TeliCamAPI

ライブラリマニュアル

Version 3.0.0 (2020/01/24)

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

DAA00563A

 www.toshiba-teli.co.jp

目次

1. はじめに 4
2. 構成 5
  2.1. Windows 版の構成 5
  2.2. Linux 版の構成 6
3. システム要件 7
  3.1. Windows 版のシステム要件 7
  3.2. Linux 版のシステム要件 8
4. ライブラリ 9
  4.1. 使用方法 10
    4.1.1. TeliCamAPI の初期化と終了 10
    4.1.2. カメラの検索 10
    4.1.3. カメラ　オープン／クローズ 10
    4.1.4. カメラの制御、情報取得 10
    4.1.5. ストリームコントロール 11
    4.1.6. カメライベント通知（メッセージ）コントロール 19
    4.1.7. カメラのアクセスモード（制御チャネル特権） 25
    4.1.8. ハートビート処理 25
  4.2. SDK のインストール 26
  4.3. SDK のアンインストール 26
  4.4. 開発環境の設定 26
    4.4.1. Windows 版　開発環境の設定 26
    4.4.2. Linux 版　開発環境の設定 27
  4.5. Linux 版におけるパフォーマンスのチューニング 28
5. ライブラリ関数 32
  5.1. システム関数 32
    5.1.1. Sys_Initialize 32
    5.1.2. Sys_Terminate 33
    5.1.3. Sys_GetInformation 34
    5.1.4. Sys_GetNumOfCameras 36
    5.1.5. Sys_CreateSignal 37
    5.1.6. Sys_CloseSignal 38
    5.1.7. Sys_WaitForSignal 39
    5.1.8. Sys_ResetSignal 41
  5.2. カメラ関数 42
    5.2.1. Cam_GetInformation 42
    5.2.2. Cam_Open 47
    5.2.3. Cam_OpenFromInfo 50
    5.2.4. Cam_Close 53
    5.2.5. Cam_ReadReg 54
    5.2.6. Cam_WriteReg 56
    5.2.7. Cam_ResetPort 57
    5.2.8. Cam_GetHeartbeat 60
    5.2.9. Cam_SetHeartbeat 61
    5.2.10. Cam_GetMulticast 62
    5.2.11. Cam_SetMulticast 63
  5.3. カメラストリーム関数 65
    5.3.1. 高水準関数 65
    5.3.2. 低水準関数 83
    5.3.3. 共通関数 96
  5.4. カメライベント通知（メッセージ）関数 100
    5.4.1. 高水準関数 100
    5.4.2. 低水準関数 104
    5.4.3. 共通関数 115
  5.5. カメラ制御関数 121
    5.5.1. ImageFormatControl 122

5.5.2. Scalable 124
5.5.3. Binning 134
5.5.4. Decimation 140
5.5.5. Reverse 146
5.5.6. PixelFormat 149
5.5.7. TestPattern 151
5.5.8. AcquisitionControl 153
5.5.9. ImageBuffer 164
5.5.10. TriggerControl 167
5.5.11. ExposureTime 180
5.5.12. DigitalIoControl 186
5.5.13. AntiGlitch / AntiChattering 202
5.5.14. TimerConrtol 207
5.5.15. Gain 213
5.5.16. BlackLevel 217
5.5.17. Gamma 219
5.5.18. WhiteBalance (BalanceRatio / BalanceWhiteAuto) 221
5.5.19. Hue 226
5.5.20. Saturation 228
5.5.21. Sharpness 233
5.5.22. ALCControl 235
5.5.23. ColorCorrectionMatrix 240
5.5.24. LUT Control 244
5.5.25. UserSetControl 248
5.5.26. SequentialShutterControl 254
5.5.27. UserDefinedName（DeviceUserID） 260
5.5.28. Chunk 262
5.5.29. FrameSynchronization 268
5.5.30. その他の関数 270
5.6. GenICam 関数 273
5.6.1. INode 系関数 273
5.6.2. ICategory 型　関数 284
5.6.3. IInteger 型　関数 287
5.6.4. IFloat 型　関数 293
5.6.5. IBoolean 型　関数 302
5.6.6. IEnumeration 型、IEnumEntry 型　関数 305
5.6.7. ICommand 型　関数 316
5.6.8. IString 型　関数 318
5.6.9. Chunk 関数 321
5.6.10. その他の関数 324
5.6.11. 旧 GenICam 関数 325
5.7. ユーティリティ関数 356
5.7.1. 画像フォーマット変換関数 356
5.7.2. その他ユーティリティ 363
5.8. ステータスコード 367
6. サンプルソース 370
6.1. Windows 版のサンプルソース 370
6.2. Linux 版のサンプルソース 371
6.2.1. コンソール サンプル 372
6.2.2. Qt サンプル 372
7. その他 373
7.1. 免責事項 373
7.2. ライセンス 373
7.3. 改定履歴 375
7.4. お問い合わせ 376

1. はじめに

TeliCamSDK は、東芝テリー製 USB3 Vision デジタルカメラシリーズおよび 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。

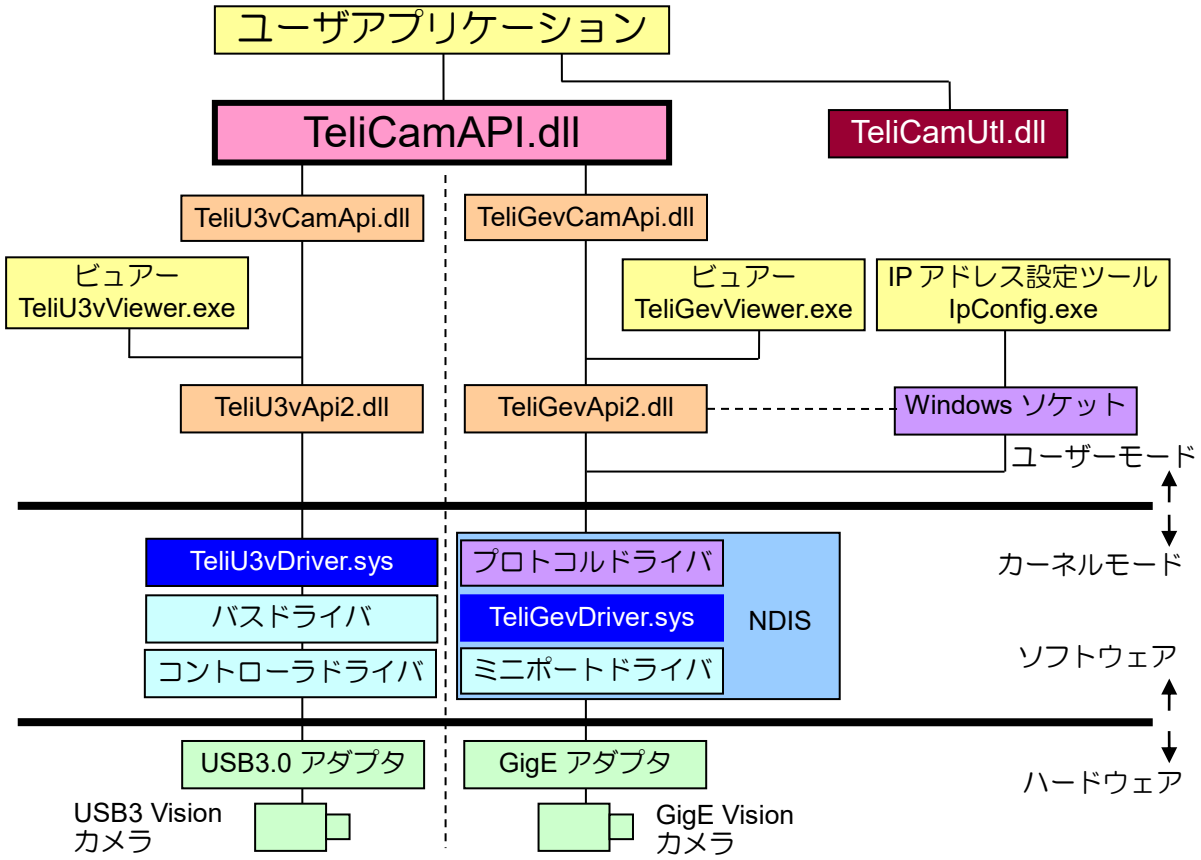
本ドキュメントは、TeliCamSDK パッケージの C++用プログラミングインタフェースである TeliCamAPI の使用方法について記載しています。 TeliCamAPI を使用することにより、カメラのインターフェースを意識することなく簡単にアプリケーションを作成することができます。

