダプタ	ルネサス エレクトロニクス製 USB3.0 ホストコントローラ搭載 アダプタ。
対応カメラ	東芝テリー製 USB3 Vision デジタルカメラ

TeliCamSDK のインストールおよびサンプル ソースコードをコンパイルするためには、PC に以下のソフトウェアがインストールされている必要があります。

sudo	ユーザーがスーパーユーザーの特権レベルでプログラムを実行することを許可するプログラム
GNU make	プログラムのビルド作業を自動化するツール
GNU gcc/g++	C/C++ コンパイラ
Qt	クロスプラットフォーム アプリケーションフレームワーク (GUI アプリケーションで使用)

## 4. ライブラリ

TeliCamAPI ライブラリでは以下のグループの関数を提供しています。



グループ	内 容
システム関数	TeliCamAPI システム制御用の関数群です
カメラ関数	カメラオープン、クローズ、レジスタ読み書きなどの基本的な制御を行うための関数群です。
カメラストリーム関数	画像ストリームインターフェース制御用の関数群です。 少ないコード記述量で画像が取得できる高水準ストリーム関数群と、ユーザアプリケーションに適した処理構成をフレキシブルに組める低水準ストリーム関数群の2種類の関数群を提供しています。
カメライベント通知関数	カメライベント通知用の関数群です。 少ないコード記述量でカメライベント情報が取得できる高水準イベント関数群と、ユーザアプリケーションに適した処理構成をフレキシブルに組める低水準イベント関数群の2種類の関数群を提供しています。
カメラ制御関数	レジスタアドレスを意識することなくカメラの各機能の制御ができる関数群です。
GenApi 関数	機能名を指定してカメラの各機能の制御ができる関数群です。 カメラ制御関数がサポートしない機能・情報の制御・取得も行うことができます。
ユーティリティ関数	画像フォーマット変換、保存などのユーティリティ関数群です。

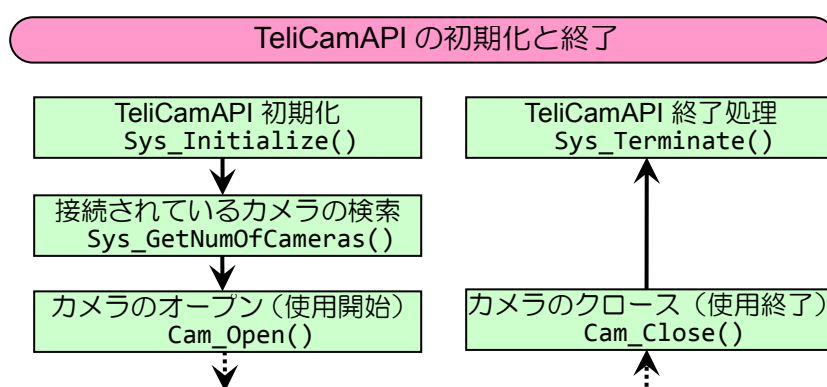
## 4.1. 使用方法

本章では TeliCamAPI の初期化・終了処理、カメラコントロール、ストリームデータ取得、カメライベント（メッセージ）データ取得の基本的な処理の流れを説明します。

### 4.1.1. TeliCamAPI の初期化と終了

ユーザアプリケーションは、TeliCamAPI の関数を使用する前に、TeliCamAPI システム初期化のために [Sys\\_Initialize\(\)](#) を一度だけ実行してください。また、TeliCamAPI の使用終了後に、TeliCamAPI システムで使用しているリソースを解放するために [Sys\\_Terminate\(\)](#) を実行してください。

カメラの使用開始から終了までの流れは、下図の通りです。



### 4.1.2. カメラの検索

カメラをオープンする前に必ず [Sys\\_GetNumOfCameras\(\)](#) を実行してください。TeliCamAPI はカメラをオープンするときに、TeliCamAPI 内部のカメラリストに保存されているカメラの情報を使用します。[Sys\\_GetNumOfCameras\(\)](#) は TeliCamAPI 内部にカメラリストを作成し、使用可能なカメラの情報を記録する動きをします。

[Cam\\_GetInformation\(\)](#) を使用するとカメラをオープンする前にカメラの情報が取得できます。

### 4.1.3. カメラ オープン/クローズ

[Cam\\_Open\(\)](#) を実行すると、指定したカメラのカメラハンドルを取得することができます。カメラを制御するときにはこのハンドルを使用してください。[Cam\\_OpenFromInfo\(\)](#) を使用すると製造番号、[UserDefinedName](#)などを指定して常に同じカメラをオープンすることができます。

他のアプリケーションが使用しているカメラもオープンすることができます。但し、複数のアプリケーションが同時にストリーム、カメライベントを使用することはできません。

カメラの使用が終了したら、必ず [Cam\\_Close\(\)](#) をコールしてリソースの解放をしてください。

### 4.1.4. カメラの制御、情報取得

TeliCamAPI は3種類のカメラの制御方法・情報取得方法を提供しています。

#### ・カメラ制御関数を使用したアクセス

この関数を使用すると、カメラのインターフェース、モデル、レジスタアドレスを気にせず、カメラの機能設定・情報取得ができます。

カメラに実装されていない機能の関数を実行するとエラーが戻ります。カメラに実装されている機能はカメラの取扱説明書で確認してください。



- **GenICam 関数**を使用したアクセス

この関数を使用すると、GenICam Standard Feature Name Convention で定められたフィーチャ名および弊社固有のフィーチャ名を指定してカメラの機能設定・情報取得を行うことができます。

[カメラ制御関数](#)が提供していない機能・情報へのアクセスもこの関数で実行可能です。

弊社のカメラで使用できる代表的なフィーチャ名は、ヘッダファイル"XMLFeatures.h"で文字列定数として宣言していますのでご使用ください。

GenICam に関する詳細情報は <http://www.genicam.org> をご覧ください。

- **レジスタアクセス関数**を使用したアクセス

[Cam\\_ReadReg\(\)](#)、[Cam\\_WriteReg\(\)](#) を使用し、レジスタアドレスを指定してアクセスする方法です。

ヘッダファイル RegisterMap\_BU.h に BU シリーズカメラのレジスタマップと代表的なレジスタの値の宣言が、RegisterMap\_BG\_Type1.h には BG シリーズカメラのレジスタマップと代表的なレジスタの値の宣言が記述されていますのでご利用ください。

#### 4.1.5. ストリームコントロール

TeliCamAPI は、高水準関数、低水準関数の2種類の画像情報を取得する関数群を提供しています。

高水準関数は、画像受信処理の大半を TeliCamAPI 内部で実行することによりユーザコードの記述量を減らした関数です。簡単に画像データを取得するコードが記述できます。

低水準関数は、画像ストリームデータの受信処理をユーザに組んでいただくことを前提とした関数群です。ユーザコードの記述量は増えますが、任意のシーケンスで受信処理をすることができ、カメラのパフォーマンスを最大限に活用し、特殊な受信処理の実装も可能にした関数群です。

##### 4.1.5.1. 高水準関数を使用したストリームデータ（画像データ）の取得

高水準関数群では、カメラから受信した画像ストリームデータは自動的に TeliCamAPI 内部に作成したストリームリクエストリングバッファに一時保存されます。ユーザアプリケーションに対しては、ストリームリクエストリングバッファ内の画像にアクセスするための関数が提供されます。

TeliCamAPI は画像ストリーム受信完了をアプリケーションに通知する方法を2種類提供しています。

- **通知シグナルを使用した通知**

ストリームオープン時に引数として指定したシグナルオブジェクトを使用して受信完了をユーザアプリケーションに通知する方法です。

ユーザアプリケーションは [Sys\\_WaitForSignal\(\)](#) を使用して受信完了を待ち、受信が完了すると、[Strm\\_ReadCurrentImage\(\)](#)などの高水準関数で画像を取得する流れです。

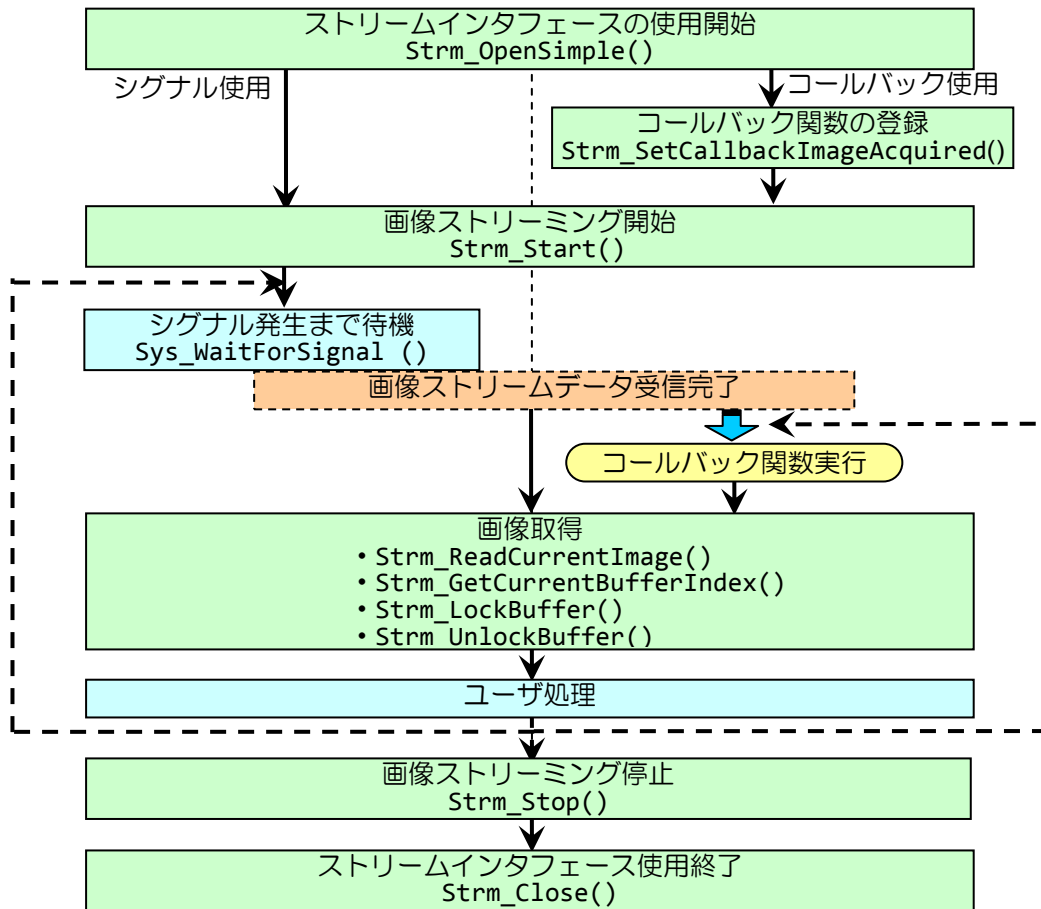
- **コールバック関数を使用した通知**

画像ストリーム受信完了時に [Strm\\_SetCallbackImageAcquired\(\)](#)関数で登録されたコールバック関数をコールすることにより受信完了をユーザアプリケーションに通知する方法です。

ユーザアプリケーションはコールバックの引数で画像を受け取ることができます。

[Strm\\_LockBuffer\(\)](#)関数を使用すればストリームリクエスト リングバッファ内の過去の画像データを取り出すこともできます。バルクトリガやシーケンシャルシャッタを使用して、複数枚のストリームデータ（画像データ）を取得する場合にもこの方法を使用することができます。

アプリケーションが行う一般的なシーケンスは下図の通りです。



#### 4.1.5.1.1. ストリームインターフェースのオープン

[Strm\\_OpenSimple\(\)](#) 高水準関数を実行するとストリームインターフェースが使用可能になり、オープンされたストリームのストリームハンドルがユーザアプリケーションに返されます。

[Strm\\_OpenSimple\(\)](#) は、内部的には、複数のストリームリクエスト構造体と、ストリームリクエスト構造体を保管するストリームリクエストリングバッファを作成して画像受信に備えます。ストリームリクエストとは、画像データやその他情報を格納するための構造体データです。

#### 4.1.5.1.2. ストリームデータ（画像データ）受信 コールバック関数

TeliCamAPIはイベントオブジェクトを使用して画像受信完了をユーザアプリケーションに通知した後、[Strm\\_SetCallbackImageAcquired\(\)](#)で設定されたコールバック関数を実行します。

コールバック関数を実行している間は、対象のストリームリクエストリングバッファの領域はロックされています。ロック中に受信した新しいストリームデータをストリームリクエストリングバッファに格納するとき、その領域がユーザアプリケーションによってロックされているとTeliCamAPIはバッファビジーエラーを発生させ、新しいストリームデータを破棄します。コールバック関数で行う処理は、極力短くしてください。

なお、[Strm\\_SetCallbackBufferBusy\(\)](#)を使用してコールバック関数が登録されている場合、バッファビジーエラーが発生すると指定された関数がコールバックされます。

#### 4.1.5.1.3. ストリーミング開始

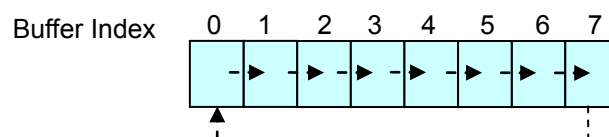
[Strm\\_Start\(\)](#)を実行するとカメラにストリーミング開始が指示され、ストリーミングが開始されます。

TeliCamAPIは、カメラから受信した画像ストリームデータのストリームリクエスト構造体への保存、ストリームリクエストリングバッファ内のストリームリクエスト構造体更新などの受信作業をバックグラウンドで実行します。なお、TeliCamAPIのストリームリクエストリングバッファは、カメラ（内部）のイメージバッファとは異なります。

ストリームリクエストリングバッファのサイズは指定可能で、デフォルトで8個に設定されます。高速で連続的に画像を取得する場合は、ユーザアプリケーションの受信処理が画像受信スピードに追いつかず、バッファビジーエラーが発生することがあります。バッファビジーエラーが発生する場合は、ストリームリクエストリングバッファのサイズを大きくしてください。

TeliCamAPIは、ストリームリクエストリングバッファのインデックス0の領域から順番に受信した画像を格納していきます。最後のインデックスに到達すると、次のストリームデータはインデックス0の領域に戻って格納されます。このとき、すでに格納されていた以前のデータは破棄されます。

ストリームリクエストリングバッファ



TeliCamAPIは新しい画像ストリームデータをストリームリクエストリングバッファに保存したとき、[Strm\\_OpenSimple\(\)](#)で引数として指定されたシグナルオブジェクトをシグナル状態にセットすることにより、アプリケーションに画像を受信したことを通知します。

画像取得の方法は3種類あります。

- **Strm ReadCurrentImage()**使用

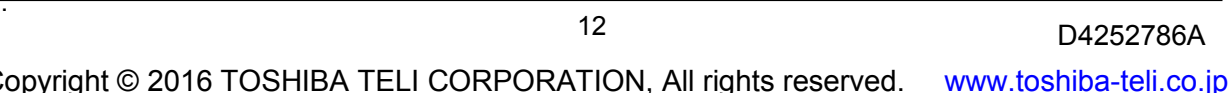
[Strm\\_ReadCurrentImage\(\)](#)を実行すると、ストリームリクエストリングバッファに格納された最新のストリームデータ（画像データ）が指定したメモリにコピーされます。

ユーザアプリケーションがストリームリクエストバッファ内の任意の画像にアクセスする際には、TeliCamAPIが画像を入れ替えないようにあらかじめその画像の領域をロックしておく必要があります。他の2方法の場合はTeliCamAPIがロックの処理を実行するのでユーザアプリケーションはロック処理不要です。

Strm\_GetCurrentBufferIndex() を使用すると最新の画像ストリームデータが格納されたストリームリクエストリングバッファのインデックスが取得できます。

- ・コールバック使用。

カメライメージバッファ転送モードが選択されている場合は、[ExecuteCamImageBufferRead\(\)](#) をコールするたびにカメラから画像が転送されます。カメラのイメージバッファが空の時は、カメラのイメージバッファに画像が入り次第転送されます。



---

#### 4.1.5.1.5. ストリーミング停止

[Strm\\_Stop\(\)](#)を実行するとカメラにストリーミング停止が指示され、その時点で撮像・転送中の画像の処理が完了した後ストリーミングが停止されます。

#### 4.1.5.1.6. ストリームインターフェースのクローズ

ストリームインターフェースの使用を終了するときは、[Strm\\_Close\(\)](#)をコールしてください。

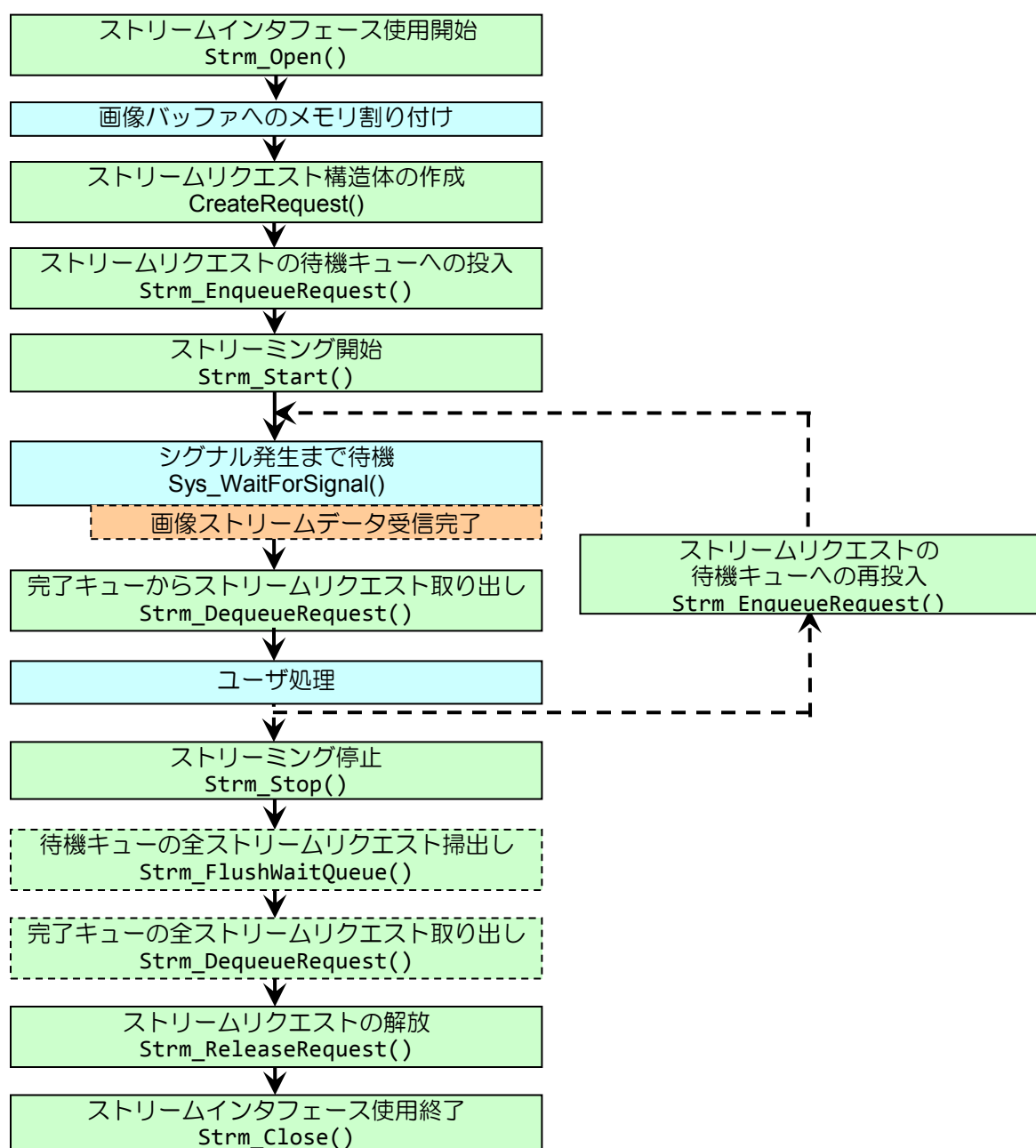
#### 4.1.5.2. 低水準関数を使用したストリームデータ（画像データ）の取得

低水準関数は、画像ストリームデータの受信処理をユーザに組んでいただくことを前提とした関数群です。ユーザコードの記述量は増えますが、任意のシーケンスで受信処理をすることができ、カメラのパフォーマンスを最大限に活用し、特殊な受信処理の実装も可能にした関数群です。

低水準関数では、受信画像と付随情報を記録するストリームリクエスト構造体を複数個作成し、TeliCamAPI 内部に作成されているストリーム受信待機キューとストリーム受信完了キューに対してストリームリクエスト構造体を投入、取り出しすることにより受信画像を取得することができます。

TeliCamAPI はカメラから受信した画像をストリーム受信待機キュー内のストリームリクエスト構造体に書き込んでストリーム受信完了キューへ移す処理をバックグラウンドで実行します。

アプリケーションが行う一般的なシーケンスは下図の通りです。



#### 4.1.5.2.1. ストリームインターフェースのオープン

[Strm\\_Open\(\)](#) 低水準関数を実行するとストリームインターフェースが使用可能になり、オープンされたストリームのストリームハンドルがユーザアプリケーションに返されます。[Strm\\_Open\(\)](#)実行時に画像受信完了シグナルを受領するためのシグナルオブジェクトを引数で指定してください。

#### 4.1.5.2.2. 画像バッファの確保

TeliCamAPI はストリームリクエストを作成する関数は提供していますが、ストリームリクエストのメンバとする画像バッファはユーザアプリケーションがメモリ割り付けしておく必要があります。

連続的取込み中のアプリケーションの処理負荷が最も重いタイミングでもとりこぼしなくストリームを受信できるよう、余裕を持った数のストリームリクエストおよび画像バッファを確保してください。

ストリームリクエスト構造体が解放されるまでは、画像データ用バッファは解放しないでください。画像データ用バッファを解放するときは以下の手順を守ってください。

1. ストリーム受信待機キュー内のすべてのストリームリクエストをストリーム受信完了キューに移動させるために[Strm\\_FlushWaitQueue\(\)](#)を実行。
2. ストリーム受信待機キューが空になるまで[Strm\\_DequeueRequest\(\)](#)を実行して、ストリーム受信完了キューから全てのストリームリクエストを取り出し。
3. [Strm\\_ReleaseRequest\(\)](#)を実行して、ストリームリクエストを解放。
4. 画像データ用バッファを解放。

#### 4.1.5.2.3. ストリームリクエストの作成

メモリ割り付け済の画像バッファを引数に指定して [Strm\\_CreateRequest\(\)](#)関数を実行すると画像データ受信用のストリームリクエストが1個作成され、そのハンドルが引数に戻されます。

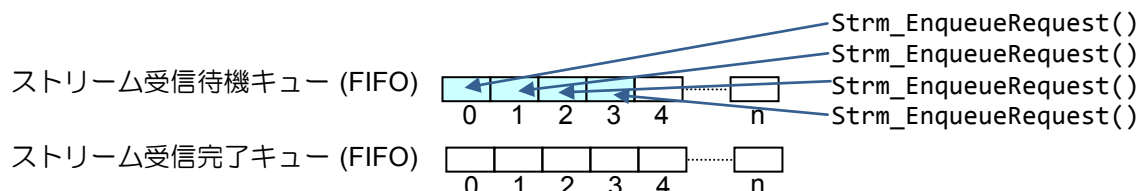
ストリームリクエストは、その画像バッファのサイズ変更、バッファの差し替えをすることはできません。画像データのサイズが変更された場合は、ストリームリクエストを解放して新たにストリームリクエストを作成しなおしてください。

アプリケーション終了時は TeliCamAPI を終了するまでにすべてのストリームリクエストを解放してください。

#### 4.1.5.2.4. ストリームリクエストを待機キューに追加

[Strm\\_EnqueueRequest\(\)](#)を実行して、ストリームリクエストをストリーム受信待機キューに投入してください。ストリーム受信待機キューにストリームリクエストが入っていると TeliCamAPI は画像ストリームデータを受信できるようになります。

例：ストリームリクエストを4つ作成し、ストリーム受信待機キューに追加



#### 4.1.5.2.5. ストリーミング開始

[Strm\\_Start\(\)](#)を実行すると、カメラにストリーミング開始が指示され、ストリーミングが開始されます。

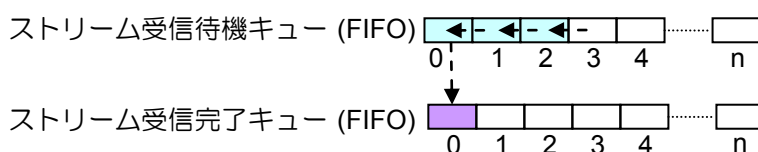


#### 4.1.5.2.6. ストリームデータ（画像データ）の受信

カメラから画像ストリームデータを受信すると、TeliCamAPI は、ストリーム受信待機キューに格納されている先頭のストリームリクエストを取り出し、受信したストリームデータを保存していきます。画像ストリームデータの受信が完了すると、TeliCamAPI はデータ書き込みが完了したストリームリクエストをストリーム受信完了キューに移動し、受信完了を通知するために受信完了シグナルオブジェクトをシグナル状態にセットします。これらの処理は自動的に実行されます。

カメライメージバッファ転送モードの場合は、アプリケーションが[ExecuteCamImageBufferRead\(\)](#)を実行するたびにカメラから画像が転送されます。カメラのイメージバッファが空の時は、カメラのイメージバッファに画像が入り次第転送されます。

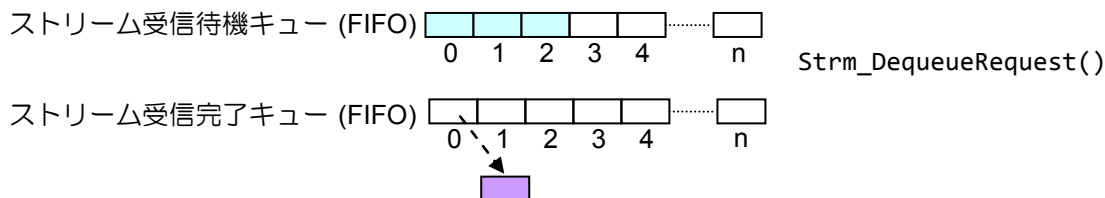
例：1 画像分のストリームデータ（画像データ）受信



#### 4.1.5.2.7. ストリームリクエストを完了キューから取り出し

ストリームデータ（画像データ）受信完了のシグナル受領後に [Strm\\_DequeueRequest\(\)](#)を実行すると、ストリーム受信完了キューから受信した画像が入っているストリームリクエストを取り出すことができます。

例：ストリームリクエストをストリーム受信完了キューから取り出し



#### 4.1.5.2.8. ストリーミング停止

[Strm\\_Stop\(\)](#)を実行するとカメラにストリーミング停止が指示され、その時点で撮像・転送中の画像の処理が完了した後ストリーミングが停止されます。



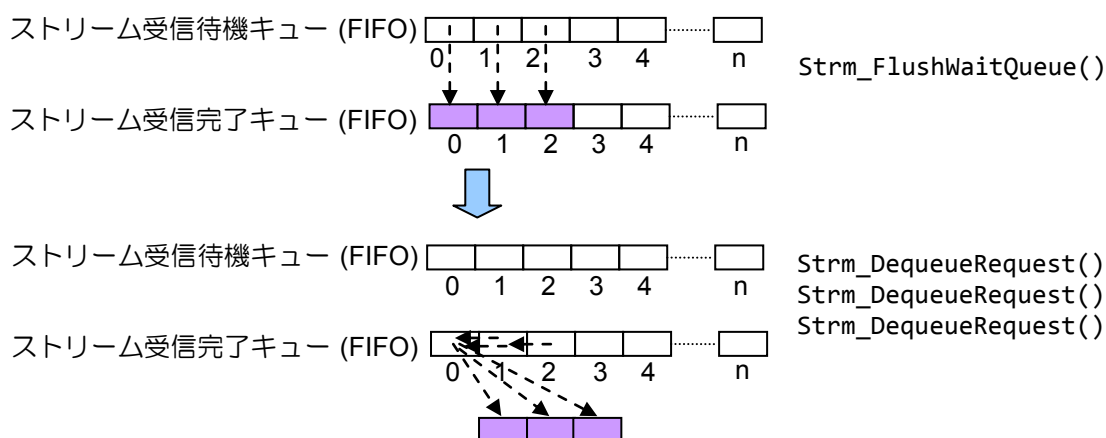
#### 4.1.5.2.9. ストリームリクエストの受信処理中止と完了キューから取り出し

ストリームインターフェースをクローズする前に、ストリーム受信待機キューおよびストリーム受信完了キューを空にしておく必要があります。

ストリーム受信待機キューが空でないときは、[Strm\\_FlushWaitQueue\(\)](#) をコールしてストリーム受信待機キュー内のすべてのストリームリクエストをストリーム受信完了キューに移動させてください。

ストリーム受信完了キューが空でないときは、ストリーム受信完了キューが空になるまで[Strm\\_DequeueRequest\(\)](#) をコールし続けて、すべてのストリームリクエストをストリーム受信完了キューから取り出してください。

例：ストリームリクエストの受信処理中止とストリーム受信完了キューから取り出し



#### 4.1.5.2.10. ストリームリクエストの解放

[Strm\\_CreateRequest\(\)](#) で作成したストリームリクエストは、それぞれ、[Strm\\_ReleaseRequest\(\)](#) を実行して解放してください。

ストリームリクエストのメンバとして作成した画像バッファは適切なタイミングで解放してください。

#### 4.1.5.2.11. ストリームインターフェースのクローズ

ストリームインターフェースの使用を終了するときは、[Strm\\_Close\(\)](#) をコールしてください。

#### 4.1.6. カメライベント（メッセージ）コントロール

「カメライベント」はカメラで発生したイベントのことを意味します。カメライベントの情報はイベントデータまたはメッセージの形でカメラから送信されます。

TeliCamAPI は、カメライベント（メッセージ）データを取得する方法として画像データ受信関数と同様に 2 種類の方法を提供しています。

高水準関数は、カメライベント（メッセージ）データ受信処理を TeliCamAPI 内部で実行することによりユーザコードの記述量を減らした関数です。ユーザは簡単にカメライベント（メッセージ）データを取得することができます。

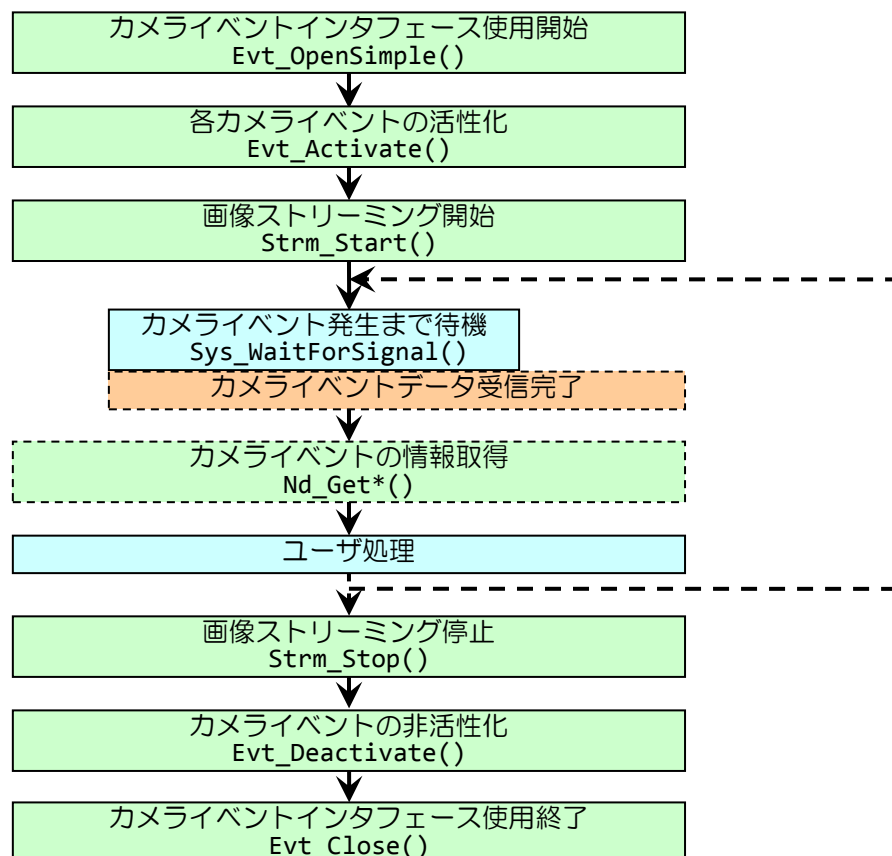
低水準関数は、カメライベントデータの受信処理をユーザに組んでいただくことを前提とした関数群です。ユーザコードの記述量は増えますが、任意のシーケンスの受信処理を組むことができ、カメラのパフォーマンスを最大限に活用し、特殊な受信処理の実装をすることも可能になります。

##### 4.1.6.1. 高水準関数を使用したカメライベント（メッセージ）の取得

高水準関数群では、カメラから受信したカメライベントは自動的に TeliCamAPI 内部に作成したイベントトリクエストリングバッファに一時保存されます。

TeliCamAPI はカメライベントデータ受信完了時にシグナルオブジェクトを使用してカメライベントデータの受信をユーザアプリケーションに通知します。発生したカメライベントのイベントデータは [GenICam 関数](#) を使用して取得することができます。

アプリケーションが行う一般的なシーケンスは下図の通りです。



---

#### 4.1.6.1.1. イベントインターフェースのオープン

[Evt\\_OpenSimple\(\)](#) 高水準関数を実行するとイベントインターフェースが使用可能になり、オープンされたイベントのイベントハンドルがユーザアプリケーションに返されます。

[Evt\\_OpenSimple\(\)](#) は、内部的には、複数のイベントリクエスト構造体と、イベントリクエスト構造体を保管するイベントリクエストリングバッファを作成してカメライベント受信に備えます。イベントリクエストとは、カメライベントデータを格納するための構造体データです。

#### 4.1.6.1.2. カメライベント（メッセージ）のアクティブ化

カメライベント通知受け取り用のシグナルオブジェクトを作成し、取得したいカメライベントの種類と作成したシグナルオブジェクトを引数にして [Evt\\_Activate\(\)](#) を実行してください。

[Evt\\_Activate\(\)](#) は指定されたカメライベントを有効にし、指定されたシグナルオブジェクトを TeliCamAPI に登録します。

#### 4.1.6.1.3. ストリーミング開始

[Strm\\_Start\(\)](#) を実行するとカメラにストリーミング開始が指示され、ストリーミングが開始されます。

カメライベントデータを受信すると、TeliCamAPI は、カメラから受信したカメライベントデータのイベントリクエスト構造体への保存、イベントリクエストリングバッファ内のイベントリクエスト構造体更新などの受信作業をバックグラウンドで実行し、[Evt\\_Activate\(\)](#) で指定されたシグナルオブジェクトをシグナル状態にセットしてユーザアプリケーションにカメライベントの受信を通知します。

#### 4.1.6.1.4. カメライベント（メッセージ）のペイロードデータ取得

カメライベント受信後は、[GenlCam 関数 \(Nd\\_GetNode\(\), Nd\\_GetIntValue\(\)\)](#) などを使用して受信したカメライベントの付随情報を取得することができます。（本ドキュメントリリース時は、USB3 Vision カメラのイベントにタイムスタンプ以外の付随情報はありません。）

#### 4.1.6.1.5. ストリーミング停止

[Strm\\_Stop\(\)](#) を実行するとカメラにストリーミング停止が指示され、その時点で撮像・転送中の画像の処理が完了した後ストリーミングが停止されます。

#### 4.1.6.1.6. カメライベント（メッセージ）の非アクティブ化

カメライベントの使用が終了したら、[Evt\\_Deactivate\(\)](#) を使用してカメライベントを無効状態にしてください。

#### 4.1.6.1.7. イベントインターフェースのクローズ

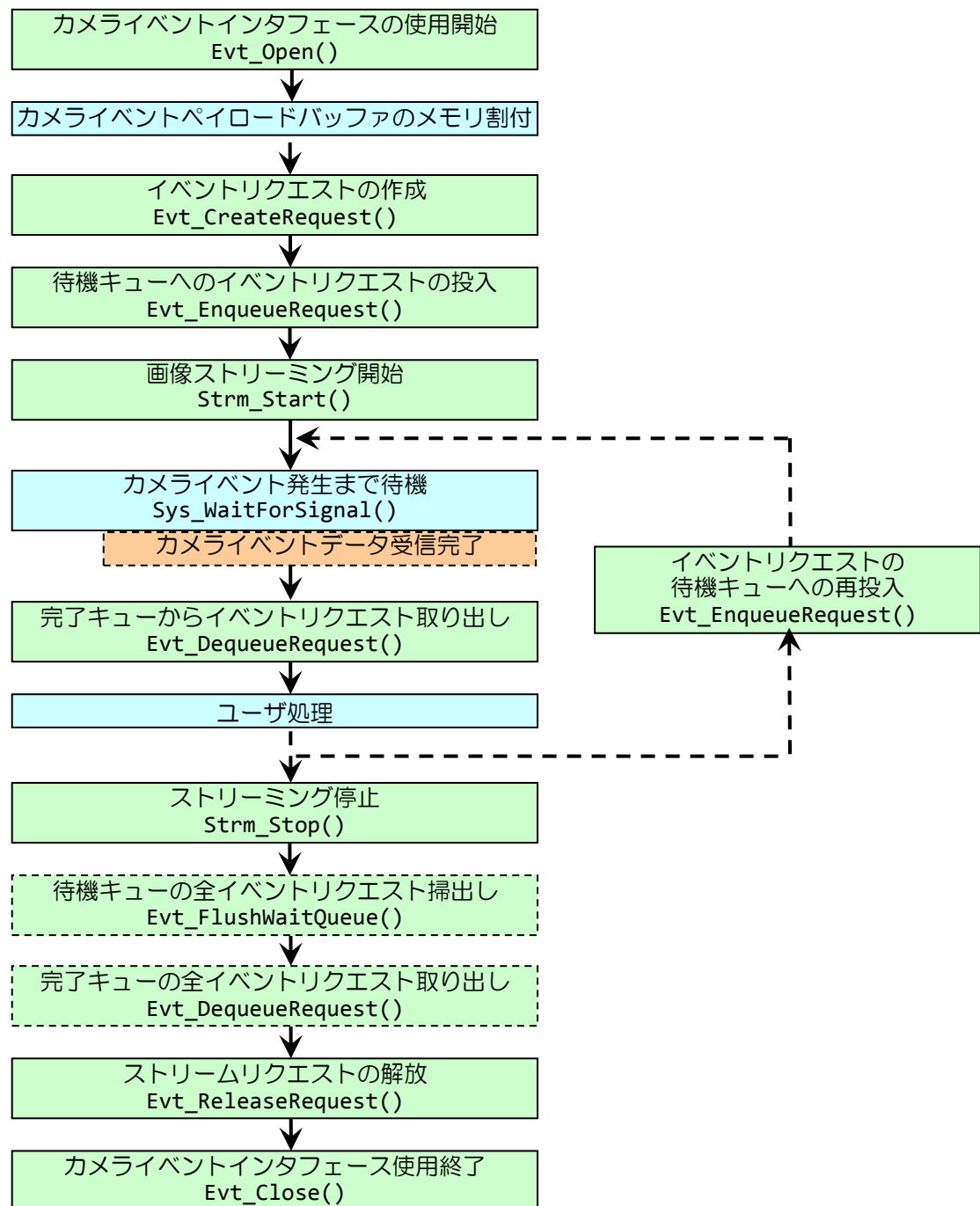
イベントインターフェースの使用を終了するときは、[Evt\\_Close\(\)](#) を実行してください。

#### 4.1.6.2. 低水準関数を使用したカメライベント（メッセージ）の取得

低水準関数は、カメライベントデータの受信処理をユーザに組んでいただくことを前提とした関数群です。ユーザコードの記述量は増えますが、任意のシーケンスの受信処理を組むことができ、カメラのパフォーマンスを最大限に活用し、特殊な受信処理の実装をすることも可能になります。

現在のバージョンでは、USB3 Vision カメラのみ低水準関数を使用することができます。

アプリケーションが行う一般的なシーケンスは下図の通りです。



#### 4.1.6.2.1. イベントインターフェースのオープン

[Evt\\_Open\(\)](#) 低水準関数を実行するとイベントインターフェースが使用可能になり、オープンされたカメライベントのイベントハンドルがユーザアプリケーションに返されます。[Evt\\_Open\(\)](#)実行時にイベントを受領するためのシグナルオブジェクトを引数で指定してください。

#### 4.1.6.2.2. カメライベント（メッセージ）ペイロードバッファの確保

TeliCamAPI はストリームイベントリクエストを作成する関数は提供していますが、ストリームイベントリクエストのメンバとするペイロードバッファはユーザアプリケーションがメモリ割り付けしておく必要があります。

連続的取込み中のアプリケーションの処理負荷が最も重いタイミングでもとりこぼしなくイベントデータを受信できるよう、余裕を持った数のイベントリクエスト（およびペイロードバッファ）を確保してください。

イベントリクエスト構造体が解放されるまでは、ペイロード用バッファは解放しないでください。ペイロードバッファを解放するときは以下の手順を守ってください。

1. イベント受信待機キュー内のすべてのイベントリクエストをイベント受信完了キューに移動させるために[Evt\\_FlushWaitQueue\(\)](#) を実行。
2. イベント受信待機キューが空になるまで[Evt\\_DequeueRequest\(\)](#) を実行して、イベント受信完了キューから全てのイベントリクエストを取り出し。
3. [Evt\\_ReleaseRequest\(\)](#)を実行して、ストリームリクエストを解放。
4. カメライベントペイロード用バッファを解放。

#### 4.1.6.2.3. イベントリクエストの作成

カメライベント（メッセージ）を取得するためのイベントリクエストは、[Evt\\_CreateRequest\(\)](#)をコールして作成します。作成されたイベントリクエストをイベント受信待機キューに追加することにより、カメライベント（メッセージ）を取得することができます。アプリケーションを終了する場合は、すべてのイベントリクエストを解放する必要があります。

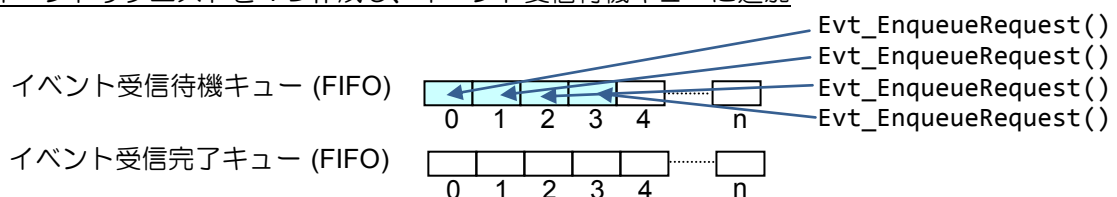
メモリ割り付け済のペイロードバッファを引数に指定して [Evt\\_CreateRequest\(\)](#)関数を実行するとカメライベント受信用のイベントリクエストが1個作成され、そのハンドルが引数に戻されます。

アプリケーション終了時は TeliCamAPI を終了するまでにすべてのイベントリクエストを解放してください。

#### 4.1.6.2.4. イベントリクエストを待機キューに追加

[Evt\\_EnqueueRequest\(\)](#)を実行して、イベントリクエストをイベント受信待機キューに投入してください。イベント受信待機キューにイベントリクエストが入っていると TeliCamAPI はカメライベントデータを受信できるようになります。

例：イベントリクエストを4つ作成し、イベント受信待機キューに追加



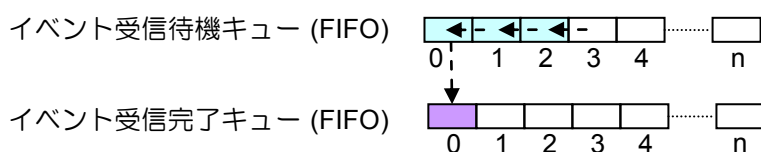
#### 4.1.6.2.5. ストリーミング開始

[Strm\\_Start\(\)](#)を実行するとカメラにストリーミング開始が指示され、ストリーミングが開始されます。

#### 4.1.6.2.6. カメライベント（メッセージ）データ受信

カメラからカメライベントデータを受信すると、TeliCamAPI は、イベント受信待機キューに格納されている先頭のイベントリクエストを取り出し、受信したイベントデータを保存していきます。カメライベントデータの受信が完了すると、TeliCamAPI はデータ書き込みが完了したイベントリクエストをイベント受信完了キューに移動し、カメライベントを通知するためにシグナルオブジェクトをシグナル状態にセットします。これらの処理は自動的に実行されます。

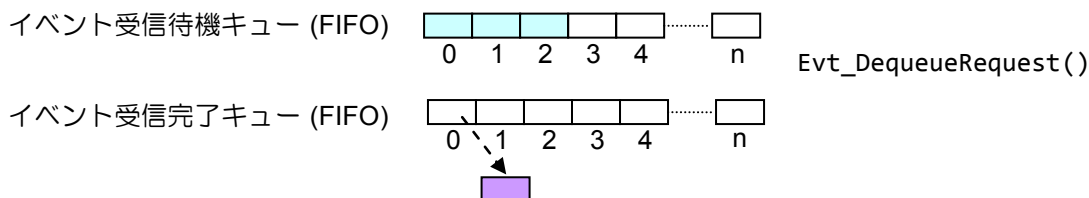
例：1つのイベント受信



#### 4.1.6.2.7. イベントリクエストを完了キューから取り出し

アプリケーションは、カメライベント受領を通知するシグナル受領後に [Evt\\_DequeueRequest\(\)](#) を実行すると、イベント受信完了キューから受信したカメライベントデータが入っているイベントリクエストを取り出すことができます。