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。

2. 構成

2.1. Windows 版の構成

TeliCamSDK のソフトウェア構成は以下の通りです。



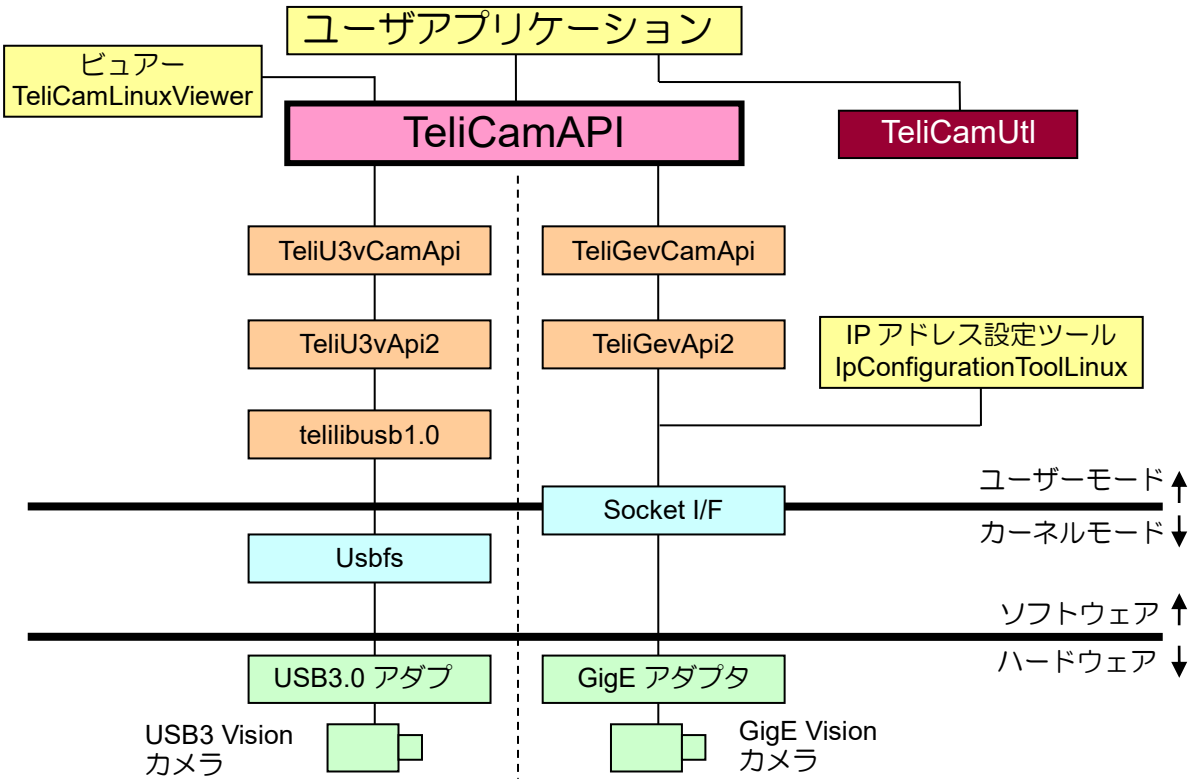
32bit OS 用	64bit OS 用	内　　容
TeliCamApi.dll	TeliCamApi64.dll	ネイティブアプリケーション開発用関数ライブラリ
TeliU3vCamApi.dll	TeliU3vCamApi_64.dll	USB3 Vision カメラ 用拡張関数ライブラリ
TeliU3vApi2.dll	TeliU3vApi2_64.dll	USB3 Vision カメラ用基本関数ライブラリ
TeliU3vDriver.sys	TeliU3vDriver64.sys	USB3 Vision カメラ用デバイスドライバ
TeliGevCamApi.dll	TeliGevCamApi_64.dll	GigE Vision カメラ用拡張関数ライブラリ
TeliGevApi2.dll	TeliGevApi2_64.dll	GigE Vision カメラ用基本関数ライブラリ
TeliGevDriver.sys	TeliGevDriver64.sys	GigE Vision カメラ用デバイスドライバ
TeliCamUtl.dll	TeliCamUtl64.dll	画像ハンドリングユーティリティ関数ライブラリ
TeliU3vViewer.exe	TeliU3vViewer64.exe	USB3 Vision ビュアー（機能と画像の確認用）
TeliGevViewer.exe	TeliGevViewer64.exe	GigE Vision ビュアー（機能と画像の確認用）
IpCnfg.exe	IpCnfg.exe	カメラの IP アドレス設定ツール

GigE Vision カメラと USB3 Vision カメラを使用して、カメラインタフェースを意識することなくコードを組める高水準 API 「TeliCamAPI」を使用してアプリケーションを作成してください。

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。

2.2. Linux 版の構成

TeliCamSDK for Linux 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
TeliGevCamApi (libTeliGevCamApi.so)	GigE Vision カメラ 用拡張関数ライブラリ
TeliGevApi2 (libTeliGevApi2.so)	GigE 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
TeliCamLinuxViewer	ビュアー（機能と画像の確認用）
IpConfigurationToolLinux	カメラの IP アドレス設定ツール

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。

尚、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

3.1. Windows 版のシステム要件

TeliCamAPI を使用するためには以下の表に記載した環境が必要です。
この環境条件はすべてのアプリケーション・使用環境についての動作を保障するものではありません。
使用条件により、更なる高性能 PC が必要となる場合があります。