例：イベントリクエストをイベント受信完了キューから取り出し



#### 4.1.6.2.8. ストリーミング停止

[Strm\\_Stop\(\)](#)を実行するとカメラにストリーミング停止が指示され、その時点で撮像・転送中の画像の処理が完了した後ストリーミングが停止されます。

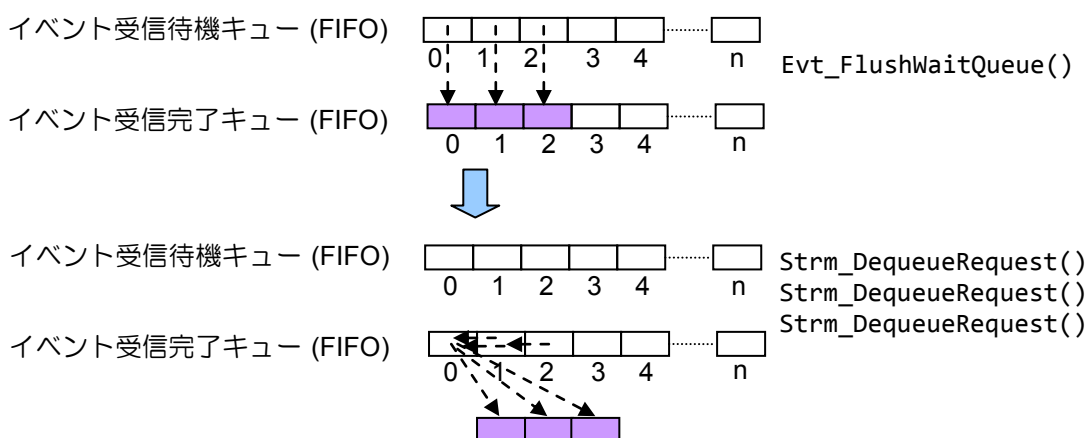
#### 4.1.6.2.9. イベントリクエストの受信処理中止と完了キューから取り出し

イベントインターフェースをクローズする前に、イベント受信待機キューおよびイベント受信完了キューを空にしておく必要があります。

イベント受信待機キューが空でないときは、[Evt\\_FlushWaitQueue\(\)](#)をコールしてイベント受信待機キュー内のすべてのイベントリクエストをイベント受信完了キューに移動させてください。

イベント受信完了キューが空でないときは、イベント受信完了キューが空になるまで[Evt\\_DequeueRequest\(\)](#)をコールし続けて、すべてのイベントリクエストをイベント受信完了キューから取り出してください。

例：イベントリクエストの受信処理中止とイベント受信完了キューから取り出し



#### 4.1.6.2.10. イベントリクエストの解放

[Evt\\_CreateRequest\(\)](#)で作成したイベントリクエストは、それぞれ、[Evt\\_ReleaseRequest\(\)](#)を実行して解放してください。

イベントリクエストのメンバとして作成したペイロードバッファは適切なタイミングで解放してください。

#### 4.1.6.2.11. イベントインターフェースのクローズ

イベントインターフェースの使用を終了するときは、[Evt\\_Close\(\)](#)をコールします。



---

#### 4.1.7. カメラのアクセスモード（制御チャネル特権）

GigE Vision カメラでは、コントロールチャネルのアクセスモード（制御チャネル特権）として、3つのレベルがあります。

- 独占アクセスモード（Exclusive 特権）  
独占アクセスモードでオープンされたアプリケーションは、カメラに独占的にアクセスすることができます。他のアプリケーションは、カメラをオープンすることはできません。
- コントロールアクセスモード（Control 特権）  
コントロールアクセスモードでオープンされたアプリケーションは、カメラにアクセスすることができます。他のアプリケーションは、オープンアクセスモードでのみカメラをオープンすることができます。
- オープンアクセスモード（Open 特権）  
オープンアクセスモードでオープンされたアプリケーションは、カメラのレジスタを読むことはできますが、書き込みはできません。

USB3 Vision カメラでは、アクセスモード機能は実装されていません。

#### 4.1.8. ハートビート処理

GigE Vision カメラでは、制御チャネル特権維持のためにアプリケーションは定期的にはカメラにアクセスする必要があります。この定期的に行うアクセスをハートビートシーケンスと呼びます。

GigE Vision カメラをオープンすると、TeliCamAPI は 15 秒のハートビートタイムアウト設定でハートビートを有効に設定されます。TeliCamAPI はカメラとの通信頻度を確認し、必要に応じてダミーコマンドを送信して通信が途切れないようバックグラウンド処理で制御します。ユーザアプリケーションは通信頻度を意識してカメラへのアクセスを調整する必要はありません。

通常はハートビート設定を変更する必要はありません。

ただし、デバッグ等で処理が長時間中断する場合は、ハートビート設定を変更しないと、カメラがハートビートタイムアウトエラーと判断して制御チャネル特権を解除してしまうため、中断解除後にカメラにアクセスすることができなくなります。このような場合は、制御チャネル特権が開場されないよう、[Cam\\_SetHeartbeat\(\)](#) を使用して、ハートビートを無効にするか、ハートビートタイムアウト時間を長くしてください。

USB3 Vision Camera では、ハートビート機能は実装されていません。



---

## 4.2. SDK のインストール

TeliCamSDK のインストール方法は、別紙の “TeliCamSDK for Linux Getting Started Guide Jpn.pdf” をご覧ください。

## 4.3. SDK のアンインストール

TeliCamSDK のインストール方法は、別紙の “TeliCamSDK for Linux Getting Started Guide Jpn.pdf” をご覧ください。

## 4.4. 開発環境の設定

TeliCamSDK を使用したアプリケーションをコンパイルしたり実行するためには、以下に示す環境変数の設定が必要となります。

```
TELICAMSDK=/opt/TeliCamSDK
export TELICAMSDK
```

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

上記環境変数の設定は、シェルを実行することにより実行できます。

```
source /opt/TeliCamSDK/set_env.sh
```

開発に必要なファイルは、以下のディレクトリにインストールされています。

```
ライブラリファイル    : /opt/TeliCamSDK/lib
インクルードファイル  : /opt/TeliCamSDK/include
```

Makefile を使ってアプリケーションをコンパイルするためには、ライブラリをインクルードファイルのディレクトリセッティングを行う必要があります。

以下に例を示します。

```
g++ -c sample.cpp -I/opt/TeliCamSDK/include -L/opt/TeliCamSDK/lib -lTeliCamApi_64
```

インストールされているサンプルプロジェクトを参考に、アプリケーションを作成してください。

---

## 4.5. パフォーマンスのチューニング

TeliCamSDK は、API 内部のパケット受信スレッドの優先順位を上げるによりパフォーマンスを向上させ、画像取り込み時間のバラつき等を最小にします。

ただし、Linux のデフォルト設定では root 権限で実行された場合にしかスレッドの優先順位を上げることはできません。

パフォーマンスを要求するアプリケーションの場合は、以下のいずれかの方法により優先順位を変更できるようにしてください。

- アプリケーションをルート権限で実行する。
- リアルタイム・プロセスの優先順位を変更できるようにシステム構成を変更する。

Pluggable Authentication Modules (PAM) for Linux の pam\_limits モジュールを使用すると、制限構成ファイルでシステム・リソースでの制限を構成できます。

デフォルトの制限は、/etc/security/limits.conf ファイル で設定されます。

例えば、

```
* - rtprio 99
```

と指定すると、すべてのユーザーがリアルタイム・プロセスの優先順位を変更できるようになります。

limits.conf を変更しても、直ちに有効にはなりません。構成変更を有効にするには、システムを再起動する必要があります。

## 5. ライブラリ関数

### 5.1. システム関数

#### 5.1.1. Sys\_Initialize

TeliCamAPI の初期化処理を実行します。

##### [構文]

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

##### [パラメータ]

パラメータ	内 容
eCamType [in]	使用するカメラのタイプです。(省略可能) すべてのタイプのカメラを指定する場合は、CAM_TYPE_ALL を指定してください。 省略した場合は、すべてのタイプのカメラが使用可能です。

##### [ CAM\_TYPE 型 ]

```
typedef enum  
{  
    CAM_TYPE_UNKNOWN    = 0x00,  
    CAM_TYPE_U3V        = 0x01,    // USB3 Vision Camera  
    CAM_TYPE_GEV        = 0x02,    // GigE Vision Camera  
    CAM_TYPE_ALL        = 0xFFFF,  
} CAM_TYPE;
```

##### [戻り値]

実行結果を返します。 戻り値は、[5.9. ステータスコード](#) を参照してください。

##### [備考]

アプリケーションは、すべての関数をコールする前に本関数を一度だけコールする必要があります。

[eCamType](#) は、使用したいカメラのタイプを OR ( | ) で複数指定することができます。

(例: Sys\_Initialize(CAM\_TYPE\_U3V | CAM\_TYPE\_GEV); )

[eCamType](#) で指定したタイプの API (TeliCamAPI が使用する共有ライブラリファイル) のロードに失敗した場合は、CAM\_API\_STS\_UNSUCCESSFUL がリターンされます。CAM\_TYPE\_ALL を指定した場合は、本 API がサポートしているタイプの API (TeliCamAPI が使用する共有ライブラリファイル) が一つもロードできなかった場合に CAM\_API\_STS\_UNSUCCESSFUL がリターンされます。

TeliCamAPI.h をインクルードする必要があります。

*現在の SDK バージョンでは、USB3 Vision カメラのみサポートしてます。*

##### [コード例]

[Sys\\_GetInformation\(\)](#) の [コード例](#) を参照してください。

---

### 5.1.2. Sys\_Terminate

TeliCamAPI の終了処理を実行します。

#### [構文]

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

#### [戻り値]

実行結果を返します。 戻り値は、[5.9. ステータスコード](#) を参照してください。

#### [備考]

アプリケーションを終了するとき、一度だけ本関数をコールする必要があります。

TeliCamAPI.h をインクルードする必要があります。

#### [コード例]

[Sys\\_GetInformation\(\)](#) の[コード例](#)を参照してください。

### 5.1.3. Sys\_GetInformation

TeliCamAPI のシステム情報を取得します。

#### [構文]

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

#### [パラメータ]

パラメータ	内 容
<i>psSysInfo</i> [in]	取得したシステム情報を格納する CAM_SYSTEM_INFO 構造体変数へのポインタです。

#### [CAM\_SYSTEM\_INFO 構造体]

```
typedef struct _CAM_SYSTEM_INFO  
{  
    U3V\_SYSTEM\_INFO      sU3vInfo;  
    GEV\_SYSTEM\_INFO      sGevInfo;  
    char                szDllVersion[MAX_INFO_STR];  
} CAM_SYSTEM_INFO, *PCAM_SYSTEM_INFO;
```

CAM_SYSTEM_INFO	内 容
sU3vInfo [out]	TeliCamAPI がロードした U3vAPI のシステム情報です。
sGevInfo [out]	TeliCamAPI がロードした GevAPI のシステム情報です。
szDllVersion [out]	TeliCamAPI のバージョンです。

#### [U3V\_SYSTEM\_INFO 構造体]

```
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;
```

U3V_SYSTEM_INFO	内 容
szDriverVersion [out]	U3v ドライバのバージョンです。 USB3 Vision カメラが一台も接続されていない場合は、バージョンを取得することはできません。
szDllVersion [out]	U3v API のバージョンです。
szDllExVersion [out]	U3v 拡張 API のバージョンです。

#### [GEV\_SYSTEM\_INFO 構造体]

```
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;
```

U3V_SYSTEM_INFO		内 容
szDriverVersion	[out]	Gev ドライバのバージョンです。 GigE Vision カメラが一台も接続されていない場合でもバージョンを取得することができます。
szDllVersion	[out]	Gev API のバージョンです。
szDllExVersion	[out]	Gev 拡張 API のバージョンです。

#### [戻り値]

実行結果を返します。 戻り値は、[5.9. ステータスコード](#) を参照してください。

#### [備考]

TeliCamAPI.h をインクルードする必要があります。

#### [コード例]

```
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  version : %s\n", sSysInfo.sU3vInfo.szDriverVersion);
printf("  TeliU3vApi2    version : %s\n", sSysInfo.sU3vInfo.szDllVersion);
printf("  TeliU3vCamApi  version : %s\n", sSysInfo.sU3vInfo.szDllExVersion);
printf("  TeliGevDriver  version : %s\n", sSysInfo.sGevInfo.szDriverVersion);
printf("  TeliGevApi2    version : %s\n", sSysInfo.sGevInfo.szDllVersion);
printf("  TeliGevCamApi  version : %s\n", sSysInfo.sGevInfo.szDllExVersion);
printf("  TeliCamAPI     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

この関数は、PC に接続されているカメラを探索して TeliCamAPI 内部に検出したカメラのリストを作成し、リスト内のカメラの数を引数の変数に設定します。

##### [構文]

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

##### [パラメータ]

パラメータ	内 容
<i>puiNum</i> [out]	検出したカメラの数を格納する変数へのポインタです。

##### [戻り値]

実行結果を返します。 戻り値は、[5.9. ステータスコード](#) を参照してください。

##### [備考]

検出されたカメラは、0 から (\*puiNum-1) までのインデックスがカメラ特定用の番号として付与されます。

この関数が実行される前は、TeliCamAPI 内部にカメラリストがまだ作成されていないため、カメラをオープンすることはできません。

TeliCamAPI.h をインクルードする必要があります。

##### [コード例]

[Sys\\_GetInformation\(\)](#) の[コード例](#)を参照してください。

---

### 5.1.5. Sys\_CreateSignal

この関数は、シグナルオブジェクトを作成し、シグナルオブジェクトのハンドルを返します。

#### [構文]

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

#### [パラメータ]

パラメータ	内 容
<i>phHandle</i> [out]	作成したシグナルオブジェクトのハンドルを格納する変数へのポインタです。

#### [戻り値]

実行結果を返します。 戻り値は、[5.9. ステータスコード](#) を参照してください。

#### [備考]

シグナルオブジェクトは、シグナルを待つために使用します。

この関数により作成されたシグナルオブジェクトは、[Sys\\_CloseSignal\(\)](#) により閉じられる必要があります。

この関数は、WINAPI の CreateEvent() 関数に類似しています。 .

TeliCamAPI.h をインクルードする必要があります。



---

### 5.1.6. Sys\_CloseSignal

この関数は、開いているシグナルオブジェクトを閉じます。.

#### [構文]

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

#### [パラメータ]

パラメータ		内 容
<i>hHandle</i>	[in]	開いているシグナルオブジェクトのハンドルです。

#### [戻り値]

実行結果を返します。 戻り値は、[5.9. ステータスコード](#) を参照してください。

#### [備考]

この関数は、WINAPI の CloseHandle() 関数に類似しています。.

TeliCamAPI.h をインクルードする必要があります。

### 5.1.7. Sys\_WaitForSignal

この関数は、指定されたシグナルオブジェクトの現在の状態を確認します。

シグナルオブジェクトがシグナル状態にセットされた場合は、直ちに制御を返します。このとき、シグナルオブジェクトは、非シグナル状態にリセットされます。

シグナルオブジェクトが非シグナル状態の場合、呼び出したスレッドはシグナルオブジェクトがシグナル状態にセットされるタイムアウト時間が経過するまで待機状態になります。

#### [構文]

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

#### [パラメータ]

パラメータ		内 容
<b>hHandle</b>	[in]	開いているシグナルオブジェクトのハンドルです。
<b>uiMilliseconds</b>	[in]	シグナル状態にセットされるまで待機するタイムアウト時間です。(単位：ミリ秒) 0 以外を指定すると、この関数は指定されたシグナルオブジェクトがシグナル状態にセットされるタイムアウト時間が経過するまで待機します。 0 を指定すると、この関数は指定されたシグナルオブジェクトの状態にかかわらず、即座に制御を返します。 CAM_SIGNAL_TIMEOUT_INFINITE (0xFFFFFFFF)を指定すると、指定されたシグナルオブジェクトがシグナル状態にセットされるまで待機します。

#### [戻り値]

以下に示す実行結果を返します。

CAM_API_STS_SUCCESS	指定されたシグナルオブジェクトがシグナル状態にセットされたことを意味します。
CAM_API_STS_TIMEOUT	タイムアウト時間が経過し、指定されたシグナルオブジェクトが非シグナル状態であったことを意味します。
CAM_API_STS_UNSUCCESSFUL	エラーが発生したことを意味します。

#### [備考]

この関数は、WINAPI の WaitForSingleObject() 関数に類似しています。

TeliCamAPI.h をインクルードする必要があります。

---

### 5.1.8. Sys\_ResetSignal

この関数は、指定されたシグナルオブジェクトを非シグナル状態にリセットします。

#### [構文]

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

#### [パラメータ]

パラメータ		内 容
hHandle	[in]	開いているシグナルオブジェクトのハンドルです。

#### [戻り値]

実行結果を返します。 戻り値は、[5.9. ステータスコード](#) を参照してください。

#### [備考]

この関数は、WINAPI の ResetEvent() 関数に類似しています。.

TeliCamAPI.h をインクルードする必要があります。

## 5.2. カメラ関数

### 5.2.1. Cam\_GetInformation

カメラの情報を取得します。

#### [構文]

```
CAM_API_STATUS Cam_GetInformation (  
    CAM_HANDLE      hCam,  
    uint32_t         uiCamIdx,  
    CAM\_INFO         *psCamInfo  
);
```

#### [パラメータ]

パラメータ		内 容
<i>hCam</i>	[in]	情報取得対象のカメラのカメラハンドルです。 カメラがオープンされていない場合は NULL を指定してください。
<i>uiCamIdx</i>	[in]	0 で始まるカメラのインデックスです。 <i>hCam</i> に NULL 以外を指定した場合、このパラメータは無視されます。
<i>psCamInfo</i>	[out]	カメラ情報を格納する <a href="#">CAM_INFO</a> 構造体変数へのポインタです。

#### [ CAM\_INFO 構造体 ]

```
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;
```

CAM_INFO		内 容
<i>eCamType</i>	[out]	カメラのタイプです。
<i>szManufacturer</i>	[out]	メーカー名です。
<i>szModelName</i>	[out]	モデル名です。
<i>szSerialNumber</i>	[out]	シリアル番号です。
<i>szUserDefinedName</i>	[out]	ユーザ定義情報です。 カメラに記憶されているユーザ定義情報が NULL で終端されていない場合、最後のデータが NULL に置き換わる場合があります。
<i>sU3vCamInfo</i>	[out]	USB3 Vision カメラに関する情報です。
<i>sGevCamInfo</i>	[out]	GigE Vision カメラに関する情報です。

[ U3V\_CAM\_INFO 構造体 ]

```
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;
```

U3V_CAM_INFO		内 容
szFamilyName	[out]	ファミリー名です。
szDeviceVersion	[out]	デバイスバージョンです。
szManufacturerInfo	[out]	メーカー情報です。
uiAdapterVendorId	[out]	カメラが接続されているUSB3.0 アダプタに組み込まれたUSB チップのベンダーID です。
uiAdapterDeviceId	[out]	カメラが接続されているUSB3.0 アダプタに組み込まれたUSB チップのデバイス ID です。
uiAdapterDfltMaxPacketSize	[out]	カメラが接続されている USB3.0 アダプタのMaxPacketSize デフォルト値です。 この値は、USB チップのベンダーID によって異なります。 <a href="#">Strm_Open()</a> 、 <a href="#">Strm_OpenSimple()</a> で <i>uiMaxPacketSize</i> に 0 を指定した場合にこの値が使用されます。

[ GEV\_CAM\_INFO 構造体 ]

```
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;
```

GEV_CAM_INFO		内 容
szDisplayName	[out]	カメラのディスプレイ名です。
aucMACAddress	[out]	カメラの MAC アドレスです。
cSupportIP_LLA	[out]	カメラのリンクローカルアドレス対応状況です。 0 の時非対応、0 以外るとき対応です。
cSupportIP_DHCP	[out]	カメラの DHCP 対応状況です。 0 の時非対応、0 以外るとき対応です。
cSupportIP_Persistent	[out]	カメラの不変 IP アドレス対応状況です。 0 の時非対応、0 以外るとき対応です。
cCurrentIP_LLA	[out]	カメラのリンクローカルアドレス アクティブ状況です。 0 の時非アクティブ、0 以外るときアクティブです。
cCurrentIP_DHCP	[out]	カメラの DHCP 対応 アクティブ状況です。 0 の時非アクティブ、0 以外るときアクティブです。
cCurrentIP_Persistent	[out]	カメラの不変 IP アドレス アクティブ状況です。 0 の時非アクティブ、0 以外るときアクティブです。
aucIPAddress	[out]	カメラの IP アドレスです。
aucSubnet	[out]	カメラのサブネットマスクです。
aucGateway	[out]	カメラのデフォルトゲートウェイです。
aucAdapterMACAddress	[out]	カメラが接続されているネットワークアダプタの MAC アドレスです。
aucAdapterIPAddress	[out]	カメラが接続されているネットワークアダプタの IP アドレスです。
aucAdapterSubnet	[out]	カメラが接続されているネットワークアダプタのサブネットマスクです。
aucAdapterGateway	[out]	カメラが接続されているネットワークアダプタのデフォルトゲートウェイです。
szAdapterDisplayName	[out]	カメラが接続されているネットワークアダプタの表示名です。

#### [戻り値]

実行結果を返します。 戻り値は、[5.9. ステータスコード](#) を参照してください。

#### [備考]

[Sys\\_GetNumOfCameras\(\)](#) をコールしてカメラリストを更新した場合は、カメラハンドル(hCam)に NULL を指定し、カメラのインデックス (*uiCamIdx*) で情報を取得してください。  
TeliCamAPI.h をインクルードする必要があります。

#### [コード例]

C++	
CAM_API_STATUS	uiStatus;
CAM_INFO	sCamInfo;
U3V_CAM_INFO	*psU3vCamInfo;
GEV_CAM_INFO	*psGevCamInfo;
uint32_t	uiNum;
// Initialize system.	
uiStatus = Sys_Initialize();	
if (uiStatus != CAM_API_STS_SUCCESS)	
return -1;	
// Get number of cameras.	
uiStatus = Sys_GetNumOfCameras(&uiNum);	

```

if (uiStatus != CAM_API_STS_SUCCESS)
    return -1;

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

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

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

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

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

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

        printf(" Gev display name : %s\n",
            psGevCamInfo->szDisplayName);
        printf(" Gev MAC address : %02X-%02X-%02X-%02X-%02X-%02X\n",
            psGevCamInfo->aucMACAddress[0],
            psGevCamInfo->aucMACAddress[1],
            psGevCamInfo->aucMACAddress[2],
            psGevCamInfo->aucMACAddress[3],
            psGevCamInfo->aucMACAddress[4],
            psGevCamInfo->aucMACAddress[5]);
        printf(" Gev support IP LLA : %d\n",
            psGevCamInfo->cSupportIP_LLA);
        printf(" Gev support IP DHCP : %d\n",
            psGevCamInfo->cSupportIP_DHCP);
        printf(" Gev Support IP Persistent-IP : %d\n",
            psGevCamInfo->cSupportIP_Persistent);
        printf(" Gev current IP LLA : %d\n",
            psGevCamInfo->cCurrentIP_LLA);
        printf(" Gev current IP DHCP : %d\n",
            psGevCamInfo->cCurrentIP_DHCP);
        printf(" Gev current IP Persistent-IP : %d\n",
            psGevCamInfo->cCurrentIP_Persistent);
        printf(" Gev IP Address : %d.%d.%d.%d\n",
            psGevCamInfo->aucIPAddress[0],
            psGevCamInfo->aucIPAddress[1],
            psGevCamInfo->aucIPAddress[2],
            psGevCamInfo->aucIPAddress[3]);
        printf(" Gev subnet mask : %d.%d.%d.%d\n",
            psGevCamInfo->aucSubnet[0],
            psGevCamInfo->aucSubnet[1],
            psGevCamInfo->aucSubnet[2],

```

```

        psGevCamInfo->aucSubnet[3]);
printf("  Gev default gateway           : %d.%d.%d.%d\n",
        psGevCamInfo->aucGateway[0],
        psGevCamInfo->aucGateway[1],
        psGevCamInfo->aucGateway[2],
        psGevCamInfo->aucGateway[3]);
printf("  Gev adapter MAC address       : %02X-%02X-%02X-%02X-%02X-%02X\n",
        psGevCamInfo->aucAdapterMACAddress[0],
        psGevCamInfo->aucAdapterMACAddress[1],
        psGevCamInfo->aucAdapterMACAddress[2],
        psGevCamInfo->aucAdapterMACAddress[3],
        psGevCamInfo->aucAdapterMACAddress[4],
        psGevCamInfo->aucAdapterMACAddress[5]);
printf("  Gev adapter IP address          : %d.%d.%d.%d\n",
        psGevCamInfo->aucAdapterIPAddress[0],
        psGevCamInfo->aucAdapterIPAddress[1],
        psGevCamInfo->aucAdapterIPAddress[2],
        psGevCamInfo->aucAdapterIPAddress[3]);
printf("  Gev adapter subnet mask         : %d.%d.%d.%d\n",
        psGevCamInfo->aucAdapterSubnet[0],
        psGevCamInfo->aucAdapterSubnet[1],
        psGevCamInfo->aucAdapterSubnet[2],
        psGevCamInfo->aucAdapterSubnet[3]);
printf("  Gev adapter default gateway     : %d.%d.%d.%d\n",
        psGevCamInfo->aucAdapterGateway[0],
        psGevCamInfo->aucAdapterGateway[1],
        psGevCamInfo->aucAdapterGateway[2],
        psGevCamInfo->aucAdapterGateway[3]);
printf("  Gev adapter display name        : %s\n",
        psGevCamInfo->szAdapterDisplayName);
    }
}

// Terminate system.
Sys_Terminate();

```



## 5.2.2. Cam\_Open

カメラをオープンし、アプリケーションでカメラを使用できるようにします。

### [構文]

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

### [パラメータ]

パラメータ	内 容
<i>uiCamIdx</i> [in]	0 から始まるカメラのインデックスです。 指定できる値の最大値は、 <a href="#">Sys_GetNumOfCameras()</a> で取得したカメラ台数-1 です。
<i>phCam</i> [out]	オープンしたカメラのカメラハンドルを格納する変数へのポインタです。
<i>hRmv</i> [in]	カメラの取り外し通知用のシグナルオブジェクトのハンドルです。  GigE Vision カメラの場合は、ハートビートタイムアウトが発生した場合にも シグナル状態にセットされます。  シグナルオブジェクトは、 <a href="#">Sys_CreateSignal()</a> で作成します。  通知する必要がなければ NULL を指定してください。  パラメータを省略した場合は NULL が指定されます。
<i>bUseGenICam</i> [in]	GenICam アクセスの有効／無効です。  true の場合、TeliCamAPI はカメラ記述ファイル(XML ファイル)をロードし、 GenApi 関数を使用可能にします。  false の場合、TeliCamAPI はカメラ記述ファイル(XML ファイル)をロードせ ず、GenApi 関数を使用不可にします。  パラメータを省略した場合は true が指定されます。 特別な理由がない場合は、true を指定してください。
<i>pvXml</i> [in]	PC 内のカメラ記述情報 (XML データ) を格納している変数へのポインタで す。 通常は NULL を指定し、カメラ内のカメラ記述ファイル (XML ファイル) を 使用します。 パラメータを省略した場合はカメラ内のカメラ記述ファイル(XML ファイル) を使用します。 GenICam アクセスの無効の時は、この変数は無視されます。

パラメータ	内 容								
<b>eAccessMode</b> [in]	<p>カメラのアクセスモード（特権）です。</p> <p>本パラメータは、GigE Vision カメラ使用以外のとき無視されます。</p> <p>パラメータを省略した場合は CAM_ACCESS_MODE_CONTROL が指定されます。</p> <table border="1"> <thead> <tr> <th>eAccessMode 指定値</th><th>内 容</th></tr> </thead> <tbody> <tr> <td>CAM_ACCESS_MODE_EXCLUSIVE</td><td>カメラの全機能を独占的に制御できます。他のアプリケーションは、カメラのオープンができなくなります。</td></tr> <tr> <td>CAM_ACCESS_MODE_CONTROL</td><td>カメラの全機能を制御できます。他のアプリケーションは、オープンアクセスモードでのみカメラのオープンをすることができます。</td></tr> <tr> <td>CAM_ACCESS_MODE_OPEN</td><td>カメラレジスタの読み込みはできますが、書き込みはできません。</td></tr> </tbody> </table>	eAccessMode 指定値	内 容	CAM_ACCESS_MODE_EXCLUSIVE	カメラの全機能を独占的に制御できます。他のアプリケーションは、カメラのオープンができなくなります。	CAM_ACCESS_MODE_CONTROL	カメラの全機能を制御できます。他のアプリケーションは、オープンアクセスモードでのみカメラのオープンをすることができます。	CAM_ACCESS_MODE_OPEN	カメラレジスタの読み込みはできますが、書き込みはできません。
eAccessMode 指定値	内 容								
CAM_ACCESS_MODE_EXCLUSIVE	カメラの全機能を独占的に制御できます。他のアプリケーションは、カメラのオープンができなくなります。								
CAM_ACCESS_MODE_CONTROL	カメラの全機能を制御できます。他のアプリケーションは、オープンアクセスモードでのみカメラのオープンをすることができます。								
CAM_ACCESS_MODE_OPEN	カメラレジスタの読み込みはできますが、書き込みはできません。								

#### [戻り値]

実行結果を返します。 戻り値は、[5.9. ステータスコード](#) を参照してください。

#### [備考]

USB3Vision カメラの場合、本関数は StreamEnable ノード（SIControl レジスタ）と EventEnable ノード（EIControl レジスタ）を 0（Disable）に初期化し、画像ストリーミングとカメライベントを停止状態に設定します。但し、ほかのアプリケーションが本カメラを既にオープンしている場合は StreamEnable ノードと EventEnable ノードの値は変更しません。

GigE Vision カメラの場合、カメラオープン時に 15 秒ハートビートタイムアウト設定でハートビートが有効に設定されます。ハートビートが有効になると、TeliCamAPI はハートビートチェック用のスレッドを作成し、定期的にハートビートチェック（カメラとの通信が継続していることの確認。一定時間通信がない場合はダミーコマンドを送信してカメラとの接続状態を確認。）を行います。ダミーコマンドに対するレスポンスが複数回連続で戻らない場合、TeliCamAPI はハートビートタイムアウトと判断し、シグナルオブジェクト [hRmv](#) をシグナル状態に設定してアプリケーションにカメラが取り除かれたことを通知します。

デバッグなどで長時間動作を停止させると、カメラがハートビートタイムアウトエラーと判断して制御チャンネル特権を解除してしまうため、動作再開後にカメラにアクセスできなくなります。オープン後に [Cam\\_SetHeartbet\(\)](#) をコールしてハートビートを無効にするか、ハートビートタイムアウト時間を長く設定することにより、ハートビートタイムアウトを回避することができます。

制御チャンネル特権については、[4.1.7 カメラのアクセスモード（制御チャンネル特権）](#) を参照してください。ハートビートについては、[4.1.8 ハートビート処理](#) を参照してください。

[bUseGenICam](#) を使用可に指定した場合、TeliCamAPI はカメラオープン時にカメラ記述ファイル（XML ファイル）をロードしてデータを内部に展開します。今まで接続したことがないカメラをオープンしするときは、XML ファイルのロード処理に数分かかる場合もあります。

[bUseGenICam](#) を使用不可に指定した場合、カメラのオープン処理は高速になりますが、GenApi 関数、カメライベント関数（USB3 Vision カメラは高水準のみ、GigE Vision カメラはすべて。）、一部を除いたカメラ制御関数が使用できなくなります。[bUseGenICam](#) を使用不可に指定したために使用できなくなった関数をアプリケーションが使用した場合、CAM\_API\_STS\_NOT\_AVAILABLE が戻り値として戻されます。

TeliCamAPI.h をインクルードする必要があります。

## [コード例]

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

指定した製造番号、カメラモデル名、ユーザ定義情報などを持つカメラをオープンします。

#### [構文]

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

#### [パラメータ]

パラメータ		内 容
<i>pszSerialNo</i>	[in]	NULL で終端されたカメラのシリアル番号文字列が格納されたバッファへのポインタです。 カメラの情報としてシリアル番号を使用しない場合は NULL を指定してください。
<i>pszModelName</i>	[in]	NULL で終端されたカメラのモデル名文字列が格納されたバッファへのポインタです。 カメラの情報としてモデル名を使用しない場合は NULL を指定してください。
<i>pszUserDefinedName</i>	[in]	NULL で終端されたカメラのユーザ定義情報文字列が格納されたバッファへのポインタです。 カメラの情報としてユーザ定義情報を使用しない場合は NULL を指定してください。
<i>phCam</i>	[out]	オープンしたカメラのカメラハンドルを格納する変数へのポインタです。
<i>hRmv</i>	[in]	カメラの取り外し通知用のシグナルオブジェクトのハンドルです。  GigE Vision カメラの場合は、ハートビートタイムアウトが発生した場合にもシグナル状態にセットされます。  シグナルオブジェクトは、 <a href="#">Sys_CreateSignal()</a> で作成します。  通知する必要がなければ NULL を指定してください。  パラメータを省略した場合は NULL が指定されます。
<i>bUseGenICam</i>	[in]	GenICam アクセスの有効／無効です。 true の場合は有効、false の場合は無効です。 パラメータを省略した場合は true が指定されます。 特別な理由がない場合は、true を設定してください。

パラメータ	内 容								
<i>pvXml</i> [in]	PC 内のカメラ記述情報（XML データ）バッファへのポインタです。 PC 内の XML データを使用する場合に指定してください。 通常は NULL を指定し、カメラ内の XML ファイルを使用します。 パラメータを省略した場合はカメラ内の XML ファイルが使用されます。 GenICam アクセスの無効の時は、この変数は無視されます。								
<i>eAccessMode</i> [in]	<p>カメラのアクセスモード（制御チャンネル特権）です。 本パラメータは、GigE Vision カメラ使用以外のとき無視されます。 パラメータを省略した場合は CAM_ACCESS_MODE_CONTROL が指定されます。</p> <table> <tr> <th>eAccessMode 指定値</th><th>内 容</th></tr> <tr> <td>CAM_ACCESS_MODE_EXCLUSIVE</td><td>カメラの全機能を独占的に制御できます。 他のアプリケーションは、カメラのオープンができなくなります。</td></tr> <tr> <td>CAM_ACCESS_MODE_CONTROL</td><td>カメラの全機能を制御できます。 他のアプリケーションは、オープンアクセスモードでのみカメラのオープンをすることができます。</td></tr> <tr> <td>CAM_ACCESS_MODE_OPEN</td><td>カメラレジスタの読み込みはできますが、書き込みはできません。</td></tr> </table>	eAccessMode 指定値	内 容	CAM_ACCESS_MODE_EXCLUSIVE	カメラの全機能を独占的に制御できます。 他のアプリケーションは、カメラのオープンができなくなります。	CAM_ACCESS_MODE_CONTROL	カメラの全機能を制御できます。 他のアプリケーションは、オープンアクセスモードでのみカメラのオープンをすることができます。	CAM_ACCESS_MODE_OPEN	カメラレジスタの読み込みはできますが、書き込みはできません。
eAccessMode 指定値	内 容								
CAM_ACCESS_MODE_EXCLUSIVE	カメラの全機能を独占的に制御できます。 他のアプリケーションは、カメラのオープンができなくなります。								
CAM_ACCESS_MODE_CONTROL	カメラの全機能を制御できます。 他のアプリケーションは、オープンアクセスモードでのみカメラのオープンをすることができます。								
CAM_ACCESS_MODE_OPEN	カメラレジスタの読み込みはできますが、書き込みはできません。								

#### [戻り値]

実行結果を返します。 戻り値は、[5.9. ステータスコード](#) を参照してください。

#### [備考]

USB3Vision カメラの場合、本関数は StreamEnable ノード（SIControl レジスタ）と EventEnable ノード（EIControl レジスタ）を 0（Disable）に初期化し、画像ストリーミングとカメライベントを停止状態に設定します。但し、ほかのアプリケーションが本カメラを既にオープンしている場合は StreamEnable ノードと EventEnable ノードの値は変更しません。

GigE Vision カメラの場合、カメラオープン時に 15 秒ハートビートタイムアウト設定でハートビートが有効に設定されます。ハートビートが有効になると、TeliCamAPI はハートビートチェック用のスレッドを作成し、定期的にハートビートチェック（カメラとの通信が継続していることの確認。一定時間通信がない場合はダミーコマンドを送信してカメラとの接続状態を確認。）を行います。ダミーコマンドに対するレスポンスが複数回連続で戻らない場合、TeliCamAPI はハートビートタイムアウトと判断し、シグナルオブジェクト [hRmv](#) をシグナル状態に設定してアプリケーションにカメラが取り除かれたことを通知します。

デバッグなどで長時間動作を停止させると、カメラがハートビートタイムアウトエラーと判断して制御チャンネル特権を解除してしまうため、動作再開後にカメラにアクセスできなくなります。オープン後に [Cam\\_SetHeartbet\(\)](#) をコールしてハートビートを無効にするか、ハートビートタイムアウト時間を長く設定することにより、ハートビートタイムアウトを回避することができます。

制御チャンネル特権については、[4.1.7 カメラのアクセスモード（制御チャンネル特権）](#) を参照してください。ハートビートについては、[4.1.8 ハートビート処理](#) を参照してください。

[bUseGenICam](#) を使用可に指定した場合、TeliCamAPI はカメラオープン時にカメラ記述ファイル（XML ファイル）をロードしてデータを内部に展開します。今まで接続したことがないカメラをオープンしするときは、XML ファイルのロード処理に数分かかる場合もあります。

[bUseGenICam](#) を使用不可に指定した場合、カメラのオープン処理は高速になりますが、GenApi 関

数、カメライベント関数（USB3 Vision カメラは高水準のみ、GigE Vision カメラはすべて。）、一部を除いたカメラ制御関数が使用できなくなります。[bUseGenICam](#) を使用不可に指定したために使用できなくなった関数をアプリケーションが使用した場合、CAM\_API\_STS\_NOT\_AVAILABLE が戻り値として戻されます。

TeliCamAPI.h をインクルードする必要があります。

#### [コード例]

```
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", (const char*)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

カメラをクローズします。

### [構文]

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

### [パラメータ]

パラメータ	内 容
hCam [in]	カメラのカメラハンドルです。

### [戻り値]

実行結果を返します。 戻り値は、[5.9. ステータスコード](#) を参照してください。

### [備考]

別のスレッドで処理を行っているときは、本関数をコールしないでください。

TeliCamAPI.h をインクルードする必要があります。

### [コード例]

[Cam\\_Open\(\)](#) の[コード例](#)を参照してください。

### 5.2.5. Cam\_ReadReg

カメラのレジスタからデータを取得します。

#### [構文]

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

#### [パラメータ]

パラメータ		内 容
<i>hCam</i>	[in]	カメラのカメラハンドルです。
<i>ullAdrs</i>	[in]	取得するレジスタの開始アドレスです。
<i>uiSizeQuadlet</i>	[in]	取得するデータサイズです。(Quadlet 単位。1～)
<i>pvData</i>	[out]	取得したデータを格納するバッファへのポインタです。

#### [戻り値]

実行結果を返します。 戻り値は、[5.9. ステータスコード](#) を参照してください。

#### [備考]

カメラのレジスタは 4 バイトでアライメントされており、4 バイト (Quadlet) 単位で読み出すことができます。

GigE Vision カメラの場合、取得したデータ (pvData) は 4 バイト (Quadlet) 単位ごとに、GigE Vision のバイトオーダー (ビッグエンディアン) から Intel プロセッサのバイトオーダー (リトルエンディアン) に変換して格納されます。文字列データなど 4 バイト以外のデータ長のデータを扱う時は、ユーザアプリケーションで本関数で読み出したデータを本来のデータの並びに修正してください。

USB3 Vision カメラの場合、TeliCamAPI は読み出しコマンド送信後、カメラからのレスポンスを 100ms まで待ちます。指定された時間内にデータを受信できなかった場合、Cam\_ReadReg() は CAM\_API\_STS\_TIMEOUT を返して処理を終了します。1 回の要求で取得できる最大データサイズは、カメラにより異なります。

GigE Vision カメラの場合、TeliCamAPI は読み出しコマンド送信後、カメラからのレスポンスを 200ms まで待ちます。指定された時間内にデータを受信できなかった場合は一度だけリトライし、それでも受信できなかった場合 Cam\_ReadReg() は CAM\_API\_STS\_TIMEOUT を返して処理を終了します。 1 回の要求で、最大 134 個の連続したレジスタを読み出すことが可能です。

TeliCamAPI.h をインクルードする必要があります。

#### [コード例]

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, 0, &sCamInfo);
if (uiStatus != CAM_API_STS_SUCCESS)
    return -1;

// Set Width address.
if (sCamInfo.eCamType == CAM_TYPE_U3V) {
    ullAddress = 0x00202098; // 0x100000(IIDC Address)
                          // + 0x102000(OffsetCategoryBlock2)
                          // + 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

カメラのレジスタにデータを書き込みます。

#### [構文]

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

#### [パラメータ]

パラメータ		内 容
<i>hCam</i>	[in]	カメラのカメラハンドルです。
<i>ullAdrs</i>	[in]	書き込むレジスタの開始アドレスです。
<i>uiSizeQuadlet</i>	[in]	書き込むデータサイズです。(Quadlet 単位。1～)
<i>pvData</i>	[in]	書き込むデータを格納したバッファへのポインタです。

#### [戻り値]

実行結果を返します。 戻り値は、[5.9. ステータスコード](#) を参照してください。

#### [備考]

カメラのレジスタは 4 バイトでアライメントされており、4 バイト (Quadlet) 単位で書き込むことができます。

GigE Vision カメラの場合、TeliCamAPI は書き込むデータ (pvData) を 4 バイト (Quadlet) 単位ごと Intel プロセッサのバイトオーダー (リトルエンディアン) から GigE Vision のバイトオーダー (ビッグエンディアン) に変換したデータを、カメラに送信します。文字列データなど 4 バイト以外のデータ長のデータを扱う時は、本来のデータの並びでデータがカメラに送信されるように並べ変えたデータを本関数のパラメータに設定してください。

USB3 Vision カメラの場合、TeliCamAPI は書き込みコマンド送信後、カメラからのレスポンスを 100ms まで待ちます。指定された時間内にレスポンスを受信できなかった場合、Cam\_WriteReg() は CAM\_API\_STS\_TIMEOUT を返して処理を終了します。1 回の要求で書き込むことができる最大データサイズは、カメラにより異なります。

GigE Vision カメラの場合、TeliCamAPI は書き込みコマンド送信後、カメラからのレスポンスを 200ms まで待ちます。指定された時間内にレスポンスを受信できなかった場合は一度だけリトライし、それでも受信できなかった場合 Cam\_WriteReg() は CAM\_API\_STS\_TIMEOUT を返して処理を終了します。 1 回の要求で、最大 134 個の連続したレジスタに書き込むことが可能です。

CAM\_API\_STS\_SUCCESS が戻り値として戻された場合、コマンド・レスポンスの送受信処理が正常に実行されています。しかし、書き込んだデータが正常に書き込まれているとは限りません。ストリーム転送中に書き込みを受け付けていないレジスタでは、書き込み処理が無視されたり保留されたりする場合があります。

データを書き込んだ後、必要に応じて [Cam\\_ReadReg\(\)](#) を実行して書き込みが正常に行われたかどうか確認してください。

TeliCamAPI.h をインクルードする必要があります。

---

**[コード例]**

[Cam\\_ReadReg\(\)](#) の[コード例](#)を参照してください。

### 5.2.7. Cam\_ResetPort

アダプタに搭載されているホストアダプタ／ホストコントローラのポートリセットを実行します。ポートは、オープンされているカメラのハンドルで指定します。

#### [構文]

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

#### [パラメータ]

パラメータ	内 容
hCam	[in] カメラのカメラハンドルです。

#### [戻り値]

実行結果を返します。 戻り値は、[5.9. ステータスコード](#) を参照してください。

#### [備考]

本関数は、USB3 Vision カメラのみ有効です。

本関数を実行すると、指定したカメラに接続されているホストアダプタ／ホストコントローラのポートがリセットされ、リセット完了後に [Cam\\_Open\(\)](#) または [Cam\\_OpenFormInfo\(\)](#) で指定したカメラ取り出しシグナルイベント [hRmv](#) がシグナル状態にセットされます。アプリケーションは本関数実行後に [Cam\\_Close\(\)](#) をコールする必要はありません。

本関数実行後に再度カメラをオープンする場合は、ストリーム関数やカメライベント（メッセージ）関数で確保したすべてのリソースを解放し、再度確保してからカメラをオープンしてください。

ポートリセット直後にプラグ&プレイ処理が走り、リセットされたポートに接続されているカメラが再探索されます。再探索には時間がかかります。再探索に必要な時間を待った後、[Sys\\_GetNumOfCameras\(\)](#) 関数を実行して接続されているカメラの数を確認してからカメラをオープンしてください。