対応 OS	• Windows 7　32/64bit • Windows 8.1　32/64bit • Windows 10　32/64bit　*5
PC 推奨スペック	• CPU　：　Intel Core2 2.40GHz 以上 • Memory　：　4Gbyte 以上 • Graphics　：　256Mbyte 以上の VRAM 搭載
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 USB3.0 アダプタ。　*1
推奨ネットワークアダプタ	Jumbo Frame（Jumbo Packet）対応（9014byte 以上）Gigabit イーサネットアダプタ。（Intel PRO/1000 シリーズなど）。　*2
必要ソフトウェア	• Microsoft Visual C++ 2013 Redistributable Package • Microsoft Visual C++ 2010 SP1 Redistributable Package • GenICam GenApi reference implementation v.3.0.1　*3,*4 • Microsoft Direct X End-User Runtime（DirectX 9.0c 以降）　*4
対応カメラ	• 東芝テリー製 USB3 Vision デジタルカメラ • 東芝テリー製 GigE Vision デジタルカメラ

注）
*1：　USB3 Vision カメラ使用時に必要です。
*2：　GigE Vision カメラ使用時に必要です。
*3：　GenApi を有効に設定したアプリケーションの開発環境と実行環境で必要です。
*4：　TeliU3vViewer および TeliGevViewer の実行に必要です。
*5：　5M ピクセル以上のカメラを使用する場合または複数台のカメラを使用する場合は、64bit OS での使用を推奨します。

3.2. Linux 版のシステム要件

TeliCamApi for Linux は Intel/AMD x86 および ARM アーキテクチャで動作します。
動作確認済みの OS は以下の通りです。

• Intel/AMD

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

• ARM

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

※1　Ubuntu 14.04 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は Strm_Stop 関数を Strm_Abort 関数に変更して、ご使用してください。
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。

※3　Ubuntu 16.04 から Ubuntu 16.04.5 まで対応。

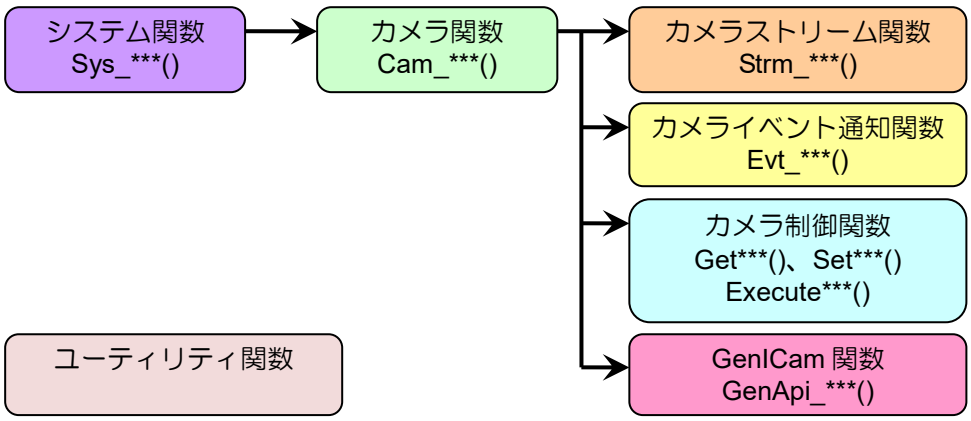
※4　Ubuntu 18.04 から Ubuntu 18.04.1 まで対応。

※5　GigE Vision デジタルカメラで、カメラの最大フレームレートで画像を取得できない場合があります。

※6　USB3 Vision デジタルカメラは使用することができません。

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。
GenICam 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。

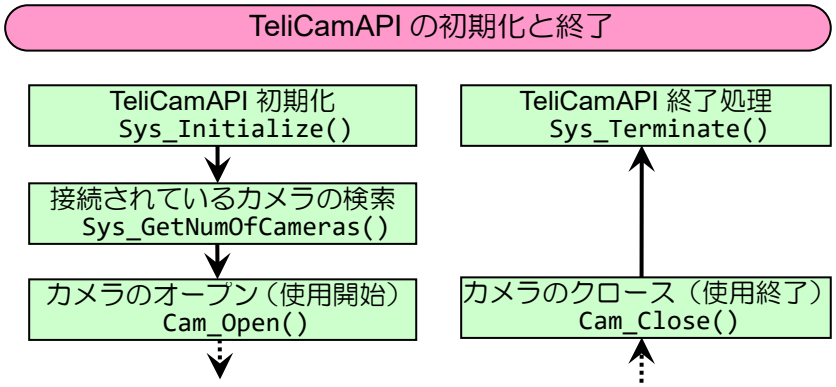
4.1. 使用方法

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、カメライベント通知（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働きをします。

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使用する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。

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。

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。

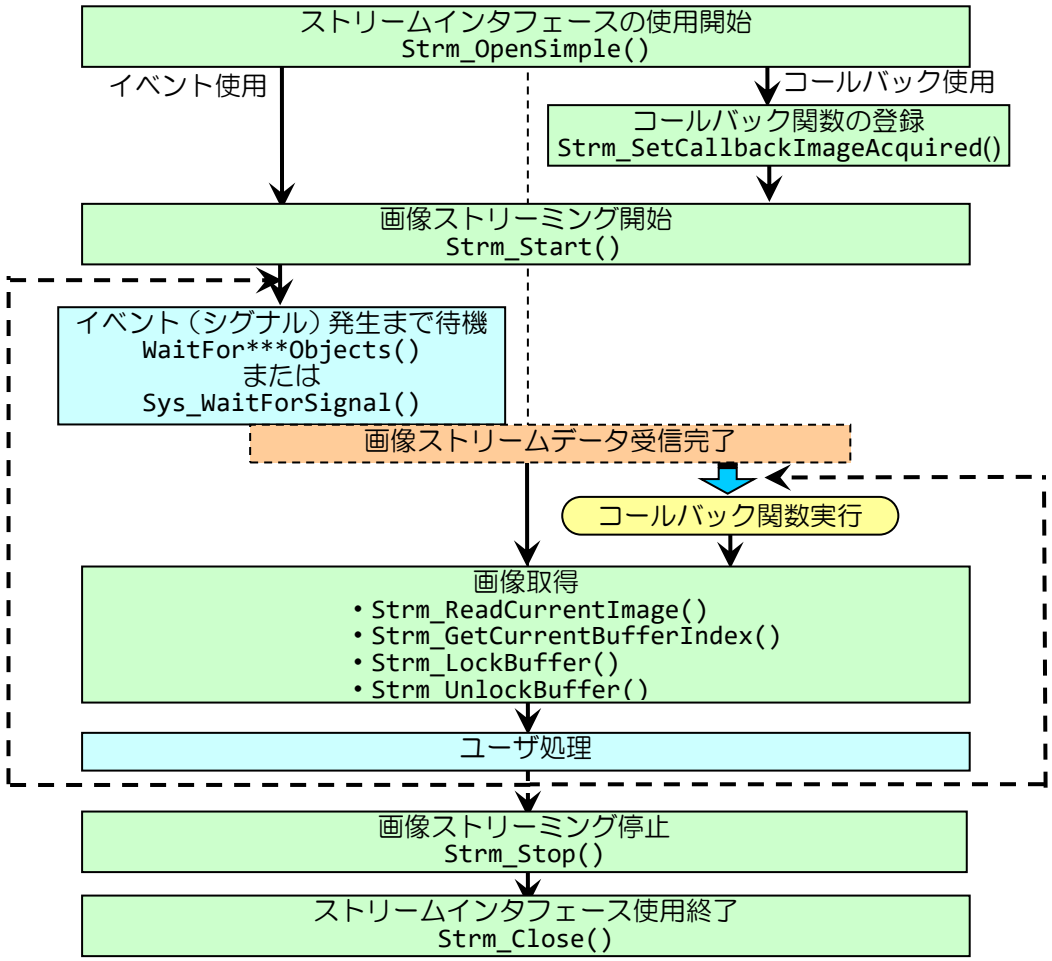
・**イベント（シグナル）オブジェクトを使用した通知**
Windows 版
ストリームオープン時に引数として指定した Windows のイベント（シグナル）オブジェクトを使用して受信完了をユーザアプリケーションに通知する方法です。
ユーザアプリケーションは WaitForSingleObject API などを使用して受信完了を待ち、受信が完了すると、Strm_ReadCurrentImage()などの高水準関数で画像を取得する流れです。