問題が発生した時にこの関数をコールしても、すべての場合で正常に動作が復帰するとは限りません。

TeliCamAPI.h をインクルードする必要があります。

#### [コード例]

C++	
CAM_API_STATUS	uiStatus;
uint32_t	uiNum, uiNumOld, i;
SIGNAL_HANDLE	hRemoveEvt;
CAM_HANDLE	hCam;
// Initialize TeliCamAPI system.	
uiStatus = Sys_Initialize();	
if (uiStatus != CAM_API_STS_SUCCESS)	
return -1;	
// Get number of cameras.	
uiStatus = Sys_GetNumOfCameras(&uiNum);	
if ((uiStatus != CAM_API_STS_SUCCESS)    (uiNum == 0))	
return -1;	

```

uiNumOld = uiNum;

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

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

printf("Cam_Open was successful.(0)\n");

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

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

// Wait until the specified signal object is signaled.
uiStatus = Sys_WaitForSignal(hRemoveEvt, 2000);
if (uiStatus != CAM_API_STS_SUCCESS)
    return -1;

// Reopen

// 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;

    usleep(100000);    // 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.(1)\n");

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

if (hRemoveEvt != (SIGNAL_HANDLE)NULL)
    Sys_CloseSignal(hRemoveEvt);

// Terminate TeliCamAPI system.
Sys_Terminate();

```

---

### 5.2.8. Cam\_GetHeartbeat

カメラのハートビート設定（有効／無効）を取得します。

#### [構文]

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

#### [パラメータ]

パラメータ		内 容
<i>hCam</i>	[in]	カメラのカメラハンドルです。
<i>pbEnable</i>	[out]	取得したハートビート設定（有効／無効）を格納する変数へのポインタです。 true とき有効、false のとき無効です。
<i>puiHbTimeout</i>	[out]	取得したハートビートタイムアウト時間を格納する変数へのポインタです。(単位：ms)

#### [戻り値]

実行結果を返します。 戻り値は、[5.9. ステータスコード](#) を参照してください。

#### [備考]

本関数は、GigE Vision カメラのみ有効です。

ハートビートについては、[4.1.8 ハートビート処理](#) を参照してください。

TeliCamAPI.h をインクルードする必要があります。

### 5.2.9. Cam\_SetHeartbeat

カメラのハートビート設定（有効／無効）を変更します。

#### [構文]

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

#### [パラメータ]

パラメータ		内 容
<i>hCam</i>	[in]	カメラのカメラハンドルです。
<i>bEnable</i>	[in]	ハートビート有効／無効です。 true のとき有効、false のとき無効です。 カメラによっては、true 以外設定できません。
<i>uiHbTimeout</i>	[in]	ハートビートタイムアウト時間です。(単位：ms) 設定できる最小値は 500(ms) です。

#### [戻り値]

実行結果を返します。 戻り値は、[5.9. ステータスコード](#) を参照してください。

#### [備考]

本関数は、GigE Vision カメラのみ有効です。

カメラによりハートビート無効に設定できないカメラがありますので注意してください。  
GiantDragon シリーズは、ハートビートを無効に設定できません。

GigE Vision カメラの場合、カメラオープン時に 15 秒のハートビートタイムアウト設定でハートビートが有効に設定されます。

通常はハートビート設定を変更する必要はありませんが、デバッグ等で長時間処理を中断させる場合には、本関数を使用してハートビート設定を変更してください。

なお、本APIでハートビート制御を一元管理しているので、[GenICAM関数](#)または[Cam\\_WriteReg\(\)](#)を使用して `GevHeartbeatTimeout` レジスタおよび `GevGCCPHeartbeatDisable` レジスタを直接変更することは避けてください。

ハートビートについては、[4.1.8 ハートビート処理](#) を参照してください。

TeliCamAPI.h をインクルードする必要があります。

#### [コード例]

[Strm\\_OpenSimple\(\)](#) の[コード例](#)を参照してください。

## 5.3. カメラストリーム関数

TeliCamAPI は、高水準関数、低水準関数の2種類の画像情報を取得する関数群を提供しています。

高水準関数は、画像受信処理の大半を TeliCamAPI 内部で実行することによりユーザコードの記述量を減らした関数です。簡単に画像データを取得するコードが記述できます。

低水準関数は、画像ストリームデータの受信処理をユーザに組んでいただくことを前提とした関数群です。ユーザコードの記述量は増えますが、任意のシーケンスで受信処理をすることができ、カメラのパフォーマンスを最大限に活用し、特殊な受信処理の実装も可能にした関数群です。

### 5.3.1. 高水準関数

TeliCamAPI 内部に作成した画像一時保管用のストリームリクエストリングバッファを介して画像を取得するための関数群です。カメラから受信した画像をストリームリクエストリングバッファに保存する処理は TeliCamAPI がバックグラウンドで実行します。ユーザアプリケーションはストリームリクエストリングバッファから画像を取り出すコードを書くだけで簡単に画像を取得できます。

詳細は、[4.1.5.1. 高水準関数を使用したストリームデータ（画像データ）の取得](#) を参照してください。

#### 5.3.1.1. Strm\_OpenSimple

TeliCamAPI 内部に画像一時保管用のストリームリクエストリングバッファを作成し、画像取得用のストリームインターフェースをオープンします。

[構文]

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

[パラメータ]

パラメータ		内 容
<i>hCam</i>	[in]	カメラのカメラハンドルです。
<i>phStrm</i>	[out]	オープンしたストリームインターフェースのストリームハンドルを格納する変数へのポインタです。
<i>puiMaxPayloadSize</i>	[out]	1 つのストリームリクエストで受信するペイロードのサイズ(画像サイズ)を格納する変数へのポインタです。(単位: byte) ペイロードサイズは画像のサイズやフォーマットなどによって変わります。
<i>hCmpEvt</i>	[in]	ストリームを受信し、ストリームリクエストリングバッファが更新されたことを通知するシグナルオブジェクトのハンドルです。 この引数は省略可能です。 シグナルオブジェクトは、 <a href="#">Sys_CreateSignal()</a> で作成します。 通知する必要がなければ NULL を指定してください。 省略した場合は、NULL が指定されます。



パラメータ	内 容
<i>uiApiBufferCount</i> [in]	TeliCamAPI 内部に作成するストリームリクエストリングバッファのサイズです。この引数は省略可能です。 0 を設定すると、推奨値が自動的に指定されます。 設定範囲は、3 から30 です。 省略した場合は、DEFAULT_API_BUFFER_CNT (8) が指定されます。
<i>uiMaxPacketSize</i> [in]	ドライバが受け取るパケットの最大サイズです。(単位: byte。) この引数は省略可能です。  0 を設定するか指定を省略すると、以下の値が使用されます。 USB3 Visionカメラ : 65536 byte GigE Visionカメラ : 1500 byte  GigE Vision カメラでジャンボフレームを設定している場合は、ジャンボフレームの設定値からイーサネットヘッダを除いた値を指定してください。 ネットワークアダプタがイーサネットヘッダを含んだ値をジャンボフレームサイズとして示している場合は、イーサネットヘッダサイズ (14byte) を減算した値を指定する必要があります。(ジャンボフレームの値が 9014 と表示されている場合は、通常イーサネットヘッダを含んだ値が表示されています。この場合、uiMaxPacketSize には 9000 を指定してください。)  USB3 Visionカメラの場合は、特別な理由がない限り 0 を指定するか、値の指定を省略してください。

#### [戻り値]

実行結果を返します。 戻り値は、[5.9. ステータスコード](#) を参照してください。

#### [備考]

本関数を使用すると簡単なコード記述でカメラから画像を取り込むことができます。カメラのパフォーマンスの最大限の活用、特殊なシーケンスの受信処理の構築が求められる場合には、低水準関数 [Strm\\_Open\(\)](#) を使用してストリームインターフェースをオープンしてください。

[Strm\\_SetCallbackImageAcquired\(\)](#) を使用してコールバック関数を登録しておくと、画像ストリームを受信してストリームリクエストリングバッファの内容が更新された時に TeliCamAPI は登録されたコールバック関数を実行します。

高水準関数でストリームをオープンしたとき、画像取得には以下の 3 種類の方法が使用できます。

- [Strm\\_SetCallbackImageAcquired\(\)](#) で登録したコールバック関数の引数使用。  
ImageAcquired のコールバック関数の引数 [psImageInfo](#) は、最新画像データとその情報を保有する [CAM\\_IMAGE\\_INFO 構造体](#) へのポインタになっています。コールバック関数実行中に限り、ユーザアプリケーションは [psImageInfo](#) が示す構造体のデータにアクセスできます。
- [Strm\\_ReadCurrentImage\(\)](#) を使用する方法  
[Strm\\_ReadCurrentImage\(\)](#) を実行するとストリームリクエストリングバッファに格納されているストリームリクエストの最新画像のデータが引数に指定したバッファにコピーされます。
- [Strm\\_LockBuffer\(\)](#) を使用する方法  
[Strm\\_LockBuffer\(\)](#) を使用するとストリームリクエストリングバッファ内の任意のストリームリクエスト内のデータを取得することができます。  
以下の手順に従って画像を取得してください。

- A. [Strm\\_GetCurrentBufferIndex\(\)](#)を実行し最新画像を保有するストリームリクエストのストリームリクエストリングバッファ内インデックスを取得。
- B. 最新画像を保有するストリームリクエストのインデックスを基準にして、取得したい画像を持つストリームリクエストのインデックスを計算。
- C. [Strm\\_LockBuffer\(\)](#)を実行して、取得したい画像を持つストリームリクエストをロック。
- D. [Strm\\_LockBuffer\(\)](#)の引数に戻されたポインタを使用して画像とその情報を取得。
- E. [Strm\\_UnlockBuffer\(\)](#)を実行して、ストリームリクエストのロックを解除。

画像データを受信すると、TeliCamAPI はストリームリクエストリングバッファの内容を更新した後、画像受信シグナルオブジェクト [hCmpEvt](#) をシグナル状態にセットし、コールバック関数を実行します。

他に優先度の高いスレッドが動作していると、シグナルオブジェクト [hCmpEvt](#) がシグナル状態にセットされたことを検知してユーザ処理の実行を開始するときには、複数の新しい画像を受信してしまっている場合があります。このとき、ユーザアプリケーションによりシグナルオブジェクト [hCmpEvt](#) がリセット ([Sys\\_WaitForSignal\(\)](#) 実行) される前に受信した画像に対して、シグナルオブジェクト [hCmpEvt](#) を再度シグナル状態にセットすることはありませんのでご注意ください。

同様に、ImageAcquired コールバック関数実行中に複数の画像を受信完了したときは、実行中のコールバック関数が終了した後に、受信完了している複数の画像をユーザアプリケーションが処理できるよう1回だけコールバック関数が実行されます。

受信したすべての画像に対してユーザ処理を行う必要があるときは、前回最後に処理した画像のバッファインデックスを記録しておき、ユーザの画像取得処理で [Strm\\_GetCurrentBufferIndex\(\)](#)、を実行して前回の処理以降何枚の画像が取得されたか調べ、[Strm\\_LockBuffer\(\)](#)、[Strm\\_UnlockBuffer\(\)](#)を使用して未処理の画像を処理するようにしてください。

TeliCamAPI 内部のストリームリクエストリングバッファは、カメラ（内部）のイメージバッファとは異なりますのでご注意ください。

TeliCamAPI.h をインクルードする必要があります。

#### [コード例]

```
C++

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

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

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

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

```

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

    // Create completion event object for stream.
    uiStatus = Sys_CreateSignal(&hStrmCmpEvt);
    if (uiStatus != CAM_API_STS_SUCCESS)
        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 *)malloc(uiPyldSize);
    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.
    uiStatus = Sys_WaitForSignal(hStrmCmpEvt, 2000);
    if (uiStatus != CAM_API_STS_SUCCESS)
        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);
}
while(false);

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

    // Close stream interface.

```

---

```
    uiStatus = Strm_Close(hStrm);
    if (uiStatus != CAM_API_STS_SUCCESS)
        printf("Strm_Close error! (0x%x)", uiStatus);
}

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

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

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

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

### 5.3.1.2. Strm\_ReadCurrentImage

TeliCamAPI 内部のストリームリクエストリングバッファから最新の画像を取得し、指定メモリにコピーします。

#### [構文]

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

#### [パラメータ]

パラメータ	内 容
<i>hStrm</i> [in]	ストリームインターフェースのストリームハンドルです。
<i>pvBuf</i> [out]	取得した最新の画像データをコピーするバッファへのポインタです。
<i>puiSize</i> [in,out]	バッファのサイズが格納された変数へのポインタです。 この値が画像サイズより小さい場合はエラーになります。 コピー後、コピーされたデータサイズが格納されます。(単位: byte)
<i>psImageInfo</i> [out]	画像付随情報を格納する <a href="#">CAM_IMAGE_INFO</a> 構造体変数へのポインタです。 付随情報が必要ない場合は NULL を指定してください。

#### [ CAM\_IMAGE\_INFO 構造体 ]

```
typedef struct _CAM_IMAGE_INFO  
{  
    uint64_t          ullTimestamp;  
    CAM_PIXEL_FORMAT  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;
```

CAM_IMAGE_INFO	内 容
<i>ullTimestamp</i> [out]	カメラが出力する映像データのタイムスタンプです。 USB3 Vision Camera のとき、単位は nsec です。 GigE Vision Camera のとき、単位は 16 nsec です。
<i>uiPixelFormat</i> [out]	カメラが出力する映像データのピクセルフォーマットです。
<i>uiSizeX</i> [out]	カメラが出力する映像データの水平有効画素数です。
<i>uiSizeY</i> [out]	カメラが出力する映像データの垂直有効画素数です。
<i>uiOffsetX</i> [out]	カメラが出力する映像データの水平方向開始位置です。
<i>uiOffsetY</i> [out]	カメラが出力する映像データの垂直方向開始位置です。

CAM_IMAGE_INFO	内 容
uiPaddingX [out]	カメラが出力する映像データの水平画素に padding データが含まれる場合に使用します。
ullBlockId [out]	カメラが出力する映像データのフレーム番号です。 映像を受信するたびにインクリメントされます。 映像停止でクリアされます。
pvBuf [out]	画像データバッファへのポインタです。
uiSize [out]	ストリームリクエストリングバッファに格納された画像データのサイズです。（単位：byte）
ullImageId [out]	API が管理している画像番号です。 block_id が取得できないカメラを使用する場合、画像データのフレーム番号として使用できます。 画像番号はストリームインターフェースをオープンしたときに 0 にクリアされます。
uiStatus [out]	画像取得時のステータスを出力します。

#### [戻り値]

実行結果を返します。 画像取得時にエラーが発生している場合は、エラーコードがリターンされます。

戻り値は、[5.9. ステータスコード](#) を参照してください。

#### [備考]

本関数は、[Strm\\_OpenSimple\(\)](#) でオープンしたストリームインターフェースに使用できます。

画像のインデックスは、正常受信・異常受信にかかわらずインクリメントされます。

TeliCamAPI.h をインクルードする必要があります。

#### [コード例]

[Strm\\_OpenSimple\(\)](#) の[コード例](#)を参照してください。

### 5.3.1.3. Strm\_GetCurrentBufferIndex

最新のストリームリクエスト（画像）を格納している、TeliCamAPI 内部のストリームリクエストリングバッファのバッファインデックスを取得します。

#### [構文]

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

#### [パラメータ]

パラメータ	内 容
<i>hStrm</i> [in]	ストリームインターフェースのストリームハンドルです。
<i>puiBufferIndex</i> [out]	取得したストリームリクエストリングバッファのバッファインデックスを格納する変数へのポインタです。 格納される値は、0 から <a href="#">Strm_OpenSimple()</a> の <a href="#">uiApiBufferCount</a> 引数で指定された値 - 1 までの値です。 ただし、ストリームリクエストリングバッファに一度もストリームリクエストが追加されていない場合は 0xFFFFFFFF が格納されます。

#### [戻り値]

実行結果を返します。 戻り値は、[5.9. ステータスコード](#) を参照してください。

#### [備考]

本関数は、[Strm\\_OpenSimple\(\)](#) でオープンしたストリームインターフェースに使用できます。

TeliCamAPI.h をインクルードする必要があります。

#### [コード例]

C++
<pre>CAM_API_STATUS    uiStatus; uint32_t          uiNum, uiPyldSize, uiBufferIndex, uiAve, i; SIGNAL_HANDLE     hStrmCmpEvt = (SIGNAL_HANDLE)NULL; CAM_HANDLE        hCam = (CAM_HANDLE)NULL; CAM_STRM_HANDLE   hStrm = (CAM_STRM_HANDLE)NULL; void*             pvPayloadBuf = NULL; CAM_IMAGE_INFO    sImageInfo;  // Initialize system. uiStatus = Sys_Initialize(); if (uiStatus != CAM_API_STS_SUCCESS)     return -1;  // Get number of cameras. uiStatus = Sys_GetNumOfCameras(&amp;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, &amp;hCam, NULL); if (uiStatus != CAM_API_STS_SUCCESS)     return -1;  do</pre>

```

{
    // Create completion event object for stream.
    uiStatus = Sys_CreateSignal(&hStrmCmpEvt);
    if (uiStatus != CAM_API_STS_SUCCESS)
        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.
    uiStatus = Sys_WaitForSignal(hStrmCmpEvt, 2000);
    if (uiStatus != CAM_API_STS_SUCCESS)
        return -1;

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

    // Lock the image 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.
    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 image ring buffer pointer.
    uiStatus = Strm_UnlockBuffer(hStrm, uiBufferIndex);
    if (uiStatus != CAM_API_STS_SUCCESS)
        return -1;
}
while(false);

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

    // Close stream interface.

```



---

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

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

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

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

#### 5.3.1.4. Strm\_LockBuffer

TeliCamAPI 内部のストリームリクエスト リングバッファの指定先をロックし、画像情報を取得します。

##### [構文]

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

##### [パラメータ]

パラメータ		内 容
<i>hStrm</i>	[in]	ストリームインターフェースのストリームハンドルです。
<i>uiBufferIndex</i>	[in]	ロックするストリームリクエストリングバッファのバッファインデックスです。 0 から <a href="#">Strm_OpenSimple()</a> の <a href="#">uiApiBufferCount</a> 引数で指定した値 - 1 までの値が指定できます。
<i>psImageInfo</i>	[out]	ロックするストリームリクエストリングバッファに格納されている画像の付随情報を格納する <a href="#">CAM_IMAGE_INFO 構造体</a> 変数へのポインタです。付随情報が必要ない場合は NULL を指定してください。

##### [戻り値]

実行結果を返します。 戻り値は、[5.9. ステータスコード](#) を参照してください。

##### [備考]

本関数は、[Strm\\_OpenSimple\(\)](#) でオープンしたストリームインターフェースに使用できます。

本関数でロックしたストリームリクエストリングバッファのバッファは、できるだけ早く [Strm\\_UnlockBuffer\(\)](#) を実行してロックを解除してください。

カメラから受信した画像を書き込んだストリームリクエストはストリームリクエストリングバッファ内の各領域に定められた順序で格納されます。長期間バッファをロックし続けると。ロックしているバッファが最新画像を書き込んだストリームリクエストの格納先になったときに BufferBusy エラーが発生します。BufferBusy エラーが発生すると TeliCamAPI は格納しようとしていた最新画像を破棄し、[Strm\\_SetCallbackBufferBusy\(\)](#) により登録されたコールバック関数を実行します。

BufferBusy エラーが発生する場合は、ストリームリクエストリングバッファのバッファをロックする期間をできる限り短くするか、[Strm\\_OpenSimple\(\)](#) の [uiApiBufferCount](#) 引数に大きな値を設定してストリームリクエストリングバッファのサイズを大きくしてください。

TeliCamAPI.h をインクルードする必要があります。

##### [コード例]

[Strm\\_GetCurrentBufferIndex\(\)](#) の [コード例](#) を参照してください。

---

### 5.3.1.5. Strm\_UnlockBuffer

TeliCamAPI 内部のストリームリクエストリングバッファの指定先のロックを解除します。

#### [構文]

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

#### [パラメータ]

パラメータ		内 容
hStrm	[in]	ストリームインターフェースのストリームハンドルです。
uiBufferIndex	[in]	ロックを解除するストリームリクエストリングバッファのバッファインデックスです。 0 から <a href="#">Strm_OpenSimple()</a> の <a href="#">uiApiBufferCount</a> 引数で指定した値 - 1 までの値が指定できます。

#### [戻り値]

実行結果を返します。 戻り値は、[5.9. ステータスコード](#) を参照してください。

#### [備考]

本関数は、[Strm\\_OpenSimple\(\)](#) でオープンしたストリームインターフェースに使用できます。

[Strm\\_LockBuffer\(\)](#) によりロックしたストリームリクエストリングバッファのバッファは、本関数により、できるだけ早くロックを解除される必要があります。

TeliCamAPI.h をインクルードする必要があります。

#### [コード例]

[Strm\\_GetCurrentBufferIndex\(\)](#) の[コード例](#)を参照してください。

### 5.3.1.6. Strm\_SetCallbackImageAcquired

TeliCamAPI 内部のストリームリクエストリングバッファの内容を正常受信した画像データで更新した時に呼び出すコールバック関数を TeliCamAPI に登録します。

#### [構文]

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

#### [パラメータ]

パラメータ	内 容
<i>hStrm</i>	[in] ストリームインターフェースのストリームハンドルです。
<i>pvContext</i>	[in] コールバック関数を実行するときに引数として渡すオブジェクトへのポインタです。
<i>pFunc</i>	[in] コールバック関数です。

#### [コールバック関数(*pFunc*) パラメータ]

パラメータ	内 容
<i>hRcvCam</i>	[out] 受信画像を送信したカメラのカメラハンドルです。
<i>hRcvStrm</i>	[out] 受信画像を送信したストリームのストリームハンドルです。
<i>psImageInfo</i>	[out] カメラから受信した画像とその付随情報です。 <a href="#">CAM_IMAGE_INFO 構造体</a> の説明は <a href="#">Strm_ReadCurrentImage()</a> をご覧ください。
<i>uiBufferIndex</i>	[out] <a href="#">psImageInfo</a> のデータが保存されているバッファのストリームリクエストリングバッファ内インデックスです。
<i>pvRcvContext</i>	[out] <a href="#">pvContext</a> で指定したオブジェクトへのポインタです。

#### [戻り値]

実行結果を返します。 戻り値は、[5.9. ステータスコード](#) を参照してください。

#### [備考]

本関数は、[Strm\\_OpenSimple\(\)](#) でオープンしたストリームインターフェースに使用できます。

コールバック関数内の処理は、できるだけ短くしてください。

コールバック関数の処理中に次の画像ストリームを受信した場合、実行中のコールバック関数が終了した後に次の画像用のコールバック関数が実行されます。

コールバック関数の処理中に複数の画像ストリームを受信したときは、実行中のコールバック関数が終了した後に、最後に受信した画像用のコールバック関数だけが実行されます。

また、ストリームリクエストリングバッファがいっぱいになると、ストリームリクエスト（画像）自体が保存せれずに破棄されます。このとき、[Strm\\_SetCallbackBufferBusy\(\)](#) によりコールバック関数が登録されている時は、コールバック関数が呼び出されます。

TeliCamAPI.h をインクルードする必要があります。

#### [コード例]

```
C++

CAM_HANDLE          s_hCam;
CAM_STRM_HANDLE     s_hStrm;

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

void 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;

        usleep(50000); // For sample
    }

    return 0;
}

```

### 5.3.1.7. Strm\_SetCallbackImageError

ストリームを正常に受信できず、TeliCamAPI 内部のストリームリクエストリングバッファの内容がエラー更新された時に呼び出すコールバック関数を TeliCamAPI に登録します。

#### [構文]

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

#### [パラメータ]

パラメータ		内 容
<i>hStrm</i>	[in]	ストリームインターフェースのストリームハンドルです。
<i>pvContext</i>	[in]	コールバック関数を実行するときに引数として渡すオブジェクトへのポインタです。
<i>pFunc</i>	[in]	コールバック関数です。

#### [コールバック関数(*pFunc*) パラメータ]

パラメータ		内 容
<i>hRcvCam</i>	[out]	受信画像を送信したカメラのカメラハンドルです。
<i>hRcvStrm</i>	[out]	受信画像を送信したストリームのストリームハンドルです。
<i>uiErrorStatus</i>	[out]	画像ストリーム受信時のエラーステータスコードです。
<i>uiBufferIndex</i>	[out]	受信エラー情報が保存されているバッファのストリームリクエストリングバッファ内インデックスです。
<i>pvRcvContext</i>	[out]	<a href="#">pvContext</a> で指定したオブジェクトへのポインタです。

#### [戻り値]

実行結果を返します。 戻り値は、[5.9. ステータスコード](#) を参照してください。

#### [備考]

本関数は、[Strm\\_OpenSimple\(\)](#) でオープンしたストリームインターフェースに使用できます。

コールバック関数内の処理は、できるだけ短くしてください。

コールバック関数の処理中に次の画像ストリームを受信した場合、実行中のコールバック関数が終了した後に次の画像用のコールバック関数が実行されます。

コールバック関数の処理中に複数の画像ストリームを受信したときは、実行中のコールバック関数が終了した後に、最後に受信した画像用のコールバック関数だけが実行されます。

また、ストリームリクエストリングバッファがいっぱいになると、ストリームリクエスト（画像）自体が保存せられずに破棄されます。 このとき、[Strm\\_SetCallbackBufferBusy\(\)](#) によりコールバ

---

ック関数が登録されている時は、コールバック関数が呼び出されます。

TeliCamAPI.h をインクルードする必要があります。

**[コード例]**

[Strm\\_SetCallbackImageAcquired\(\)](#) の[コード例](#)を参照してください。



### 5.3.1.8. Strm\_SetCallbackBufferBusy

画像ストリーム受信時、TeliCamAPI 内部のストリームリクエストリングバッファ書き込み先がロック中のために、最新画像データを保有したストリームリクエストを格納できなかった場合に呼び出すコールバック関数を TeliCamAPI に登録します。

#### [構文]

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

#### [パラメータ]

パラメータ	内 容
<i>hStrm</i> [in]	ストリームインターフェースのストリームハンドルです。
<i>pvContext</i> [in]	コールバック関数を実行するときに引数として渡すオブジェクトへのポインタです。
<i>pFunc</i> [in]	コールバック関数です。

#### [ コールバック関数(*pFunc*) パラメータ ]

パラメータ	内 容
<i>hRcvCam</i> [out]	受信画像を送信したカメラのカメラハンドルです。
<i>hRcvStrm</i> [out]	受信画像を送信したストリームのストリームハンドルです。
<i>uiBufferIndex</i> [out]	ロック中で格納できなかったストリームリクエストリングバッファのバッファインデックスです。
<i>pvRcvContext</i> [out]	<a href="#">pvContext</a> で指定したオブジェクトへのポインタです。

#### [戻り値]

実行結果を返します。 戻り値は、[5.9. ステータスコード](#) を参照してください。

#### [備考]

本関数は、[Strm\\_OpenSimple\(\)](#) でオープンしたストリームインターフェースに使用できます。

バッファビジーエラーは、[Strm\\_LockBuffer\(\)](#) によりストリームリクエストリングバッファの書き込み先がロックされているか、コールバック関数の処理コストが大きく、書き込み先のバッファがビジー状態の時に発生します。

コールバック関数内の処理は、できるだけ短くしてください。

コールバック関数の処理中に次の画像ストリームを受信した場合、実行中のコールバック関数が終了した後に次の画像用のコールバック関数が実行されます。

コールバック関数の処理中に複数の画像ストリームを受信したときは、実行中のコールバック関数が終了した後に、最後に受信した画像用のコールバック関数だけが実行されます。

---

また、ストリームリクエストリングバッファがいっぱいになると、ストリームリクエスト（画像）自体が保存せれずに破棄されます。このとき、[Strm\\_SetCallbackBufferBusy\(\)](#) によりコールバック関数が登録されている時は、コールバック関数が呼び出されます。

TeliCamAPI.h をインクルードする必要があります。

**[コード例]**

[Strm\\_SetCallbackImageAcquired\(\)](#) の[コード例](#)を参照してください。

### 5.3.2. 低水準関数

カメラのパフォーマンスを最大限に活用したり、高度なアプリケーションを作成したい場合に使用します。 ストリームリクエストキューの管理などを行う必要があります。

詳細は、[4.1.5.2 低水準関数を使用したストリームデータ（画像データ）の取得](#) を参照してください。

#### 5.3.2.1. Strm\_Open

画像取得用のストリームインターフェースをオープンします。

##### [構文]

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

##### [パラメータ]

パラメータ		内 容
<i>hCam</i>	[in]	カメラのカメラハンドルです。
<i>hCmpEvt</i>	[in]	ストリームを受信し、ストリームリクエストリングバッファが更新されたことを通知するシグナルオブジェクトのハンドルです。 シグナルオブジェクトは、 <a href="#">Sys_CreateSignal()</a> で作成します。 通知する必要がある場合は NULL を指定してください。
<i>puiMaxPayloadSize</i>	[in,out]	1 つのストリームリクエストで受信の可能性がある1ブロックの最大ペイロードサイズ（画像サイズ）が格納されている変数へのポインタです。（単位：byte）  ペイロードサイズは画像のサイズやフォーマットなどによって変わります。  0 を指定した場合は、 <a href="#">GetCamStreamPayloadSize()</a> で得られる値が使用されます。  通常は、0 または <a href="#">GetCamStreamPayloadSize()</a> で取得した値を指定してください。
<i>phStrm</i>	[out]	オープンしたストリームインターフェースのストリームハンドルを格納する変数へのポインタです。

パラメータ	内 容
<i>uiMaxPacketSize</i> [in]	<p>ドライバが受け取るパケットの最大サイズです。（単位：byte。）この引数は省略可能です。</p> <p>0 を設定するか指定を省略すると、以下の値が使用されます。</p> <p>USB3 Visionカメラ : 65536 byte</p> <p>GigE Visionカメラ : 1500 byte</p> <p>GigE Vision カメラでジャンボフレームを設定している場合は、ジャンボフレームの設定値からイーサネットヘッダを除いた値を指定してください。</p> <p>ネットワークアダプタがイーサネットヘッダを含んだ値をジャンボフレームサイズとして示している場合は、イーサネットヘッダサイズ（14byte）を減算した値を指定する必要があります。（ジャンボフレームの値が 9014 と表示されている場合は、通常イーサネットヘッダを含んだ値が表示されています。この場合、<i>uiMaxPacketSize</i> には 9000 を指定してください。）</p> <p>USB3 Visionカメラの場合は、特別な理由がない限り 0 を指定するか、値の指定を省略してください。</p>

#### [戻り値]

実行結果を返します。 戻り値は、[5.9. ステータスコード](#) を参照してください。

#### [備考]

本関数は、カメラのパフォーマンスの最大限の活用、独自の画像取得シーケンスの構築が求められる場合にご使用ください。低水準関数をご使用の場合、ストリームリクエストを操作する処理をユーザアプリケーションに記述していただく必要があります。

ユーザコードを簡潔にしたい場合は、本関数の代わりに高水準関数 [Strm\\_OpenSimple\(\)](#) を使ってください。

低水準関数では TeliCamAPI 内部のストリーム受信待機キューとストリーム受信完了キューに対してストリームリクエストを投入、取得することによりユーザアプリケーションはカメラから受信した画像を受け取ります。

TeliCamAPI は、画像データを受信すると、シグナルオブジェクト [hCmpEvt](#) をシグナル状態にセットします。

TeliCamAPI.h をインクルードする必要があります。

#### [コード例]

C++	
<pre>const int STRM_REQUEST_NUM = 5;  CAM_API_STATUS      uiStatus; uint32_t            uiNum, uiPyldSize, uiRcvSize, i; SIGNAL_HANDLE       hStrmCmpEvt = (SIGNAL_HANDLE)NULL; CAM_HANDLE          hCam = (CAM_HANDLE)NULL; CAM_STRM_HANDLE      hStrm = (CAM_STRM_HANDLE)NULL; void*               pvPayloadBuf = NULL; CAM_STRM_REQUEST_HANDLE hStrmReq[STRM_REQUEST_NUM]; void*               pvRcvPayloadBuf = NULL; CAM_STRM_REQUEST_HANDLE hRcvStrmReq = (CAM_STRM_REQUEST_HANDLE)NULL;</pre>	

```

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

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

do
{
    // 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, (SIGNAL_HANDLE)NULL);
    if (uiStatus != CAM_API_STS_SUCCESS)
        return -1;

    // Create completion event object for stream.
    uiStatus = Sys_CreateSignal(&hStrmCmpEvt);
    if (uiStatus != CAM_API_STS_SUCCESS)
        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 *)malloc(uiPyldSize * STRM_REQUEST_NUM);
    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.
    uiStatus = Sys_WaitForSignal(hStrmCmpEvt, 2000);
    if (uiStatus != CAM_API_STS_SUCCESS)
        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, (uint32_t)hRcvStrmReq);

    // TODO: add your handling code here.

    // Re-enqueue a StreamRequest to stream wait queue.
    uiStatus = Strm_EnqueueRequest(hStrm, hRcvStrmReq);
    if (uiStatus != CAM_API_STS_SUCCESS)
        return -1;
}
while(false);

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

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

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

    // Close stream interface.

```

---

```
    uiStatus = Strm_Close(hStrm);
    if (uiStatus != CAM_API_STS_SUCCESS)
        printf("Strm_Close error! (0x%x)", uiStatus);
}

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

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

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

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

### 5.3.2.2. Strm\_CreateRequest

画像ストリームデータ取得に使用するストリームリクエストを作成します。

ユーザアプリケーションが予めメモリ割り付けした画像バッファを、ストリームリクエスト構造体のメンバ変数用として本関数実行時に引数で指定してください。

#### [構文]

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

#### [パラメータ]

パラメータ	内 容
<i>hStrm</i> [in]	ストリームインターフェースのストリームハンドルです。
<i>pvPayloadBuf</i> [in]	受信した画像データを格納するバッファへのポインタです。 バッファは、 <i>uiPayloadSize</i> バイトの領域をあらかじめ確保しておく必要があります。
<i>uiPayloadSize</i> [in]	1フレーム分の最大ペイロードサイズ（画像サイズ）です。（単位：byte） 通常は、 <a href="#">Strm_Open()</a> をコールした時に <a href="#">puiMaxPayloadSize</a> に指定（取得）したと同じ値を指定してください。
<i>phStrmRequest</i> [out]	作成されたストリームリクエストのハンドルを格納する変数へのポインタです。

#### [戻り値]

実行結果を返します。 戻り値は、[5.9. ステータスコード](#) を参照してください。

#### [備考]

本関数は、[Strm\\_Open\(\)](#) でオープンしたストリームインターフェースに使用できます。

[Strm\\_OpenSimple\(\)](#) でオープンしたストリームインターフェースでは正常に動作しません。

作成したストリームリクエストは [Strm\\_EnqueueRequest\(\)](#) を実行してストリーム受信待機キューに投入してください。TeliCamAPI は画像を受信するたびに、ストリーム受信待機キューからストリームリクエストをとりだして受信した画像データを保存し、ストリーム受信完了キューに投入する処理を自動実行します。

[Strm\\_DequeueRequest\(\)](#) を実行すると、画像データが保存されたストリームリクエストがストリーム受信完了キューから取り出されます。

画像の縦横画素数や出力フォーマットを変更すると、画像データのサイズが変わります。ストリームリクエスト内の画像バッファを変更する機能は提供していないので、画像データサイズが変更された場合は既存のストリームリクエストを解放し、ストリームリクエストを再作成してください。

ストリームリクエストを解放する前に [pvPayloadBuf](#) で指定したバッファを解放すると予期せぬエラーが発生することがあります。[pvPayloadBuf](#) で指定したバッファを解放する場合は以下の手順で行ってください。



- 
1. [Strm\\_Stop\(\)](#) を実行してストリーム転送を停止。
  2. [Strm\\_Flash\\_WaitQueue\(\)](#) を実行し、ストリームリクエストの受信処理中止、ストリーム受信待機キューに投入されているすべてのストリームリクエストをストリーム受信完了キューへ移動。
  3. ストリーム受信完了キューが空になるまで [Strm\\_DequeueRequest\(\)](#) を繰り返し実行し、ストリーム受信完了キューからストリームリクエストを取り出し。
  4. [Strm\\_ReleaseRequest\(\)](#) を実行し、ストリームリクエストを解放。
  5. pvPayloadBuf で指定したバッファを解放。

TeliCamAPI.h をインクルードする必要があります。

**[コード例]**

[Strm\\_Open\(\)](#) の[コード例](#)を参照してください。

---

### 5.3.2.3. Strm\_ReleaseRequest

ストリームリクエストを解放します。

#### [構文]

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

#### [パラメータ]

パラメータ		内 容
<i>hStrm</i>	[in]	ストリームインターフェースのストリームハンドルです。
<i>hStrmRequest</i>	[in]	ストリームリクエストのハンドルです。

#### [戻り値]

実行結果を返します。 戻り値は、[5.9. ステータスコード](#) を参照してください。

#### [備考]

本関数は、[Strm\\_Open\(\)](#) でオープンしたストリームインターフェースに使用できます。

[Strm\\_OpenSimple\(\)](#) でオープンしたストリームインターフェースでは正常に動作しません。

[Strm\\_CreateRequest\(\)](#) で作成したストリームリクエストは、アプリケーションを終了する前に必ず本関数により解放してください。

TeliCamAPI.h をインクルードする必要があります。

#### [コード例]

[Strm\\_Open\(\)](#) の[コード例](#)を参照してください。

---

### 5.3.2.4. Strm\_EnqueueRequest

ストリームリクエストを、ストリーム受信待機キューに投入します。

#### [構文]

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

#### [パラメータ]

パラメータ		内 容
<i>hStrm</i>	[in]	ストリームインターフェースのストリームハンドルです。
<i>hStrmRequest</i>	[in]	ストリームリクエストのハンドルです。

#### [戻り値]

実行結果を返します。 戻り値は、[5.9. ステータスコード](#) を参照してください。

#### [備考]

本関数は、[Strm\\_Open\(\)](#) でオープンしたストリームインターフェースに使用できます。

[Strm\\_OpenSimple\(\)](#) でオープンしたストリームインターフェースでは正常に動作しません。

画像ストリームデータを受信すると、TeliCamAPI はストリーム受信待機キューの先頭からストリームリクエストを取り出し、そのストリームリクエストに画像データを格納します。受信が完了すると、TeliCamAPI はストリームリクエストをストリーム受信待機キューからストリーム受信完了キューに移します。

画像ストリームデータを取りこぼしなく連続して取り込むために、ストリーム受信待機キューに常に複数のストリームリクエストが存在するようストリームリクエストの補充を行ってください。

TeliCamAPI.h をインクルードする必要があります。

#### [コード例]

[Strm\\_Open\(\)](#) の[コード例](#)を参照してください。

### 5.3.2.5. Strm\_DequeueRequest

ストリーム受信完了キューから、ストリームリクエストを一つ取り出します。

#### [構文]

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

#### [パラメータ]

パラメータ		内 容
<i>hStrm</i>	[in]	ストリームインターフェースのストリームハンドルです。
<i>phStrmRequest</i>	[out]	取り出したストリームリクエストを格納する変数へのポインタです。
<i>ppvPayloadBuf</i>	[out]	取り出したストリームリクエストの、ペイロード（画像）データが格納されているバッファへのポインタを格納する変数へのポインタです。
<i>puiPayloadSize</i>	[out]	取り出したストリームリクエストの、ペイロード（画像）サイズを格納する変数へのポインタです。

#### [戻り値]

実行結果を返します。 戻り値は、[5.9. ステータスコード](#) を参照してください。

#### [備考]

本関数は、[Strm\\_Open\(\)](#) でオープンしたストリームインターフェースに使用できます。

[Strm\\_OpenSimple\(\)](#) でオープンしたストリームインターフェースでは正常に動作しません。

本関数は、ストリーム受信完了キューに移動された古いストリームリクエストから順番にストリームリクエストを取り出します。

ストリーム受信完了キューが空の場合は、CAM\_API\_STS\_EMPTY\_COMPLETE\_QUEUE を戻り値として返します。

TeliCamAPI.h をインクルードする必要があります。

#### [コード例]

[Strm\\_Open\(\)](#) の[コード例](#)を参照してください。

---

### 5.3.2.6. Strm\_FlushWaitQueue

画像ストリームの受信処理を中止し、ストリーム受信待機キューに投入されているすべてのストリームリクエストをストリーム受信完了キューに移動させます。

#### [構文]

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

#### [パラメータ]

パラメータ	内 容
<i>hStrm</i> [in]	ストリームインターフェースのストリームハンドルです。

#### [戻り値]

実行結果を返します。 戻り値は、[5.9. ステータスコード](#) を参照してください。

#### [備考]

本関数は、[Strm\\_Open\(\)](#) でオープンしたストリームインターフェースに使用できます。

[Strm\\_OpenSimple\(\)](#) でオープンしたストリームインターフェースでは正常に動作しません。

[Strm\\_Close\(\)](#) でストリームインターフェースをクローズするまでに、ストリーム受信待機キューとストリーム受信完了キューは空にしておく必要があります。本関数はストリーム受信待機キュー内のすべてのストリームリクエストをストリーム受信完了キューに移動させます。ストリーム受信完了キューが空になるまで [Strm\\_DequeueRequest\(\)](#) を繰り返し実行することによりストリーム受信完了キューを空にすることができます。

[Strm\\_DequeueRequest\(\)](#) で取り出したストリームリクエストが [Strm\\_FlushWaitQueue\(\)](#) 実行により移動されたストリームリクエストの場合、有効な画像データが保存されていないことを示すために CAM\_API\_STS\_FLUSH\_REQUESTED が戻り値として戻されます。

TeliCamAPI.h をインクルードする必要があります。

#### [コード例]

[Strm\\_Open\(\)](#) の[コード例](#)を参照してください。

### 5.3.2.7. Strm\_GetStrmReqInfo

ストリーム受信完了キューから取り出したストリームリクエストの情報を取得します。

#### [構文]

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

#### [パラメータ]

パラメータ	内 容
<i>hStrm</i> [in]	ストリームインターフェースのストリームハンドルです。
<i>hStrmRequest</i> [in]	情報を取得するストリームリクエストのハンドルです。
<i>psStrmReqInfo</i> [out]	取得した情報を格納するバッファへのポインタです。

#### [CAM\_STRM\_REQUEST\_INFO 構造体]

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

CAM_STRM_REQUEST_INFO	内 容
<i>sU3vInfo</i> [out]	USB3 Vision Camera のとき、ストリームリクエストの情報です。
<i>sGevInfo</i> [out]	GigE Vision Camera のとき、ストリームリクエストの情報です。

#### [U3V\_STRM\_REQUEST\_INFO 構造体]

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

U3V_STRM_REQUEST_INFO	内 容
<i>pvLeader</i> [out]	ストリームのリーダーへのポインタです。
<i>pvPayload</i> [out]	ストリームのペイロードへのポインタです。
<i>pvTrailer</i> [out]	ストリームのトレーラへのポインタです。
<i>uiPayloadSize</i> [out]	実際に受信したペイロード（画像）のサイズです。

[ GEV\_STRM\_REQUEST\_INFO 構造体 ]

```
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;
```

GEV_STRM_REQUEST_INFO		内 容
pvLeader	[out]	ストリームのリーダーへのポインタです。
pvPayload	[out]	ストリームのペイロードへのポインタです。
pvTrailer	[out]	ストリームのトレーラへのポインタです。
uiNumOfPayloadPacket	[out]	実際に受信したペイロードパケット数です。
uiPayloadSize	[out]	実際に受信したペイロード（画像）のサイズです。
uiNumOfResendPacket	[out]	リクエストが完了するまでに行われた再送要求数です。

[戻り値]

実行結果を返します。 戻り値は、[5.9. ステータスコード](#) を参照してください。

[備考]

本関数は、[Strm\\_Open\(\)](#) でオープンしたストリームインターフェースに使用できます。

[Strm\\_OpenSimple\(\)](#) でオープンしたストリームインターフェースでは正常に動作しません。

本関数を使用するためには、USB3 Vision 規格および GigE Vision 規格の知識が必要です。 ストリームのリーダー、トレーラを使用するためには、別の構造体にキャストして使用する必要があります。

通常は、本関数を使用する必要はありません。 特別な理由がない限り、本関数は使用しないでください。

TeliCamAPI.h をインクルードする必要があります。

---

### 5.3.3. 共通関数

高水準関数と低水準関数共通の関数です。

#### 5.3.3.1. Strm\_Close

画像取得用のストリームインターフェースをクローズします。

##### [構文]

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

##### [パラメータ]

パラメータ	内 容
<i>hStrm</i> [in]	ストリームインターフェースのストリームハンドルです。

##### [戻り値]

実行結果を返します。 戻り値は、[5.9. ステータスコード](#) を参照してください。

##### [備考]

[Strm\\_Open\(\)](#) および [Strm\\_OpenSimple\(\)](#) でストリームインターフェースをオープンした場合ストリームの使用が終了した後に、必ず本関数を実行してストリームインターフェースをクローズさせてください。

別のスレッドで使用しているストリームをクローズすると、そのスレッドでエラーが発生する可能性があります。すべてのスレッドでストリームの使用が終了してから本関数を実行してください。

TeliCamAPI.h をインクルードする必要があります。

##### [コード例]

[Strm\\_OpenSimple\(\)](#) の[コード例](#)を参照してください。



### 5.3.3.2. Strm\_Start

画像ストリームの転送開始をカメラに要求します。

#### [構文]

```
CAM_API_STATUS Strm_Start (  
    CAM_STRM_HANDLE      hStrm,  
    CAM\_ACQ\_MODE\_TYPE     eAcqMode = CAM_ACQ_MODE_CONTINUOUS  
);
```

#### [パラメータ]

パラメータ		内 容
<i>hStrm</i>	[in]	ストリームインターフェースのストリームハンドルです。
<i>eAcqMode</i>	[in]	映像ストリーム転送モードです。 省略した場合は、連続映像ストリーム転送モード (CAM_ACQ_MODE_CONTINUOUS) になります。

#### [ CAM\_ACQ\_MODE\_TYPE 型 ]

```
typedef enum  
{  
    CAM_ACQ_MODE_CONTINUOUS      = 8,    // Continuous  
    CAM_ACQ_MODE_MULTI_FRAME     = 9,    // MultiFrame  
    CAM_ACQ_MODE_IMAGE_BUFFER_READ = 10,  // Camera Image Buffer Mode  
} CAM_ACQ_MODE_TYPE;
```

#### [戻り値]

実行結果を返します。 戻り値は、[5.9. ステータスコード](#) を参照してください。

#### [備考]

以下の表に各画像取得モードの説明を記載します。

使用可能な画像取得モードに関してはカメラの取扱説明書で確認してください。

CAM_ACQ_MODE_TYPE	内 容
CAM_ACQ_MODE_CONTINUOUS	<a href="#">Strm_Stop()</a> が実行されるまで継続的に画像を撮像・転送するモードです。
CAM_ACQ_MODE_MULTI_FRAME	<a href="#">SetCamAcquisitionFrameCount()</a> で設定した枚数の画像を撮像・転送するモードです。  設定された枚数の画像取得が完了して画像転送が停止していても、 <a href="#">Strm_Stop()</a> を実行してストリームをクローズする必要があります。
CAM_ACQ_MODE_IMAGE_BUFFER_READ	<a href="#">Strm_Stop()</a> が実行されるまで継続的に画像を撮像し、カメラ内のバッファに保存するモードです。  <a href="#">ExecuteCamImageBuffer()</a> を実行するとカメラ内のバッファから画像が転送されます。 詳細は <a href="#">5.5.9.ImageBuffer</a> を参照してください。

CAM\_ACQ\_MODE\_IMAGE\_BUFFER\_READ を使用する場合は、[SetCamImageBufferMode\(\)](#)でイメージバッファモードを ON に、CAM\_ACQ\_MODE\_IMAGE\_BUFFER\_READ 以外の時は OFF に設定してください。

---

本関数を実行すると、GenICam アクセスを有効に設定している場合は TLPParamsLocked が 1 に設定されます。

TeliCamAPI.h をインクルードする必要があります。

**[コード例]**

[Strm\\_OpenSimple\(\)](#) の[コード例](#)を参照してください。

---

### 5.3.3.3. Strm\_Stop

画像取得用のストリーム転送停止をカメラに要求します。

#### [構文]

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

#### [パラメータ]

パラメータ	内 容
<i>hStrm</i> [in]	ストリームインターフェースのストリームハンドルです。

#### [戻り値]

実行結果を返します。 戻り値は、[5.9. ステータスコード](#) を参照してください。

#### [備考]

本関数を実行すると、GenICam アクセスを有効に設定している場合は TLPParamsLocked が 0 に設定されます。

TeliCamAPI.h をインクルードする必要があります。

#### [コード例]

[Strm\\_OpenSimple\(\)](#) の[コード例](#)を参照してください。

## 5.4. カメライベント（メッセージ）通知関数

TeliCamAPI は高水準関数と低水準関数の2種類のカメライベント通知関数を提供しています。  
詳細は [4.1.6.カメライベント（メッセージ）コントロール](#) を参照してください。

### 5.4.1. 高水準関数

高水準関数はカメライベント受信処理の大半を TeliCamAPI 内でバックグラウンド実行することにより、ユーザアプリケーションは簡単なコード記述でカメライベント情報を取得できる関数群です。

カメラオープン時に、GenICam アクセスを無効に設定している場合は、高水準関数は使用できません。

詳細は、[4.1.6.1. 高水準関数を使用したカメライベント（メッセージ）の取得](#) を参照してください。

#### 5.4.1.1. Evt\_OpenSimple

カメライベント（メッセージ）取得用のイベントインターフェースをオープンし、カメライベント（メッセージ）を取得するためのイベントリクエストとイベントリクエストリングバッファを作成します。

イベントリクエストとイベントリクエストリングバッファの管理は、TeliCamAPI 内部で行われます。

##### [構文]

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

##### [パラメータ]

パラメータ		内 容
<i>hCam</i>	[in]	カメラのカメラハンドルです。
<i>phEvt</i>	[out]	オープンしたイベントインターフェースのイベントハンドルを格納する変数へのポインタです。
<i>uiApiBufferCount</i>	[in]	TeliCamAPI 内部に作成するイベントリクエストリングバッファのサイズです。この引数は省略可能です。 0 を設定すると、推奨値が自動的に指定されます。 設定範囲は、3 から30 です。 省略した場合は、DEFAULT_API_BUFFER_CNT (8) が指定されます。

##### [戻り値]

実行結果を返します。 戻り値は、[5.9. ステータスコード](#) を参照してください。

##### [備考]

本関数は、TeliCamAPI 内部にカメライベントデータを一時保存するイベントリクエスト構造体とカメライベントデータが格納されたイベントリクエスト構造体を一時保管するイベントリクエストリングバッファを作成し、カメライベント（メッセージ）取得用のイベントインターフェースをオープンします。

カメライベント発生通知を受けた後は、[GenICAM 関数](#)を使用して発生したカメライベント（メッセ

ージ)の付随情報を取得することができます。

1つのカメラに複数のカメライベント(メッセージ)通知を設定すると、多数のカメライベントが短時間に発生し、処理が間に合わなくなったり、イベントリクエストリングバッファがいっぱいになり、バッファビジーエラーが発生する恐れがあります。バッファビジーエラーが発生しても通知はされません。連続的にストリームデータ(画像データ)を取得する場合は、複数のカメライベント(メッセージ)通知を設定しないことを推奨いたします。

TeliCamAPI.h をインクルードする必要があります。

#### [コード例]

```
C++

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

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

do
{
    // 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, (SIGNAL_HANDLE)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.
    uiStatus = Sys_CreateSignal(&hFrmTrgEvt);
    if (uiStatus != CAM_API_STS_SUCCESS)
        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.
uiStatus = Sys_CreateSignal(&hStrmCmpEvt);
if (uiStatus != CAM_API_STS_SUCCESS)
    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 *)malloc(uiPyldSize);
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.
    uiStatus = Sys_WaitForSignal(hFrmTrgEvt, 2000);
    if (uiStatus != CAM_API_STS_SUCCESS)
        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 = %llu\n", i, (long long unsigned int)llVal);

    // Wait for receiving image completion event.
    uiStatus = Sys_WaitForSignal(hStrmCmpEvt, 2000);
    if (uiStatus != CAM_API_STS_SUCCESS)
        return -1;
}
}
while(false);

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

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

    // Close event interface.
    uiStatus = Evt_Close(hEvent);
}

```

---

```

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

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

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

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

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

    // Terminate system.
    Sys_Terminate();

```

### 5.4.1.2. Evt\_Activate

指定されたカメライベントを有効に設定します。

#### [構文]

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

#### [パラメータ]

パラメータ	内 容
hEvt	[in] イベントインターフェースのイベントハンドルです。
pszEvtName	[in] カメライベント（メッセージ）のノード名（レジスタ名）です。
hCmpEvt	[in] カメライベント（メッセージ）を受信したことを通知するシグナルオブジェクトのハンドルです。 シグナルオブジェクトは、 <a href="#">Sys_CreateSignal()</a> で作成します。 通知する必要がなければ NULL を指定してください。

#### [戻り値]

実行結果を返します。 戻り値は、[5.9. ステータスコード](#) を参照してください。

#### [備考]

本関数は、[Evt\\_OpenSimple\(\)](#) でオープンしたイベントインターフェースに使用できます。

本関数により有効化されたカメライベント（メッセージ）の発生通知を受け取った後は、[GenICAM 関数](#)を使用して発生したカメライベントの付随情報を取得することができます。

TeliCamAPI はカメライベントデータを受信するたびに本関数で指定されたシグナルオブジェクト [hCmpEvt](#) をシグナル状態にセットします。

ただし、ユーザアプリケーションがシグナルオブジェクト [hCmpEvt](#) を非シグナル状態にリセット（[Sys\\_WaitForSignal\(\)](#) 実行）する前に同じカメライベントを受信した場合、そのカメライベントに対するシグナルオブジェクト [hCmpEvt](#) の設定は行いません。

以下の表に [pszEvtName](#) に指定できるカメライベント名（EventSelector ノードに設定できる値）と付随情報取得に使用する GenICam ノード名の一部を示します。

指定できるカメライベント名に関してはカメラの取扱説明書をご覧ください。

カメライベント名	内 容	カメライベント情報ノード名
FrameTrigger	トリガ受付	EventFrameTriggerTimestamp
FrameTriggerError	トリガエラー	EventFrameTriggerErrorTimestamp
FrameTriggerWait	トリガ受付待ち開始	EventFrameTriggerWaitTimestamp
FrameTransferStart	映像転送開始	EventFrameTransferStartTimestamp
FrameTransferEnd	映像転送終了	EventFrameTransferEndTimestamp
ExposureStart	露光開始	EventExposureStartTimestamp
ExposureEnd	露光終了	EventExposureEndTimestamp
Timer0Start	Timer0 開始	EventTimer0StartTimestamp
Timer0End	Timer0 終了	EventTimer0EndTimestamp



カメライベント名	内 容	カメライベント情報ノード名
ALCLatestInformation	ALC 動作更新時の値	EventALCLatestInformationTimestamp
		EventALCLatestInformationTotalLuminance
		EventALCLatestInformationAverageLuminance
		EventALCLatestInformationExposureTime
		EventALCLatestInformationGain
ALCConverged	ALC 動作収束時の値	EventALCConvergedTimestamp
		EventALCConvergedLuminanceTotal
		EventALCConvergedLuminanceAverage
		EventALCConvergedExposureTime
		EventALCConvergedGain

なお、GenICAM 関数 [Nd\\_GetNode\(\)](#) で、[pszEvtName](#) で設定したノード名の前に前置詞“Event”をつけたノード名（EventFrameTrigger、EventFrameTriggerError など）のノードにアクセスすると、シグナルオブジェクト [hCmpEvt](#) がシグナル状態にセットされます。その際は、[Sys\\_ResetSignal\(\)](#) でシグナルオブジェクト [hCmpEvt](#) を非シグナル状態にリセットする必要がありますのでご注意ください。

特別な理由がない限り、前置詞“Event”をつけたノード名のノードにアクセスしないでください。

カメラによっては、レジスタの設定順序の制限によりエラーになる場合がありますので、ご注意ください。一部の GigE-Vision カメラでは、FrameTrigger イベントは TriggerMode = On の時にしか設定できません。

TeliCamAPI.h をインクルードする必要があります。

#### [コード例]

[Evt\\_OpenSimple\(\)](#) の[コード例](#)を参照してください。

---

### 5.4.1.3. Evt\_Deactivate

指定されたカメライベント（メッセージ）を無効に設定します。

#### [構文]

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

#### [パラメータ]

パラメータ		内 容
<i>hEvt</i>	[in]	イベントインターフェースのイベントハンドルです。
<i>pszEvtName</i>	[in]	解除するカメライベント（メッセージ）のレジスタ名（ノード名）です。

#### [戻り値]

実行結果を返します。 戻り値は、[5.9. ステータスコード](#) を参照してください。

#### [備考]

本関数は、[Evt\\_OpenSimple\(\)](#) でオープンしたイベントインターフェースに使用できます。

[Evt\\_Activate](#) 関数で登録したカメライベント（メッセージ）は、必ず本関数で無効に設定してからイベントインターフェースをクローズしてください。

TeliCamAPI.h をインクルードする必要があります。

#### [コード例]

[Evt\\_OpenSimple\(\)](#) の[コード例](#)を参照してください。

## 5.4.2. 低水準関数

カメラのパフォーマンスを最大限に活用したり、高度なアプリケーションを作成したい場合に使用します。 イベントリクエストキューの管理などを行う必要があります。 現在のバージョンでは、GigE Vision カメラは使用できません。

詳細は、[4.1.6.2. 低水準関数を使用したカメライベント（メッセージ）の取得](#) を参照してください。

### 5.4.2.1. Evt\_Open

カメライベント（メッセージ）取得用のイベントインターフェースをオープンします。

#### [構文]

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

#### [パラメータ]

パラメータ		内 容
<i>hCam</i>	[in]	カメラのカメラハンドルです。
<i>phEvt</i>	[out]	オープンしたイベントインターフェースのイベントハンドルを格納する変数へのポインタです。
<i>puiMaxPayloadSize</i>	[in,out]	1 つのイベントリクエストで受信の可能性がある1ブロックの最大ペイロードサイズが格納されている変数へのポインタです。 (単位: byte) 0 を指定した場合は、TeliCamAPI 内部で自動的にペイロードサイズを設定し、設定した値を <i>puiMaxPayloadSize</i> に格納します。 通常は 0 を指定してください。
<i>hCmpEvt</i>	[in]	カメライベント（メッセージ）を受信したことを通知するシグナルオブジェクトのハンドルです。 シグナルオブジェクトは、 <a href="#">Sys_CreateSignal()</a> で作成します。 通知する必要がなければ NULL を指定してください。

#### [戻り値]

実行結果を返します。 戻り値は、[5.9. ステータスコード](#) を参照してください。

#### [備考]

1 つのカメラに複数のカメライベント（メッセージ）通知を設定すると、非常に短い間隔でカメライベント（メッセージ）通知が発生します。ご注意ください。

カメライベント（メッセージ）を受信してすると、TeliCamAPI はイベント受信待機キューからのイベントリクエストの取り出し、取り出したイベントリクエストへの受信したカメライベントデータの書き込み、カメライベントデータを書き込んだイベントリクエストのイベント受信完了キューへの投入、を行った後、本関数で指定されたシグナルオブジェクト [hCmpEvt](#) をシグナル状態にセットします。

[Evt\\_FlushRequestQueue\(\)](#) を実行してイベントリクエストをイベント受信完了キューに移動した

ときはシグナルオブジェクト [hCmpEvt](#) はシグナル状態にセットされません。

TeliCamAPI.h をインクルードする必要があります。

#### [コード例]

```
C++

// Only USB Vision Camera.
CAM_API_STATUS          uiStatus;
uint32_t                uiNum, uiPyldSize, i;
CAM_HANDLE              hCam = (CAM_HANDLE)NULL;
CAM_STRM_HANDLE         hStrm = (CAM_STRM_HANDLE)NULL;
CAM_EVT_HANDLE          hEvent = (CAM_EVT_HANDLE)NULL;
SIGNAL_HANDLE           hStrmCmpEvt = (SIGNAL_HANDLE)NULL;
SIGNAL_HANDLE           hEventCmpEvt = (SIGNAL_HANDLE)NULL;
void*                   pvPayloadBuf = NULL;
void*                   pvEvtPayloadBuf = NULL;
uint32_t                uiEvtMaxPayloadSize = 0;
uint32_t                uiData;
uint64_t                ullAddress;
CAM_EVT_REQUEST_HANDLE  hEvtRequest = (CAM_EVT_REQUEST_HANDLE)NULL;
CAM_EVT_REQUEST_HANDLE  hRcvEvtRequest = (CAM_EVT_REQUEST_HANDLE)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;

do
{
#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.
        uiStatus = Sys_CreateSignal(&hEventCmpEvt);
        if (uiStatus != CAM_API_STS_SUCCESS)
            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 *)malloc(uiEvtMaxPayloadSize);
        if (pvEvtPayloadBuf == NULL)
            return -1;

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

    // Create completion event object for stream.
    uiStatus = Sys_CreateSignal(&hStrmCmpEvt);
    if (uiStatus != CAM_API_STS_SUCCESS)
        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 *)malloc(uiPyldSize);
    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.
        uiStatus = Sys_WaitForSignal(hEventCmpEvt, 2000);
        if (uiStatus != CAM_API_STS_SUCCESS)

```

```

        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 = %llu, pPayload = 0x%11X\n",
           (long long unsigned int)sEventReqInfo.sU3vInfo.u11Timestamp,
           (long long unsigned int)sEventReqInfo.sU3vInfo.pvPayload);

    // Wait for receiving image completion event.
    uiStatus = Sys_WaitForSignal(hStrmCmpEvt, 2000);
    if (uiStatus != CAM_API_STS_SUCCESS)
        return -1;
}

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

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

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

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

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

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

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

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

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

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

```

---

```
    free(pvEvtPayloadBuf);

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

    // Terminate system.
    Sys_Terminate();

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

### 5.4.2.2. Evt\_CreateRequest

カメライベント（メッセージ）データー時保管用のイベントリクエストを作成します。

#### [構文]

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

#### [パラメータ]

パラメータ		内 容
hEvt	[in]	イベントインターフェースのイベントハンドルです。
pvPayloadBuf	[in]	受信したカメライベント（メッセージ）データを格納するバッファへのポインタです。バッファは、 <a href="#">uiPayloadSize</a> バイトの領域をあらかじめ確保しておく必要があります。
uiPayloadSize	[in]	1つのカメライベント（メッセージ）分の最大ペイロードサイズです。（単位：byte） 通常は、 <a href="#">Evt_Open()</a> 実行時に <a href="#">puiMaxPayloadSize</a> に指定（取得）した値を指定してください。
phEvtRequest	[out]	作成されたイベントリクエストのハンドルを格納する変数へのポインタです。

#### [戻り値]

実行結果を返します。 戻り値は、[5.9. ステータスコード](#) を参照してください。

#### [備考]

本関数は、[Evt\\_Open\(\)](#)でオープンしたイベントインターフェースに使用できます。

[Evt\\_OpenSimple\(\)](#)でオープンしたイベントインターフェースでは正常に動作しません。

作成したイベントリクエストは [Evt\\_EnqueueRequest\(\)](#) を使用してイベント受信待機キューに投入してください。

カメライベントデータを受信すると、TeliCamAPI はイベント受信待機キューから取り出したイベントリクエストに受信データを書き込み、そのイベントリクエストをイベント受信完了キューに投入します。[Evt\\_DequeueRequest\(\)](#) を実行すればイベント受信完了キューからイベントデータが書き込まれたイベントリクエストを取り出すことができます。

[pvPayloadBuf](#) で指定したバッファは [Evt\\_ReleaseRequest\(\)](#) を実行してから解放してください。イベントリクエストを解放する前に [pvPayloadBuf](#) で指定したバッファを解放すると、予期せぬエラーが発生することがあります。

TeliCamAPI.h をインクルードする必要があります。

#### [コード例]

[Evt\\_Open\(\)](#) の[コード例](#)を参照してください。



---

### 5.4.2.3. Evt\_ReleaseRequest

カメライベント（メッセージ）データー時保管用のイベントリクエストを解放します。

#### [構文]

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

#### [パラメータ]

パラメータ		内 容
<i>hEvt</i>	[in]	イベントインターフェースのイベントハンドルです。
<i>hEvtRequest</i>	[in]	イベントリクエストのハンドルです。

#### [戻り値]

実行結果を返します。 戻り値は、[5.9. ステータスコード](#) を参照してください。

#### [備考]

本関数は、[Evt\\_Open\(\)](#) でオープンしたイベントインターフェースに使用できます。

[Evt\\_OpenSimple\(\)](#) でオープンしたイベントインターフェースでは正常に動作しません。

[Evt\\_CreateRequest\(\)](#) で作成したストリームリクエストは、アプリケーションを終了する前に必ず本関数により解放してください。

TeliCamAPI.h をインクルードする必要があります。

#### [コード例]

[Evt\\_Open\(\)](#) の[コード例](#)を参照してください。

---

#### 5.4.2.4. Evt\_EnqueueRequest

イベントリクエストを、イベント受信待機キューに投入します。

##### [構文]

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

##### [パラメータ]

パラメータ		内 容
hEvt	[in]	イベントインターフェースのイベントハンドルです。
hEvtRequest	[in]	イベントリクエストのハンドルです。

##### [戻り値]

実行結果を返します。 戻り値は、[5.9. ステータスコード](#) を参照してください。

##### [備考]

本関数は、[Evt\\_Open\(\)](#) でオープンしたイベントインターフェースに使用できます。

[Evt\\_OpenSimple\(\)](#) でオープンしたイベントインターフェースでは正常に動作しません。

カメライベントデータを受信すると、TeliCamAPI はイベント受信待機キューから取り出したイベントリクエストに受信データを書き込み、そのイベントリクエストをイベント受信完了キューに投入します。

イベント受信待機キューが空の状態カメライベントデータを受信すると、TeliCamAPI は受信したデータを破棄します。カメライベントデータを取りこぼしなく取り込むためには、ユーザアプリケーションはイベント受信待機キューに常に複数のイベントリクエストが投入されているよう管理する必要があります。

TeliCamAPI.h をインクルードする必要があります。

##### [コード例]

[Evt\\_Open\(\)](#) の[コード例](#)を参照してください。

---

### 5.4.2.5. Evt\_DequeueRequest

イベント受信完了キューから、イベントリクエストを一つ取り出します。

#### [構文]

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

#### [パラメータ]

パラメータ	内 容
hEvt [in]	イベントインターフェースのイベントハンドルです。
phEvtRequest [out]	取り出したイベントリクエストを格納する変数へのポインタです。
psEvtReqInfo [out]	取得した情報を格納するバッファへのポインタです。

#### [戻り値]

実行結果を返します。 戻り値は、[5.9. ステータスコード](#) を参照してください。

#### [備考]

本関数は、[Evt\\_Open\(\)](#) でオープンしたイベントインターフェースに使用できます。

[Evt\\_OpenSimple\(\)](#) でオープンしたイベントインターフェースでは正常に動作しません。

本関数はイベント受信完了キュー内の最初に投入されたイベントリクエストを取り出します。

イベント受信完了キューが空のとき、本関数は CAM\_API\_STS\_EMPTY\_COMPLETE\_QUEUE を返します。

TeliCamAPI.h をインクルードする必要があります。

#### [コード例]

[Evt\\_Open\(\)](#) の[コード例](#)を参照してください。

---

### 5.4.2.6. Evt\_FlushWaitQueue

全てのカメライベントデータの受信処理を中止し、イベント受信待機キュー内のすべてのイベントリクエストをイベント受信完了キューに移動させます。

#### [構文]

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

#### [パラメータ]

パラメータ	内 容
hEvt [in]	イベントインターフェースのイベントハンドルです。

#### [戻り値]

実行結果を返します。 戻り値は、[5.9. ステータスコード](#) を参照してください。

#### [備考]

本関数は、[Evt\\_Open\(\)](#)でオープンしたイベントインターフェースに使用できます。

[Evt\\_OpenSimple\(\)](#)でオープンしたイベントインターフェースでは正常に動作しません。

[Evt\\_Close\(\)](#)でイベントインターフェースをクローズする時、イベント受信待機キューとイベント受信完了キューは空である必要があります。

本関数を実行して、イベント受信待機キュー内のすべてのイベントリクエストをイベント受信完了キューに移動させた後、イベント受信完了キューが空になるまで [Evt\\_DequeueRequest\(\)](#)を実行してイベントリクエストをイベント受信完了キューから取り出してください。

[Evt\\_DequeueRequest\(\)](#)で取り出したイベントリクエストが、本関数によりイベント受信完了キューに移動されたイベントリクエストの場合、有効なイベントデータがイベントリクエストに保存されていないことを示すために TeliCamAPI は CAM\_API\_STS\_FLUSH\_REQUESTE を返します。

TeliCamAPI.h をインクルードする必要があります。

#### [コード例]

[Evt\\_Open\(\)](#) の[コード例](#)を参照してください。

---

### 5.4.3. 共通関数

高水準関数と低水準関数共通の関数です。

#### 5.4.3.1. Evt\_Close

カメライベント（メッセージ）取得用のイベントインターフェースをクローズします。

##### [構文]

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

##### [パラメータ]

パラメータ	内 容
hEvt [in]	イベントインターフェースのイベントハンドルです。

##### [戻り値]

実行結果を返します。 戻り値は、[5.9. ステータスコード](#) を参照してください。

##### [備考]

イベントインターフェースをオープンした場合、必ず本関数を実行してイベントインターフェースをクローズしてください。

別のスレッドで使用しているイベントインタフェースをクローズすると、そのスレッドでエラーが発生する可能性があります。すべてのスレッドでイベントインタフェースの使用が終了してから本関数を実行してください。

TeliCamAPI.h をインクルードする必要があります。

##### [コード例]

[Evt\\_OpenSimple\(\)](#)の[コード例](#)を参照してください。

---

## 5.5. カメラ制御関数

カメラインタフェースの種類。カメラの型、レジスタアドレスなどを気にせずにカメラの機能を制御するための関数群です。

GigE Vision カメラの場合、本章のすべての関数に GenICam GenApi ライブラリを内部で使用しています。USB3 Vision カメラの場合、処理パフォーマンスを高めるために本章のほとんどの関数で GenICam GenApi ライブラリを使用せず、カメラのレジスタに直接アクセスしています。カメラオープン時に GenApi モジュール無効を指定した場合、GenICam GenApi ライブラリを使用している関数は使用できなくなりますのでご注意ください。

カメラによって使用できる関数が異なります。機能が実装されているかどうかは、使用するカメラの取扱説明書をご覧ください。

カメラインタフェースの違い、インタフェースバージョンの違いなどにより、同じ機能が本関数群で使用している名称と異なる名称で取扱説明書に説明されている場合がありますのでご注意ください。

TriggerSource などの一部の機能では、同じ機能状態に設定するためのレジスタ設定数値がカメラインタフェースによって異なる場合があります。TeliCam,Api のカメラ制御関数で取得・設定する enum 値は、インタフェースを意識することなく扱うための値であり、[Cam\\_ReadReg\(\)](#)、[Cam\\_WriteReg\(\)](#)で扱う実際のカメラレジスタの値、GenICam 関数で扱う設定値とは異なる場合があります。

---

### 5.5.1. ImageFormatControl

カメラの映像フォーマットの制御を行います。  
映像フォーマットの説明は、カメラの取扱説明書をご覧ください。

#### 5.5.1.1. GetCamImageFormatSelector

カメラの映像フォーマットを取得します。

##### [構文]

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

##### [パラメータ]

パラメータ		内 容
<i>hCam</i>	[in]	カメラのカメラハンドルです。
<i>peFormat</i>	[out]	取得した映像フォーマットを格納する変数へのポインタです。

##### [戻り値]

実行結果を返します。 戻り値は、[5.9. ステータスコード](#) を参照してください。

##### [備考]

映像フォーマット制御機能（ImageFormatSelector レジスタ）が実装されていないカメラで実行するとエラーになります。

CAM\_IMAGE\_FORMAT\_SELECTOR\_TYPE 型は、TeliCamAPI.h で定義されています。

TeliCamAPI.h をインクルードする必要があります。

---

### 5.5.1.2. SetCamImageFormatSelector

カメラの映像フォーマットを設定します。

#### [構文]

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

#### [パラメータ]

パラメータ		内 容
<i>hCam</i>	[in]	カメラのカメラハンドルです。
<i>eFormat</i>	[in]	設定するフォーマットです。

#### [戻り値]

実行結果を返します。 戻り値は、[5.9. ステータスコード](#) を参照してください。

#### [備考]

映像フォーマット制御機能（ImageFormatSelector レジスタ）が実装されていないカメラで実行するとエラーになります。

CAM\_IMAGE\_FORMAT\_SELECTOR\_TYPE 型は、TeliCamAPI.h で定義されています。

TeliCamAPI.h をインクルードする必要があります。



---

## 5.5.2. Scalable

カメラのスケーラブル機能の設定を行います。  
スケーラブル機能の説明は、カメラの取扱説明書をご覧ください。

### 5.5.2.1. GetCamSensorWidth

カメラの水平有効画素数を取得します。

#### [構文]

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

#### [パラメータ]

パラメータ		内 容
<i>hCam</i>	[in]	カメラのカメラハンドルです。
<i>puiSensorWidth</i>	[out]	取得した水平有効画素数を格納する変数へのポインタです。

#### [戻り値]

実行結果を返します。 戻り値は、[5.9. ステータスコード](#) を参照してください。

#### [備考]

TeliCamAPI.h をインクルードする必要があります。

---

### 5.5.2.2. GetCamSensorHeight

カメラの垂直有効画素数を取得します。

#### [構文]

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

#### [パラメータ]

パラメータ		内 容
<i>hCam</i>	[in]	カメラのカメラハンドルです。
<i>puiSensorHeight</i>	[out]	取得した垂直有効画素数を格納する変数へのポインタです。

#### [戻り値]

実行結果を返します。 戻り値は、[5.9. ステータスコード](#) を参照してください。

#### [備考]

TeliCamAPI.h をインクルードする必要があります。

---

### 5.5.2.3. GetCamRoi

カメラの ROI（領域）を取得します。

#### [構文]

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

#### [パラメータ]

パラメータ		内 容
<i>hCam</i>	[in]	カメラのカメラハンドルです。
<i>puiWidth</i>	[out]	取得した映像の幅を格納する変数へのポインタです。
<i>puiHeight</i>	[out]	取得した映像の高さを格納する変数へのポインタです。
<i>puiOffsetX</i>	[out]	取得した映像の水平方向開始位置を格納する変数へのポインタです。
<i>puiOffsetY</i>	[out]	取得した映像の垂直方向開始位置を格納する変数へのポインタです。

#### [戻り値]

実行結果を返します。 戻り値は、[5.9. ステータスコード](#) を参照してください。

#### [備考]

TeliCamAPI.h をインクルードする必要があります。

---

#### 5.5.2.4. SetCamRoi

カメラの ROI（領域）を設定します。

##### [構文]

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

##### [パラメータ]

パラメータ		内 容
<i>hCam</i>	[in]	カメラのカメラハンドルです。
<i>uiWidth</i>	[in]	設定する映像の幅です。
<i>uiHeight</i>	[in]	設定する映像の高さです。
<i>uiOffsetX</i>	[in]	設定する映像の水平方向開始位置です。
<i>uiOffsetY</i>	[in]	設定する映像の垂直方向開始位置です。

##### [戻り値]

実行結果を返します。 戻り値は、[5.9. ステータスコード](#) を参照してください。

##### [備考]

ストリーム転送が開始されている場合は設定を行うことはできません。

TeliCamAPI.h をインクルードする必要があります。

---

### 5.5.2.5. GetCamWidthMinMax

カメラに設定できる映像の幅の最小値・最大値・増加量を取得します。

#### [構文]

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

#### [パラメータ]

パラメータ		内 容
<i>hCam</i>	[in]	カメラのカメラハンドルです。
<i>puiWidthMin</i>	[out]	取得した最小値を格納する変数へのポインタです。
<i>puiWidthMax</i>	[out]	取得した最大値を格納する変数へのポインタです。
<i>puiWidthInc</i>	[out]	取得した増加量を格納する変数へのポインタです。

#### [戻り値]

実行結果を返します。 戻り値は、[5.9. ステータスコード](#) を参照してください。

#### [備考]

TeliCamAPI.h をインクルードする必要があります。

---

### 5.5.2.6. GetCamWidth

カメラの映像の幅を取得します。

#### [構文]

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

#### [パラメータ]

パラメータ		内 容
<i>hCam</i>	[in]	カメラのカメラハンドルです。
<i>puiWidth</i>	[out]	取得した映像の幅を格納する変数へのポインタです。

#### [戻り値]

実行結果を返します。 戻り値は、[5.9. ステータスコード](#) を参照してください。

#### [備考]

TeliCamAPI.h をインクルードする必要があります。

---

### 5.5.2.7. SetCamWidth

カメラの映像の幅を設定します。

#### [構文]

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

#### [パラメータ]

パラメータ		内 容
<i>hCam</i>	[in]	カメラのカメラハンドルです。
<i>uiWidth</i>	[in]	設定する映像の幅です。

#### [戻り値]

実行結果を返します。 戻り値は、[5.9. ステータスコード](#) を参照してください。

#### [備考]

ストリーム転送が開始されている場合は設定を行うことはできません。

TeliCamAPI.h をインクルードする必要があります。

---

### 5.5.2.8. GetCamHeightMinMax

カメラに設定できる映像の高さの最小値・最大値・増加量を取得します。

#### [構文]

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

#### [パラメータ]

パラメータ		内 容
<i>hCam</i>	[in]	カメラのカメラハンドルです。
<i>puiHeightMin</i>	[out]	取得した最小値を格納する変数へのポインタです。
<i>puiHeightMax</i>	[out]	取得した最大値を格納する変数へのポインタです。
<i>puiHeightInc</i>	[out]	取得した増加量を格納する変数へのポインタです。

#### [戻り値]

実行結果を返します。 戻り値は、[5.9. ステータスコード](#) を参照してください。

#### [備考]

TeliCamAPI.h をインクルードする必要があります。



---

### 5.5.2.9. GetCamHeight

カメラの映像の高さを取得します。

#### [構文]

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

#### [パラメータ]

パラメータ		内 容
<i>hCam</i>	[in]	カメラのカメラハンドルです。
<i>puiHeight</i>	[out]	取得した映像の高さを格納する変数へのポインタです。

#### [戻り値]

実行結果を返します。 戻り値は、[5.9. ステータスコード](#) を参照してください。

#### [備考]

TeliCamAPI.h をインクルードする必要があります。

---

### 5.5.2.10. SetCamHeight

カメラの映像の高さを設定します。

#### [構文]

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

#### [パラメータ]

パラメータ		内 容
<i>hCam</i>	[in]	カメラのカメラハンドルです。
<i>uiHeight</i>	[in]	設定する映像の高さです。

#### [戻り値]

実行結果を返します。 戻り値は、[5.9. ステータスコード](#) を参照してください。

#### [備考]

ストリーム転送が開始されている場合は設定を行うことはできません。  
TeliCamAPI.h をインクルードする必要があります。

---

### 5.5.2.11. GetCamOffsetXMinMax

カメラに設定できる映像の水平方向開始位置の最小値・最大値・増加量を取得します。

#### [構文]

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

#### [パラメータ]

パラメータ		内 容
<i>hCam</i>	[in]	カメラのカメラハンドルです。
<i>puiOffsetXMin</i>	[out]	取得した最小値を格納する変数へのポインタです。
<i>puiOffsetXMax</i>	[out]	取得した最大値を格納する変数へのポインタです。
<i>puiOffsetXInc</i>	[out]	取得した増加量を格納する変数へのポインタです。

#### [戻り値]

実行結果を返します。 戻り値は、[5.9. ステータスコード](#) を参照してください。

#### [備考]

TeliCamAPI.h をインクルードする必要があります。

---

### 5.5.2.12. GetCamOffsetX

カメラの映像の水平方向開始位置を取得します。

#### [構文]

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

#### [パラメータ]

パラメータ		内 容
<i>hCam</i>	[in]	カメラのカメラハンドルです。
<i>puiOffsetX</i>	[out]	取得した映像の水平方向開始位置を格納する変数へのポインタです。

#### [戻り値]

実行結果を返します。 戻り値は、[5.9. ステータスコード](#) を参照してください。

#### [備考]

TeliCamAPI.h をインクルードする必要があります。

---

### 5.5.2.13. SetCamOffsetX

カメラの映像の水平方向開始位置を設定します。

#### [構文]

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

#### [パラメータ]

パラメータ		内 容
<i>hCam</i>	[in]	カメラのカメラハンドルです。
<i>uiOffsetX</i>	[in]	設定する映像の水平方向開始位置です。

#### [戻り値]

実行結果を返します。 戻り値は、[5.9. ステータスコード](#) を参照してください。

#### [備考]

ストリーム転送が開始されている場合は設定を行うことはできません。

TeliCamAPI.h をインクルードする必要があります。

---

### 5.5.2.14. GetCamOffsetYMinMax

カメラに設定できる映像の垂直方向開始位置の最小値・最大値・増加量を取得します。

#### [構文]

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

#### [パラメータ]

パラメータ		内 容
<i>hCam</i>	[in]	カメラのカメラハンドルです。
<i>puiOffsetYMin</i>	[out]	取得した最小値を格納する変数へのポインタです。
<i>puiOffsetYMax</i>	[out]	取得した最大値を格納する変数へのポインタです。
<i>puiOffsetYInc</i>	[out]	取得した増加量を格納する変数へのポインタです。

#### [戻り値]

実行結果を返します。 戻り値は、[5.9. ステータスコード](#) を参照してください。

#### [備考]

TeliCamAPI.h をインクルードする必要があります。

---

### 5.5.2.15. GetCamOffsetY

カメラの映像の垂直方向開始位置を取得します。

#### [構文]

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

#### [パラメータ]

パラメータ		内 容
<i>hCam</i>	[in]	カメラのカメラハンドルです。
<i>puiOffsetY</i>	[out]	取得した映像の垂直方向開始位置を格納する変数へのポインタです。

#### [戻り値]

実行結果を返します。 戻り値は、[5.9. ステータスコード](#) を参照してください。

#### [備考]

TeliCamAPI.h をインクルードする必要があります。

---

### 5.5.2.16. SetCamOffsetY

カメラの映像の垂直方向開始位置を設定します。

#### [構文]

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

#### [パラメータ]

パラメータ		内 容
<i>hCam</i>	[in]	カメラのカメラハンドルです。
<i>puiOffsetY</i>	[in]	設定する映像の垂直方向開始位置です。

#### [戻り値]

実行結果を返します。 戻り値は、[5.9. ステータスコード](#) を参照してください。

#### [備考]

ストリーム転送が開始されている場合は設定を行うことはできません。

TeliCamAPI.h をインクルードする必要があります。



---

### 5.5.3. Binning

カメラのビニング機能の制御を行います。  
ビニング機能の説明は、カメラの取扱説明書をご覧ください。

#### 5.5.3.1. GetCamBinningHorizontalMinMax

カメラに設定できる水平方向ビニング設定値の最小値と最大値を取得します。

##### [構文]

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

##### [パラメータ]

パラメータ		内 容
<i>hCam</i>	[in]	カメラのカメラハンドルです。
<i>puiMin</i>	[out]	取得した最小値を格納する変数へのポインタです。
<i>puiMax</i>	[out]	取得した最大値を格納する変数へのポインタです。

##### [戻り値]

実行結果を返します。 戻り値は、[5.9. ステータスコード](#) を参照してください。

##### [備考]

水平方向ビニング機能（BinningHorizontal レジスタ）が実装されていないカメラで実行するとエラーになります。

USB3 Vision カメラの場合、ImageFormatSelector レジスタに Format1（Binning Mode）が設定されていない場合は機能が無効になっているため、有効な値を取得できません。

TeliCamAPI.h をインクルードする必要があります。

---

### 5.5.3.2. GetCamBinningHorizontal

カメラの水平方向ビニング設定値を取得します。

#### [構文]

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

#### [パラメータ]

パラメータ		内 容
<i>hCam</i>	[in]	カメラのカメラハンドルです。
<i>puiValue</i>	[out]	取得した水平方向ビニング設定値を格納する変数へのポインタです。

#### [戻り値]

実行結果を返します。 戻り値は、[5.9. ステータスコード](#) を参照してください。

#### [備考]

水平方向ビニング機能（BinningHorizontal レジスタ）が実装されていないカメラで実行するとエラーになります。

USB3 Vision カメラの場合、ImageFormatSelector レジスタに Format1（Binning Mode）が設定されていない場合は機能が無効になっているため、有効な値を取得できません。

TeliCamAPI.h をインクルードする必要があります。

---

### 5.5.3.3. SetCamBinningHorizontal

カメラの水平方向ビニングを設定します。

#### [構文]

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

#### [パラメータ]

パラメータ		内 容
<i>hCam</i>	[in]	カメラのカメラハンドルです。
<i>uiValue</i>	[in]	水平方向ビニング設定値です。

#### [戻り値]

実行結果を返します。 戻り値は、[5.9. ステータスコード](#) を参照してください。

#### [備考]

水平方向ビニング機能（BinningHorizontal レジスタ）が実装されていないカメラで実行するとエラーになります。

ストリーム転送が開始されている場合は設定を行うことはできません。

USB3 Vision カメラの場合、ImageFormatSelector レジスタに Format1（Binning Mode）が設定されていない場合は機能が無効になっているため、設定できません。

TeliCamAPI.h をインクルードする必要があります。

---

#### 5.5.3.4. GetCamBinningVerticalMinMax

カメラに設定できる垂直方向ビニング設定値の最小値と最大値を取得します。

##### [構文]

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

##### [パラメータ]

パラメータ	内 容
<i>hCam</i> [in]	カメラのカメラハンドルです。
<i>puiMin</i> [out]	取得した最小値を格納する変数へのポインタです。
<i>puiMax</i> [out]	取得した最大値を格納する変数へのポインタです。

##### [戻り値]

実行結果を返します。 戻り値は、[5.9. ステータスコード](#) を参照してください。

##### [備考]

垂直方向ビニング機能（BinningVertical レジスタ）が実装されていないカメラで実行するとエラーになります。

USB3 Vision カメラの場合、ImageFormatSelector レジスタが Format1（Binning Mode）が設定されていない場合は機能が無効になっているため、有効な値を取得できません。

TeliCamAPI.h をインクルードする必要があります。

---

### 5.5.3.5. GetCamBinningVertical

カメラの垂直方向ビニング設定値を取得します。

#### [構文]

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

#### [パラメータ]

パラメータ		内 容
<i>hCam</i>	[in]	カメラのカメラハンドルです。
<i>puiValue</i>	[out]	取得した垂直方向ビニング設定値を格納する変数へのポインタです。

#### [戻り値]

実行結果を返します。 戻り値は、[5.9. ステータスコード](#) を参照してください。

#### [備考]

垂直方向ビニング機能（BinningVertical レジスタ）が実装されていないカメラで実行するとエラーになります。

USB3 Vision カメラの場合、ImageFormatSelector レジスタに Format1（Binning Mode）が設定されていない場合は機能が無効になっているため、有効な値を取得できません。

TeliCamAPI.h をインクルードする必要があります。

---

### 5.5.3.6. SetCamBinningVertical

カメラの垂直方向ビニングを設定します。

#### [構文]

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

#### [パラメータ]

パラメータ		内 容
<i>hCam</i>	[in]	カメラのカメラハンドルです。
<i>uiValue</i>	[in]	垂直方向ビニング設定値です。

#### [戻り値]

実行結果を返します。 戻り値は、[5.9. ステータスコード](#) を参照してください。

#### [備考]

垂直方向ビニング機能（BinningVertical レジスタ）が実装されていないカメラで実行するとエラーになります。

ストリーム転送が開始されている場合は設定を行うことはできません。

USB3 Vision カメラの場合、ImageFormatSelector レジスタに Format1（Binning Mode）が設定されていない場合は機能が無効になっているため、設定できません。

TeliCamAPI.h をインクルードする必要があります。

#### 5.5.4. Decimation

カメラのデシメーション（間引き）機能の制御を行います。  
デシメーション（間引き）機能の説明は、カメラの取扱説明書をご覧ください。

##### 5.5.4.1. GetCamDecimationHorizontalMinMax

カメラに設定できる水平方向デシメーション（間引き）設定値の最小値と最大値を取得します。

###### [構文]

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

###### [パラメータ]

パラメータ		内 容
<i>hCam</i>	[in]	カメラのカメラハンドルです。
<i>puiMin</i>	[out]	取得した最小値を格納する変数へのポインタです。
<i>puiMax</i>	[out]	取得した最大値を格納する変数へのポインタです。

###### [戻り値]

実行結果を返します。 戻り値は、[5.9. ステータスコード](#) を参照してください。

###### [備考]

水平方向デシメーション（間引き）機能（DecimationHorizontal レジスタ）が実装されていないカメラで実行するとエラーになります。

USB3 Vision カメラの場合、ImageFormatSelector レジスタに Format2（Decimation Mode）が設定されていない場合は機能が無効になっているため、有効な値を取得できません。

TeliCamAPI.h をインクルードする必要があります。

---

#### 5.5.4.2. GetCamDecimationHorizontal

カメラの水平方向デシメーション（間引き）設定値を取得します。

##### [構文]

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

##### [パラメータ]

パラメータ		内 容
<i>hCam</i>	[in]	カメラのカメラハンドルです。
<i>puiValue</i>	[out]	取得した水平方向デシメーション（間引き）設定値を格納する変数へのポインタです。

##### [戻り値]

実行結果を返します。 戻り値は、[5.9. ステータスコード](#) を参照してください。

##### [備考]

水平方向デシメーション（間引き）機能（DecimationHorizontal レジスタ）が実装されていないカメラで実行するとエラーになります。

USB3 Vision カメラの場合、ImageFormatSelector レジスタに Format2（Decimation Mode）が設定されていない場合は機能が無効になっているため、有効な値を取得できません。

TeliCamAPI.h をインクルードする必要があります。



---

### 5.5.4.3. SetCamDecimationHorizontal

カメラの水平方向デシメーション（間引き）を設定します。

#### [構文]

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

#### [パラメータ]

パラメータ		内 容
<i>hCam</i>	[in]	カメラのカメラハンドルです。
<i>uiValue</i>	[in]	水平方向デシメーション（間引き）設定値です。

#### [戻り値]

実行結果を返します。 戻り値は、[5.9. ステータスコード](#) を参照してください。

#### [備考]

デシメーション機能は無効な設定値を指定してもエラーにならない場合があります。無効な設定値が設定された場合、正常な画像が出力されない場合があります。必ず使用するカメラの取扱説明書をご覧ください。

水平方向デシメーション（間引き）機能（DecimationHorizontal レジスタ）が実装されていないカメラで実行するとエラーになります。

ストリーム転送が開始されている場合は設定を行うことはできません。

USB3 Vision カメラの場合、ImageFormatSelector レジスタに Format2（Decimation Mode）が設定されていない場合は機能が無効になっているため、設定できません。