Linux 版
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()関数を使用すると、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。

 www.toshiba-teli.co.jp

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（シグナル）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

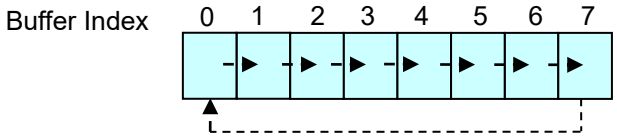
Strm_Start()を実行するとカメラにストリーミング開始が指示され、ストリーミングが開始されます。

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Windowsのイベント（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。

4.1.5.1.4. 画像データの取得

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。

- **Strm_LockBuffer()使用。**
  ユーザアプリケーションがストリームリクエストリングバッファ内の任意の画像にアクセスするときには、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。

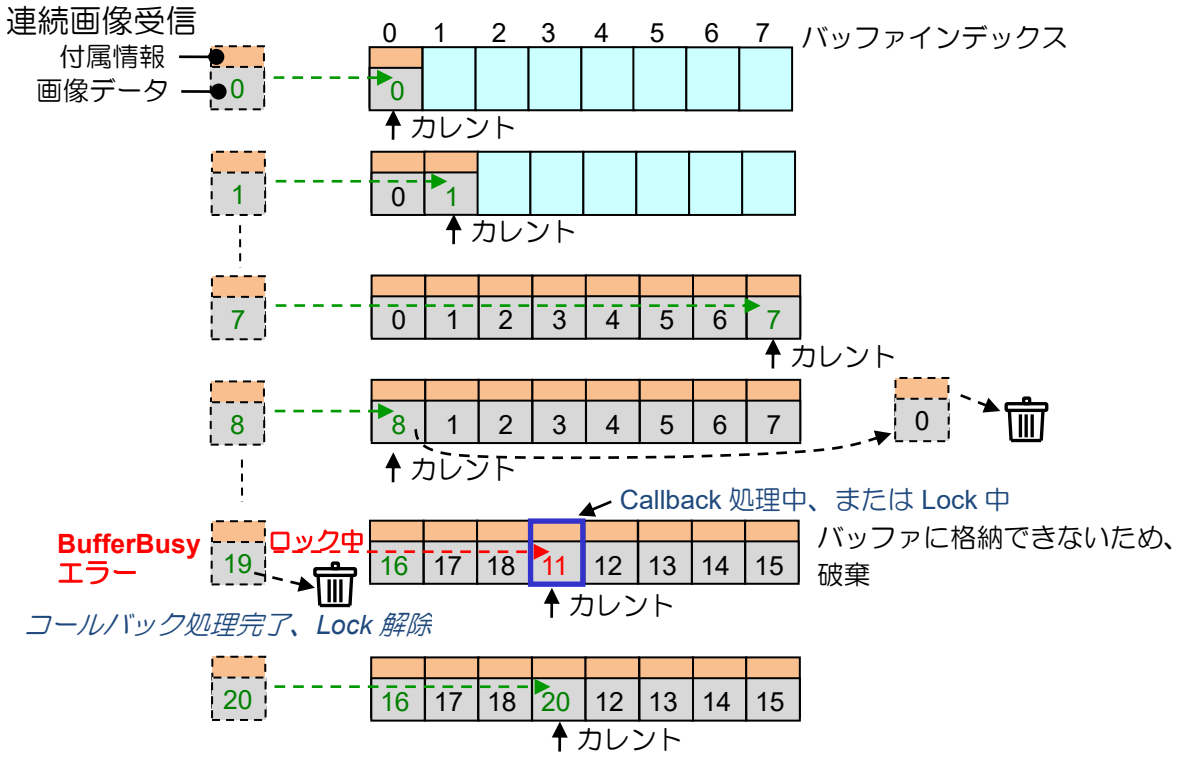
Strm_LockBuffer()を使用すると、ストリームリクエスト リングバッファの指定したインデックスの領域がロックされ、画像データへのポインタがStrm_LockBuffer()引数に設定されます。Strm_GetCurrentBufferIndex() を使用すると最新の画像ストリームデータが格納されたストリームリクエストリングバッファのインデックスが取得できます。

Strm_LockBuffer()を実行した後は、取得したい画像を短時間で処理するか、別バッファにコピーして、速やかにStrm_LockBuffer()を実行してロックを解除してください。長期間バッファをロックし続けるとバッファビジーエラーが発生する可能性があります。

・コールバック使用。

Strm_SetCallbackImageAcquired()を使用してコールバック関数を登録しておくと、画像受信時にコールバック関数が実行されます。コールバック関数が実行されたとき、最新画像へのポインタが引数として渡されます。ユーザアプリケーションはコールバック実行期間中に限り、このポインタを使用して画像データにアクセスすることができます。