TeliCamAPI.h をインクルードする必要があります。

---

#### 5.5.4.4. GetCamDecimationVerticalMinMax

カメラに設定できる垂直方向デシメーション（間引き）設定値の最小値と最大値を取得します。

##### [構文]

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

##### [パラメータ]

パラメータ	内 容
<i>hCam</i> [in]	カメラのカメラハンドルです。
<i>puiMin</i> [out]	取得した最小値を格納する変数へのポインタです。
<i>puiMax</i> [out]	取得した最大値を格納する変数へのポインタです。

##### [戻り値]

実行結果を返します。 戻り値は、[5.9. ステータスコード](#) を参照してください。

##### [備考]

垂直方向デシメーション（間引き）機能（DecimationVertical レジスタ）が実装されていないカメラで実行するとエラーになります。

USB3 Vision カメラの場合、ImageFormatSelector レジスタが Format2（Decimation Mode）が設定されていない場合は機能が無効になっているため、有効な値を取得できません。

TeliCamAPI.h をインクルードする必要があります。

---

#### 5.5.4.5. GetCamDecimationVertical

カメラの垂直方向デシメーション（間引き）設定値を取得します。

##### [構文]

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

##### [パラメータ]

パラメータ		内 容
<i>hCam</i>	[in]	カメラのカメラハンドルです。
<i>puiValue</i>	[out]	取得した垂直方向デシメーション（間引き）設定値を格納する変数へのポインタです。

##### [戻り値]

実行結果を返します。 戻り値は、[5.9. ステータスコード](#) を参照してください。

##### [備考]

垂直方向デシメーション（間引き）機能（DecimationVertical レジスタ）が実装されていないカメラで実行するとエラーになります。

USB3 Vision カメラの場合、ImageFormatSelector レジスタに Format2（Decimation Mode）が設定されていない場合は機能が無効になっているため、有効な値を取得できません。

TeliCamAPI.h をインクルードする必要があります。

---

#### 5.5.4.6. SetCamDecimationVertical

カメラの垂直方向デシメーション（間引き）を設定します。

##### [構文]

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

##### [パラメータ]

パラメータ		内 容
<i>hCam</i>	[in]	カメラのカメラハンドルです。
<i>uiValue</i>	[in]	垂直方向デシメーション（間引き）設定値です。

##### [戻り値]

実行結果を返します。 戻り値は、[5.9. ステータスコード](#) を参照してください。

##### [備考]

デシメーション機能は無効な設定値を指定してもエラーにならない場合があります。無効な設定値が設定された場合、正常な画像が出力されない場合があります。必ず使用するカメラの取扱説明書をご覧ください。

垂直方向デシメーション（間引き）機能（DecimationVertical レジスタ）が実装されていないカメラで実行するとエラーになります。

ストリーム転送が開始されている場合は設定を行うことはできません。

USB3 Vision カメラの場合、ImageFormatSelector レジスタに Format2（Decimation Mode）が設定されていない場合は機能が無効になっているため、設定できません。

TeliCamAPI.h をインクルードする必要があります。

---

### 5.5.5. Reverse

カメラの映像反転機能の制御を行います。  
映像反転機能の説明は、カメラの取扱説明書をご覧ください。

#### 5.5.5.1. GetCamReverseX

カメラの水平方向の映像反転設定値を取得します。

##### [構文]

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

##### [パラメータ]

パラメータ		内 容
<i>hCam</i>	[in]	カメラのカメラハンドルです。
<i>pbValue</i>	[out]	取得した水平方向の映像反転設定値を格納する変数へのポインタです。 true の場合 反転 ON、false の場合 反転 OFF です。

##### [戻り値]

実行結果を返します。 戻り値は、[5.9. ステータスコード](#) を参照してください。

##### [備考]

水平方向映像反転機能（ReverseX レジスタ）が実装されていないカメラで実行するとエラーになります。

TeliCamAPI.h をインクルードする必要があります。

---

### 5.5.5.2. SetCamReverseX

カメラの水平方向の映像反転 ON/OFF を設定します。

#### [構文]

```
CAM_API_STATUS SetCamReverseX (  
    CAM_HANDLE      hCam,  
    bool8_t         bValue  
);
```

#### [パラメータ]

パラメータ		内 容
<i>hCam</i>	[in]	カメラのカメラハンドルです。
<i>bValue</i>	[in]	設定する水平方向の映像反転 ON/OFF です。 true の場合 反転 ON、false の場合 反転 OFF です。

#### [戻り値]

実行結果を返します。 戻り値は、[5.9. ステータスコード](#) を参照してください。

#### [備考]

水平方向映像反転機能（ReverseX レジスタ）が実装されていないカメラで実行するとエラーになります。

ストリーム転送が開始されている場合は設定を行うことはできません。

TeliCamAPI.h をインクルードする必要があります。

---

### 5.5.5.3. GetCamReverseY

カメラの垂直方向の映像反転設定値を取得します。

#### [構文]

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

#### [パラメータ]

パラメータ		内 容
<i>hCam</i>	[in]	カメラのカメラハンドルです。
<i>pbValue</i>	[out]	取得した垂直方向の映像反転設定値を格納する変数へのポインタです。 true の場合 反転 ON、false の場合 反転 OFF です。

#### [戻り値]

実行結果を返します。 戻り値は、[5.9. ステータスコード](#) を参照してください。

#### [備考]

垂直方向映像反転機能（ReverseY レジスタ）が実装されていないカメラで実行するとエラーになります。

TeliCamAPI.h をインクルードする必要があります。

---

#### 5.5.5.4. SetCamReverseY

カメラの垂直方向の映像反転 ON/OFF を設定します。

##### [構文]

```
CAM_API_STATUS SetCamReverseY (  
    CAM_HANDLE      hCam,  
    bool8_t          bValue  
);
```

##### [パラメータ]

パラメータ		内 容
<i>hCam</i>	[in]	カメラのカメラハンドルです。
<i>bValue</i>	[in]	設定する垂直方向の映像反転 ON/OFF です。 true の場合 反転 ON、false の場合 反転 OFF です。

##### [戻り値]

実行結果を返します。 戻り値は、[5.9. ステータスコード](#) を参照してください。

##### [備考]

垂直方向映像反転機能（ReverseY レジスタ）が実装されていないカメラで実行するとエラーになります。

ストリーム転送が開始されている場合は設定を行うことはできません。

TeliCamAPI.h をインクルードする必要があります。



---

### 5.5.6. PixelFormat

カメラのストリームのピクセルフォーマットの制御を行います。  
ストリームのピクセルフォーマットの説明は、カメラの取扱説明書をご覧ください。

#### 5.5.6.1. GetCamPixelFormat

カメラの映像ストリームのピクセルフォーマット設定値を取得します。

##### [構文]

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

##### [パラメータ]

パラメータ		内 容
<i>hCam</i>	[in]	カメラのカメラハンドルです。
<i>puiPixelFormat</i>	[out]	取得したピクセルフォーマット設定値を格納する変数へのポインタです。

##### [戻り値]

実行結果を返します。 戻り値は、[5.9. ステータスコード](#) を参照してください。

##### [備考]

ピクセルフォーマット制御機能（PixelFormat レジスタ）が実装されていないカメラで実行するとエラーになります。

TeliCamAPI.h をインクルードする必要があります。

---

### 5.5.6.2. SetCamPixelFormat

カメラの映像ストリームのピクセルフォーマットを設定します。

#### [構文]

```
CAM_API_STATUS SetCamPixelFormat (  
    CAM_HANDLE          hCam,  
    CAM_PIXEL_FORMAT     uiPixelFormat  
);
```

#### [パラメータ]

パラメータ		内 容
<i>hCam</i>	[in]	カメラのカメラハンドルです。
<i>uiPixelFormat</i>	[in]	設定するピクセルフォーマットです。

#### [戻り値]

実行結果を返します。 戻り値は、[5.9. ステータスコード](#) を参照してください。

#### [備考]

USB3 Vision カメラの場合、PixelCoding レジスタと PixelSize レジスタの設定を行います。

GigE Vision カメラの場合、PixelFormat レジスタの設定を行います。

ピクセルフォーマット制御機能（上記レジスタ）が実装されていないカメラで実行するとエラーになります。

ストリーム転送が開始されている場合は設定を行うことはできません。

TeliCamAPI.h をインクルードする必要があります。

---

### 5.5.7. TestPattern

カメラのテストパターン機能の制御を行います。  
テストパターン機能の説明は、カメラの取扱説明書をご覧ください。

#### 5.5.7.1. GetCamTestPattern

カメラのテストパターン設定値を取得します。

##### [構文]

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

##### [パラメータ]

パラメータ		内 容
<i>hCam</i>	[in]	カメラのカメラハンドルです。
<i>peTestPattern</i>	[out]	取得したテストパターン設定値を格納する変数へのポインタです。

##### [戻り値]

実行結果を返します。 戻り値は、[5.9. ステータスコード](#) を参照してください。

##### [備考]

テストパターン機能（TestPattern、または TestImageSelector レジスタ）が実装されていないカメラで実行するとエラーになります。

CAM\_TEST\_PATTERN\_TYPE 型は、TeliCamAPI.h で定義されています。

TeliCamAPI.h をインクルードする必要があります。

---

### 5.5.7.2. SetCamTestPattern

カメラのテストパターンを設定します。

#### [構文]

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

#### [パラメータ]

パラメータ		内 容
<i>hCam</i>	[in]	カメラのカメラハンドルです。
<i>eTestPattern</i>	[in]	設定するテストパターンです。

#### [戻り値]

実行結果を返します。 戻り値は、[5.9. ステータスコード](#) を参照してください。

#### [備考]

テストパターン機能（TestPattern、または TestImageSelector レジスタ）が実装されていないカメラで実行するとエラーになります。

ストリーム転送が開始されている場合は設定を行うことはできません。

CAM\_TEST\_PATTERN\_TYPE 型は、TeliCamAPI.h で定義されています。

TeliCamAPI.h をインクルードする必要があります。

---

## 5.5.8. AcquisitionControl

カメラの映像出力についての実行・設定を行います。  
詳細については、カメラの取扱説明書をご覧ください。

### 5.5.8.1. GetCamStreamPayloadSize

カメラのレジスタから、映像ストリーム ペイロードサイズ（画像サイズ）を取得します。

#### [構文]

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

#### [パラメータ]

パラメータ		内 容
<i>hCam</i>	[in]	カメラのカメラハンドルです。
<i>puiPayloadSize</i>	[out]	取得した映像ストリーム ペイロードサイズ（画像サイズ）を格納する変数へのポインタです。（単位：byte）。

#### [戻り値]

実行結果を返します。 戻り値は、[5.9. ステータスコード](#) を参照してください。

#### [備考]

TeliCamAPI.h をインクルードする必要があります。

---

### 5.5.8.2. GetCamStreamEnable

カメラのレジスタから、ストリーム状態を取得します。

#### [構文]

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

#### [パラメータ]

パラメータ		内 容
<i>hCam</i>	[in]	カメラのカメラハンドルです。
<i>pbEnable</i>	[out]	取得したストリームの状態を格納する変数へのポインタです。 true のとき有効、false のとき無効です。

#### [戻り値]

実行結果を返します。 戻り値は、[5.9. ステータスコード](#) を参照してください。

#### [備考]

本関数は USB3 Vision カメラでのみ使用することができます。

TeliCamAPI.h をインクルードする必要があります。

---

### 5.5.8.3. GetCamAcquisitionFrameCountMinMax

映像ストリーム転送モードがマルチフレーム映像ストリーム転送モード、およびカメライメージバッファ転送モードに設定されているときの映像ストリーム転送枚数 最小値と最大値を取得します。

#### [構文]

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

#### [パラメータ]

パラメータ		内 容
<i>hCam</i>	[in]	カメラのカメラハンドルです。
<i>puiAcqFrameCountMin</i>	[out]	取得した最小値を格納する変数へのポインタです。
<i>puiAcqFrameCountMax</i>	[out]	取得した最大値を格納する変数へのポインタです。

#### [戻り値]

実行結果を返します。 戻り値は、[5.9. ステータスコード](#) を参照してください。

#### [備考]

映像出力設定機能（AcquisitionFrameCount レジスタ）が実装されていないカメラで実行するとエラーになります。

TeliCamAPI.h をインクルードする必要があります。

---

#### 5.5.8.4. GetCamAcquisitionFrameCount

映像ストリーム転送モードがマルチフレーム映像ストリーム転送モード、およびカメライメージバッファ転送モードに設定されているときの映像ストリーム転送枚数を取得します。

##### [構文]

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

##### [パラメータ]

パラメータ		内 容
<i>hCam</i>	[in]	カメラのカメラハンドルです。
<i>puiAcqFrameCount</i>	[out]	取得した映像ストリーム転送枚数を格納する変数へのポインタです。

##### [戻り値]

実行結果を返します。 戻り値は、[5.9. ステータスコード](#) を参照してください。

##### [備考]

映像出力設定機能（AcquisitionFrameCount レジスタ）が実装されていないカメラで実行するとエラーになります。

TeliCamAPI.h をインクルードする必要があります。



---

### 5.5.8.5. SetCamAcquisitionFrameCount

映像ストリーム転送モードがマルチフレーム映像ストリーム転送モード、およびカメライメージバッファ転送モードに設定されているときの映像ストリーム転送枚数を設定します。

#### [構文]

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

#### [パラメータ]

パラメータ	内 容
<i>hCam</i> [in]	カメラのカメラハンドルです。
<i>uiAcqFrameCount</i> [in]	設定する映像ストリーム転送枚数です。

#### [戻り値]

実行結果を返します。 戻り値は、[5.9. ステータスコード](#) を参照してください。

#### [備考]

映像出力設定機能（AcquisitionFrameCount レジスタ）が実装されていないカメラで実行するとエラーになります。

ストリーム転送が開始されている場合は設定を行うことはできません。

TeliCamAPI.h をインクルードする必要があります。

---

### 5.5.8.6. GetCamAcquisitionFrameRateControl

映像のフレームレート設定を取得します。

#### [構文]

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

#### [パラメータ]

パラメータ		内 容
<i>hCam</i>	[in]	カメラのカメラハンドルです。
<i>peFrameRateControl</i>	[out]	取得したフレームレート設定を格納する変数へのポインタです。

#### [戻り値]

実行結果を返します。 戻り値は、[5.9. ステータスコード](#) を参照してください。

#### [備考]

映像出力設定機能（AcquisitionFrameRateControl レジスタ）が実装されていないカメラで実行するとエラーになります。

CAM\_ACQ\_FRAME\_RATE\_CTRL\_TYPE 型は、TeliCamAPI.h で定義されています。

TeliCamAPI.h をインクルードする必要があります。

---

### 5.5.8.7. SetCamAcquisitionFrameRateControl

映像のフレームレート設定を設定します。

#### [構文]

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

#### [パラメータ]

パラメータ		内 容
<i>hCam</i>	[in]	カメラのカメラハンドルです。
<i>eFrameRateControl</i>	[in]	設定するフレームレート設定です。

#### [戻り値]

実行結果を返します。 戻り値は、[5.9. ステータスコード](#) を参照してください。

#### [備考]

映像出力設定機能（AcquisitionFrameRateControl レジスタ）が実装されていないカメラで実行するとエラーになります。

NoSpecify（CAM\_ACQ\_FRAME\_RATE\_CTRL\_NO\_SPECIFY）を設定した場合、AcquisitionFrameRate レジスタの値が変わる場合があります。使用するカメラの動作を確認してからご使用ください。

CAM\_ACQ\_FRAME\_RATE\_CTRL\_TYPE 型は、TeliCamAPI.h で定義されています。

TeliCamAPI.h をインクルードする必要があります。

---

#### 5.5.8.8. GetCamAcquisitionFrameRateMinMax

映像のフレームレート最小値と最大値を取得します。

##### [構文]

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

##### [パラメータ]

パラメータ		内 容
<i>hCam</i>	[in]	カメラのカメラハンドルです。
<i>pdAcqFrameRateMin</i>	[out]	取得した最小値を格納する変数へのポインタです。
<i>pdAcqFrameRateMax</i>	[out]	取得した最大値を格納する変数へのポインタです。

##### [戻り値]

実行結果を返します。 戻り値は、[5.9. ステータスコード](#) を参照してください。

##### [備考]

映像出力設定機能（AcquisitionFrameRate レジスタ）が実装されていないカメラで実行するとエラーになります。

TeliCamAPI.h をインクルードする必要があります。

---

### 5.5.8.9. GetCamAcquisitionFrameRate

映像のフレームレート設定値を取得します。

#### [構文]

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

#### [パラメータ]

パラメータ		内 容
<i>hCam</i>	[in]	カメラのカメラハンドルです。
<i>pdAcqFrameRate</i>	[out]	取得したフレームレート設定値を格納する変数へのポインタです。

#### [戻り値]

実行結果を返します。 戻り値は、[5.9. ステータスコード](#) を参照してください。

#### [備考]

映像出力設定機能（AcquisitionFrameRate レジスタ）が実装されていないカメラで実行するとエラーになります。

TeliCamAPI.h をインクルードする必要があります。

---

### 5.5.8.10. SetCamAcquisitionFrameRate

映像のフレームレートを設定します。

#### [構文]

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

#### [パラメータ]

パラメータ		内 容
<i>hCam</i>	[in]	カメラのカメラハンドルです。
<i>dAcqFrameRate</i>	[in]	設定するフレームレートです。

#### [戻り値]

実行結果を返します。 戻り値は、[5.9. ステータスコード](#) を参照してください。

#### [備考]

映像出力設定機能（AcquisitionFrameRate レジスタ）が実装されていないカメラで実行するとエラーになります。

ストリーム転送が開始されている場合にも設定できるカメラと、設定できないカメラがあります。

TeliCamAPI.h をインクルードする必要があります。

---

## 5.5.9. ImageBuffer

カメラのイメージバッファ機能の制御を行います。  
イメージバッファ機能の説明は、カメラの取扱説明書をご覧ください。

### 5.5.9.1. GetCamImageBufferMode

カメラのイメージバッファモード設定値を取得します。

#### [構文]

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

#### [パラメータ]

パラメータ		内 容
<i>hCam</i>	[in]	カメラのカメラハンドルです。
<i>peMode</i>	[out]	取得したイメージバッファモード設定値を格納する変数へのポインタです。

#### [戻り値]

実行結果を返します。 戻り値は、[5.9. ステータスコード](#) を参照してください。

#### [備考]

イメージバッファ機能（ImageBufferMode レジスタ）が実装されていないカメラで実行するとエラーになります。

カメラのイメージバッファの使用方法については、ハードウェアの取扱説明書をご覧ください。

CAM\_IMAGE\_BUFFER\_MODE\_TYPE 型は、TeliCamAPI.h で定義されています。

TeliCamAPI.h をインクルードする必要があります。

---

### 5.5.9.2. SetCamImageBufferMode

カメラのイメージバッファモード ON/OFF を設定します。

#### [構文]

```
CAM_API_STATUS SetCamImageBufferMode (  
    CAM_HANDLE          hCam,  
    CAM_IMAGE_BUFFER_MODE_TYPE eMode  
);
```

#### [パラメータ]

パラメータ		内 容
<i>hCam</i>	[in]	カメラのカメラハンドルです。
<i>eMode</i>	[in]	設定するイメージバッファモード ON/OFF です。

#### [戻り値]

実行結果を返します。 戻り値は、[5.9. ステータスコード](#) を参照してください。

#### [備考]

イメージバッファ機能（ImageBufferMode レジスタ）が実装されていないカメラで実行するとエラーになります。

カメラのイメージバッファの使用方法については、ハードウェアの取扱説明書をご覧ください。

ストリーム転送が開始されている場合は設定を行うことはできません。

CAM\_IMAGE\_BUFFER\_MODE\_TYPE 型は、TeliCamAPI.h で定義されています。

TeliCamAPI.h をインクルードする必要があります。



---

### 5.5.9.3. GetCamImageBufferFrameCount

カメラのイメージバッファに取り込まれた画像枚数を取得します。

#### [構文]

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

#### [パラメータ]

パラメータ		内 容
<i>hCam</i>	[in]	カメラのカメラハンドルです。
<i>puiCount</i>	[out]	取得したイメージバッファに取り込まれた画像枚数を格納する変数へのポインタです。

#### [戻り値]

実行結果を返します。 戻り値は、[5.9. ステータスコード](#) を参照してください。

#### [備考]

イメージバッファ機能（ImageBufferCount レジスタ）が実装されていないカメラで実行するとエラーになります。

カメラのイメージバッファの使用方法については、ハードウェアの取扱説明書をご覧ください。

TeliCamAPI.h をインクルードする必要があります。

---

#### 5.5.9.4. ExecuteCamImageBufferRead

カメラのイメージバッファ読み出しコマンドを実行します。

##### [構文]

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

##### [パラメータ]

パラメータ	内 容
<i>hCam</i> [in]	カメラのカメラハンドルです。

##### [戻り値]

実行結果を返します。 戻り値は、[5.9. ステータスコード](#) を参照してください。

##### [備考]

イメージバッファ機能（ImageBufferRead レジスタ）が実装されていないカメラで実行するとエラーになります。

1 回のコマンド実行でカメラが出力するストリームの転送枚数は、[SetCamAcquisitionFrameCount\(\)](#) で設定できます。

カメラから出力されたストリームは、[5.3 カメラ ストリーム関数](#) を使用して取り込んでください。

カメラのイメージバッファの使用方法については、ハードウェアの取扱説明書をご覧ください。

TeliCamAPI.h をインクルードする必要があります。

---

## 5.5.10.TriggerControl

カメラのトリガ機能の制御を行います。

### 5.5.10.1. GetCamTriggerMode

カメラのトリガ動作モード設定値を取得します。

#### [構文]

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

#### [パラメータ]

パラメータ		内 容
<i>hCam</i>	[in]	カメラのカメラハンドルです。
<i>pbValue</i>	[out]	取得したトリガ動作モード設定値を格納する変数へのポインタです。 true の場合 トリガ ON、false の場合 トリガ OFF です。

#### [戻り値]

実行結果を返します。 戻り値は、[5.9. ステータスコード](#) を参照してください。

#### [備考]

トリガ制御機能（TriggerMode レジスタ）が実装されていないカメラで実行するとエラーになります。

TeliCamAPI.h をインクルードする必要があります。

---

### 5.5.10.2. SetCamTriggerMode

カメラのトリガ動作モードを設定します。

#### [構文]

```
CAM_API_STATUS SetCamTriggerMode (  
    CAM_HANDLE      hCam,  
    bool8_t          bValue  
);
```

#### [パラメータ]

パラメータ		内 容
<i>hCam</i>	[in]	カメラのカメラハンドルです。
<i>bValue</i>	[in]	設定するトリガ動作モードです。 true の場合 トリガ ON、false の場合 トリガ OFF です。

#### [戻り値]

実行結果を返します。 戻り値は、[5.9. ステータスコード](#) を参照してください。

#### [備考]

トリガ制御機能（TriggerMode レジスタ）が実装されていないカメラで実行するとエラーになります。

ストリーム転送が開始されている場合にも設定できるカメラと、設定できないカメラがあります。

TeliCamAPI.h をインクルードする必要があります。

### 5.5.10.3. GetCamTriggerSequence

カメラのトリガ ON 時における露光時間制御モード設定値を取得します。

#### [構文]

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

#### [パラメータ]

パラメータ	内 容
<i>hCam</i> [in]	カメラのカメラハンドルです。
<i>peTriggerSequence</i> [out]	取得した露光時間制御モード設定値を格納する変数へのポインタです。

#### [戻り値]

実行結果を返します。 戻り値は、[5.9. ステータスコード](#) を参照してください。

#### [備考]

トリガ制御機能（TriggerSequence、または ExposureMode レジスタ）が実装されていないカメラで実行するとエラーになります。

CAM\_TRIGGER\_SEQUENCE\_TYPE 型 は、以下の通りです。

value	Description
<i>CAM_TRIGGER_SEQUENCE0</i>	Edge モード。 露光時間は電子シャッタの設定値 USB3 Vision カメラ : TriggerSequence レジスタ = TriggerSequence0 GigE Vision カメラ : ExposureMode レジスタ = Timed TriggerSelector レジスタ = FrameStart
<i>CAM_TRIGGER_SEQUENCE1</i>	Level モード。 露光時間はトリガ信号のパルス幅 USB3 Vision カメラ: TriggerSequence レジスタ = TriggerSequence1 GigE Vision カメラ : ExposureMode レジスタ = TriggerWidth TriggerSelector レジスタ = FrameStart
<i>CAM_TRIGGER_SEQUENCE6</i>	Bulk (FrameBurst) モード。 1 回のトリガ信号入力で、連続して複数回の露光と映像出力を行います。 USB3 Vision カメラ : TriggerSequence レジスタ = TriggerSequence6 GigE Vision カメラ :

---

	ExposureMode レジスタ = Timed TriggerSelector レジスタ = FrameBurstStart
--	---

カメラによって、読み出すレジスタが異なります。

上記レジスタの説明は、カメラの取扱説明書をご覧ください。

CAM\_TRIGGER\_SEQUENCE\_TYPE 型は、TeliCamAPI.h で定義されています。

TeliCamAPI.h をインクルードする必要があります。

---

#### 5.5.10.4. SetCamTriggerSequence

カメラのトリガ ON 時における露光時間制御モードを設定します。

##### [構文]

```
CAM_API_STATUS SetCamTriggerSequence (  
    CAM_HANDLE          hCam,  
    CAM_TRIGGER_SEQUENCE_TYPE eTriggerSequence  
);
```

##### [パラメータ]

パラメータ		内 容
<i>hCam</i>	[in]	カメラのカメラハンドルです。
<i>eTriggerSequence</i>	[in]	設定する露光時間制御モードです。

##### [戻り値]

実行結果を返します。 戻り値は、[5.9. ステータスコード](#) を参照してください。

##### [備考]

トリガ制御機能（TriggerSequence、または ExposureMode レジスタ）が実装されていないカメラで実行するとエラーになります。

CAM\_TRIGGER\_SEQUENCE\_TYPE 型 は、[5.5.10.3 GetCamTriggerSequence](#) を参照してください。

上記レジスタの説明は、カメラの取扱説明書をご覧ください。

CAM\_TRIGGER\_SEQUENCE\_TYPE 型は、TeliCamAPI.h で定義されています。

TeliCamAPI.h をインクルードする必要があります。

---

### 5.5.10.5. GetCamTriggerSource

カメラのランダムトリガシャッタのトリガソース設定値を取得します。

#### [構文]

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

#### [パラメータ]

パラメータ		内 容
<i>hCam</i>	[in]	カメラのカメラハンドルです。
<i>peTriggerSource</i>	[out]	取得したトリガソース設定値を格納する変数へのポインタです。

#### [戻り値]

実行結果を返します。 戻り値は、[5.9. ステータスコード](#) を参照してください。

#### [備考]

トリガ制御機能（TriggerSource レジスタ）が実装されていないカメラで実行するとエラーになります。

CAM\_TRIGGER\_SOURCE\_TYPE 型は、TeliCamAPI.h で定義されています。

TeliCamAPI.h をインクルードする必要があります。



---

### 5.5.10.6. SetCamTriggerSource

カメラのランダムトリガシャッタのトリガソースを設定します。

#### [構文]

```
CAM_API_STATUS SetCamTriggerSource (  
    CAM_HANDLE          hCam,  
    CAM_TRIGGER_SOURCE_TYPE eTriggerSource  
);
```

#### [パラメータ]

パラメータ		内 容
<i>hCam</i>	[in]	カメラのカメラハンドルです。
<i>eTriggerSource</i>	[in]	設定するトリガソース設定値です。

#### [戻り値]

実行結果を返します。 戻り値は、[5.9. ステータスコード](#) を参照してください。

#### [備考]

トリガ制御機能（TriggerSource レジスタ）が実装されていないカメラで実行するとエラーになります。

CAM\_TRIGGER\_SOURCE\_TYPE 型は、TeliCamAPI.h で定義されています。

TeliCamAPI.h をインクルードする必要があります。

---

### 5.5.10.7. GetCamTriggerAdditionalParameterMinMax

Bulk (FrameBurst) モード動作設定時の露光回数の最小値と最大値を取得します。

#### [構文]

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

#### [パラメータ]

パラメータ	内 容
<i>hCam</i> [in]	カメラのカメラハンドルです。
<i>puiAdditionalParameterMin</i> [out]	取得した最小値を格納する変数へのポインタです。
<i>puiAdditionalParameterMax</i> [out]	取得した最大値を格納する変数へのポインタです。

#### [戻り値]

実行結果を返します。 戻り値は、[5.9. ステータスコード](#) を参照してください。

#### [備考]

Bulk (FrameBurst) モード動作 設定時、カメラは 1 回のトリガで、設定された枚数のフレームを転送します。

USB3 Vision カメラのときは、TriggerAdditionalParameter レジスタの最小値と最大値を取得します。

GigE Vision カメラのときは、AcquisitionBurstFrameCount レジスタの最小値と最大値を取得します。

トリガ制御機能（TriggerAdditionalParameter レジスタ、または AcquisitionBurstFrameCount レジスタ）が実装されていないカメラで実行するとエラーになります。

TeliCamAPI.h をインクルードする必要があります。

---

### 5.5.10.8. GetCamTriggerAdditionalParameter

Bulk (FrameBurst) モード動作設定時の露光回数を取得します。

#### [構文]

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

#### [パラメータ]

パラメータ	内 容
<i>hCam</i> [in]	カメラのカメラハンドルです。
<i>puiAdditionalParameter</i> [out]	取得した露光回数を格納する変数へのポインタです。

#### [戻り値]

実行結果を返します。 戻り値は、[5.9. ステータスコード](#) を参照してください。

#### [備考]

Bulk (FrameBurst) モード動作 設定時、カメラは 1 回のトリガで、設定された枚数のフレームを転送します。

USB3 Vision カメラのときは、TriggerAdditionalParameter レジスタの値を取得します。

GigE Vision カメラのときは、AcquisitionBurstFrameCount レジスタの値を取得します。

トリガ制御機能（TriggerAdditionalParameter レジスタ、または AcquisitionBurstFrameCount レジスタ）が実装されていないカメラで実行するとエラーになります。

TeliCamAPI.h をインクルードする必要があります。

---

### 5.5.10.9. SetCamTriggerAdditionalParameter

Bulk (FrameBurst) モード動作設定時の露光回数を設定します。

#### [構文]

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

#### [パラメータ]

パラメータ	内 容
<i>hCam</i> [in]	カメラのカメラハンドルです。
<i>uiAdditionalParameter</i> [in]	設定する露光回数です。

#### [戻り値]

実行結果を返します。 戻り値は、[5.9. ステータスコード](#) を参照してください。

#### [備考]

Bulk (FrameBurst) モード動作 設定時、カメラは 1 回のトリガで、設定された枚数のフレームを転送します。

USB3 Vision カメラのときは、TriggerAdditionalParameter レジスタの値を設定します。

GigE Vision カメラのときは、AcquisitionBurstFrameCount レジスタの値を設定します。

トリガ制御機能（TriggerAdditionalParameter レジスタ、または AcquisitionBurstFrameCount レジスタ）が実装されていないカメラで実行するとエラーになります。

TeliCamAPI.h をインクルードする必要があります。

---

#### 5.5.10.10. GetCamTriggerDelayMinMax

カメラのトリガ信号検出から露光開始までの遅延量の最小値と最大値を取得します。

##### [構文]

```
CAM_API_STATUS GetCamTriggerDelayMinMax (  
    CAM_HANDLE      hCam,  
    float64_t        *pdDealyUsMin,  
    float64_t        *pdDealyUsMax  
);
```

##### [パラメータ]

パラメータ		内 容
hCam	[in]	カメラのカメラハンドルです。
pdDealyUsMin	[out]	取得した最小値を格納する変数へのポインタです。 (単位: $\mu$ sec)
pdDealyUsMax	[out]	取得した最大値を格納する変数へのポインタです。 (単位: $\mu$ sec)

##### [戻り値]

実行結果を返します。 戻り値は、[5.9. ステータスコード](#) を参照してください。

##### [備考]

トリガ制御機能 (TriggerDelay レジスタ) が実装されていないカメラで実行するとエラーになります。

TeliCamAPI.h をインクルードする必要があります。

---

### 5.5.10.11. GetCamTriggerDelay

カメラのトリガ信号検出から露光開始までの遅延量を取得します。

#### [構文]

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

#### [パラメータ]

パラメータ		内 容
<i>hCam</i>	[in]	カメラのカメラハンドルです。
<i>pdDeaLyUs</i>	[out]	取得した遅延量を格納する変数へのポインタです。 (単位： $\mu$ sec)。

#### [戻り値]

実行結果を返します。 戻り値は、[5.9. ステータスコード](#) を参照してください。

#### [備考]

トリガ制御機能（TriggerDelay レジスタ）が実装されていないカメラで実行するとエラーになります。

TeliCamAPI.h をインクルードする必要があります。

---

### 5.5.10.12. SetCamTriggerDelay

カメラのトリガ信号検出から露光開始までの遅延量を設定します。

#### [構文]

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

#### [パラメータ]

パラメータ		内 容
<i>hCam</i>	[in]	カメラのカメラハンドルです。
<i>dDealyUs</i>	[in]	設定する遅延量です。（単位：μsec）。

#### [戻り値]

実行結果を返します。 戻り値は、[5.9. ステータスコード](#) を参照してください。

#### [備考]

トリガ制御機能（TriggerDelay レジスタ）が実装されていないカメラで実行するとエラーになります。

TeliCamAPI.h をインクルードする必要があります。

---

### 5.5.10.13. ExecuteCamSoftwareTrigger

カメラのソフトウェアトリガを実行します。

#### [構文]

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

#### [パラメータ]

パラメータ	内 容
<i>hCam</i> [in]	カメラのカメラハンドルです。

#### [戻り値]

実行結果を返します。 戻り値は、[5.9. ステータスコード](#) を参照してください。

#### [備考]

正常終了しても、カメラがソフトウェアトリガ受付可能状態でない場合は、何も実行されない場合があります。

TeliCamAPI.h をインクルードする必要があります。



### 5.5.11.ExposureTime

カメラの露光時間の制御を行います。  
露光時間の説明は、カメラの取扱説明書をご覧ください。

#### 5.5.11.1. GetCamExposureTimeControl

カメラの露光時間 制御モードの設定値を取得します。

##### [構文]

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

##### [パラメータ]

パラメータ	内 容
<i>hCam</i> [in]	カメラのカメラハンドルです。
<i>peExpControl</i> [out]	取得した制御モード設定値を格納する変数へのポインタです。

##### [戻り値]

実行結果を返します。 戻り値は、[5.9. ステータスコード](#) を参照してください。

##### [備考]

露光時間制御機能（ExposureTimeControl、または ExposureAuto レジスタ）が実装されていないカメラで実行するとエラーになります。

CAM\_EXPOSURE\_TIME\_CONTROL\_TYPE 型 は、以下の通りです。

value	Description
<i>CAM_EXPOSURE_TIME_CONTROL_AUTO</i>	自動露光時間制御 USB3 Vision カメラ : ExposureTimeControl レジスタ = Auto GigE Vision カメラ : ExposureAuto レジスタ = Continuous
<i>CAM_EXPOSURE_TIME_CONTROL_MANUAL</i>	ExposureTime の設定値優先 USB3 Vision カメラ: ExposureTimeControl レジスタ = Manual GigE Vision カメラ : ExposureAuto レジスタ = Off
<i>CAM_EXPOSURE_TIME_CONTROL_NO_SPECIFY</i>	AcquisitionFrameRate の設定値優先 USB3 Vision カメラ : ExposureTimeControl レジスタ = NoSpecify

---

	GigE Vision カメラ : 利用不可
--	---------------------------

カメラによって、読み出すレジスタが異なります。読み出すレジスタの説明は、カメラの取扱説明書をご覧ください。

CAM\_EXPOSURE\_TIME\_CONTROL\_TYPE 型は、TeliCamAPI.h で定義されています。

TeliCamAPI.h をインクルードする必要があります。

---

### 5.5.11.2. SetCamExposureTimeControl

カメラの露光時間 制御モードを設定します。

#### [構文]

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

#### [パラメータ]

パラメータ		内 容
<i>hCam</i>	[in]	カメラのカメラハンドルです。
<i>eExpControl</i>	[in]	設定する制御モードです。

#### [戻り値]

実行結果を返します。 戻り値は、[5.9. ステータスコード](#) を参照してください。

#### [備考]

露光時間制御機能（ExposureTimeControl、または ExposureAuto レジスタ）が実装されていないカメラで実行するとエラーになります。

CAM\_EXPOSURE\_TIME\_CONTROL\_TYPE 型は、[5.5.11.1 GetCamExposureTimeControl](#) を参照してください。

カメラによって、書き込むレジスタが異なります。書き込むレジスタの説明は、カメラの取扱説明書をご覧ください。

CAM\_EXPOSURE\_TIME\_CONTROL\_TYPE 型は、TeliCamAPI.h で定義されています。

TeliCamAPI.h をインクルードする必要があります。

---

### 5.5.11.3. GetCamExposureTimeMinMax

カメラの露光時間制御モードが Manual 設定時の、露光時間 最小値と最大値を取得します。

#### [構文]

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

#### [パラメータ]

パラメータ		内 容
<i>hCam</i>	[in]	カメラのカメラハンドルです。
<i>pdExpTimeUsMin</i>	[out]	取得した最小値を格納する変数へのポインタです。
<i>pdExpTimeUsMax</i>	[out]	取得した最大値を格納する変数へのポインタです。

#### [戻り値]

実行結果を返します。 戻り値は、[5.9. ステータスコード](#) を参照してください。

#### [備考]

露光時間制御機能（ExposureTime レジスタ）が実装されていないカメラで実行するとエラーになります。

TeliCamAPI.h をインクルードする必要があります。

---

#### 5.5.11.4. GetCamExposureTime

カメラの露光時間制御モードが Manual 設定時の、露光時間を取得します。

##### [構文]

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

##### [パラメータ]

パラメータ		内 容
<i>hCam</i>	[in]	カメラのカメラハンドルです。
<i>pdExpTimeUs</i>	[out]	取得した露光時間を格納する変数へのポインタです。（単位：μsec）

##### [戻り値]

実行結果を返します。 戻り値は、[5.9. ステータスコード](#) を参照してください。

##### [備考]

露光時間制御機能（ExposureTime レジスタ）が実装されていないカメラで実行するとエラーになります。

TeliCamAPI.h をインクルードする必要があります。

---

### 5.5.11.5. SetCamExposureTime

カメラの露光時間制御モードが Manual 設定時の、露光時間を設定します。

#### [構文]

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

#### [パラメータ]

パラメータ		内 容
<i>hCam</i>	[in]	カメラのカメラハンドルです。
<i>dExpTimeUs</i>	[in]	設定する露光時間です。（単位：μ sec）

#### [戻り値]

実行結果を返します。 戻り値は、[5.9. ステータスコード](#) を参照してください。

#### [備考]

露光時間制御機能（ExposureTime レジスタ）が実装されていないカメラで実行するとエラーになります。

TeliCamAPI.h をインクルードする必要があります。

---

## 5.5.12.DigitalIoControl

カメラのデジタル I/O の制御を行います。  
デジタル I/O の説明は、カメラの取扱説明書をご覧ください。

### 5.5.12.1. GetCamLineModeAll

カメラの全ラインの入出力設定を取得します。

#### [構文]

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

#### [パラメータ]

パラメータ	内 容
<i>hCam</i> [in]	カメラのカメラハンドルです。
<i>puiValue</i> [out]	取得した全ラインの入出力設定を格納する変数へのポインタです。 各 bit が各 Line に対応しています。(Line0 : Bit0 , Line1 : Bit1 , Line2 : Bit2 , ... ) bit データが 0 のラインは入力、1 のラインは出力です。 ( <i>puiValue</i> = 0x06 のとき、Line0 : 入力 , Line1 : 出力 , Line2 : 出力 , ...)

#### [戻り値]

実行結果を返します。 戻り値は、[5.9. ステータスコード](#) を参照してください。

#### [備考]

デジタル I/O 制御機能 (LineModeAll、または LineSelector と LineMode レジスタ) が実装されていないカメラで実行するとエラーになります。

カメラによって、読み出すレジスタが異なります。読み出すレジスタの説明は、カメラの取扱説明書をご覧ください。

TeliCamAPI.h をインクルードする必要があります。

---

### 5.5.12.2. GetCamLineInverterAll

カメラの全ラインの極性設定を取得します。

#### [構文]

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

#### [パラメータ]

パラメータ		内 容
<i>hCam</i>	[in]	カメラのカメラハンドルです。
<i>puiValue</i>	[out]	取得した全ラインの極性設定を格納する変数へのポインタです。 各 bit が各 Line に対応しています。(Line0 : Bit0 , Line1 : Bit1 , Line2 : Bit2 , ... ) bit データが 0 のラインは Invert なし、1 のラインは Invert あり です。 ( <i>puiValue</i> = 0x02 のとき、Line0: Invert なし , Line1 : Invert あり , Line2 : Invert なし , ...)

#### [戻り値]

実行結果を返します。 戻り値は、[5.9. ステータスコード](#) を参照してください。

#### [備考]

デジタル I/O 制御機能 (LineInverterAll、または LineSelector と LineInverter レジスタ) が実装されていないカメラで実行するとエラーになります。

カメラによって、読み出すレジスタが異なります。読み出すレジスタの説明は、カメラの取扱説明書をご覧ください。

TeliCamAPI.h をインクルードする必要があります。



### 5.5.12.3. SetCamLineInverterAll

カメラの全ラインの極性を設定します。

#### [構文]

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

#### [パラメータ]

パラメータ		内 容
<i>hCam</i>	[in]	カメラのカメラハンドルです。
<i>uiValue</i>	[in]	設定する全ラインの極性です。 各 bit が各 Line に対応しています。(Line0 : Bit0 , Line1 : Bit1 , Line2 : Bit2 , ... ) bit データが 0 のラインは Invert なし、1 のラインは Invert あり です。 ( <i>uiValue</i> = 0x02 のとき、Line0: Invert なし , Line1 : Invert あり , Line2 : Invert なし , ...)

#### [戻り値]

実行結果を返します。 戻り値は、[5.9. ステータスコード](#) を参照してください。

#### [備考]

デジタル I/O 制御機能 (LineInverterAll、または LineSelector と LineInverter レジスタ) が実装されていないカメラで実行するとエラーになります。

カメラによって、書き込むレジスタおよび、書き込み可能な Line が異なります。書き込みができない Line の bit データに 1 (Invert あり) が設定されている場合、CAM\_API\_STS\_INVALID\_PARAMETER がリターンされる場合があります。(GigE Vision カメラは、Line0 に 1 (Invert あり) を設定することはできません。)

上記レジスタの説明は、カメラの取扱説明書をご覧ください。

TeliCamAPI.h をインクルードする必要があります。

---

#### 5.5.12.4. GetCamLineStatusAll

カメラの全ラインの現在の状態を取得します。

##### [構文]

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

##### [パラメータ]

パラメータ	内 容
<i>hCam</i> [in]	カメラのカメラハンドルです。
<i>puiValue</i> [out]	取得した全ラインの現在の状態を格納する変数へのポインタです。 各 bit が各 Line に対応しています。 (Line0 : Bit0 , Line1 : Bit1 , Line2 : Bit2 , ... )

##### [戻り値]

実行結果を返します。 戻り値は、[5.9. ステータスコード](#) を参照してください。

##### [備考]

デジタル I/O 制御機能 (LineStatusAll レジスタ) が実装されていないカメラで実行するとエラーになります。

TeliCamAPI.h をインクルードする必要があります。

---

### 5.5.12.5. GetCamUserOutputValueAll

カメラの全ラインのユーザ出力設定値を取得します。

#### [構文]

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

#### [パラメータ]

パラメータ	内 容
<i>hCam</i> [in]	カメラのカメラハンドルです。
<i>puiValue</i> [out]	取得した全ラインのユーザ出力設定値を格納する変数へのポインタです。 各 bit が各 Line に対応しています。 (Line0 : Bit0 , Line1 : Bit1 , Line2 : Bit2 , ... )

#### [戻り値]

実行結果を返します。 戻り値は、[5.9. ステータスコード](#) を参照してください。

#### [備考]

デジタル I/O 制御機能 (UserOutputValueAll レジスタ) が実装されていないカメラで実行するとエラーになります。

TeliCamAPI.h をインクルードする必要があります。

---

### 5.5.12.6. SetCamUserOutputValueAll

カメラの全ラインのユーザ出力設定値を設定します。

#### [構文]

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

#### [パラメータ]

パラメータ		内 容
<i>hCam</i>	[in]	カメラのカメラハンドルです。
<i>uiValue</i>	[in]	設定する全ラインのユーザ出力設定値です。 各 bit が各 Line に対応しています。(Line0 : Bit0 , Line1 : Bit1 , Line2 : Bit2 , ... )

#### [戻り値]

実行結果を返します。 戻り値は、[5.9. ステータスコード](#) を参照してください。

#### [備考]

信号出力が UserOutput (CAM\_LINE\_SOURCE\_USER\_OUTPUT) に設定されている Line 以外の出力設定値は無視されます。

デジタル I/O 制御機能 (UserOutputValueAll レジスタ) が実装されていないカメラで実行するとエラーになります。

TeliCamAPI.h をインクルードする必要があります。

---

### 5.5.12.7. GetCamLineSource

カメラのラインの信号種類設定値を取得します。

#### [構文]

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

#### [パラメータ]

パラメータ		内 容
<i>hCam</i>	[in]	カメラのカメラハンドルです。
<i>eLineSelector</i>	[in]	取得する LINE です。
<i>peLineSource</i>	[out]	取得した信号種類設定値を格納する変数へのポインタです。

#### [戻り値]

実行結果を返します。 戻り値は、[5.9. ステータスコード](#) を参照してください。

#### [備考]

デジタル I/O 制御機能（LineSelector、 LineSource レジスタ）が実装されていないカメラで実行するとエラーになります。

CAM\_LINE\_SELECTOR\_TYPE 型と CAM\_LINE\_SOURCE\_TYPE 型は、TeliCamAPI.h で定義されています。

TeliCamAPI.h をインクルードする必要があります。

---

### 5.5.12.8. SetCamLineSource

カメラのラインの信号種類を設定します。

#### [構文]

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

#### [パラメータ]

パラメータ		内 容
<i>hCam</i>	[in]	カメラのカメラハンドルです。
<i>eLineSelector</i>	[in]	設定する LINE です。
<i>eLineSource</i>	[in]	設定する信号種類です。

#### [戻り値]

実行結果を返します。 戻り値は、[5.9. ステータスコード](#) を参照してください。

#### [備考]

デジタル I/O 制御機能 (LineSelector、LineSource レジスタ) が実装されていないカメラで実行するとエラーになります。

カメラにより、設定できる信号種類が異なります。カメラの取扱説明書をご覧ください。

CAM\_LINE\_SELECTOR\_TYPE 型と CAM\_LINE\_SOURCE\_TYPE 型は、TeliCamAPI.h で定義されています。

TeliCamAPI.h をインクルードする必要があります。

### 5.5.13.TimerConrtol

カメラの TimerConrtol 機能（Timer0Active 信号）の制御を行います。

カメラのタイマーは TimerActive 信号の生成に使用されます。 .