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。

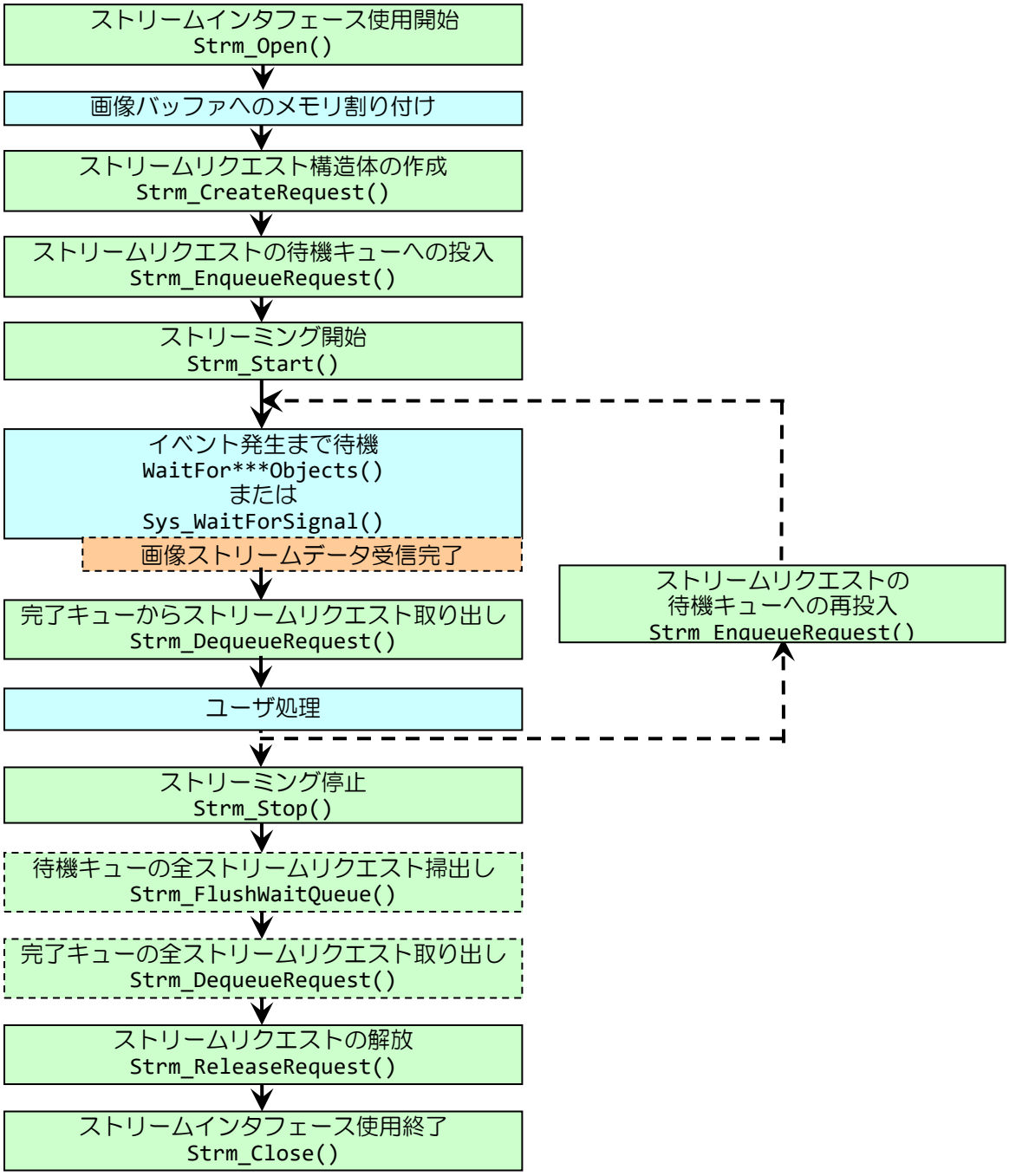
4.1.5.2. 低水準関数を使用したストリームデータ（画像データ）の取得

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イベントを受領するためのイベント（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。

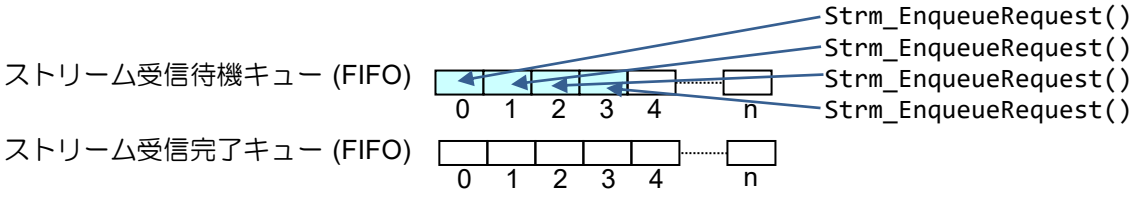
ストリームリクエストは、その画像バッファのサイズ変更、バッファの差し替えをすることはできません。画像データのサイズが変更された場合は、ストリームリクエストを解放して新たにストリームリクエストを作成しなおしてください。

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

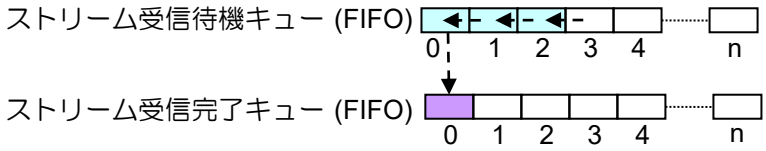
Strm_Start()を実行すると、カメラにストリーミング開始が指示され、ストリーミングが開始されます。

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イベント（シグナル）オブジェクトをシグナル状態にします。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。

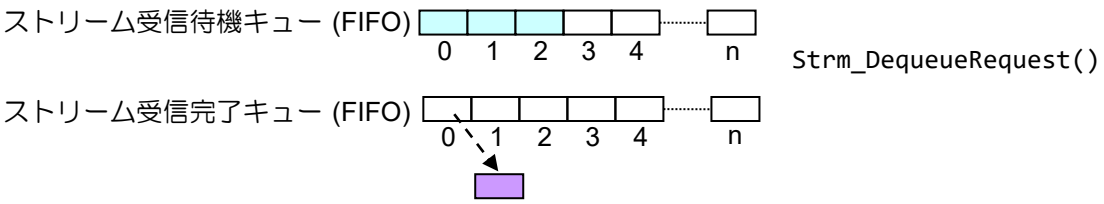
例：1 画像分のストリームデータ（画像データ）受信



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）受信完了のイベント受領後に `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。

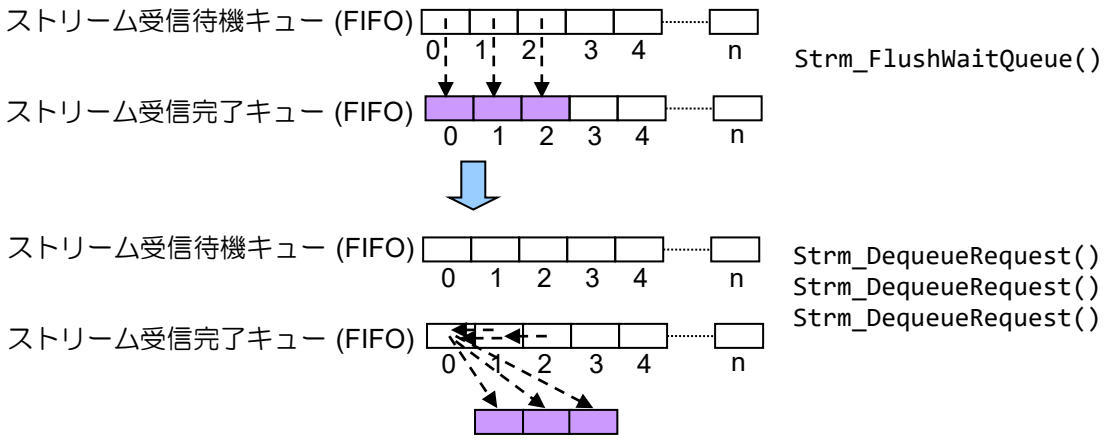
4.1.5.2.9. ストリームリクエストの受信処理中止と完了キューから取り出し

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。

 www.toshiba-teli.co.jp

4.1.6. カメライベント通知（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、カメライベント通知（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、カメライベント通知（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通知（メッセージ）データを取得することができます。

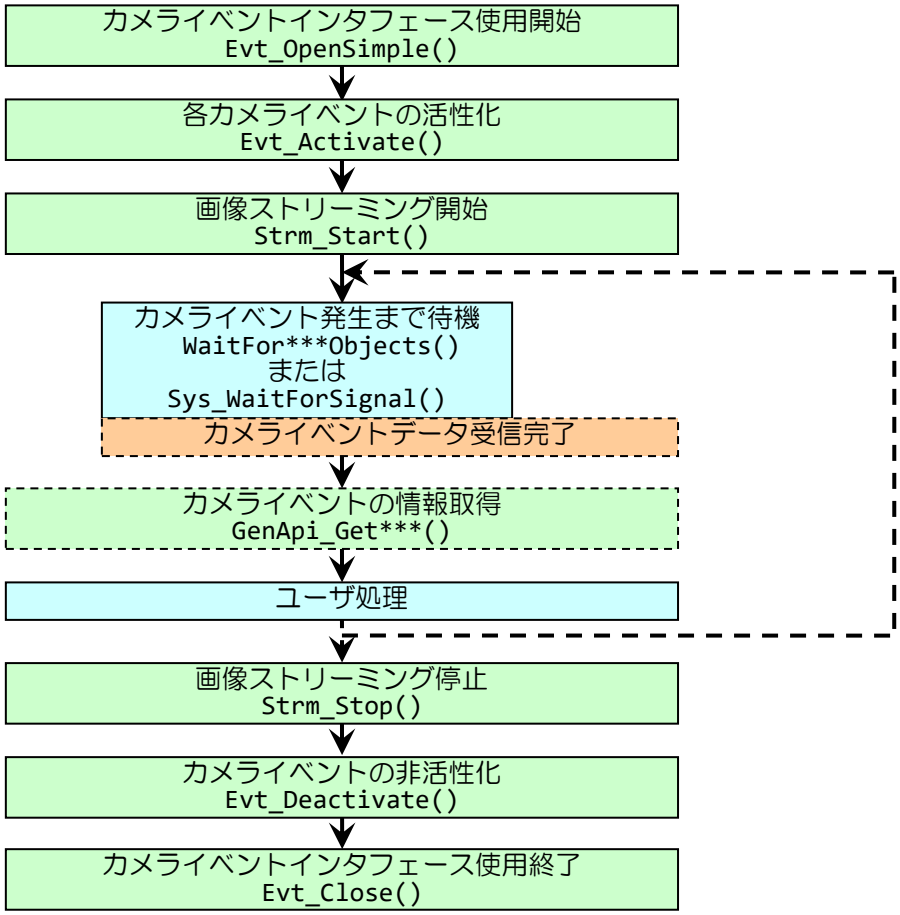
低水準関数は、カメライベントデータの受信処理をユーザに組んでいただくことを前提とした関数群です。ユーザコードの記述量は増えますが、任意のシーケンスの受信処理を組むことができ、カメラのパフォーマンスを最大限に活用し、特殊な受信処理の実装をすることも可能になります。

4.1.6.1. 高水準関数を使用したカメライベント通知（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 Windows のイベント（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. カメライベント通知（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 Windows イベント（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イベント（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、指定されたイベント（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()で指定されたイベント（シグナル）オブジェクトをシグナル状態に設定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. カメライベント通知（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、GenICam 関数 (GenApi_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. カメライベント通知（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

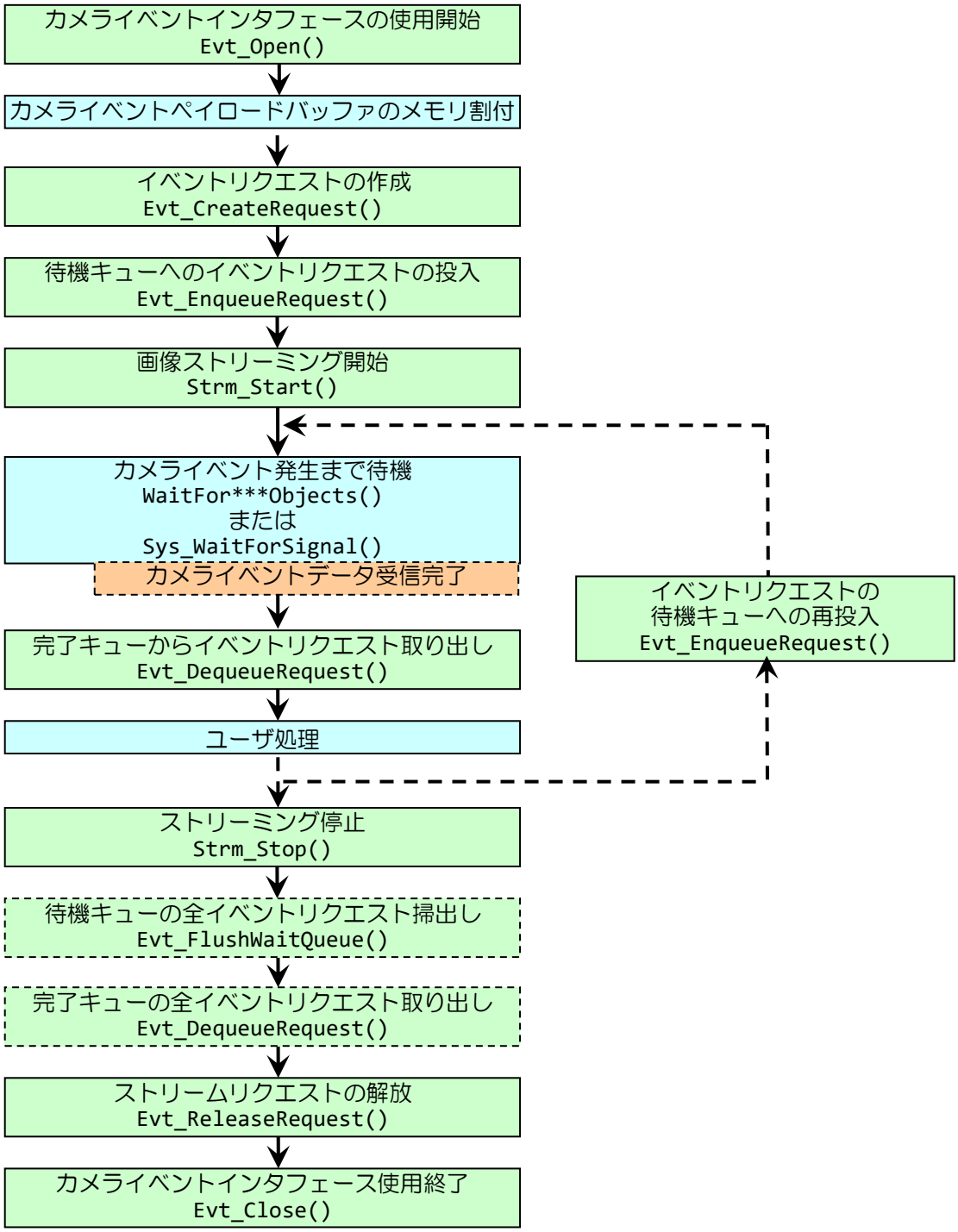
イベントインターフェースの使用を終了するときは、Evt_Close()を実行してください。