現状の BU、BG シリーズカメラはタイマーを1本のみ保有しているなのでこの章の関数の制御対象は Timer0Active 信号に固定されます。

TimerConrtol 機能（Timer0Active 信号）信号の説明は、カメラの取扱説明書をご覧ください。

#### 5.5.13.1. GetCamTimerDurationMinMax

カメラの Timer0Active 信号の幅の最小値と最大値を取得します。

##### [構文]

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

##### [パラメータ]

パラメータ		内 容
<i>hCam</i>	[in]	カメラのカメラハンドルです。
<i>pdTimerDurationMin</i>	[out]	取得した最小値を格納する変数へのポインタです。 (単位: $\mu\text{sec}$ )
<i>pdTimerDurationMax</i>	[out]	取得した最大値を格納する変数へのポインタです。 (単位: $\mu\text{sec}$ )

##### [戻り値]

実行結果を返します。 戻り値は、[5.9. ステータスコード](#) を参照してください。

##### [備考]

TimerConrtol 機能（TimerDuration レジスタ）が実装されていないカメラで実行するとエラーになります。

TeliCamAPI.h をインクルードする必要があります。

---

### 5.5.13.2. GetCamTimerDuration

カメラの Timer0Active 信号の幅の設定値を取得します。

#### [構文]

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

#### [パラメータ]

パラメータ		内 容
<i>hCam</i>	[in]	カメラのカメラハンドルです。
<i>pdTimerDuration</i>	[out]	取得した Timer0Active 信号の幅の設定値を格納する変数へのポインタです。（単位：μsec）

#### [戻り値]

実行結果を返します。 戻り値は、[5.9. ステータスコード](#) を参照してください。

#### [備考]

TimerConrtol 機能（TimerDuration レジスタ）が実装されていないカメラで実行するとエラーになります。

TeliCamAPI.h をインクルードする必要があります。



---

### 5.5.13.3. SetCamTimerDuration

カメラの Timer0Active 信号の幅を設定します。

#### [構文]

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

#### [パラメータ]

パラメータ		内 容
<i>hCam</i>	[in]	カメラのカメラハンドルです。
<i>dTimerDuration</i>	[in]	設定する Timer0Active 信号の幅です。(単位: $\mu\text{sec}$ )

#### [戻り値]

実行結果を返します。 戻り値は、[5.9. ステータスコード](#) を参照してください。

#### [備考]

TimerConrtol 機能 (TimerDuration レジスタ) が実装されていないカメラで実行するとエラーになります。

TeliCamAPI.h をインクルードする必要があります。

---

#### 5.5.13.4. GetCamTimerDelayMinMax

カメラの Timer0Active 信号の遅延量の最小値と最大値を取得します。

##### [構文]

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

##### [パラメータ]

パラメータ		内 容
<i>hCam</i>	[in]	カメラのカメラハンドルです。
<i>pdTimerDelayMin</i>	[out]	取得した最小値を格納する変数へのポインタです。 (単位: $\mu$ sec)
<i>pdTimerDelayMax</i>	[out]	取得した最大値を格納する変数へのポインタです。 (単位: $\mu$ sec)

##### [戻り値]

実行結果を返します。 戻り値は、[5.9. ステータスコード](#) を参照してください。

##### [備考]

TimerConrtol 機能 (TimerDelay レジスタ) が実装されていないカメラで実行するとエラーになります。

TeliCamAPI.h をインクルードする必要があります。

---

### 5.5.13.5. GetCamTimerDelay

カメラの Timer0Active 信号の遅延量の設定値を取得します。

#### [構文]

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

#### [パラメータ]

パラメータ		内 容
<i>hCam</i>	[in]	カメラのカメラハンドルです。
<i>pdTimerDelay</i>	[out]	取得した Timer0Active 信号の遅延量の設定値を格納する変数へのポインタです。（単位：μsec）

#### [戻り値]

実行結果を返します。 戻り値は、[5.9. ステータスコード](#) を参照してください。

#### [備考]

TimerConrtol 機能（TimerDelay レジスタ）が実装されていないカメラで実行するとエラーになります。

TeliCamAPI.h をインクルードする必要があります。

---

### 5.5.13.6. SetCamTimerDelay

カメラの Timer0Active 信号の遅延量を設定します。

#### [構文]

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

#### [パラメータ]

パラメータ		内 容
<i>hCam</i>	[in]	カメラのカメラハンドルです。
<i>dTimerDelay</i>	[in]	設定する Timer0Active 信号の遅延量です。 (単位: $\mu$ sec)

#### [戻り値]

実行結果を返します。 戻り値は、[5.9. ステータスコード](#) を参照してください。

#### [備考]

TimerConrtol 機能 (TimerDelay レジスタ) が実装されていないカメラで実行するとエラーになります。

TeliCamAPI.h をインクルードする必要があります。

---

### 5.5.13.7. GetCamTimerTriggerSource

カメラの Timer0Active 信号の基準信号 設定値を取得します。

#### [構文]

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

#### [パラメータ]

パラメータ		内 容
<i>hCam</i>	[in]	カメラのカメラハンドルです。
<i>peTimerTriggerSource</i>	[out]	取得した Timer0Active 信号の基準信号の設定値を格納する変数へのポインタです。

#### [戻り値]

実行結果を返します。 戻り値は、[5.9. ステータスコード](#) を参照してください。

#### [備考]

TimerConrtol 機能（TimerTriggerSource レジスタ）が実装されていないカメラで実行するとエラーになります。

CAM\_TIMER\_TRIGGER\_SOURCE\_TYPE 型は、TeliCamAPI.h で定義されています。

TeliCamAPI.h をインクルードする必要があります。

---

### 5.5.13.8. SetCamTimerTriggerSource

カメラの Timer0Active 信号の基準信号を設定します。

#### [構文]

```
CAM_API_STATUS SetCamTimerTriggerSource (  
    CAM_HANDLE          hCam,  
    CAM_TIMER_TRIGGER_SOURCE_TYPE eTimerTriggerSource  
);
```

#### [パラメータ]

パラメータ		内 容
<i>hCam</i>	[in]	カメラのカメラハンドルです。
<i>eTimerTriggerSource</i>	[in]	設定する Timer0Active 信号の基準信号です。

#### [戻り値]

実行結果を返します。 戻り値は、[5.9. ステータスコード](#) を参照してください。

#### [備考]

TimerConrtol 機能（TimerTriggerSource レジスタ）が実装されていないカメラで実行するとエラーになります。

カメラにより、設定できる基準信号種類が異なります。カメラの取扱説明書をご覧ください。

CAM\_TIMER\_TRIGGER\_SOURCE\_TYPE 型は、TeliCamAPI.h で定義されています。

TeliCamAPI.h をインクルードする必要があります。

## 5.5.14.Gain

カメラのゲイン調整機能の制御を行います。  
ゲイン調整機能の説明は、カメラの取扱説明書をご覧ください。

### 5.5.14.1. GetCamGainMinMax

カメラのゲインの最小値と最大値を取得します。

#### [構文]

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

#### [パラメータ]

パラメータ		内 容
<i>hCam</i>	[in]	カメラのカメラハンドルです。
<i>pdGainMin</i>	[out]	取得した最小値を格納する変数へのポインタです。 (単位：dB)
<i>pdGainMax</i>	[out]	取得した最大値を格納する変数へのポインタです。 (単位：dB)

#### [戻り値]

実行結果を返します。 戻り値は、[5.9. ステータスコード](#) を参照してください。

#### [備考]

ゲイン調整機能（Gain レジスタ）が実装されていないカメラで実行するとエラーになります。

TeliCamAPI.h をインクルードする必要があります。

---

### 5.5.14.2. GetCamGain

カメラのゲインの設定値を取得します。

#### [構文]

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

#### [パラメータ]

パラメータ	内 容
<i>hCam</i> [in]	カメラのカメラハンドルです。
<i>pdGain</i> [out]	取得したゲインの設定値を格納する変数へのポインタです。（単位：dB）

#### [戻り値]

実行結果を返します。 戻り値は、[5.9. ステータスコード](#) を参照してください。

#### [備考]

ゲイン調整機能（Gain レジスタ）が実装されていないカメラで実行するとエラーになります。

TeliCamAPI.h をインクルードする必要があります。



---

### 5.5.14.3. SetCamGain

カメラのゲインを設定します。

#### [構文]

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

#### [パラメータ]

パラメータ		内 容
<i>hCam</i>	[in]	カメラのカメラハンドルです。
<i>dGain</i>	[in]	設定するゲインです。（単位：dB）

#### [戻り値]

実行結果を返します。 戻り値は、[5.9. ステータスコード](#) を参照してください。

#### [備考]

ゲイン調整機能（Gain レジスタ）が実装されていないカメラで実行するとエラーになります。

TeliCamAPI.h をインクルードする必要があります。

---

#### 5.5.14.4. GetCamGainAuto

カメラの AGC 動作モードの設定値を取得します。

##### [構文]

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

##### [パラメータ]

パラメータ		内 容
<i>hCam</i>	[in]	カメラのカメラハンドルです。
<i>peGainAuto</i>	[out]	取得した AGC 動作モードの設定値を格納する変数へのポインタです。

##### [戻り値]

実行結果を返します。 戻り値は、[5.9. ステータスコード](#) を参照してください。

##### [備考]

ゲイン調整機能（GainAuto レジスタ）が実装されていないカメラで実行するとエラーになります。

CAM\_GAIN\_AUTO\_TYPE 型は、TeliCamAPI.h で定義されています。

TeliCamAPI.h をインクルードする必要があります。

---

### 5.5.14.5. SetCamGainAuto

カメラの AGC 動作モードを設定します。

#### [構文]

```
CAM_API_STATUS SetCamGainAuto (  
    CAM_HANDLE          hCam,  
    CAM_GAIN_AUTO_TYPE  eGainAuto  
);
```

#### [パラメータ]

パラメータ		内 容
<i>hCam</i>	[in]	カメラのカメラハンドルです。
<i>eGainAuto</i>	[in]	設定する AGC 動作モードです。

#### [戻り値]

実行結果を返します。 戻り値は、[5.9. ステータスコード](#) を参照してください。

#### [備考]

ゲイン調整機能（GainAuto レジスタ）が実装されていないカメラで実行するとエラーになります。

CAM\_GAIN\_AUTO\_TYPE 型は、TeliCamAPI.h で定義されています。

TeliCamAPI.h をインクルードする必要があります。

---

### 5.5.15.BlackLevel

カメラの黒レベル調整機能の制御を行います。  
黒レベル調整機能の説明は、カメラの取扱説明書をご覧ください。

#### 5.5.15.1. GetCamBlackLevelMinMax

カメラの黒レベルの最小値と最大値を取得します。

##### [構文]

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

##### [パラメータ]

パラメータ		内 容
hCam	[in]	カメラのカメラハンドルです。
pdBlackLevelMin	[out]	取得した最小値を格納する変数へのポインタです。（単位：％）
pdBlackLevelMax	[out]	取得した最大値を格納する変数へのポインタです。（単位：％）

##### [戻り値]

実行結果を返します。 戻り値は、[5.9. ステータスコード](#) を参照してください。

##### [備考]

黒レベル調整機能（BlackLevel レジスタ）が実装されていないカメラで実行するとエラーになります。

TeliCamAPI.h をインクルードする必要があります。

---

### 5.5.15.2. GetCamBlackLevel

カメラの黒レベルの設定値を取得します。

#### [構文]

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

#### [パラメータ]

パラメータ	内 容
<i>hCam</i> [in]	カメラのカメラハンドルです。
<i>pdBlackLevel</i> [out]	取得した黒レベルの設定値を格納する変数へのポインタです。（単位：％）

#### [戻り値]

実行結果を返します。 戻り値は、[5.9. ステータスコード](#) を参照してください。

#### [備考]

黒レベル調整機能（BlackLevel レジスタ）が実装されていないカメラで実行するとエラーになります。

TeliCamAPI.h をインクルードする必要があります。

---

### 5.5.15.3. SetCamBlackLevel

カメラの黒レベルを設定します。

#### [構文]

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

#### [パラメータ]

パラメータ		内 容
<i>hCam</i>	[in]	カメラのカメラハンドルです。
<i>dBlackLevel</i>	[in]	設定する黒レベルです。（単位：％）

#### [戻り値]

実行結果を返します。 戻り値は、[5.9. ステータスコード](#) を参照してください。

#### [備考]

黒レベル調整機能（BlackLevel レジスタ）が実装されていないカメラで実行するとエラーになります。

TeliCamAPI.h をインクルードする必要があります。

---

## 5.5.16.Gamma

カメラのガンマ補正機能の制御を行います。  
ガンマ補正機能の説明は、カメラの取扱説明書をご覧ください。

### 5.5.16.1. GetCamGammaMinMax

カメラのガンマ補正值の最小値と最大値を取得します。

#### [構文]

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

#### [パラメータ]

パラメータ		内 容
<i>hCam</i>	[in]	カメラのカメラハンドルです。
<i>pdGammaMin</i>	[out]	取得した最小値を格納する変数へのポインタです。
<i>pdGammaMax</i>	[out]	取得した最大値を格納する変数へのポインタです。

#### [戻り値]

実行結果を返します。 戻り値は、[5.9. ステータスコード](#) を参照してください。

#### [備考]

ガンマ補正機能（Gamma レジスタ）が実装されていないカメラで実行するとエラーになります。

TeliCamAPI.h をインクルードする必要があります。

---

### 5.5.16.2. GetCamGamma

カメラのガンマ補正值の設定値を取得します。

#### [構文]

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

#### [パラメータ]

パラメータ		内 容
<i>hCam</i>	[in]	カメラのカメラハンドルです。
<i>pdGamma</i>	[out]	取得したガンマ補正值の設定値を格納する変数へのポインタです。

#### [戻り値]

実行結果を返します。 戻り値は、[5.9. ステータスコード](#) を参照してください。

#### [備考]

ガンマ補正機能（Gamma レジスタ）が実装されていないカメラで実行するとエラーになります。

TeliCamAPI.h をインクルードする必要があります。



---

### 5.5.16.3. SetCamGamma

カメラのガンマ補正値を設定します。

#### [構文]

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

#### [パラメータ]

パラメータ		内 容
<i>hCam</i>	[in]	カメラのカメラハンドルです。
<i>dGamma</i>	[in]	設定するガンマ補正値です。

#### [戻り値]

実行結果を返します。 戻り値は、[5.9. ステータスコード](#) を参照してください。

#### [備考]

ガンマ補正機能（Gamma レジスタ）が実装されていないカメラで実行するとエラーになります。

TeliCamAPI.h をインクルードする必要があります。

## 5.5.17.WhiteBalance

カメラのホワイトバランス調整機能の制御を行います。

この章の関数はカラーカメラで使用できます。

カメラはGコンポーネントを基準とした R と B コンポーネントのゲインの比率を手動または自動で調整することによりホワイトバランスを制御しています。

ホワイトバランス調整機能の説明は、カメラの取扱説明書をご覧ください。

### 5.5.17.1. GetCamBalanceRatioMinMax

カメラのホワイトバランスゲイン（倍率）の最小値と最大値を取得します。

#### [構文]

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

#### [パラメータ]

パラメータ		内 容
<i>hCam</i>	[in]	カメラのカメラハンドルです。
<i>eSelector</i>	[in]	ホワイトバランスゲイン設定の対象となる要素です。
<i>pdBalanceRatioMin</i>	[out]	取得した最小値を格納する変数へのポインタです。
<i>pdBalanceRatioMax</i>	[out]	取得した最大値を格納する変数へのポインタです。

#### [戻り値]

実行結果を返します。 戻り値は、[5.9. ステータスコード](#) を参照してください。

#### [備考]

ホワイトバランス調整機能（BalanceRatioSelector、BalanceRatio レジスタ）が実装されていないカメラで実行するとエラーになります。

CAM\_BALANCE\_RATIO\_SELECTOR\_TYPE 型は、TeliCamAPI.h で定義されています。

TeliCamAPI.h をインクルードする必要があります。

---

### 5.5.17.2. GetCamBalanceRatio

カメラのホワイトバランスゲイン（倍率）の設定値を取得します。

#### [構文]

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

#### [パラメータ]

パラメータ		内 容
<i>hCam</i>	[in]	カメラのカメラハンドルです。
<i>eSelector</i>	[in]	ホワイトバランスゲイン設定の対象となる要素です。
<i>pdBalanceRatio</i>	[out]	取得したホワイトバランスゲイン（倍率）の設定値を格納する変数へのポインタです。

#### [戻り値]

実行結果を返します。 戻り値は、[5.9. ステータスコード](#) を参照してください。

#### [備考]

ホワイトバランス調整機能（BalanceRatioSelector、BalanceRatio レジスタ）が実装されていないカメラで実行するとエラーになります。

CAM\_BALANCE\_RATIO\_SELECTOR\_TYPE 型は、TeliCamAPI.h で定義されています。

TeliCamAPI.h をインクルードする必要があります。

---

### 5.5.17.3. SetCamBalanceRatio

カメラのホワイトバランスゲイン（倍率）を設定します。

#### [構文]

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

#### [パラメータ]

パラメータ		内 容
<i>hCam</i>	[in]	カメラのカメラハンドルです。
<i>eSelector</i>	[in]	ホワイトバランスゲイン設定の対象となる要素です。
<i>dBalanceRatio</i>	[in]	設定するホワイトバランスゲイン（倍率）です。

#### [戻り値]

実行結果を返します。 戻り値は、[5.9. ステータスコード](#) を参照してください。

#### [備考]

ホワイトバランス調整機能（BalanceRatioSelector、BalanceRatio レジスタ）が実装されていないカメラで実行するとエラーになります。

CAM\_BALANCE\_RATIO\_SELECTOR\_TYPE 型は、TeliCamAPI.h で定義されています。

TeliCamAPI.h をインクルードする必要があります。

---

#### 5.5.17.4. GetCamBalanceWhiteAuto

カメラのホワイトバランスゲイン自動調整モードの設定値を取得します。

##### [構文]

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

##### [パラメータ]

パラメータ		内 容
<i>hCam</i>	[in]	カメラのカメラハンドルです。
<i>peBalanceWhiteAuto</i>	[out]	取得したホワイトバランスゲイン自動調整モードの設定値を格納する変数へのポインタです。

##### [戻り値]

実行結果を返します。 戻り値は、[5.9. ステータスコード](#) を参照してください。

##### [備考]

ホワイトバランス調整機能（BalanceWhiteAuto レジスタ）が実装されていないカメラで実行するとエラーになります。

CAM\_BALANCE\_WHITE\_AUTO\_TYPE 型は、TeliCamAPI.h で定義されています。

TeliCamAPI.h をインクルードする必要があります。

---

### 5.5.17.5. SetCamBalanceWhiteAuto

カメラのホワイトバランスゲイン自動調整モードを設定します。

#### [構文]

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

#### [パラメータ]

パラメータ		内 容
<i>hCam</i>	[in]	カメラのカメラハンドルです。
<i>BalanceWhiteAuto</i>	[in]	設定するホワイトバランスゲイン自動調整モードです。

#### [戻り値]

実行結果を返します。 戻り値は、[5.9. ステータスコード](#) を参照してください。

#### [備考]

ホワイトバランス調整機能（BalanceWhiteAuto レジスタ）が実装されていないカメラで実行するとエラーになります。

CAM\_BALANCE\_WHITE\_AUTO\_TYPE 型は、TeliCamAPI.h で定義されています。

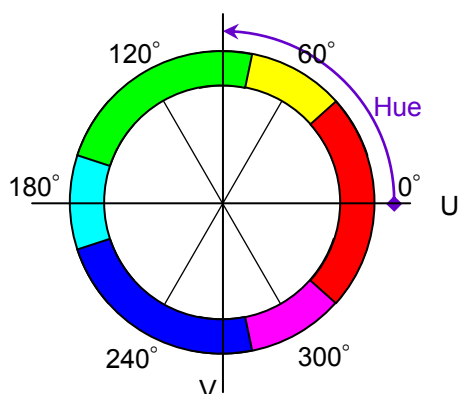
TeliCamAPI.h をインクルードする必要があります。

## 5.5.18.Hue

カメラの色相調整機能の制御を行います。

この章の関数はカラーカメラで使用できます。

色相調整機能の説明は、カメラの取扱説明書をご覧ください。



### 5.5.18.1. GetCamHueMinMax

カメラの色相の最小値と最大値を取得します。

#### [構文]

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

#### [パラメータ]

パラメータ	内 容
<i>hCam</i> [in]	カメラのカメラハンドルです。
<i>pdHueMin</i> [out]	取得した最小値を格納する変数へのポインタです。(単位:°)
<i>pdHueMax</i> [out]	取得した最大値を格納する変数へのポインタです。(単位:°)

#### [戻り値]

実行結果を返します。 戻り値は、[5.9. ステータスコード](#) を参照してください。

#### [備考]

色相調整機能（Hue レジスタ）が実装されていないカメラで実行するとエラーになります。

TeliCamAPI.h をインクルードする必要があります。

---

### 5.5.18.2. GetCamHue

カメラの色相の設定値を取得します。

#### [構文]

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

#### [パラメータ]

パラメータ		内 容
<i>hCam</i>	[in]	カメラのカメラハンドルです。
<i>pdHue</i>	[out]	取得したカメラの色相の設定値を格納する変数へのポインタです。(単位:°)

#### [戻り値]

実行結果を返します。 戻り値は、[5.9. ステータスコード](#) を参照してください。

#### [備考]

色相調整機能（Hue レジスタ）が実装されていないカメラで実行するとエラーになります。

TeliCamAPI.h をインクルードする必要があります。



---

### 5.5.18.3. SetCamHue

カメラの色相を設定します。

#### [構文]

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

#### [パラメータ]

パラメータ		内 容
<i>hCam</i>	[in]	カメラのカメラハンドルです。
<i>dHue</i>	[in]	設定するカメラの色相です。（単位：°）

#### [戻り値]

実行結果を返します。 戻り値は、[5.9. ステータスコード](#) を参照してください。

#### [備考]

色相調整機能（Hue レジスタ）が実装されていないカメラで実行するとエラーになります。

TeliCamAPI.h をインクルードする必要があります。

---

## 5.5.19.Saturation

カメラの彩度調整機能の制御を行います。  
彩度調整機能の説明は、カメラの取扱説明書をご覧ください。

### 5.5.19.1. GetCamSaturationMinMax

カメラの彩度の最小値と最大値を取得します。

#### [構文]

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

#### [パラメータ]

パラメータ		内 容
<i>hCam</i>	[in]	カメラのカメラハンドルです。
<i>pdSaturationMin</i>	[out]	取得した最小値を格納する変数へのポインタです。（単位：倍）
<i>pdSaturationMax</i>	[out]	取得した最大値を格納する変数へのポインタです。（単位：倍）

#### [戻り値]

実行結果を返します。 戻り値は、[5.9. ステータスコード](#) を参照してください。

#### [備考]

彩度調整機能（Saturation レジスタ）が実装されていないカメラで実行するとエラーになります。

TeliCamAPI.h をインクルードする必要があります。

---

### 5.5.19.2. GetCamSaturation

カメラの彩度の設定値を取得します。

#### [構文]

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

#### [パラメータ]

パラメータ	内 容
<i>hCam</i> [in]	カメラのカメラハンドルです。
<i>pdSaturation</i> [out]	取得したカメラの彩度の設定値を格納する変数へのポインタです。 (単位：倍)

#### [戻り値]

実行結果を返します。 戻り値は、[5.9. ステータスコード](#) を参照してください。

#### [備考]

彩度調整機能（Saturation レジスタ）が実装されていないカメラで実行するとエラーになります。

TeliCamAPI.h をインクルードする必要があります。

---

### 5.5.19.3. SetCamSaturation

カメラの彩度を設定します。

#### [構文]

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

#### [パラメータ]

パラメータ		内 容
<i>hCam</i>	[in]	カメラのカメラハンドルです。
<i>dSaturation</i>	[in]	設定するカメラの彩度です。（単位：倍）

#### [戻り値]

実行結果を返します。 戻り値は、[5.9. ステータスコード](#) を参照してください。

#### [備考]

彩度調整機能（Saturation レジスタ）が実装されていないカメラで実行するとエラーになります。

TeliCamAPI.h をインクルードする必要があります。

---

## 5.5.20.Sharpness

カメラの画像のエッジ強度調整機能の制御を行います。  
エッジ強度調整機能の説明は、カメラの取扱説明書をご覧ください。

### 5.5.20.1. GetCamSharpnessMinMax

カメラの画像のエッジ強度の最小値と最大値を取得します。

#### [構文]

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

#### [パラメータ]

パラメータ		内 容
<i>hCam</i>	[in]	カメラのカメラハンドルです。
<i>puiSharpnessMin</i>	[out]	取得した最小値を格納する変数へのポインタです。
<i>puiSharpnessMax</i>	[out]	取得した最大値を格納する変数へのポインタです。

#### [戻り値]

実行結果を返します。 戻り値は、[5.9. ステータスコード](#) を参照してください。

#### [備考]

エッジ強度調整機能（Sharpness レジスタ）が実装されていないカメラで実行するとエラーになります。

TeliCamAPI.h をインクルードする必要があります。

---

### 5.5.20.2. GetCamSharpness

カメラの画像のエッジ強度の設定値を取得します。

#### [構文]

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

#### [パラメータ]

パラメータ		内 容
<i>hCam</i>	[in]	カメラのカメラハンドルです。
<i>puiSharpness</i>	[out]	取得した画像のエッジ強度の設定値を格納する変数へのポインタです。

#### [戻り値]

実行結果を返します。 戻り値は、[5.9. ステータスコード](#) を参照してください。

#### [備考]

エッジ強度調整機能（Sharpness レジスタ）が実装されていないカメラで実行するとエラーになります。

TeliCamAPI.h をインクルードする必要があります。

---

### 5.5.20.3. SetCamSharpness

カメラの画像のエッジ強度を設定します。

#### [構文]

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

#### [パラメータ]

パラメータ		内 容
<i>hCam</i>	[in]	カメラのカメラハンドルです。
<i>uiSharpness</i>	[in]	設定する画像のエッジ強度です。

#### [戻り値]

実行結果を返します。 戻り値は、[5.9. ステータスコード](#) を参照してください。

#### [備考]

エッジ強度調整機能（Sharpness レジスタ）が実装されていないカメラで実行するとエラーになります。

TeliCamAPI.h をインクルードする必要があります。

### 5.5.21.ColorCorrectionMatrix

カメラの色補正マトリックス調整機能の制御を行います。

この章の関数はカラーカメラで使用できます。

色補正マトリックス調整機能の説明は、カメラの取扱説明書をご覧ください。

以下に原画像の画素値(R, G, B)と色補正後の画素値 (R'、G'、B') の関係式を示します。

$$\begin{pmatrix} R' \\ G' \\ B' \end{pmatrix} = \begin{pmatrix} 1 & -mask\_rg & -mask\_rb \\ -mask\_gr & 1 & -mask\_gb \\ -mask\_br & -mask\_bg & 1 \end{pmatrix} \begin{pmatrix} R & (G-R) & (B-R) \\ (R-G) & G & (B-G) \\ (R-B) & (G-B) & B \end{pmatrix}$$

$$R' = (1 - mask\_rg - mask\_rb) \times R + mask\_rg \times G + mask\_rb \times B$$

$$G' = mask\_gr \times R + (1 - mask\_gr - mask\_gb) \times G + mask\_gb \times B$$

$$B' = mask\_br \times R + mask\_bg \times G + (1 - mask\_br - mask\_bg) \times B$$

GenlCam GenApi では「SelectoorI」と「SelectorJ」の2個のセレクトラを使用して GenApi のノードで読み書きする色補正係数を指定するようになっています。以下に「SelectoorI」と「SelectorJ」で選択される係数を示します。

	SelectorJ = R	SelectorJ = G	SelectorJ = B
SelectoorI = R		mask_rg	mask_rb
SelectoorI = G	mask_gr		mask_gb
SelectoorI = B	mask_br	mask_bg	

本章の関数は「SelectoorI」と「SelectorJ」をまとめた CAM\_COLOR\_CORRECTION\_MATRIX\_TYPE 列挙型の値を使用して関数で扱う係数を指定します。



---

### 5.5.21.1. GetCamColorCorrectionMatrixMinMax

カメラの色補正マトリクスの係数の最小値と最大値を取得します。

#### [構文]

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

#### [パラメータ]

パラメータ		内 容
<i>hCam</i>	[in]	カメラのカメラハンドルです。
<i>eType</i>	[in]	取得する色補正マトリクスの要素です。
<i>pdMatrixMin</i>	[out]	取得した最小値を格納する変数へのポインタです。
<i>pdMatrixMax</i>	[out]	取得した最大値を格納する変数へのポインタです。

#### [戻り値]

実行結果を返します。 戻り値は、[5.9. ステータスコード](#) を参照してください。

#### [備考]

色補正マトリクス調整機能（ColorCorrectionMatrix、ColorCorrectionMatrixSelectorI、ColorCorrectionMatrixSelectorJ レジスタ）が実装されていないカメラで実行するとエラーになります。

CAM\_COLOR\_CORRECTION\_MATRIX\_TYPE 型は、TeliCamAPI.h で定義されています。

TeliCamAPI.h をインクルードする必要があります。

---

### 5.5.21.2. GetCamColorCorrectionMatrix

カメラの色補正マトリクスの係数の設定値を取得します。

#### [構文]

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

#### [パラメータ]

パラメータ	内 容
<i>hCam</i> [in]	カメラのカメラハンドルです。
<i>eType</i> [in]	取得する色補正マトリクスの要素です。
<i>pdMatrix</i> [out]	取得した色補正マトリクスの係数の設定値を格納する変数へのポインタです。

#### [戻り値]

実行結果を返します。 戻り値は、[5.9. ステータスコード](#) を参照してください。

#### [備考]

色補正マトリックス調整機能（ColorCorrectionMatrix、ColorCorrectionMatrixSelectorI、ColorCorrectionMatrixSelectorJ レジスタ）が実装されていないカメラで実行するとエラーになります。

CAM\_COLOR\_CORRECTION\_MATRIX\_TYPE 型は、TeliCamAPI.h で定義されています。

TeliCamAPI.h をインクルードする必要があります。

---

### 5.5.21.3. SetCamColorCorrectionMatrix

カメラの色補正マトリクスの係数を設定します。

#### [構文]

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

#### [パラメータ]

パラメータ		内 容
<i>hCam</i>	[in]	カメラのカメラハンドルです。
<i>eType</i>	[in]	取得する色補正マトリクスの要素です。
<i>dMatrix</i>	[in]	設定する色補正マトリクスの係数です。

#### [戻り値]

実行結果を返します。 戻り値は、[5.9. ステータスコード](#) を参照してください。

#### [備考]

色補正マトリックス調整機能（ColorCorrectionMatrix、ColorCorrectionMatrixSelectorI、ColorCorrectionMatrixSelectorJ レジスタ）が実装されていないカメラで実行するとエラーになります。

CAM\_COLOR\_CORRECTION\_MATRIX\_TYPE 型は、TeliCamAPI.h で定義されています。

TeliCamAPI.h をインクルードする必要があります。

---

## 5.5.22.LUTControl

カメラの LUT（ルックアップテーブル）調整機能の制御を行います。  
LUT 調整機能の説明は、カメラの取扱説明書をご覧ください。

### 5.5.22.1. GetCamLutEnable

カメラの LUT モード（有効 / 無効）を取得します。

#### [構文]

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

#### [パラメータ]

パラメータ		内 容
<i>hCam</i>	[in]	カメラのカメラハンドルです。
<i>pbEnable</i>	[out]	取得した LUT モード（有効 / 無効）を格納する変数へのポインタです。 true とき有効、false のとき無効です。

#### [戻り値]

実行結果を返します。 戻り値は、[5.9. ステータスコード](#) を参照してください。

#### [備考]

LUT 調整機能（LUTEnable レジスタ）が実装されていないカメラで実行するとエラーになります。

TeliCamAPI.h をインクルードする必要があります。

---

### 5.5.22.2. SetCamLutEnable

カメラの LUT モード（有効 / 無効）を設定します。

#### [構文]

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

#### [パラメータ]

パラメータ		内 容
<i>hCam</i>	[in]	カメラのカメラハンドルです。
<i>bEnable</i>	[in]	設定する LUT モード（有効 / 無効）です。 true とき有効、false のとき無効です。

#### [戻り値]

実行結果を返します。 戻り値は、[5.9. ステータスコード](#) を参照してください。

#### [備考]

LUT 調整機能（LUTEnable レジスタ）が実装されていないカメラで実行するとエラーになります。

TeliCamAPI.h をインクルードする必要があります。

---

### 5.5.22.3. GetCamLutValue

カメラの LUT の出力値を取得します。

#### [構文]

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

#### [パラメータ]

パラメータ		内 容
<i>hCam</i>	[in]	カメラのカメラハンドルです。
<i>uiLutIndex</i>	[in]	LUT の入力値です。（設定範囲：0 ～ 1023）
<i>puiLutValue</i>	[out]	取得した LUT の出力値を格納する変数へのポインタです。 （取得範囲：0 ～ 1023）

#### [戻り値]

実行結果を返します。 戻り値は、[5.9. ステータスコード](#) を参照してください。

#### [備考]

LUT 調整機能（LUTIndex レジスタと、LUTValue または LUTEntry レジスタ）が実装されていないカメラで実行するとエラーになります。

TeliCamAPI.h をインクルードする必要があります。

---

#### 5.5.22.4. SetCamLutValue

カメラの LUT の出力値を設定します。

##### [構文]

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

##### [パラメータ]

パラメータ		内 容
<i>hCam</i>	[in]	カメラのカメラハンドルです。
<i>uiLutIndex</i>	[in]	LUT の入力値です。（設定範囲：0 ～ 1023）
<i>puiLutValue</i>	[in]	設定する LUT の出力値です。（設定範囲：0 ～ 1023）

##### [戻り値]

実行結果を返します。 戻り値は、[5.9. ステータスコード](#) を参照してください。

##### [備考]

LUT 調整機能（LUTIndex レジスタと、LUTValue または LUTEntry レジスタ）が実装されていないカメラで実行するとエラーになります。

TeliCamAPI.h をインクルードする必要があります。

---

## 5.5.23. UserSetControl

カメラのユーザ設定機能の制御を行います。  
ユーザ設定機能の説明は、カメラの取扱説明書をご覧ください。

### 5.5.23.1. ExecuteCamUserSetLoad

カメラに実装されている不揮発性メモリ（ユーザメモリ）から、設定パラメータをカメラにロードします。

#### [構文]

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

#### [パラメータ]

パラメータ		内 容
<i>hCam</i>	[in]	カメラのカメラハンドルです。
<i>eSelector</i>	[in]	ロードするユーザ設定チャンネルです。

#### [戻り値]

実行結果を返します。 戻り値は、[5.9. ステータスコード](#) を参照してください。

#### [備考]

ユーザ設定機能（UserSetSelector、UserSetLoad レジスタ）が実装されていないカメラで実行するとエラーになります。

ロードされるパラメータは、カメラの取扱説明書をご覧ください。

CAM\_USER\_SET\_SELECTOR\_TYPE 型は、TeliCamAPI.h で定義されています。

TeliCamAPI.h をインクルードする必要があります。



---

### 5.5.23.2. ExecuteCamUserSetSave

現在カメラに設定されているパラメータを、カメラに実装されている不揮発性メモリ（ユーザメモリ）にセーブします。

#### [構文]

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

#### [パラメータ]

パラメータ		内 容
<i>hCam</i>	[in]	カメラのカメラハンドルです。
<i>eSelector</i>	[in]	セーブするユーザ設定チャンネルです。 CAM_USER_SET_SELECTOR_DEFAULT は指定できません。

#### [戻り値]

実行結果を返します。 戻り値は、[5.9. ステータスコード](#) を参照してください。

#### [備考]

ユーザ設定機能（UserSetSelector、UserSetSave レジスタ）が実装されていないカメラで実行するとエラーになります。

セーブされるパラメータは、カメラの取扱説明書をご覧ください。

CAM\_USER\_SET\_SELECTOR\_TYPE 型は、TeliCamAPI.h で定義されています。

TeliCamAPI.h をインクルードする必要があります。

### 5.5.23.3. ExecuteCamUserSetSaveAndSetDefault

現在カメラに設定されているパラメータを、カメラに実装されている不揮発性メモリ（ユーザメモリ）にセーブし、カメラ起動時にユーザ設定チャンネルを設定します。

#### [構文]

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

#### [パラメータ]

パラメータ	内 容
<i>hCam</i> [in]	カメラのカメラハンドルです。
<i>eSelector</i> [in]	セーブを実行し、カメラ起動時にロードするユーザ設定チャンネルです。 CAM_USER_SET_SELECTOR_DEFAULT が指定された場合はセーブは実行されず、カメラ起動時にロードするユーザ設定チャンネルのみ設定されます。

#### [戻り値]

実行結果を返します。 戻り値は、[5.9. ステータスコード](#) を参照してください。

#### [備考]

ユーザ設定機能（UserSetSelector、UserSetSave、UserSetDefault レジスタ）が実装されていないカメラで実行するとエラーになります。

*eSelector* に CAM\_USER\_SET\_SELECTOR\_DEFAULT 以外を指定したときは、指定されたユーザ設定チャンネルに現在のパラメータがセーブされます。カメラ起動時は指定されたユーザ設定チャンネルのパラメータで起動されます。

*eSelector* に CAM\_USER\_SET\_SELECTOR\_DEFAULT を指定したときは、工場出荷時設定がロードされ、現在のパラメータのセーブは行われません。カメラ起動時は工場出荷時設定のパラメータで起動されます。

CAM\_USER\_SET\_SELECTOR\_TYPE 型は、TeliCamAPI.h で定義されています。

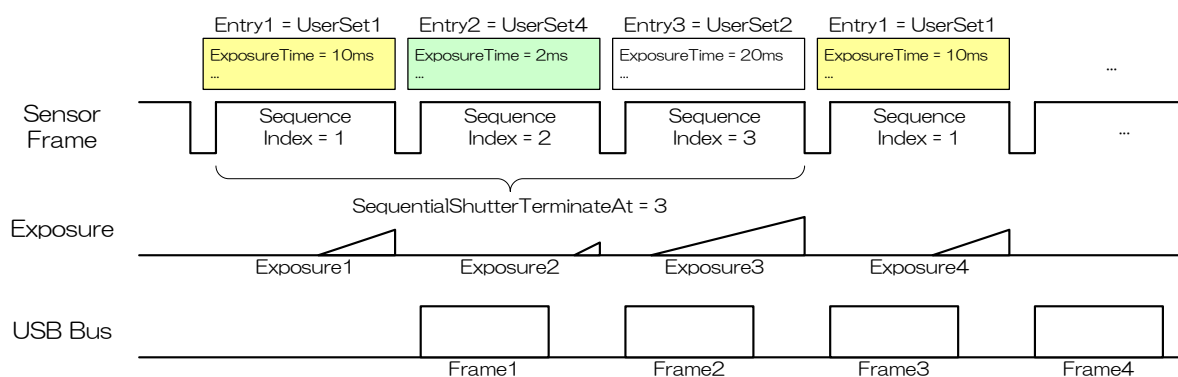
TeliCamAPI.h をインクルードする必要があります。

## 5.5.24.SequentialShutterControl

カメラのシーケンシャルシャッター機能の設定を行います。

シーケンシャルシャッター機能の説明は、カメラの取扱説明書をご覧ください。

シーケンシャルシャッターはあらかじめ指定したインデックスの UserSet メモリを各フレームごとにロードし、撮像条件の違う画像を連続して撮影する機能です。



### 5.5.24.1. GetCamSequentialShutterEnable

カメラのシーケンシャルシャッターモード（有効 / 無効）を取得します。

#### [構文]

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

#### [パラメータ]

パラメータ	内 容
<i>hCam</i> [in]	カメラのカメラハンドルです。
<i>pbEnable</i> [out]	取得したシーケンシャルシャッターモード（有効 / 無効）を格納する変数へのポインタです。 true とき有効、false のとき無効です。

#### [戻り値]

実行結果を返します。 戻り値は、[5.9. ステータスコード](#) を参照してください。

#### [備考]

シーケンシャルシャッター機能（SequentialShutterEnable レジスタ）が実装されていないカメラで実行するとエラーになります。

TeliCamAPI.h をインクルードする必要があります。

---

### 5.5.24.2. SetCamSequentialShutterEnable

カメラのシーケンシャルシャッターモード（有効 / 無効）を設定します。

#### [構文]

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

#### [パラメータ]

パラメータ		内 容
<i>hCam</i>	[in]	カメラのカメラハンドルです。
<i>bEnable</i>	[in]	設定するシーケンシャルシャッターモード（有効 / 無効）です。 true とき有効、false のとき無効です。

#### [戻り値]

実行結果を返します。 戻り値は、[5.9. ステータスコード](#) を参照してください。

#### [備考]

シーケンシャルシャッター機能（SequentialShutterEnable レジスタ）が実装されていないカメラで実行するとエラーになります。

TeliCamAPI.h をインクルードする必要があります。

---

### 5.5.24.3. GetCamSequentialShutterTerminateAtMinMax

カメラのシーケンシャルシャッター機能の繰り返しを行う回数の最小値と最大値を取得します。

#### [構文]

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

#### [パラメータ]

パラメータ		内 容
<i>hCam</i>	[in]	カメラのカメラハンドルです。
<i>puiMin</i>	[out]	取得した最小値を格納する変数へのポインタです。
<i>puiMax</i>	[out]	取得した最大値を格納する変数へのポインタです。

#### [戻り値]

実行結果を返します。 戻り値は、[5.9. ステータスコード](#) を参照してください。

#### [備考]

シーケンシャルシャッター機能 (SequentialShutterTerminateAt レジスタ) が実装されていないカメラで実行するとエラーになります。

TeliCamAPI.h をインクルードする必要があります。

---

#### 5.5.24.4. GetCamSequentialShutterTerminateAt

カメラのシーケンシャルシャッター機能の繰り返しを行う回数を取得します。

##### [構文]

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

##### [パラメータ]

パラメータ		内 容
<i>hCam</i>	[in]	カメラのカメラハンドルです。
<i>puiValue</i>	[out]	取得したシーケンシャルシャッター機能の繰り返しを行う回数を格納する変数へのポインタです。

##### [戻り値]

実行結果を返します。 戻り値は、[5.9. ステータスコード](#) を参照してください。

##### [備考]

シーケンシャルシャッター機能（SequentialShutterTerminateAt レジスタ）が実装されていないカメラで実行するとエラーになります。

TeliCamAPI.h をインクルードする必要があります。

---

#### 5.5.24.5. SetCamSequentialShutterTerminateAt

カメラのシーケンシャルシャッター機能の繰り返しを行う回数を設定します。

##### [構文]

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

##### [パラメータ]

パラメータ		内 容
<i>hCam</i>	[in]	カメラのカメラハンドルです。
<i>uiValue</i>	[in]	設定するシーケンシャルシャッター機能の繰り返しを行う回数です。

##### [戻り値]

実行結果を返します。 戻り値は、[5.9. ステータスコード](#) を参照してください。

##### [備考]

シーケンシャルシャッター機能（SequentialShutterTerminateAt レジスタ）が実装されていないカメラで実行するとエラーになります。

TeliCamAPI.h をインクルードする必要があります。

---

#### 5.5.24.6. GetCamSequentialShutterIndexMinMax

カメラのシーケンシャルシャッター機能の登録を行うシーケンス番号の最小値と最大値を取得します。

##### [構文]

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

##### [パラメータ]

パラメータ		内 容
<i>hCam</i>	[in]	カメラのカメラハンドルです。
<i>puiMin</i>	[out]	取得した最小値を格納する変数へのポインタです。
<i>puiMax</i>	[out]	取得した最大値を格納する変数へのポインタです。