4.1.6.2. 低水準関数を使用したカメライベント通知（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。



 www.toshiba-teli.co.jp

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イベント（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. カメライベント通知（メッセージ）ペイロードバッファの確保

TeliCamAPI は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、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. イベント通知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通知受信完了キューに移動させるためにEvt_FlushWaitQueue() を実行。
2. イベント通知受信待機キューが空になるまでEvt_DequeueRequest() を実行して、イベント通知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

カメライベント通知（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通知受信待機キューに追加することにより、カメライベント通知（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。

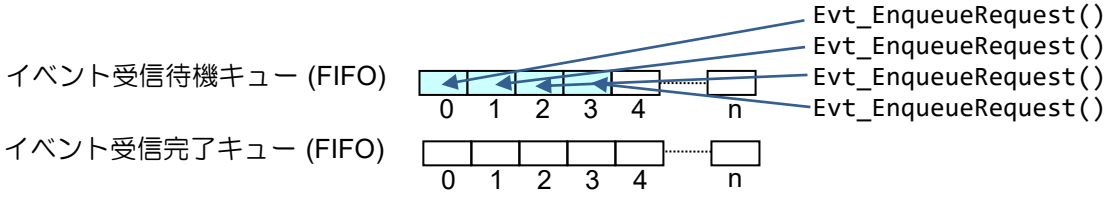
メモリ割り付け済のペイロードバッファを引数に指定して Evt_CreateRequest()関数を実行するとカメライベント受信用のイベントリクエストが１個作成され、そのハンドルが引数に戻されます。

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通知受信待機キューに投入してください。イベント通知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



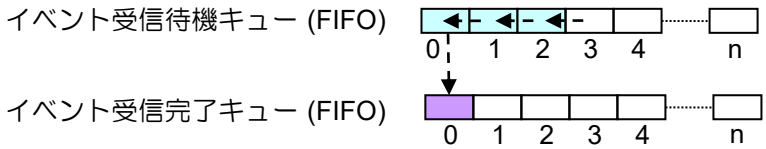
4.1.6.2.5. ストリーミング開始

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. カメライベント通知（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、イベント通知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通知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イベント（シグナル）オブジェクトをシグナル状態にします。これらの処理は自動的に実行されます。

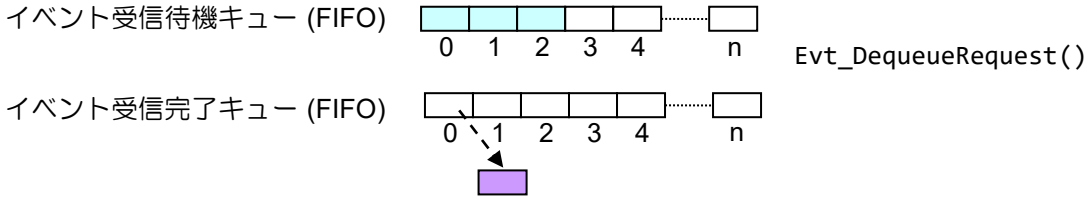
例：1 つのイベント受信



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イベントの受領後に Evt_DequeueRequest() を実行すると、イベント通知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。

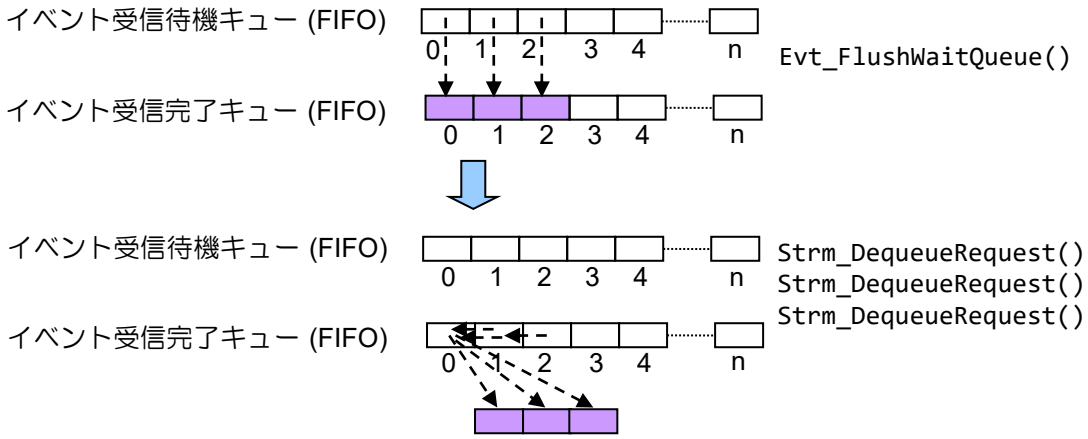
4.1.6.2.9. イベントリクエストの受信処理中止と完了キューから取り出し

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

イベント通知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通知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通知受信完了キューに移動させてください。

イベント通知受信完了キューが空でないときは、イベント通知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通知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 はカメラとの交信頻度を確認し、必要に応じてダミーコマンドを送信して交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交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 カメラでは、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、以下のドキュメントをご覧ください。
これらのドキュメントは、TeliCamSDK インストールフォルダの"Ducuments"フォルダに格納されています。

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

4.3. SDK のアンインストール

TeliCamSDK のアンインストール方法は、4.2 項に記載のドキュメントをご覧ください。

4.4. 開発環境の設定

4.4.1. Windows 版　開発環境の設定

Visual Studio を使用してアプリケーションの開発する場合は、以下のようにプロジェクトを設定してください。（Visual Studio のバージョンにより、設定項目が異なる場合があります。）

・「構成プロパティ」－「C/C++」－「全般」－「追加のインクルードディレクトリ」に以下のディレクトリを追加。

“$(TeliCamSDK)TeliCamAPI\include”

・「構成プロパティ」－「リンカー」－「全般」－「追加のライブラリディレクトリ」に以下のディレクトリを追加。

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

・「構成プロパティ」－「リンカー」－「入力」－「追加の依存ファイル」に以下のファイルを追加。

32bit OS の場合　：　“TeliCamAPI.lib;TeliCamUtl.lib”
64bit OS の場合　：　“TeliCamAPI64.lib;TeliCamUtl64.lib”