##### [戻り値]

実行結果を返します。 戻り値は、[5.9. ステータスコード](#) を参照してください。

##### [備考]

シーケンシャルシャッター機能（SequentialShutterIndex レジスタ）が実装されていないカメラで実行するとエラーになります。

TeliCamAPI.h をインクルードする必要があります。



---

### 5.5.24.7. GetCamSequentialShutterEntryMinMax

カメラのシーケンシャルシャッター機能のシーケンスに登録するユーザ設定チャンネル（UserSet 番号）の最小値と最大値を取得します。

#### [構文]

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

#### [パラメータ]

パラメータ		内 容
<i>hCam</i>	[in]	カメラのカメラハンドルです。
<i>puiMin</i>	[out]	取得した最小値を格納する変数へのポインタです。
<i>puiMax</i>	[out]	取得した最大値を格納する変数へのポインタです。

#### [戻り値]

実行結果を返します。 戻り値は、[5.9. ステータスコード](#) を参照してください。

#### [備考]

シーケンシャルシャッター機能（SequentialShutterEntry レジスタ）が実装されていないカメラで実行するとエラーになります。

TeliCamAPI.h をインクルードする必要があります。

---

#### 5.5.24.8. GetCamSequentialShutterEntry

カメラのシーケンシャルシャッター機能のシーケンスに登録されているユーザ設定チャンネル（UserSet 番号）を取得します。

##### [構文]

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

##### [パラメータ]

パラメータ		内 容
<i>hCam</i>	[in]	カメラのカメラハンドルです。
<i>uiIndex</i>	[in]	取得するシーケンス番号です。
<i>puiEntry</i>	[out]	取得したユーザ設定チャンネル（UserSet 番号）を格納する変数へのポインタです。

##### [戻り値]

実行結果を返します。 戻り値は、[5.9. ステータスコード](#) を参照してください。

##### [備考]

シーケンシャルシャッター機能（SequentialShutterIndex、SequentialShutterEntry レジスタ）が実装されていないカメラで実行するとエラーになります。

TeliCamAPI.h をインクルードする必要があります。

---

#### 5.5.24.9. SetCamSequentialShutterEntry

カメラのシーケンシャルシャッター機能のシーケンスにユーザ設定チャンネル（UserSet 番号）を登録します。

##### [構文]

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

##### [パラメータ]

パラメータ		内 容
<i>hCam</i>	[in]	カメラのカメラハンドルです。
<i>uiIndex</i>	[in]	登録するシーケンス番号です。
<i>uiEntry</i>	[in]	登録するユーザ設定チャンネル（UserSet 番号）です。

##### [戻り値]

実行結果を返します。 戻り値は、[5.9. ステータスコード](#) を参照してください。

##### [備考]

シーケンシャルシャッター機能（SequentialShutterIndex、SequentialShutterEntry レジスタ）が実装されていないカメラで実行するとエラーになります。

TeliCamAPI.h をインクルードする必要があります。

### 5.5.25. UserDefinedName (DeviceUserID)

カメラのユーザ定義情報（UserDefinedName または、DeviceUserID レジスタ）の制御を行います。

UserDefinedName または DeviceUserID レジスタは、カメラ内部の不揮発性メモリに任意の文字列を保存することができるレジスタ（メモリ）です。複数台カメラを使用する場合、カメラを特定する手段として利用することができます。

ユーザ定義情報（UserDefinedName または、DeviceUserID レジスタ）の説明は、カメラの取扱説明書をご覧ください。

#### 5.5.25.1. GetCamUserDefinedName

カメラのユーザ定義情報を取得します。

##### [構文]

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

##### [パラメータ]

パラメータ	内 容
<i>hCam</i> [in]	カメラのカメラハンドルです。
<i>pszName</i> [out]	ユーザ定義情報の説明を格納するバッファへのポインタです。 NULL を指定すると、 <i>puiSize</i> に必要なバッファサイズが格納されます。
<i>puiSize</i> [in,out]	ユーザ定義情報を格納するバッファのサイズを格納している変数へのポインタです。（単位：byte） 取得に成功した場合は、 <a href="#">pszName</a> に格納したサイズが格納されます。 <a href="#">pszName</a> にNULLが指定された場合は、必要なバッファサイズが格納されます。

##### [戻り値]

実行結果を返します。 戻り値は、[5.9. ステータスコード](#) を参照してください。

##### [備考]

ユーザ定義情報の機能（UserDefinedName または、DeviceUserID レジスタ）が実装されていないカメラで実行するとエラーになります。

カメラに記憶されているユーザ定義情報が NULL で終端されていない場合は、最後のデータが NULL に置き換わる場合があります。

カメラのユーザ定義情報を変更した場合は、[Sys\\_GetNumOfCameras\(\)](#) を実行しなければ情報が更新されない場合があります。

TeliCamAPI.h をインクルードする必要があります。

---

### 5.5.25.2. SetCamUserDefinedName

カメラのユーザ定義情報を設定します。

#### [構文]

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

#### [パラメータ]

パラメータ		内 容
<i>hCam</i>	[in]	カメラのカメラハンドルです。
<i>pszName</i>	[in]	NULL で終端されたユーザ定義情報（文字列）です。

#### [戻り値]

実行結果を返します。 戻り値は、[5.9. ステータスコード](#) を参照してください。

#### [備考]

ユーザ定義情報の機能（UserDefinedName または、DeviceUserID レジスタ）が実装されていないカメラで実行するとエラーになります。

設定できる文字数は、カメラにより異なります。

GigE Vision カメラは最大 16 文字(16byte)、USB3 Vision カメラは最大 64 文字(64byte)です。（終端の NULL 文字は、1 文字とカウントされます。）

TeliCamAPI.h をインクルードする必要があります。

## 5.6. GenICam 関数

この章の関数は GenICam GenAPI ライブラリの各クラスのメソッドをラップした関数です。ユーザアプリケーションはカメラのレジスタアドレスの代わりに機能名を指定してカメラのレジスタに簡単にアクセスすることができます。

GenICam に関する詳細情報は、<http://www.genicam.org> を参照してください。

### 5.6.1. Node 系関数

#### 5.6.1.1. Nd\_GetNode

ノード名を指定して、指定したカメラ機能を扱うノードのノードハンドルを取得します。

##### [構文]

```
CAM_API_STATUS Nd_GetNode (  
    CAM_HANDLE      hCam,  
    const char      *pszName,  
    CAM_NODE_HANDLE *phNode  
);
```

##### [パラメータ]

パラメータ	内 容
<i>hCam</i> [in]	カメラのカメラハンドルです。
<i>pszName</i> [in]	NULL で終端されたカメラ機能の機能名（ノード名）です。
<i>phNode</i> [out]	取得したノードハンドルを格納する変数へのポインタです。

##### [戻り値]

実行結果を返します。 戻り値は、[5.9. ステータスコード](#) を参照してください。

##### [備考]

XmFeatures.h に BU、BG シリーズカメラで使用可能な標準機能名と弊社固有機能名の宣言が記述されていますのでご活用ください。Visual Studio の IntelliSense が使用可能な場合、名前空間名「Teli::XmlFeatures」をキー入力すれば、定義されている機能名が選択肢として表示されます。

本章の関数群は本関数で取得したノードハンドルを使用してノードにアクセスします。カメラにより、使用できる機能名（ノード名）が異なります。使用するカメラの取扱説明書をご覧ください。

TeliCamAPI.h をインクルードする必要があります。

## [コード例]

```
C++

CAM_API_STATUS      uiStatus;
uint32_t            uiNum, uiSize;
CAM_HANDLE          hCam = (CAM_HANDLE)NULL;
CAM_NODE_HANDLE     hNode = (CAM_NODE_HANDLE)NULL;
TC_NODE_TYPE        eNodeType = TC_NODE_TYPE_UNKNOWN;
TC_NODE_ACCESS_MODE eNodeAccessMode = TC_NODE_ACCESS_MODE_UNKNOWN;
TC_NODE_VISIBILITY  eNodeVisibility = TC_NODE_VISIBILITY_UNKNOWN;
TC_NODE_CACHING_MODE eNodeCachingMode = TC_NODE_CACHING_MODE_UNKNOWN;
TC_NODE_REPRESENTATION nodeRepresentation = TC_NODE_REPRESENTATION_UNKNOWN;
char                *pszBuf = NULL;

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

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

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

do
{
    // Get node handle.
    uiStatus = Nd_GetNode(hCam, "Gain", &hNode);
    if (uiStatus != CAM_API_STS_SUCCESS)
        return -1;
    printf("hNode = 0x%llu\n", (long long unsigned int)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 *)malloc(uiSize);
```

```

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.
free(pszBuf);
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 *)malloc(uiSize);
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.
free(pszBuf);
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 *)malloc(uiSize);
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.
free(pszBuf);
pszBuf = NULL;
}
while(false);

if (pszBuf != NULL) {
    free( pszBuf);
}

// Close camera.

```



---

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

---

### 5.6.1.2. Nd\_GetType

ノードのノード型を取得します。

#### [構文]

```
CAM_API_STATUS Nd_GetType (  
    CAM_HANDLE      hCam,  
    CAM_NODE_HANDLE hNode,  
    TC\_NODE\_TYPE     *peNodeType  
);
```

#### [パラメータ]

パラメータ	内 容
<i>hCam</i> [in]	カメラのカメラハンドルです。
<i>hNode</i> [in]	タイプを取得するノードのノードハンドルです。
<i>peNodeType</i> [out]	取得したノード型を格納する変数へのポインタです。

#### [TC\_NODE\_TYPE 型]

```
typedef enum  
{  
    TC_NODE_TYPE_VALUE = 0,  
    TC_NODE_TYPE_BASE,  
    TC_NODE_TYPE_INTEGER,  
    TC_NODE_TYPE_BOOLEAN,  
    TC_NODE_TYPE_COMMAND,  
    TC_NODE_TYPE_FLOAT,  
    TC_NODE_TYPE_STRING,  
    TC_NODE_TYPE_REGISTER,  
    TC_NODE_TYPE_CATEGORY,  
    TC_NODE_TYPE_ENUMERATION,  
    TC_NODE_TYPE_ENUM_ENTRY,  
    TC_NODE_TYPE_PORT,  
    TC_NODE_TYPE_UNKNOWN  
} TC_NODE_TYPE;
```

#### [戻り値]

実行結果を返します。 戻り値は、[5.9. ステータスコード](#) を参照してください。

#### [備考]

TeliCamAPI.h をインクルードする必要があります。

#### [コード例]

[Nd\\_GetNode\(\)](#) の[コード例](#)を参照してください。

---

### 5.6.1.3. Nd\_GetName

ノード名を取得します。

#### [構文]

```
CAM_API_STATUS Nd_GetName (  
    CAM_HANDLE      hCam,  
    CAM_NODE_HANDLE hNode,  
    CAM_NODE_NAME*  pszNodeName  
);
```

#### [パラメータ]

パラメータ	内 容
<i>hCam</i> [in]	カメラのカメラハンドルです。
<i>hNode</i> [in]	ノード名を取得するノードのノードハンドルです。
<i>pszNode</i> [out]	取得したノード名を格納する構造体変数へのポインタです。

#### [CAM\_NODE\_NAME 構造体]

```
typedef struct _UNI_NODE_NAME  
{  
    char    szNodeName[NODE_NAME_LENGTH_MAX];  
} UNI_NODE_NAME, *PUNI_NODE_NAME;
```

#### [戻り値]

実行結果を返します。 戻り値は、[5.9.ステータスコード](#) を参照してください。

#### [備考]

TeliCamAPI.h をインクルードする必要があります。

#### [コード例]

[Nd\\_GetNumOfFeatures\(\)](#) の[コード例](#)を参照してください。

#### 5.6.1.4. Nd\_GetAccessMode

ノードのアクセスモードを取得します。

##### [構文]

```
CAM_API_STATUS Nd_GetAccessMode (
    CAM_HANDLE          Cam,
    CAM_NODE_HANDLE     hNode,
    TC\_NODE\_ACCESS\_MODE *peAccessMode
);
```

##### [パラメータ]

パラメータ	内 容
<i>hCam</i> [in]	カメラのカメラハンドルです。
<i>hNode</i> [in]	アクセスモードを取得するノードのノードハンドルです。
<i>peAccessMode</i> [out]	取得したアクセスモードを格納する変数へのポインタです。

##### [ TC\_NODE\_ACCESS\_MODE 型 ]

```
typedef enum
{
    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
} TC_NODE_ACCESS_MODE;
```

##### [戻り値]

実行結果を返します。 戻り値は、[5.9. ステータスコード](#) を参照してください。

##### [備考]

TeliCamAPI.h をインクルードする必要があります。

##### [コード例]

[Nd\\_GetNode\(\)](#) の[コード例](#)を参照してください。

### 5.6.1.5. Nd\_GetVisibility

ノードの推奨可視レベルを取得します。

#### [構文]

```
CAM_API_STATUS Nd_GetVisibility (  
    CAM_HANDLE          Cam,  
    CAM_NODE_HANDLE     hNode,  
    TC\_NODE\_VISIBILITY   *peVisibility  
);
```

#### [パラメータ]

パラメータ	内 容
<i>hCam</i> [in]	カメラのカメラハンドルです。
<i>hNode</i> [in]	可推奨可視レベルを取得するノードのノードハンドルです。
<i>peVisibility</i> [out]	取得した推奨可視レベルを格納する変数へのポインタです。

#### [ TC\_NODE\_VISIBILITY 型 ]

```
typedef enum  
{  
    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  
} TC_NODE_VISIBILITY;
```

#### [戻り値]

実行結果を返します。 戻り値は、[5.9. ステータスコード](#) を参照してください。

#### [備考]

TeliCamAPI.h をインクルードする必要があります。

#### [コード例]

[Nd\\_GetNode\(\)](#) の[コード例](#)を参照してください。

---

### 5.6.1.6. Nd\_GetCachingMode

ノードのキャッシュ設定を取得します。

#### [構文]

```
CAM_API_STATUS Nd_GetCachingMode (  
    CAM_HANDLE          Cam,  
    CAM_NODE_HANDLE     hNode,  
    TC\_NODE\_CACHING\_MODE *peCachingMode  
);
```

#### [パラメータ]

パラメータ	内 容
<i>hCam</i> [in]	カメラのカメラハンドルです。
<i>hNode</i> [in]	キャッシュ設定を取得するノードのノードハンドルです。
<i>peCachingMode</i> [out]	取得したキャッシュ設定を格納する変数へのポインタです。

#### [ TC\_NODE\_CACHING\_MODE 型 ]

```
typedef enum  
{  
    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  
} TC_NODE_CACHING_MODE;
```

#### [戻り値]

実行結果を返します。 戻り値は、[5.9. ステータスコード](#) を参照してください。

#### [備考]

TeliCamAPI.h をインクルードする必要があります。

#### [コード例]

[Nd\\_GetNode\(\)](#) の[コード例](#)を参照してください。

---

### 5.6.1.7. Nd\_GetDescription

ノードの説明文字列を取得します。

#### [構文]

```
CAM_API_STATUS Nd_GetDescription (  
    CAM_HANDLE          Cam,  
    CAM_NODE_HANDLE     hNode,  
    char                 *pszBuf,  
    uint32_t             *puiSize  
);
```

#### [パラメータ]

パラメータ		内 容
<i>hCam</i>	[in]	カメラのカメラハンドルです。
<i>hNode</i>	[in]	説明文字列を取得するノードのノードハンドルです。
<i>pszBuf</i>	[out]	ノードの説明文字列を格納するバッファへのポインタです。 NULLを指定すると、 <a href="#">puiSize</a> に必要なバッファサイズが格納されます。
<i>puiSize</i>	[in,out]	ノードの説明を格納するバッファのサイズを格納している変数へのポインタです。（単位：byte） <a href="#">pszBuf</a> にNULL が指定された場合は、必要なバッファサイズが格納されます。

#### [戻り値]

実行結果を返します。 戻り値は、[5.9. ステータスコード](#) を参照してください。

#### [備考]

TeliCamAPI.h をインクルードする必要があります。

#### [コード例]

[Nd\\_GetNode\(\)](#) の[コード例](#)を参照してください。

---

### 5.6.1.8. Nd\_GetToolTip

ノードのツールチップを取得します。

#### [構文]

```
CAM_API_STATUS Nd_GetToolTip (  
    CAM_HANDLE          Cam,  
    CAM_NODE_HANDLE     hNode,  
    char                 *pszBuf,  
    uint32_t             *puiSize  
);
```

#### [パラメータ]

パラメータ		内 容
<i>hCam</i>	[in]	カメラのカメラハンドルです。
<i>hNode</i>	[in]	ツールチップを取得するノードのノードハンドルです。
<i>pszBuf</i>	[out]	ノードのツールチップを格納するバッファへのポインタです。 NULL を指定すると、 <a href="#">puiSize</a> に必要なバッファサイズが格納されます。
<i>puiSize</i>	[in,out]	ノードのツールチップを格納するバッファのサイズを格納している変数へのポインタです。（単位：byte） <a href="#">pszBuf</a> にNULL が指定された場合は、必要なバッファサイズが格納されます。

#### [戻り値]

実行結果を返します。 戻り値は、[5.9. ステータスコード](#) を参照してください。

#### [備考]

TeliCamAPI.h をインクルードする必要があります。

#### [コード例]

[Nd\\_GetNode\(\)](#) の[コード例](#)を参照してください。



---

### 5.6.1.9. Nd\_GetRepresentation

ノードの推奨表現を取得します。

#### [構文]

```
CAM_API_STATUS Nd_GetRepresentation (  
    CAM_HANDLE          Cam,  
    CAM_NODE_HANDLE     hNode,  
    TC\_NODE\_REPRESENTATION *peRepresentation  
);
```

#### [パラメータ]

パラメータ		内 容
hCam	[in]	カメラのカメラハンドルです。
hNode	[in]	推奨表現を取得するノードのノードハンドルです。
peRepresentation	[out]	取得した推奨表現を格納する変数へのポインタです。

#### [ TC\_NODE\_REPRESENTATION 型 ]

```
typedef enum  
{  
    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  
} TC_NODE_REPRESENTATION;
```

#### [戻り値]

実行結果を返します。 戻り値は、[5.9. ステータスコード](#) を参照してください。

#### [備考]

TeliCamAPI.h をインクルードする必要があります。

#### [コード例]

[Nd\\_GetNode\(\)](#) の[コード例](#)を参照してください。

---

### 5.6.1.10. Nd\_GetUnit

ノードの単位名を取得します。

#### [構文]

```
CAM_API_STATUS Nd_GetUnit (  
    CAM_HANDLE          Cam,  
    CAM_NODE_HANDLE     hNode,  
    char                *pszBuf,  
    uint32_t            *puiSize  
);
```

#### [パラメータ]

パラメータ	内 容
<i>hCam</i> [in]	カメラのカメラハンドルです。
<i>hNode</i> [in]	単位名を取得するノードのノードハンドルです。
<i>pszBuf</i> [out]	ノードの単位名を格納するバッファへのポインタです。 NULL を指定すると、 <a href="#">puiSize</a> に必要なバッファサイズが格納されます。
<i>puiSize</i> [in,out]	ノードの単位名を格納するバッファのサイズを格納している変数へのポインタです。（単位：byte） <a href="#">pszBuf</a> にNULL が指定された場合は、必要なバッファサイズが格納されます。

#### [戻り値]

実行結果を返します。 戻り値は、[5.9. ステータスコード](#) を参照してください。

#### [備考]

TeliCamAPI.h をインクルードする必要があります。

#### [コード例]

[Nd\\_GetNode\(\)](#) の[コード例](#)を参照してください。

## 5.6.2. Node 系 ICategory 型 関数

### 5.6.2.1. Nd\_GetNumOfFeatures

ICategory 型ノードが持つ子 Feature ノードの数を取得します。

#### [構文]

```
CAM_API_STATUS Nd_GetNumOfFeatures (  
    CAM_HANDLE      hCam,  
    CAM_NODE_HANDLE hNode,  
    uint32_t         *puiNum  
);
```

#### [パラメータ]

パラメータ	内 容
<i>hCam</i>	[in] カメラのカメラハンドルです。
<i>hNode</i>	[in] フィーチャ数を取得するノードのノードハンドルです。
<i>puiNum</i>	[out] 取得したFeatureノード数を格納する変数へのポインタです。

#### [戻り値]

実行結果を返します。 戻り値は、[5.9. ステータスコード](#) を参照してください。

#### [備考]

本関数は、ICategory 型ノード専用です。それ以外のノードで実行するとエラーになります。

TeliCamAPI.h をインクルードする必要があります。

#### [コード例]

C++
<pre>CAM_API_STATUS      uiStatus; uint32_t             uiNum, uiFeatureNum, i; CAM_HANDLE           hCam = (CAM_HANDLE)NULL; CAM_NODE_HANDLE      hNode = (CAM_NODE_HANDLE)NULL; CAM_NODE_HANDLE      hCatNode = (CAM_NODE_HANDLE)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(&amp;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, &amp;hCam, NULL); if (uiStatus != CAM_API_STS_SUCCESS)     return -1;</pre>

```

do
{
#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 = %llx\n", (long long unsigned int)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);
    }
}
while(false);

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

// Terminate system.
Sys_Terminate();

```

---

### 5.6.2.2. Nd\_GetFeatureByIndex

ICategory 型ノードに含まれるフィーチャノードの中の、指定インデックスのノードのハンドルを取得します。

#### [構文]

```
CAM_API_STATUS Nd_GetFeatureByIndex (  
    CAM_HANDLE      hCam,  
    CAM_NODE_HANDLE hNode,  
    uint32_t         uiNodeIndex,  
    CAM_NODE_HANDLE *phNode  
);
```

#### [パラメータ]

パラメータ		内 容
<i>hCam</i>	[in]	カメラのカメラハンドルです。
<i>hNode</i>	[in]	ICategory 型ノードのノードハンドルです。
<i>uiNodeIndex</i>	[in]	フィーチャノードのインデックスです。（0 以上の整数値）
<i>phNode</i>	[out]	取得したノードハンドルを格納する変数へのポインタです。

#### [戻り値]

実行結果を返します。 戻り値は、[5.9. ステータスコード](#) を参照してください。

#### [備考]

本関数は、ICategory 型ノード専用です。それ以外のノードで実行するとエラーになります。

TeliCamAPI.h をインクルードする必要があります。

#### [コード例]

[Nd\\_GetNumOfFeatures\(\)](#) の[コード例](#)を参照してください。

## 5.6.3. Node 系 Integer 型 関数

### 5.6.3.1. Nd\_GetIntMin

Integer 型ノードの現在有効な最小値を取得します。

#### [構文]

```
CAM_API_STATUS Nd_GetIntMin (  
    CAM_HANDLE      hCam,  
    CAM_NODE_HANDLE hNode,  
    int64_t         *pLLMin  
);
```

#### [パラメータ]

パラメータ	内 容
<i>hCam</i>	[in] カメラのカメラハンドルです。
<i>hNode</i>	[in] 最小値を取得する Integer 型ノードのノードハンドルです。
<i>pLLMin</i>	[out] 取得した最小値を格納する変数へのポインタです。

#### [戻り値]

実行結果を返します。 戻り値は、[5.9. ステータスコード](#) を参照してください。

#### [備考]

本関数は、Integer 型ノード専用です。それ以外のノードで実行するとエラーになります。

TeliCamAPI.h をインクルードする必要があります。

#### [コード例]

C++
<pre>CAM_API_STATUS    uiStatus; uint32_t          uiNum; CAM_HANDLE        hCam = (CAM_HANDLE)NULL; CAM_NODE_HANDLE   hNode = (CAM_NODE_HANDLE)NULL; int64_t           llMin, llMax, llInc, llRdValue, llWrValue;  // Initialize system. uiStatus = Sys_Initialize(); if (uiStatus != CAM_API_STS_SUCCESS)     return -1;  // Get number of cameras. uiStatus = Sys_GetNumOfCameras(&amp;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, &amp;hCam, (SIGNAL_HANDLE)NULL); if (uiStatus != CAM_API_STS_SUCCESS)     return -1;  do {     // Get node handle.     uiStatus = Nd_GetNode(hCam, "Width", &amp;hNode);</pre>

```

if (uiStatus != CAM_API_STS_SUCCESS)
    return -1;
printf("hNode = %I64x\n", (long long unsigned int)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);
}
while(false);

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

// Terminate system.
Sys_Terminate();

```

---

### 5.6.3.2. Nd\_GetIntMax

Integer 型ノードの現在有効な最大値を取得します。

#### [構文]

```
CAM_API_STATUS Nd_GetIntMax (  
    CAM_HANDLE      hCam,  
    CAM_NODE_HANDLE hNode,  
    int64_t          *pLLMax  
);
```

#### [パラメータ]

パラメータ	内 容
<i>hCam</i> [in]	カメラのカメラハンドルです。
<i>hNode</i> [in]	最大値を取得する Integer 型ノードのノードハンドルです。
<i>pLLMax</i> [out]	取得した最大値を格納する変数へのポインタです。

#### [戻り値]

実行結果を返します。 戻り値は、[5.9. ステータスコード](#) を参照してください。

#### [備考]

本関数は、Integer 型ノード専用です。それ以外のノードで実行するとエラーになります。

TeliCamAPI.h をインクルードする必要があります。

#### [コード例]

[Nd\\_GetIntMin\(\)](#) の[コード例](#)を参照してください。



---

### 5.6.3.3. Nd\_GetIntInc

Integer 型ノードの増加量を取得します。

#### [構文]

```
CAM_API_STATUS Nd_GetIntInc (  
    CAM_HANDLE      hCam,  
    CAM_NODE_HANDLE hNode,  
    int64_t          *pLLInc  
);
```

#### [パラメータ]

パラメータ	内 容
<i>hCam</i> [in]	カメラのカメラハンドルです。
<i>hNode</i> [in]	増加量を取得する Integer 型ノードのノードハンドルです。
<i>pLLInc</i> [out]	取得した増加量を格納する変数へのポインタです。

#### [戻り値]

実行結果を返します。 戻り値は、[5.9. ステータスコード](#) を参照してください。

#### [備考]

本関数は、Integer 型ノード専用です。それ以外のノードで実行するとエラーになります。

TeliCamAPI.h をインクルードする必要があります。

#### [コード例]

[Nd\\_GetIntMin\(\)](#) の[コード例](#)を参照してください。

#### 5.6.3.4. Nd\_GetIntValue

Integer 型ノードの値を取得します。

##### [構文]

```
CAM_API_STATUS Nd_GetIntValue (  
    CAM_HANDLE      hCam,  
    CAM_NODE_HANDLE hNode,  
    int64_t          *pLLValue,  
    bool8_t          bVerify = false,  
    bool8_t          bIgnoreCache = false,  
);
```

##### [パラメータ]

パラメータ		内 容
<i>hCam</i>	[in]	カメラのカメラハンドルです。
<i>hNode</i>	[in]	値を取得する Integer 型ノードのノードハンドルです。
<i>pLLValue</i>	[out]	取得した値を格納する変数へのポインタです。
<i>bVerify</i>	[in]	trueを設定した場合、アクセスモードおよび、設定範囲をチェックを実行します。 falseを設定した場合、チェックを実行しません。 通常はfalseを指定してください。この引数は省略可能です。 省略した場合は、falseになります。
<i>bIgnoreCache</i>	[in]	trueを設定した場合、キャッシュ値を無視し、カメラのレジスタ値を取得します。 falseを設定した場合、キャッシュ値を取得します。 通常はfalseを指定してください。この引数は省略可能です。 省略した場合は、falseになります。

##### [戻り値]

実行結果を返します。 戻り値は、[5.9. ステータスコード](#) を参照してください。

##### [備考]

本関数は、Integer 型ノード専用です。それ以外のノードで実行するとエラーになります。

TeliCamAPI.h をインクルードする必要があります。

##### [コード例]

[Nd\\_GetIntMin\(\)](#) の[コード例](#)を参照してください。

### 5.6.3.5. Nd\_SetIntValue

Integer 型ノードの値を設定します。

#### [構文]

```
CAM_API_STATUS Nd_SetIntValue (  
    CAM_HANDLE      hCam,  
    CAM_NODE_HANDLE hNode,  
    int64_t          llValue,  
    bool8_t          bVerify = true  
);
```

#### [パラメータ]

パラメータ	内 容
<i>hCam</i> [in]	カメラのカメラハンドルです。
<i>hNode</i> [in]	値を設定する Integer 型ノードのノードハンドルです。
<i>llValue</i> [in]	設定する値です。
<i>bVerify</i> [in]	trueを設定した場合、アクセスモードおよび、設定範囲をチェックを実行します。 falseを設定した場合、チェックを実行しません。 通常はtrueを設定してください。この引数は省略可能です。 省略した場合は、trueになります。

#### [戻り値]

実行結果を返します。 戻り値は、[5.9. ステータスコード](#) を参照してください。

#### [備考]

本関数は、Integer 型ノード専用です。それ以外のノードで実行するとエラーになります。

TeliCamAPI.h をインクルードする必要があります。

#### [コード例]

[Nd\\_GetIntMin\(\)](#) の[コード例](#)を参照してください。

## 5.6.4. Node 系 IFloat 型 関数

### 5.6.4.1. Nd\_GetFloatMin

IFloat 型ノードの現在有効な最小値を取得します。

#### [構文]

```
CAM_API_STATUS Nd_GetFloatMin (  
    CAM_HANDLE      hCam,  
    CAM_NODE_HANDLE hNode,  
    float64_t       *pdMin  
);
```

#### [パラメータ]

パラメータ	内 容
<i>hCam</i>	[in] カメラのカメラハンドルです。
<i>hNode</i>	[in] 最小値を取得する IFloat 型ノードのノードハンドルです。
<i>pdMin</i>	[out] 取得した最小値を格納する変数へのポインタです。

#### [戻り値]

実行結果を返します。 戻り値は、[5.9. ステータスコード](#) を参照してください。

#### [備考]

本関数は、IFloat 型ノード専用です。それ以外のノードで実行するとエラーになります。

TeliCamAPI.h をインクルードする必要があります。

#### [コード例]

C++
<pre>CAM_API_STATUS      uiStatus; uint32_t            uiNum; CAM_HANDLE          hCam = (CAM_HANDLE)NULL; CAM_NODE_HANDLE     hNode = (CAM_NODE_HANDLE)NULL; float64_t           dMin, dMax, dInc, dRdValue, dWrValue; bool8_t             bInc; TC_NODE_DISPLAY_NOTATION eDisplayNotation; int64_t             llPrecision;  // Initialize system. uiStatus = Sys_Initialize(); if (uiStatus != CAM_API_STS_SUCCESS)     return -1;  // Get number of cameras. uiStatus = Sys_GetNumOfCameras(&amp;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, &amp;hCam, (SIGNAL_HANDLE)NULL); if (uiStatus != CAM_API_STS_SUCCESS)     return -1;  do</pre>

```

{
    // Get node handle.
    uiStatus = Nd_GetNode(hCam, "Gain", &hNode);
    if (uiStatus != CAM_API_STS_SUCCESS)
        return -1;
    printf("hNode = %llx\n", (long long unsigned int)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);
}
while(false);

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

```

---

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

---

### 5.6.4.2. Nd\_GetFloatMax

IFloat 型ノードの現在有効な最大値を取得します。

#### [構文]

```
CAM_API_STATUS Nd_GetFloatMax (  
    CAM_HANDLE      hCam,  
    CAM_NODE_HANDLE hNode,  
    float64_t        *pdMax  
);
```

#### [パラメータ]

パラメータ	内 容
<i>hCam</i> [in]	カメラのカメラハンドルです。
<i>hNode</i> [in]	最大値を取得する IFloat 型ノードのノードハンドルです。
<i>dMax</i> [out]	取得した最大値を格納する変数へのポインタです。

#### [戻り値]

実行結果を返します。 戻り値は、[5.9. ステータスコード](#) を参照してください。

#### [備考]

本関数は、IFloat 型ノード専用です。それ以外のノードで実行するとエラーになります。

TeliCamAPI.h をインクルードする必要があります。

#### [コード例]

[Nd\\_GetFloatMin\(\)](#) の[コード例](#)を参照してください。

---

### 5.6.4.3. Nd\_GetFloatHasInc

IFloat 型ノードの値が固定の増加量を持つ離散値か連続値かを取得します。

#### [構文]

```
CAM_API_STATUS Nd_GetFloatHasInc (  
    CAM_HANDLE      hCam,  
    CAM_NODE_HANDLE hNode,  
    bool8_t          *pbInc  
);
```

#### [パラメータ]

パラメータ		内 容
<i>hCam</i>	[in]	カメラのカメラハンドルです。
<i>hNode</i>	[in]	離散値／連続値を確認する IFloat 型ノードのノードハンドルです。
<i>pbInc</i>	[out]	離散値を表すフラグを格納する変数へのポインタです。 true のとき離散値、false のとき連続値です。

#### [戻り値]

実行結果を返します。 戻り値は、[5.9. ステータスコード](#) を参照してください。

#### [備考]

本関数は、IFloat 型ノード専用です。それ以外のノードで実行するとエラーになります。

TeliCamAPI.h をインクルードする必要があります。

#### [コード例]

[Nd\\_GetFloatMin\(\)](#) の[コード例](#)を参照してください。



---

#### 5.6.4.4. Nd\_GetFloatInc

IFloat 型ノードの増加量を取得します。

##### [構文]

```
CAM_API_STATUS Nd_GetFloatInc (  
    CAM_HANDLE      hCam,  
    CAM_NODE_HANDLE hNode,  
    float64_t        *pdInc  
);
```

##### [パラメータ]

パラメータ	内 容
<i>hCam</i> [in]	カメラのカメラハンドルです。
<i>hNode</i> [in]	増加量を取得する IFloat 型ノードのノードハンドルです。
<i>pdInc</i> [out]	取得した増加量を格納する変数へのポインタです。

##### [戻り値]

実行結果を返します。 戻り値は、[5.9. ステータスコード](#) を参照してください。

##### [備考]

本関数は、IFloat 型ノード専用です。それ以外のノードで実行するとエラーになります。

IFloat 型ノードの値が離散値でない場合、エラーになります。

TeliCamAPI.h をインクルードする必要があります。

##### [コード例]

[Nd\\_GetFloatMin\(\)](#) の[コード例](#)を参照してください。

---

#### 5.6.4.5. Nd\_GetFloatDisplayNotation

IFloat 型ノードの推奨数値表示形式を取得します。

##### [構文]

```
CAM_API_STATUS Nd_GetFloatDisplayNotation (  
    CAM_HANDLE          hCam,  
    CAM_NODE_HANDLE     hNode,  
    TC\_NODE\_DISPLAY\_NOTATION *peDisplayNotation  
);
```

##### [パラメータ]

パラメータ		内 容
<i>hCam</i>	[in]	カメラのカメラハンドルです。
<i>hNode</i>	[in]	数値表示形式を取得する IFloat 型ノードのノードハンドルです。
<i>peDisplayNotation</i>	[out]	取得した数値表示形式を格納する変数へのポインタです。

[ TC\_NODE\_DISPLAY\_NOTATION 型 ]

```
typedef enum  
{  
    TC_NODE_DISPLAY_NOTATION_AUTOMATIC,  
    TC_NODE_DISPLAY_NOTATION_FIXED,  
    TC_NODE_DISPLAY_NOTATION_SCIENTIFIC,  
    TC_NODE_DISPLAY_NOTATION_UNDEFINED,  
    TC_NODE_DISPLAY_NOTATION_UNKNOWN  
} TC_NODE_DISPLAY_NOTATION;
```

##### [戻り値]

実行結果を返します。 戻り値は、[5.9. ステータスコード](#) を参照してください。

##### [備考]

本関数は、IFloat 型ノード専用です。それ以外のノードで実行するとエラーになります。

TeliCamAPI.h をインクルードする必要があります。

##### [コード例]

[Nd\\_GetFloatMin\(\)](#) の[コード例](#)を参照してください。

---

#### 5.6.4.6. Nd\_GetFloatDisplayPrecision

IFloat 型ノードの値を表示する時の小数点以下の表示精度を取得します。

##### [構文]

```
CAM_API_STATUS Nd_GetFloatDisplayPrecision (  
    CAM_HANDLE      hCam,  
    CAM_NODE_HANDLE hNode,  
    int64_t          *pLLPrecision  
);
```

##### [パラメータ]

パラメータ		内 容
<i>hCam</i>	[in]	カメラのカメラハンドルです。
<i>hNode</i>	[in]	表示精度を取得する IFloat 型ノードのノードハンドルです。
<i>pLLPrecision</i>	[out]	取得した表示精度を格納する変数へのポインタです。

##### [戻り値]

実行結果を返します。 戻り値は、[5.9. ステータスコード](#) を参照してください。

##### [備考]

本関数は、IFloat 型ノード専用です。それ以外のノードで実行するとエラーになります。

TeliCamAPI.h をインクルードする必要があります。

##### [コード例]

[Nd\\_GetFloatMin\(\)](#) の[コード例](#)を参照してください。

#### 5.6.4.7. Nd\_GetFloatValue

IFloat 型ノードの値を取得します。

##### [構文]

```
CAM_API_STATUS Nd_GetFloatValue (  
    CAM_HANDLE      hCam,  
    CAM_NODE_HANDLE hNode,  
    float64_t        *pdValue,  
    bool8_t          bVerify = false,  
    bool8_t          bIgnoreCache = false,  
);
```

##### [パラメータ]

パラメータ		内 容
<i>hCam</i>	[in]	カメラのカメラハンドルです。
<i>hNode</i>	[in]	値を取得する IFloat 型ノードのノードハンドルです。
<i>pdValue</i>	[out]	取得した値を格納する変数へのポインタです。
<i>bVerify</i>	[in]	trueを設定した場合、アクセスモードおよび、設定範囲をチェックを実行します。 falseを設定した場合、チェックを実行しません。 通常はfalseですを指定してください。この引数は省略可能です。 省略した場合は、falseになります。
<i>bIgnoreCache</i>	[in]	trueを設定した場合、キャッシュ値を無視し、カメラのレジスタ値を取得します。 falseを設定した場合、キャッシュ値を取得します。 通常はfalseですを指定してください。この引数は省略可能です。 省略した場合は、falseになります。

##### [戻り値]

実行結果を返します。 戻り値は、[5.9. ステータスコード](#) を参照してください。

##### [備考]

本関数は、IFloat 型ノード専用です。それ以外のノードで実行するとエラーになります。

TeliCamAPI.h をインクルードする必要があります。

##### [コード例]

[Nd\\_GetFloatMin\(\)](#) の[コード例](#)を参照してください。

---

#### 5.6.4.8. Nd\_SetFloatValue

IFloat 型ノードの値を設定します。

##### [構文]

```
CAM_API_STATUS Nd_SetFloatValue (  
    CAM_HANDLE      hCam,  
    CAM_NODE_HANDLE hNode,  
    float64_t        dValue,  
    bool8_t          bVerify = true  
);
```

##### [パラメータ]

パラメータ		内 容
<i>hCam</i>	[in]	カメラのカメラハンドルです。
<i>hNode</i>	[in]	値を設定する IFloat 型ノードのノードハンドルです。
<i>dValue</i>	[in]	設定する値です。
<i>bVerify</i>	[in]	trueを設定した場合、アクセスモードおよび、設定範囲をチェックを実行します。 falseを設定した場合、チェックを実行しません。 通常はtrueを指定してください。この引数は省略可能です、省略した場合は、trueになります。

##### [戻り値]

実行結果を返します。 戻り値は、[5.9. ステータスコード](#) を参照してください。

##### [備考]

本関数は、IFloat 型ノード専用です。それ以外のノードで実行するとエラーになります。

TeliCamAPI.h をインクルードする必要があります。

##### [コード例]

[Nd\\_GetFloatMin\(\)](#) の[コード例](#)を参照してください。

## 5.6.5. Node 系 IBoolean 型 関数

### 5.6.5.1. Nd\_GetBoolValue

IBoolean 型ノードの値を取得します。

#### [構文]

```
CAM_API_STATUS Nd_GetBoolValue (  
    CAM_HANDLE      hCam,  
    CAM_NODE_HANDLE hNode,  
    bool8_t          *pbValue,  
    bool8_t          bVerify = false,  
    bool8_t          bIgnoreCache = false,  
);
```

#### [パラメータ]

パラメータ		内 容
<b>hCam</b>	[in]	カメラのカメラハンドルです。
<b>hNode</b>	[in]	値を取得する IBoolean 型ノードのノードハンドルです。
<b>pbValue</b>	[out]	取得した値を格納する変数へのポインタです。
<b>bVerify</b>	[in]	trueを設定した場合、アクセスモードおよび、設定範囲をチェックを実行します。 falseを設定した場合、チェックを実行しません。 通常はfalseを指定してください。この引数は省略可能です。 省略した場合は、falseになります。
<b>bIgnoreCache</b>	[in]	trueを設定した場合、キャッシュ値を無視し、カメラのレジスタ値を取得します。 falseを設定した場合、キャッシュ値を取得します。 通常はfalseを指定してください。この引数は省略可能です。 省略した場合は、falseになります。

#### [戻り値]

実行結果を返します。 戻り値は、[5.9. ステータスコード](#) を参照してください。

#### [備考]

本関数は、IBoolean 型ノード専用です。それ以外のノードで実行するとエラーになります。

TeliCamAPI.h をインクルードする必要があります。

#### [コード例]

C++	
CAM_API_STATUS	uiStatus;
uint32_t	uiNum;
CAM_HANDLE	hCam = (CAM_HANDLE)NULL;
CAM_NODE_HANDLE	hNode = (CAM_NODE_HANDLE)NULL;
bool8_t	bRdValue, bWrValue;
// Initialize system.	
uiStatus = Sys_Initialize();	

```

if (uiStatus != CAM_API_STS_SUCCESS)
    return -1;

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

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

do
{
    // 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 = %llx\n", (long long unsigned int)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);
}
while(false);

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

// Terminate system.
Sys_Terminate();

```

---

### 5.6.5.2. Nd\_SetBoolValue

IBoolean 型ノードの値を設定します。

#### [構文]

```
CAM_API_STATUS Nd_SetBoolValue (  
    CAM_HANDLE      hCam,  
    CAM_NODE_HANDLE hNode,  
    bool8_t         bValue,  
    bool8_t         bVerify = true  
);
```

#### [パラメータ]

パラメータ	内 容
<i>hCam</i> [in]	カメラのカメラハンドルです。
<i>hNode</i> [in]	値を設定する IBoolean 型ノードのノードハンドルです。
<i>bValue</i> [in]	設定する値です。
<i>bVerify</i> [in]	trueを設定した場合、アクセスモードおよび、設定範囲をチェックを実行します。 falseを設定した場合、チェックを実行しません。 通常はtrueですを指定してください。この引数は省略可能です。 省略した場合は、trueになります。

#### [戻り値]

実行結果を返します。 戻り値は、[5.9. ステータスコード](#) を参照してください。

#### [備考]

本関数は、IBoolean 型ノード専用です。それ以外のノードで実行するとエラーになります。

TeliCamAPI.h をインクルードする必要があります。

#### [コード例]

[Nd\\_GetBoolValue\(\)](#) の[コード例](#)を参照してください。



## 5.6.6. Node 系 IEnumeration 型 関数

### 5.6.6.1. Nd\_GetNumOfEnumEntries

IEnumeration 型ノードのエントリー数を取得します。

#### [構文]

```
CAM_API_STATUS Nd_GetNumOfEnumEntries (  
    CAM_HANDLE      hCam,  
    CAM_NODE_HANDLE hNode,  
    uint32_t         *piNum  
);
```

#### [パラメータ]

パラメータ	内 容
<i>hCam</i> [in]	カメラのカメラハンドルです。
<i>hNode</i> [in]	エントリー数を取得する IEnumeration 型ノードのノードハンドルです。
<i>piNum</i> [out]	取得したエントリー数を格納する変数へのポインタです。

#### [戻り値]

実行結果を返します。 戻り値は、[5.9. ステータスコード](#) を参照してください。

#### [備考]

本関数は、IEnumeration 型ノード専用です。それ以外のノードで実行するとエラーになります。

取得したエントリー数のすべてのエントリーが有効であるとは限りません。

TeliCamAPI.h をインクルードする必要があります。

#### [コード例]

C++
<pre>CAM_API_STATUS      uiStatus; uint32_t             uiNum, uiSize; CAM_HANDLE           hCam = (CAM_HANDLE)NULL; CAM_NODE_HANDLE      hEnumEntryNode = (CAM_NODE_HANDLE)NULL; CAM_NODE_HANDLE      hNode = (CAM_NODE_HANDLE)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(&amp;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, &amp;hCam, (SIGNAL_HANDLE)NULL); if (uiStatus != CAM_API_STS_SUCCESS)     return -1;</pre>

```

do
{
    // 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 = %llx\n", (long long unsigned int)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, (uint32_t)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 *)malloc(uiSize);
        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);

        free(pszBuf);
        pszBuf = NULL;
    }
}
while(false);

if (pszBuf != NULL) {
    free( pszBuf);
}

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

// Terminate system.
Sys_Terminate();

```

### 5.6.6.2. Nd\_GetEnumIntValue

IEnumeration 型ノードの値を整数値で取得します。

#### [構文]

```
CAM_API_STATUS Nd_GetEnumIntValue (  
    CAM_HANDLE      hCam,  
    CAM_NODE_HANDLE hNode,  
    int64_t          *pLLValue,  
    bool8_t          bVerify = false,  
    bool8_t          bIgnoreCache = false  
);
```

#### [パラメータ]

パラメータ		内 容
<i>hCam</i>	[in]	カメラのカメラハンドルです。
<i>hNode</i>	[in]	値を取得する IEnumeration 型ノードのノードハンドルです。
<i>pLLValue</i>	[out]	取得した整数値を格納する変数へのポインタです。
<i>bVerify</i>	[in]	trueを設定した場合、アクセスモードおよび、設定範囲をチェックを実行します。 falseを設定した場合、チェックを実行しません。 通常はfalseですを指定してください。この引数は省略可能です。 省略した場合は、falseになります。
<i>bIgnoreCache</i>	[in]	trueを設定した場合、キャッシュ値を無視し、カメラのレジスタ値を取得します。 falseを設定した場合、キャッシュ値を取得します。 通常はfalseを指定してください。この引数は省略可能です。 省略した場合は、falseになります。

#### [戻り値]

実行結果を返します。 戻り値は、[5.9. ステータスコード](#) を参照してください。

#### [備考]

本関数は、IEnumeration 型ノード専用です。それ以外のノードで実行するとエラーになります。

TeliCamAPI.h をインクルードする必要があります。

#### [コード例]

C++	
<pre>CAM_API_STATUS      uiStatus; uint32_t             uiNum; CAM_HANDLE           hCam = (CAM_HANDLE)NULL; CAM_NODE_HANDLE      hNode = (CAM_NODE_HANDLE)NULL; int64_t              llRdValue, llWrValue;  // Initialize system. uiStatus = Sys_Initialize(); if (uiStatus != CAM_API_STS_SUCCESS)     return -1;  // Get number of cameras. uiStatus = Sys_GetNumOfCameras(&amp;uiNum);</pre>	

```

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, (SIGNAL_HANDLE)NULL);
if (uiStatus != CAM_API_STS_SUCCESS)
    return -1;

do
{
#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 = %llx\n", (long long unsigned int)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);
}
while(false);

// 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

IEnumeration 型ノードの値を整数値で設定します。

#### [構文]

```
CAM_API_STATUS Nd_SetEnumIntValue (  
    CAM_HANDLE          hCam,  
    CAM_NODE_HANDLE     hNode,  
    int64_t             llValue,  
    bool8_t              bVerify = true  
);
```

#### [パラメータ]

パラメータ		内 容
<i>hCam</i>	[in]	カメラのカメラハンドルです。
<i>hNode</i>	[in]	値を設定する IEnumeration 型ノードのノードハンドルです。
<i>llValue</i>	[in]	設定する整数値です。
<i>bVerify</i>	[in]	trueを設定した場合、アクセスモードおよび、設定範囲をチェックを実行します。 falseを設定した場合、チェックを実行しません。 通常はtrueですを指定してください。この引数は省略可能です。 省略した場合は、trueになります。

#### [戻り値]

実行結果を返します。 戻り値は、[5.9. ステータスコード](#) を参照してください。

#### [備考]

本関数は、IEnumeration 型ノード専用です。それ以外のノードで実行するとエラーになります。

TeliCamAPI.h をインクルードする必要があります。

#### [コード例]

[Nd\\_GetEnumIntValue\(\)](#) [Nd\\_GetNode](#) の[コード例](#)を参照してください。

#### 5.6.6.4. Nd\_GetEnumStrValue

IEnumeration 型ノードの値を文字列で取得します。

##### [構文]

```
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,  
);
```

##### [パラメータ]

パラメータ	内 容
<i>hCam</i> [in]	カメラのカメラハンドルです。
<i>hNode</i> [in]	値を取得する IEnumeration 型ノードのノードハンドルです。
<i>pszBuf</i> [out]	取得した文字列を格納するバッファへのポインタです。 NULLを指定すると、 <a href="#">puiSize</a> に必要なバッファサイズが格納されます。
<i>puiSize</i> [in,out]	文字列を格納するバッファのサイズを格納している変数へのポインタです。（単位：byte） <a href="#">pszBuf</a> にNULL が指定された場合は、必要なバッファサイズが格納されます。
<i>bVerify</i> [in]	trueを設定した場合、アクセスモードおよび、設定範囲をチェックを実行します。 falseを設定した場合、チェックを実行しません。 通常はfalseを指定してください。この引数は省略可能です。 省略した場合は、falseになります。
<i>bIgnoreCache</i> [in]	trueを設定した場合、キャッシュ値を無視し、カメラのレジスタ値を取得します。 falseを設定した場合、キャッシュ値を取得します。 通常はfalseですを指定してください。この引数は省略可能です。 省略した場合は、falseになります。

##### [戻り値]

実行結果を返します。 戻り値は、[5.9. ステータスコード](#) を参照してください。

##### [備考]

本関数は、IEnumeration 型ノード専用です。それ以外のノードで実行するとエラーになります。

TeliCamAPI.h をインクルードする必要があります。

##### [コード例]

C++	
CAM_API_STATUS	uiStatus;
uint32_t	uiNum, uiSize;
CAM_HANDLE	hCam = (CAM_HANDLE)NULL;

```

CAM_NODE_HANDLE      hNode = (CAM_NODE_HANDLE)NULL;
char                  szbuf[32];

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

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

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

do
{
#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);
}
while(false);

// 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

IEnumeration 型ノードの値を文字列で設定します。

#### [構文]

```
CAM_API_STATUS Nd_SetEnumStrValue (  
    CAM_HANDLE      hCam,  
    CAM_NODE_HANDLE hNode,  
    const char      *pszBuf,  
    bool8_t         bVerify = true  
);
```

#### [パラメータ]

パラメータ	内 容
<i>hCam</i> [in]	カメラのカメラハンドルです。
<i>hNode</i> [in]	値を設定する IEnumeration 型ノードのノードハンドルです。
<i>pszBuf</i> [in]	設定する値の文字列を格納したバッファへのポインタです。
<i>bVerify</i> [in]	trueを設定した場合、アクセスモードおよび、設定範囲をチェックを実行します。 falseを設定した場合、チェックを実行しません。 通常はtrueですを指定してください。この引数は省略可能です。 省略した場合は、trueになります。

#### [戻り値]

実行結果を返します。 戻り値は、[5.9. ステータスコード](#) を参照してください。

#### [備考]

本関数は、IEnumeration 型ノード専用です。それ以外のノードで実行するとエラーになります。

TeliCamAPI.h をインクルードする必要があります。

#### [コード例]

[Nd\\_GetEnumStrValue\(\)](#) [Nd\\_GetNode](#) の[コード例](#)を参照してください。



---

#### 5.6.6.6. Nd\_GetEnumEntryByIndex

IEnumeration 型ノードが保有する IEnumEntry 型ノードリスト内のインデックスを指定し、IEnumEntry 型のノードハンドルを取得します。

##### [構文]

```
CAM_API_STATUS Nd_GetEnumEntryByIndex (  
    CAM_HANDLE      hCam,  
    CAM_NODE_HANDLE hNode,  
    uint32_t         uiEnumIdx,  
    CAM_NODE_HANDLE *phEnumEntryNode  
);
```

##### [パラメータ]

パラメータ		内 容
<i>hCam</i>	[in]	カメラのカメラハンドルです。
<i>hNode</i>	[in]	IEnumeration 型ノードのノードハンドルです。
<i>uiEnumIdx</i>	[out]	IEnumEntry型ノードリスト内のインデックスです。
<i>phEnumEntryNode</i>	[out]	取得したIEnumEntry型ノードのノードハンドルを格納する変数へのポインタです。

##### [戻り値]

実行結果を返します。 戻り値は、[5.9. ステータスコード](#) を参照してください。

##### [備考]

本関数は、IEnumeration 型ノード専用です。それ以外のノードで実行するとエラーになります。

TeliCamAPI.h をインクルードする必要があります。

##### [コード例]

[Nd\\_GetEnumStrValue\(\)](#) [Nd\\_GetNode](#) の[コード例](#)を参照してください。

---

## 5.6.7. Node 系 IEnumEntry 型 関数

### 5.6.7.1. Nd\_GetEnumEntryIntValue

IEnumEntry 型ノードの値を整数値で取得します。

#### [構文]

```
CAM_API_STATUS Nd_GetEnumEntryIntValue (  
    CAM_HANDLE      hCam,  
    CAM_NODE_HANDLE hNode,  
    int64_t          *pLLValue  
);
```

#### [パラメータ]

パラメータ	内 容
<i>hCam</i> [in]	カメラのカメラハンドルです。
<i>hNode</i> [in]	IEnumEntry 型ノードのノードハンドルです。
<i>pLLValue</i> [out]	取得した整数値を書き込む変数へのポインタです。

#### [戻り値]

実行結果を返します。 戻り値は、[5.9. ステータスコード](#) を参照してください。

#### [備考]

本関数は、IEnumEntry 型ノード専用です。それ以外のノードで実行するとエラーになります。

TeliCamAPI.h をインクルードする必要があります。

#### [コード例]

[Nd\\_GetNumOfEnumEntries\(\)](#) の[コード例](#)を参照してください。

---

### 5.6.7.2. Nd\_GetEnumEntryStrValue

IEnumEntry 型ノードの値を文字列で取得します。

#### [構文]

```
CAM_API_STATUS Nd_GetEnumEntryStrValue (  
    CAM_HANDLE      hCam,  
    CAM_NODE_HANDLE hNode,  
    char             *pszBuf,  
    uint32_t         *puiSize  
);
```

#### [パラメータ]

パラメータ		内 容
<i>hCam</i>	[in]	カメラのカメラハンドルです。
<i>hNode</i>	[in]	IEnumEntry 型ノードのノードハンドルです。
<i>pszBuf</i>	[out]	取得した文字列を格納するバッファへのポインタです。 NULLを指定すると、 <a href="#">puiSize</a> に必要なバッファサイズが格納されます。
<i>puiSize</i>	[in,out]	取得した文字列を格納するバッファのサイズを格納している変数へのポインタです。（単位：byte） <a href="#">pszBuf</a> に NULL が指定された場合は、必要なバッファサイズが格納されます。

#### [戻り値]

実行結果を返します。 戻り値は、[5.9. ステータスコード](#) を参照してください。

#### [備考]

本関数は、IEnumEntry 型ノード専用です。それ以外のノードで実行するとエラーになります。

TeliCamAPI.h をインクルードする必要があります。

#### [コード例]

[Nd\\_GetNumOfEnumEntries\(\)](#) の[コード例](#)を参照してください。

## 5.6.8. Node 系 ICommand 型 関数

### 5.6.8.1. Nd\_CmdExecute

ICommand 型ノードのコマンドを実行します。

#### [構文]

```
CAM_API_STATUS Nd_CmdExecute (  
    CAM_HANDLE      hCam,  
    CAM_NODE_HANDLE hNode,  
    bool8_t          bVerify = true  
);
```

#### [パラメータ]

パラメータ	内 容
<i>hCam</i>	[in] カメラのカメラハンドルです。
<i>hNode</i>	[in] 実行する ICommand 型ノードのノードハンドルです。
<i>bVerify</i>	[in] trueを設定した場合、アクセスモードおよび、設定範囲をチェックを実行します。 falseを設定した場合、チェックを実行しません。 通常はtrueを指定してください。この引数は省略可能です。 省略した場合は、trueになります。

#### [戻り値]

実行結果を返します。 戻り値は、[5.9. ステータスコード](#) を参照してください。

#### [備考]

本関数は、ICommand 型ノード専用です。 それ以外のノードで実行するとエラーになります。  
TeliCamAPI.h をインクルードする必要があります。

#### [コード例]

C++
<pre>... // Get node handle. uiStatus = Nd_GetNode(hCam, "TriggerSoftware", &amp;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, &amp;bDone);     if (uiStatus != CAM_API_STS_SUCCESS)         return -1;      if (bDone == true)         break;      Sleep(0); } ...</pre>

### 5.6.8.2. Nd\_GetCmdIsDone

ICommand 型ノードのコマンド実行状態を取得します。

#### [構文]

```
CAM_API_STATUS Nd_GetCmdIsDone (  
    CAM_HANDLE      hCam,  
    CAM_NODE_HANDLE hNode,  
    bool18_t        *pbDone,  
    bool18_t        bVerify = false  
);
```

#### [パラメータ]

パラメータ		内 容
<i>hCam</i>	[in]	カメラのカメラハンドルです。
<i>hNode</i>	[in]	実行状態を取得する ICommand 型ノードのノードハンドルです。
<i>pbDone</i>	[out]	取得したコマンドの実行状態を格納する変数へのポインタです。 true が取得された場合はコマンド処理終了、 false が取得された場合はコマンド処理実行中です。
<i>bVerify</i>	[in]	trueを設定した場合、アクセスモードおよび、設定範囲をチェックを実行します。 falseを設定した場合、チェックを実行しません。 通常はfalseですを指定してください。この引数は省略可能です。 省略した場合は、falseになります。

#### [戻り値]

実行結果を返します。 戻り値は、[5.9. ステータスコード](#) を参照してください。

#### [備考]

本関数は、ICommand 型ノード専用です。それ以外のノードで実行するとエラーになります。

TeliCamAPI.h をインクルードする必要があります。

#### [コード例]

[Nd\\_CmdExecute\(\)](#) の[コード例](#)を参照してください。

## 5.6.9. Node 系 IString 型 関数

### 5.6.9.1. Nd\_GetStrValue

IString 型ノードの値（文字列）を取得します。

#### [構文]

```
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,  
);
```

#### [パラメータ]

パラメータ		内 容
<i>hCam</i>	[in]	カメラのカメラハンドルです。
<i>hNode</i>	[in]	値（文字列）を取得する IString 型ノードのノードハンドルです。
<i>pszBuf</i>	[out]	取得した文字列を格納するバッファへのポインタです。 NULLを指定すると、 <a href="#">puiSize</a> に必要なバッファサイズが格納されます。
<i>puiSize</i>	[in,out]	文字列を格納するバッファのサイズを格納している変数へのポインタ です。（単位：byte） <a href="#">pszBuf</a> にNULLが指定された場合は、必要なバッファサイズが格納され ます。
<i>bVerify</i>	[in]	trueを設定した場合、アクセスモードおよび、設定範囲をチェックを 実行します。 falseを設定した場合、チェックを実行しません。 通常はfalseを指定してください。この引数は省略可能です。 省略した場合は、falseになります。
<i>bIgnoreCache</i>	[in]	trueを設定した場合、キャッシュ値を無視し、カメラのレジスタ値を 取得します。 falseを設定した場合、キャッシュ値を取得します。 通常はfalseを指定してください。この値は省略可能です。 省略した場合は、falseになります。

#### [戻り値]

実行結果を返します。 戻り値は、[5.9. ステータスコード](#) を参照してください。

#### [備考]

本関数は、IString 型ノード専用です。それ以外のノードで実行するとエラーになります。

TeliCamAPI.h をインクルードする必要があります。

---

## [コード例]

```
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

IString 型ノードの値（文字列）を設定します。

#### [構文]

```
CAM_API_STATUS Nd_SetStrValue (  
    CAM_HANDLE      hCam,  
    CAM_NODE_HANDLE hNode,  
    const char      *pszBuf,  
    bool8_t         bVerify = true  
);
```

#### [パラメータ]

パラメータ		内 容
<i>hCam</i>	[in]	カメラのカメラハンドルです。
<i>hNode</i>	[in]	値（文字列）を設定する IString 型ノードのノードハンドルです。
<i>pszBuf</i>	[in]	文字列を格納したバッファへのポインタです。
<i>bVerify</i>	[in]	trueを設定した場合、アクセスモードおよび、設定範囲をチェックを実行します。 falseを設定した場合、チェックを実行しません。 通常はtrueを指定してください。この引数は省略可能です。 省略した場合は、trueになります。

#### [戻り値]

実行結果を返します。 戻り値は、[5.9. ステータスコード](#) を参照してください。

#### [備考]

本関数は、IString 型ノード専用です。それ以外のノードで実行するとエラーになります。

TeliCamAPI.h をインクルードする必要があります。

#### [コード例]

[Nd\\_GetStrValue\(\)](#) のコード例を参照してください。



---

## 5.7. ユーティリティ関数

### 5.7.1. 画像フォーマット変換関数

#### 5.7.1.1. PrepareLUT

画像フォーマット変換関数で使用するルックアップテーブルを使用可能な状態に設定します。

**[構文]**

```
CAM_API_STATUS  PrepareLUT (void);
```

**[戻り値]**

実行結果を返します。 戻り値は、[5.9. ステータスコード](#) を参照してください。

**[備考]**

画像フォーマット変換関数を実行する前にこの関数を必ず1回実行してください。

TeliCamUtil.h をインクルードする必要があります。

### 5.7.1.2. Conv\*ToBGRA

引数の原画像データを BGRA フォーマットのデータに変換します。  
このユーティリティでは 23 種類の原画像フォーマットに対応した画像変換関数を提供しています。

BGRA フォーマットは 1 画素のデータが 4 個の 8bit コンポーネント (B: blue、G: green、R: red、A: alpha (透明度)) で構成され、アドレスの若い順で B、G、R、A の順序でコンポーネントを配置するフォーマットです。32bit ARGB フォーマット Bitmap の画像データ部分と同じデータ配置になります。  
この関数では透明度 A は常に 255 の値になります。

#### [構文]

```
CAM_API_STATUS Conv*ToBGRA (  
    void          *pvDstBGRA,  
    void          *pvSrc,  
    uint32_t      uiWidth,  
    uint32_t      uiHeight  
);
```

#### [関数名]

23 種の同じ syntax を持つ関数を提供します。

関 数 名	原画像 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

---

## [パラメータ]

パラメータ		内 容
<i>puiDstARGB</i>	[out]	出力画像データバッファへのポインタです。このバッファに BGRA フォーマットに変換されたデータが格納されます。バッファには予めメモリを割り付けておく必要があります。
<i>pvSrc</i>	[in]	原画像データへのポインタです。 この画像データが BGRA フォーマットに変換されます。
<i>uiWidth</i>	[in]	原画像データの幅です。(単位：画素) 画像幅は 4 の倍数である必要があります。
<i>uiHeight</i>	[in]	原画像データの高さです。(単位：画素)

## [戻り値]

実行結果を返します。 戻り値は、[5.9. ステータスコード](#) を参照してください。

## [備考]

TeliCamUtil.h をインクルードする必要があります。

### 5.7.1.3. Conv\*ToBGR

引数の原画像データを BGR フォーマットのデータに変換します。 このユーティリティでは 23 種類の原画像フォーマットに対応した画像変換関数を提供しています。

BGR フォーマットは 1 画素のデータが 3 個の 8bit コンポーネント (B: blue、G: green、R: red) で構成され、アドレスの若い順で B、G、R の順序でコンポーネントを配置するフォーマットです。 24bit RGB フォーマット Bitmap の画像データ部分と同じデータ配置になります。

#### [構文]

```
CAM_API_STATUS Conv*ToBGR (  
    void          *pvDstBGR,  
    void          *pvSrc,  
    uint32_t      uiWidth,  
    uint32_t      uiHeight  
);
```

#### [Functions]

23 種の同じ syntax を持つ関数を提供します。

関 数 名	原画像 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_UYVY (YUV411Packed)	0x020C001E
ConvYUV422PToBGR	YUV422_8_UYVY (YUV422Packed)	0x0210001F
ConvYUV444PToBGR	YUV8_UYV (YUV444Packed)	0x02180020

---

## [パラメータ]

パラメータ		内 容
<i>puiDstBGR</i>	[out]	出力画像データバッファへのポインタです。このバッファに BGR フォーマットに変換されたデータが格納されます。 バッファには予めメモリを割り付けておく必要があります。
<i>pvSrc</i>	[in]	原画像データへのポインタです。 この画像データが BGR フォーマットに変換されます。
<i>uiWidth</i>	[in]	原画像データの幅です。(単位：画素) 画像幅は 4 の倍数である必要があります。
<i>uiHeight</i>	[in]	原画像データの高さです。(単位：画素)

## [戻り値]

実行結果を返します。 戻り値は、[5.9. ステータスコード](#) を参照してください。

## [備考]

TeliCamUtil.h をインクルードする必要があります。

### 5.7.1.4. ConvImage

各種画像フォーマットのデータを BGRA フォーマットまたは BGR フォーマットに変換します。  
この関数は 5.7.1.2 章と 5.7.1.3 章に記載した画像フォーマット変換関数を使用して画像を変換します。

#### [構文]

```
CAM_API_STATUS ConvImage (  
    DST_FORMAT          eDstFormat,  
    CAM_PIXEL_FORMAT    uiSrcPixelFormat,  
    bool8_t             bBayreConversion  
    void                *pvDst,  
    void                *pvSrc,  
    uint32_t            uiWidth,  
    uint32_t            uiHeight  
);
```

#### [eters]

パラメータ		内 容
<i>eDstFormat</i>	[in]	以下のいずれかの値で出力画像フォーマットです。 DST_FMT_BGRA32 : BGRA フォーマット(32bit)に変換。 DST_FMT_BGR24 : BGR フォーマット(24bit)に変換。
<i>uiSrcPixelFormat</i>	[in]	原画像データの PixelFormat です。
<i>bBayerConversion</i>	[in]	原画像がバイヤータイプで、この値が偽のとき、原画像をモノクロ画像として扱い、画像を変換します。 原画像がバイヤータイプで、この値が真のときは、原画像をバイヤーフィルタ付画像として扱い、カラー画像に変換します。 原画像がバイヤータイプでない場合、この値は無視されます。
<i>puiDst</i>	[out ]	出力画像データバッファへのポインタです。 変換された画像データが、このバッファに格納されます。 バッファには予めメモリを割り付けておく必要があります。
<i>pvSrc</i>	[in]	原画像データへのポインタです。 この画像データが BGR フォーマットに変換されます。
<i>uiWidth</i>	[in]	原画像データの幅です。(単位：画素) 画像幅は 4 の倍数である必要があります。
<i>uiHeight</i>	[in]	原画像データの高さです。(単位：画素)

#### [戻り値]

実行結果を返します。 戻り値は、[5.9. ステータスコード](#) を参照してください。

#### [備考]

TeliCamUtil.h をインクルードする必要があります。

## [コード例]

```
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. その他ユーティリティ

### 5.7.2.1. BitPerPixel

引数の PixelFormat データの 1 画素あたりのビット数を返します。

#### [構文]

```
uint8_t BitPerPixel (  
    CAM_PIXEL_FORMAT    uiPixelFormat  
);
```

#### [パラメータ]

パラメータ	内 容
uiPixelFormat [in]	PixelFormat です。

#### [戻り値]

1 画素あたりビット数を返します。例：RGB8 フォーマットの場合 24 を返します。

#### [備考]

TeliCamUtil.h をインクルードする必要があります。

### 5.7.2.2. DataDepth

引数の PixelFormat データのデータ深度をビット数として返します。1 画素のデータが複数のコンポーネントで構成される場合、コンポーネントのデータ深度を求める関数になります。

#### [構文]

```
uint8_t DataDepth (  
    CAM_PIXEL_FORMAT    uiPixelFormat  
);
```

#### [パラメータ]

パラメータ	内 容
uiPixelFormat [in]	PixelFormat です。

#### [戻り値]

データ深度を返します。例：RGB8 フォーマットの場合 8 を返します。

#### [備考]

TeliCamUtil.h をインクルードする必要があります。



---

### 5.7.2.3. IsMonochromic

引数の PixelFormat がモノクロームタイプか否かを返します。  
PixelFormat 値の最上位バイトの値が“01”のフォーマットがモノクロームタイプフォーマットです。  
Mono8、Mono10、Mono12、Mono16 とベイヤーフォーマットがモノクロームに該当します。

#### [構文]

```
bool8_t IsMonochromic (  
    CAM_PIXEL_FORMAT      uiPixelFormat  
);
```

#### [パラメータ]

パラメータ	内 容
uiPixelFormat [in]	PixelFormat です。

#### [戻り値]

PixelFormat が Mono8、Mono10、Mono12、Mono16 またはベイヤーフォーマットの時 true を返します。 それ以外の時、false を返します。

#### [備考]

TeliCamUtil.h をインクルードする必要があります。

### 5.7.2.4. IsPixelBayer

引数の Pixel フォーマットがベイヤータイプか否かを返します。

#### [構文]

```
bool8_t IsBayer (  
    CAM_PIXEL_FORMAT      uiPixelFormat  
);
```

#### [パラメータ]

パラメータ	内 容
uiPixelFormat [in]	PixelFormat です。

#### [戻り値]

PixelFormat がベイヤータイプフォーマットの時 true を返します。それ以外の時、false を返します。

#### [備考]

TeliCamUtil.h をインクルードする必要があります。

### 5.7.2.5. SaveBmp\*

引数の画像データを Bitmap ファイルとして保存します。

32bitARGB フォーマット、24bitRGB フォーマット、モノクロ 8bit フォーマットの 3 種類の関数を提供しています。

既存ファイルが Path で指定された場合、既存ファイルを新しい Bitmap ファイルで上書きします。

#### [構文]

```
CAM_API_STATUS SaveBmp* (  
    void          *pvTgt,  
    uint32_t      uiWidth,  
    uint32_t      uiHeight,  
    const char    *pszPath  
);
```

#### [関数名]

Bitmap フォーマットが異なる 3 種の関数を提供します。.

関 数 名	Bitmap フォーマット t	備 考
SaveBmpARGB	Format32bppArgb	
SaveBmpRGB	Format24bppRgb	
SaveBmpMono	Format8bppIndexed	モノクロ画像用カラーパレット使用。

#### [パラメータ]

パラメータ	内 容
<i>pvTgt</i> [in]	保存対象画像データの先頭画素のポインタです。
<i>uiWidth</i> [in]	保存対象画像の幅方向画素数です。
<i>uiHeight</i> [in]	保存対象画像の高さ方向画素数です。
<i>pszPath</i> [in]	画像保存先のファイル Path です。 指定されたファイルのディレクトリが実在し、書き込み可能である必要があります。

#### [戻り値]

実行結果を返します。 戻り値は、[5.9. ステータスコード](#) を参照してください。

#### [備考]

TeliCamUtil.h をインクルードする必要があります。

### 5.7.2.6. ReverseImg

引数の画像を左右方向、上下方向で反転した画像を作成します。

Bayer 形式の画像を反転対象とした場合、画像データをモノクロデータとして扱い、画像の反転を行います。このため、反転後の画像データの色フィルタ配置は反転前の画像データと異なる配置になります。

例えば、BG 配置の画像を左右反転すると GB 配置の画像データが得られ、上下反転すると GR 配置の画像データが得られます。

処理対象 PixelFormat は以下の通りです。

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

#### [構文]

```
CAM_API_STATUS ReverseImg (  
    void                *pvDst,  
    void                *pvSrc,  
    CAM_PIXEL_FORMAT    uiPixelFormat,  
    uint32_t            uiWidth,  
    uint32_t            uiHeight,  
    bool8_t             bRevX,  
    bool8_t             bRevY  
);
```

#### [パラメータ]

パラメータ		内 容
<i>pvDst</i>	[in]	変換先画像データです。
<i>pvSrc</i>	[in]	変換元の画像データです。
<i>uiPixelFormat</i>	[in]	変換元画像の PixelFormat です。
<i>uiWidth</i>	[in]	変換元画像の幅方向画素数です。
<i>uiHeight</i>	[in]	変換元画像の高さ方向画素数です。
<i>bRevX</i>	[in]	X 方向の反転 ON/OFF です。 true のとき左右方向の反転を施します。
<i>bRevY</i>	[in]	Y 方向の反転 ON/OFF です。 true のとき上下方向の反転を施します。

#### [戻り値]

実行結果を返します。 戻り値は、[5.9. ステータスコード](#) を参照してください。

#### [備考]

TeliCamAPI.h と TeliCamUtil.h をインクルードする必要があります。

---

## 5.8. その他の関数

### 5.8.1. GetCamIndexFromCamHandle

カメラハンドルから、オープンしたときのカメラのインデックスを取得します。

#### [構文]

```
CAM_API_STATUS GetCamIndexFromCamHandle (  
    CAM_HANDLE      hCam,  
    uint32_t         *puiCamIndex  
);
```

#### [パラメータ]

パラメータ	内 容
<i>hCam</i> [in]	カメラのカメラハンドルです。
<i>puiCamIndex</i> [out]	取得したカメラのインデックスを格納する変数へのポインタです。

#### [戻り値]

実行結果を返します。 戻り値は、[5.9. ステータスコード](#) を参照してください。

#### [備考]

新しいカメラが接続されたり他のカメラが外された後に [Sys\\_GetNumOfCameras\(\)](#) がコールされると、取得したインデックスと現在 TeliCamAPI 内部で管理しているインデックスが異なる場合があります。

TeliCamAPI.h をインクルードする必要があります。

---

## 5.8.2. GetCamTypeFromCamHandle

カメラハンドルから、カメラのタイプを取得します。

### [構文]

```
CAM_API_STATUS GetCamTypeFromCamHandle (  
    CAM_HANDLE      hCam,  
    CAM_TYPE        *peType  
);
```

### [パラメータ]

パラメータ		内 容
<i>hCam</i>	[in]	カメラのカメラハンドルです。
<i>peType</i>	[out]	取得したカメラのタイプを格納する変数へのポインタです。

### [戻り値]

実行結果を返します。 戻り値は、[5.9. ステータスコード](#) を参照してください。

### [備考]

CAM\_TYPE 型は、TeliCamAPI.h で定義されています。

TeliCamAPI.h をインクルードする必要があります。

---

### 5.8.3. GetCamTLParamsLocked

TLParamsLocked の値を取得します。

TLParamsLocked は、画像取得中にトランスポートレイヤーでクリティカルな機能の設定変更を禁止するために使用されます。

TLParamsLocked が 0 のとき、ロックされている機能はありません。

TLParamsLocked が 1 のとき、トランスポートレイヤーとカメラのクリティカルな機能はロックされ、設定を変更することができません。

#### [構文]

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

#### [パラメータ]

パラメータ	内 容
<i>hCam</i> [in]	カメラのカメラハンドルです。
<i>puiValue</i> [out]	取得した TLParamsLocked の値を格納する変数へのポインタです。

#### [戻り値]

実行結果を返します。 戻り値は、[5.9. ステータスコード](#) を参照してください。

#### [備考]

GenICam アクセスを無効に設定している場合はエラーがリターンされます。

TLParamsLocked の制御は TeliCamAPI 内部で行っているため、ユーザが設定する必要はありません。

レジスタアクセス関数を使用してレジスタに直接設定を行った場合は、ロックされていても設定可能なレジスタがあります。(ただし、処理が実行されない、または設定可能な状態になるまで処理が保留される場合があります。)

TeliCamAPI.h をインクルードする必要があります。

## 5.8.4. Misc\_GetLastGenICamError

CAM\_API\_STS\_GENICAM\_ERR が発生したときの、エラー情報を取得します。

### [構文]

```
CAM_API_STATUS Misc_GetLastGenICamError (  
    CAM_GENICAM_ERR_MSG *peErrMsg  
);
```

### [パラメータ]

パラメータ	内 容
peErrMsg [out]	取得したエラー情報を格納する変数へのポインタです。

### [CAM\_GENICAM\_ERR\_MSG 構造体]

```
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;
```

CAM_GENICAM_ERR_MSG	内 容
pszDescription [out]	エラー内容です。
pszSourceFileName [out]	エラーが発生したファイル名です。
uiSourceLine [out]	エラーが発生したファイルの行番号です。

### [戻り値]

実行結果を返します。 戻り値は、[5.9. ステータスコード](#) を参照してください。

### [備考]

CAM\_API\_STS\_GENICAM\_ERR は、[5.6 GenICam 関数](#) をコールしたときに発生することがあります。また、[5.5 カメラ 制御関数](#) の一部は GenICam 関数を使用して実行されるため、CAM\_API\_STS\_GENICAM\_ERR が発生することがあります。

TeliCamAPI はこのエラー情報を TeliCamAPI 全体で管理しており、スレッドごとの情報管理は行っておりません。このため、本関数を実行する前または実行中に、他のスレッドで CAM\_API\_STS\_GENICAM\_ERR が発生した場合は、取得したいエラー情報が取得できない場合があります。

TeliCamAPI.h をインクルードする必要があります。

## 5.9. ステータスコード

TeliCamAPI の関数実行時に返されるステータスコードは以下の通りです。

CAM_API_STATUS	値	内 容
CAM_API_STS_SUCCESS	0x00000000	成功しました。
CAM_API_STS_NOT_INITIALIZED	0x00000001	API の初期化が行われていません。
CAM_API_STS_ALREADY_INITIALIZED	0x00000002	API の初期化がすでに行われています。
CAM_API_STS_NOT_FOUND	0x00000003	カメラが見つかりません。
CAM_API_STS_ALREADY_OPENED	0x00000004	すでにオープンされています。
CAM_API_STS_ALREADY_ACTIVATED	0x00000005	すでにアクティブ化されています。
CAM_API_STS_INVALID_CAMERA_INDEX	0x00000006	指定されたカメラインデックスが不正です。
CAM_API_STS_INVALID_CAMERA_HANDLE	0x00000007	指定されたカメラハンドルが不正です。
CAM_API_STS_INVALID_NODE_HANDLE	0x00000008	指定されたノードハンドルが不正です。
CAM_API_STS_INVALID_STREAM_HANDLE	0x00000009	指定されたストリームハンドルが不正です。
CAM_API_STS_INVALID_REQUEST_HANDLE	0x0000000A	指定されたリクエストハンドルが不正です。
CAM_API_STS_INVALID_EVENT_HANDLE	0x0000000B	指定されたイベントハンドルが不正です。
CAM_API_STS_INVALID_PARAMETER	0x0000000D	指定されたパラメータが不正です。
CAM_API_STS_BUFFER_TOO_SMALL	0x0000000E	指定されたバッファが小さすぎます。
CAM_API_STS_NO_MEMORY	0x0000000F	TeliCamAPI 内部のメモリ確保に失敗しました。
CAM_API_STS_MEMORY_NO_ACCESS	0x00000010	指定されたメモリへのアクセスに失敗しました。
CAM_API_STS_NOT_IMPLEMENTED	0x00000011	未実装の機能です。 レジスタ名（ノード名）が間違っている場合にもこのエラーが発生します。
CAM_API_STS_TIMEOUT	0x00000012	タイムアウトが発生しました。
CAM_API_STS_CAMERA_NOT_RESPONDING	0x00000013	カメラを検出できませんでした。 カメラを接続しているケーブルが外れている可能性があります。 接続を確認してください。
CAM_API_STS_EMPTY_COMPLETE_QUEUE	0x00000014	完了したリクエストが、完了キューに存在しません。
CAM_API_STS_NOT_READY	0x00000015	準備が完了していません。
CAM_API_STS_ACCESS_MODE_SET_ERR	0x00000016	アクセスモードの設定に失敗しました。
CAM_API_STS_XML_LOAD_ERR	0x00000101	XMLファイルのロードに失敗しました。
CAM_API_STS_GENICAM_ERR	0x00000102	GenICamエラーが発生しました。
CAM_API_STS_DLL_LOAD_ERR	0x00000103	共有ライブラリファイルのロードに失敗しました。
CAM_API_STS_INVALID_ADDRESS	0x00000801	無効なアドレスが指定されました。
CAM_API_STS_WRITE_PROTECT	0x00000802	指定アドレスがライトプロテクトされているため、処理に失敗しました。
CAM_API_STS_BAD_ALIGNMENT	0x00000803	不整列なアドレスにアクセスしました。
CAM_API_STS_ACCESS_DENIED	0x00000804	アクセスが拒否されました。
CAM_API_STS_BUSY	0x00000805	ビジー状態です。



CAM_API_STATUS	値	内 容
CAM_API_STS_NOT_READABLE	0x00000806	読み出し可能な状態ではありません。
CAM_API_STS_NOT_WRITABLE	0x00000807	書き込み（実行）可能な状態ではありません。
CAM_API_STS_NOT_AVAILABLE	0x00000808	実行された関数が利用できません。 または、設定パラメータが有効ではありません。 GenICamのアクセスを無効にしてカメラをオープンした場合、利用できないカメラ制御関数があります。
CAM_API_STS_REQUEST_TIMEOUT	0x00001001	リクエストがタイムアウトしました。
CAM_API_STS_RESEND_TIMEOUT	0x00001002	再送要求を行わない場合、あるパケットを受信してから時間内に次のパケットを受信できず、タイムアウトしました。（GigE Visionカメラのみ）
CAM_API_STS_RESPONSE_TIMEOUT	0x00001003	再送要求を行ってから時間内に要求したパケットを受信できず、タイムアウトしました。（GigE Visionカメラのみ）
CAM_API_STS_BUFFER_FULL	0x00001004	受信データのサイズが指定の最大値を超えました。
CAM_API_STS_UNEXPECTED_BUFFER_SIZE	0x00001005	実際の受信データサイズが、トレーラに記載されているサイズと一致しませんでした。
CAM_API_STS_UNEXPECTED_NUMBER	0x00001006	受信パケットが最大数を超えました。
CAM_API_STS_PACKET_STATUS_ERROR	0x00001007	パケットのステータスがエラーでした。
CAM_API_STS_RESEND_NOT_IMPLEMENTED	0x00001008	カメラがパケット再送コマンドに対応していません。
CAM_API_STS_PACKET_UNAVAILABLE	0x00001009	ストリームパケットは利用できません。
CAM_API_STS_MISSING_PACKETS	0x0000100A	フレームデータの完了前に、次のブロックのリーダーが受信されました。（パケットが欠落している可能性があります。）
CAM_API_STS_FLUSH_REQUESTED	0x0000100B	ユーザ操作により、リクエストがフラッシュされました。
CAM_API_STS_TOO_MANY_PACKET_MISSING	0x0000100C	パケットの消失が規定値を超えました。 このエラーがGigE Visionカメラで発生するとき、ネットワークの帯域が不足している可能性があります。ジャンボフレームを有効にしたり、カメラのフレームレートを低下させて、帯域幅を超えないようにしてください。
CAM_API_STS_FLUSHED_BY_D0EXIT	0x0000100D	パワーステートが変更されて、リクエストがフラッシュされました。
CAM_API_STS_FLUSHED_BY_CAMERA_REMOVED	0x0000100E	カメラが取り外されて、リクエストがフラッシュされました。
CAM_API_STS_DRIVER_LOAD_ERROR	0x0000100F	ドライバのロードに失敗しました。
CAM_API_STS_FILE_OPEN_ERROR	0x00002001	ファイルのオープンに失敗しました。
CAM_API_STS_FILE_WRITE_ERROR	0x00002002	ファイルへのデータ書き込みに失敗しました。
CAM_API_STS_FILE_READ_ERROR	0x00002003	ファイルからデータ読み出しに失敗しました。
CAM_API_STS_FILE_NOT_FOUND	0x00002004	ファイルが見つかりませんでした。
CAM_API_STS_DATA_DISCARDED	0x0000A100	この block_id のいくつかのパケットが破棄されました。このステータスは、トレーラパケットで使用されます。
CAM_API_STS_UNSUCCESSFUL	0xFFFFFFFF	その他のエラーが発生しました。

---

## 6. サンプル ソースコード

TeliCamSDK はユーザアプリケーション実装の参考として以下の表に記載したサンプルソースコードを同梱しています。 サンプルソースコードは順次追加予定です。

サンプル名	UI	機能
GrabStream_FreerunUsingCallback	CUI	画像の輝度値表示 （コールバックを使用したフリーランによる画像取り込み）
GrabStream_FreerunUsingSignal	CUI	画像の輝度値表示 （シグナルを使用したフリーランによる画像取り込み）
GrabStream_SWTrigUsingSignal	CUI	画像の輝度値表示 （シグナルを使用したソフトウェアトリガによる画像取り込み）
GrabStream_ViaNode	CUI	画像の輝度値表示 （GenICam ノードを使用した画像取り込み）
GrabStream_RegBU	CUI	画像の輝度値表示 （BU シリーズのレジスタアクセスによる画像取り込み）
GrabEvent	CUI	FrameTrigger イベント取得
MultiCamera	GUI	最大4カメラ画像同時表示

これらのサンプルソースコードは、以下のディレクトリにインストールされています。

`$HOME/TeliCamSDK/samples`

TeliCamSDK を利用したアプリケーションを実行するためには、以下の通り環境変数を設定しなければなりません。

```
TELICAMSDK=/opt/TeliCamSDK
export TELICAMSDK
```

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

上記環境変数は、シェルスクリプトを実行することにより設定できます。

```
source /opt/TeliCamSDK/set_env.sh
```

---

## 6.1. コンソール サンプル

コンソールサンプルをコンパイルする手順は以下の通りです。

1. ターミナル (gnome-terminal) を起動します。
2. コンソール サンプルがインストールされているディレクトリに移動します。

```
cd $HOME/TeliCamSDK/samples/CPP/ConsoleSamples
```

3. すべてのプロジェクトのコンパイル実行

```
make
```

コンパイルに成功すると、各プロジェクトディレクトリにバイナリファイルが生成されます。  
実行するには、各プロジェクトディレクトリでシェルを実行します。

以下は GrabStream\_FreerunUsingCallback サンプルを実行する方法です。

```
cd ./GrabStream_FreerunUsingCallback
```

```
sh ./execute_GrabStream_FreerunUsingCallback.sh
```

## 6.2. Qt サンプル

Qt サンプルをコンパイルするには、Qt がインストールされている必要があります。

Qt サンプルをコンパイルする手順は以下の通りです。

1. ターミナル (gnome-terminal) を起動します。
2. Qt サンプルがインストールされているディレクトリに移動します。

```
cd $HOME/TeliCamSDK/samples/CPP/QtSamples/Qt5/MultiCamera
```

3. 環境変数を設定し、Qt Creator を実行します。

```
sh ./set_qt_env.sh
```

## 7. その他

### 7.1. 改定履歴

Date	Version	内容
2016/01/26	1.0.0	初版

### 7.2. 免責事項

このソフトウェアの免責事項は、別途付属の” License Agreement TeliCamSDK for Linux Jpn.txt”に記載されています。必ずご一読の上、ご利用されますようお願い致します。

ライセンスに関するドキュメントは /opt/TeliCamSDK/documents/licenses にインストールされています。

### 7.3. ライセンス

TeliCamSDK は、複数の独立したソフトウェアコンポーネントを使用しています。 個々のソフトウェアコンポーネントは、それぞれ第三者の著作権が存在します。

TeliCamSDK は、第三者が規定したエンドユーザーライセンスアグリーメントあるいは著作権通知(以下、「EULA」といいます)に基づきフリーウェアとして配布されるソフトウェアコンポーネントを使用しております。

「EULA」の中には、実行形式のソフトウェアコンポーネントを配布する条件として、当該コンポーネントのソースコードの入手を可能とするよう求めているものがあります。 当該「EULA」の対象となるソフトウェアコンポーネントのお問い合わせに関しては、弊社営業担当までお問い合わせください。TeliCamSDKで使用している、対象となるソフトウェアコンポーネントの「EULA」は以下のディレクトリにインストールされています。

/opt/TeliCamSDK/documents/licenses

東芝テリーは、東芝テリーが定める条件の基で TeliCamSDK の動作を保証します。(” License Agreement TeliCamSDK for Linux Jpn.txt” と “License Agreement TeliCamSDK for Linux Sample Jpn.txt” をご覧ください。) ただし、「EULA」に基づいて配布されるソフトウェアコンポーネントには、著作権者または弊社を含む第三者の保証がないことを前提に、お客様がご自身でご利用になれることが認められるものであります。 この場合、当該ソフトウェアコンポーネントは無償でお客様に使用許諾されますので、適用法令の範囲内で、当該ソフトウェアコンポーネントの保証は一切ありません。 ここでいう保証とは、市場性や特定目的適合性についての黙示の保証も含まれますが、それに限定されるものではありません。 当該ソフトウェアコンポーネントの品質や性能に関するすべてのリスクはお客様が追うものとします。 また、当該ソフトウェアコンポーネントに欠陥があると分かった場合、それに伴う一切の派生費用や修理・訂正に要する費用は、東芝テリーは一切の責任を負いません。 適用法令の定め、または書面による合意がある場合を除き、著作権者や上記許諾を受けて当該ソフトウェアコンポーネントを使用したこと、または使用できないことに起因する一切の損害について何らの責任も負いません。著作権者や第三者が、そのような損害の発生する可能性について知らされていた場合でも同様です。なお、ここでいう損害には、通常損害、特別損害、偶発損害、間接損害が含まれます(データの消失、またはその正確さの喪失、お客様や第三者が被った損失、他のソフトウェアとのインタフェースの不適合化等も含まれますが、これに限定されるものではありません)。 当該ソフトウェアコンポーネントの使用条件や遵守いただかなければならない事項等の詳細は、各「EULA」をお読みください。

TeliCamSDKで使用している「EULA」の対象となるソフトウェアコンポーネントは、以下の表のとおりです。 これらのソフトウェアコンポーネントをお客様自身でご利用いただく場合は、対応する「EULA」をよく読んでから、ご利用くださるようお願いいたします。

対応ソフトウェアモジュール	ライセンス
gcc libgcc	GPLv3.txt and gcc-exception.txt (GPLv3 with GCC Runtime Library Exception)
gcc libstdc++	GPLv3.txt and gcc-exception.txt (GPLv3 with GCC Runtime Library Exception)
glibc	LGPLv2.1
libteliusb (libusb)	LGPLv2.1
GenICam	GenICam license
Qt	LGPLv2.1 and Digia Qt LGPL Exception version 1.1

本ドキュメントで使用されている商品名は、各社の商標もしくは登録商標です。

## 7.4. お問い合わせ

TeliCamSDK に関して不明な点がございましたら、下記のヘルプのフォームからお問い合わせください。

[http://www.toshiba-teli.co.jp/support/contact/industrial\\_j.htm](http://www.toshiba-teli.co.jp/support/contact/industrial_j.htm)

または、下記窓口までご連絡ください。

191-0065

東京都日野市旭が丘 4-7-1

東芝テリー株式会社

営業部 ソリューション・技術担当

TEL : 042-589-8772

FAX : 042-589-8774