4.4.2. Linux 版　開発環境の設定

TeliCamSDK for Linux 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
```

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. Linux 版におけるパフォーマンスのチューニング

TeliCamSDK for Linux は、API 内部のパケット受信スレッドの優先順位を上げること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。

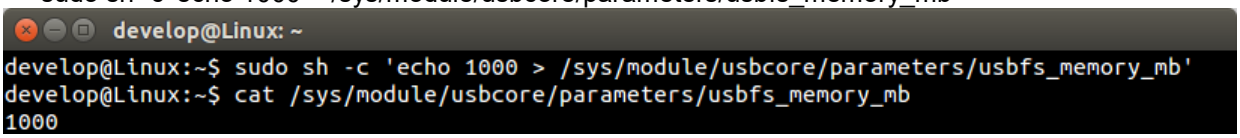
パフォーマンスを要求するアプリケーションの場合は、以下のいずれかの方法により優先順位を変更できるようにしてください。
・アプリケーションをルート権限で実行する。
・リアルタイム・プロセスの優先順位を変更できるようにシステム構成を変更する。
Pluggable Authentication Modules (PAM) for Linux の pam_limits モジュールを使用すると、制限構成ファイルでシステム・リソースでの制限を構成できます。
デフォルトの制限は、/etc/security/limits.conf ファイル で設定されます。
例えば、
* - rtprio 99
と指定すると、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。

・USB3 Vision カメラを複数台使用する、または CAM_API_STS_IO_DEVICE_ERROR が発生した場合は usbfs のメモリ制限を変更してください。
設定方法は以下となります。

1. ターミナル（gnome-terminal）を起動します。
2. デフォルトの値を確認します。
cat /sys/module/usbcore/parameters/usbfs_memory_mb

3. メモリ制限の変更します。
例：メモリ制限を 1000[MB]に変更する場合
sudo sh -c 'echo 1000 > /sys/module/usbcore/parameters/usbfs_memory_mb'



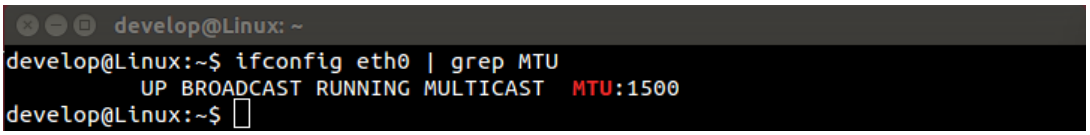
または
/sys/module/usbcore/parameters/内の usbfs_memory_mb を直接エディタで編集してください。

再起動を行うと変更した値はデフォルトの 16[MB]に戻るため、再度設定を行ってください。

・GigE Vision カメラを使用中に CAM_API_STS_TOO_MANY_PACKET_MISSING エラー（エラーコード：0x100C) が発生する場合、ジャンボフレーム設定値、または受信バッファサイズ、もしくは Network Interface Cards (NICs) のパケット設定が不足しているか、HUB の QoS 設定が適切でない可能性があります。
以下の方法で設定を行い、再度お試しください。

・ジャンボフレーム設定
ifconfig コマンドを使用する場合：

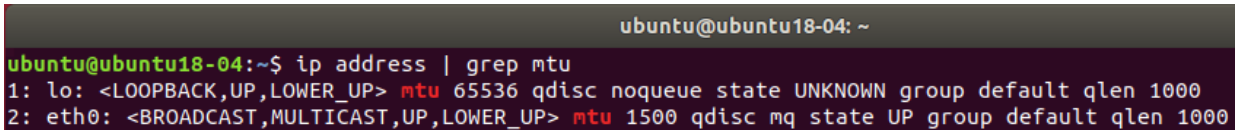
1. ターミナル（gnome-terminal）を起動します。
2. デフォルトの設定値を確認します。
   ifconfig [interface] | grep MTU



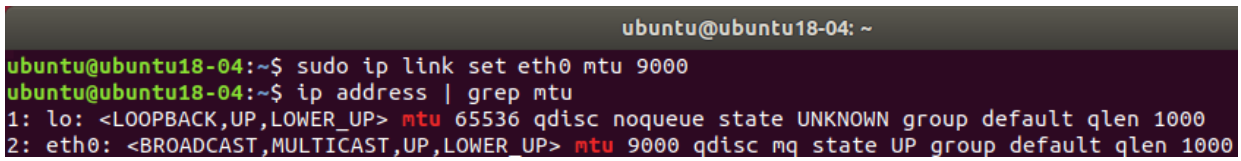
3. ジャンボフレームの設定を変更します。
   例: sudo ifconfig [interface] mtu 9000

ip コマンドを使用する場合：

1. ターミナル（gnome-terminal）を起動します。
2. デフォルトの設定値を確認します。
   ip address | grep mtu



3. ジャンボフレームの設定を変更します。
   例: sudo ip link set [interface] mtu 9000

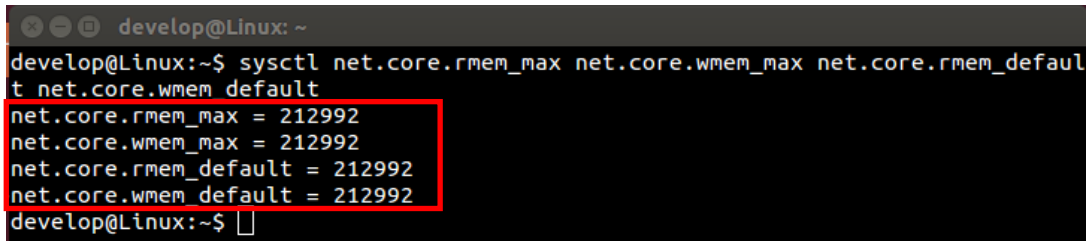


 www.toshiba-teli.co.jp

• 最大 UDP 受信バッファサイズ設定

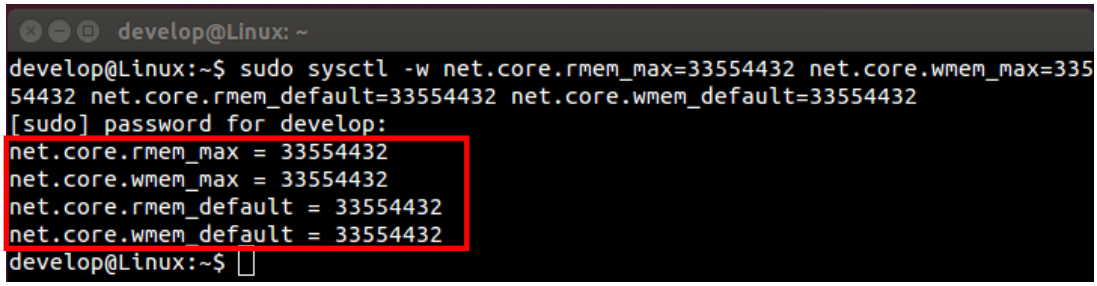
1. ターミナル（gnome-terminal）を起動します。
2. デフォルトの設定値を確認します。

sysctl net.core.rmem_max net.core.wmem_max net.core.rmem_default net.core.wmem_default



3. バッファサイズの設定を変更します。

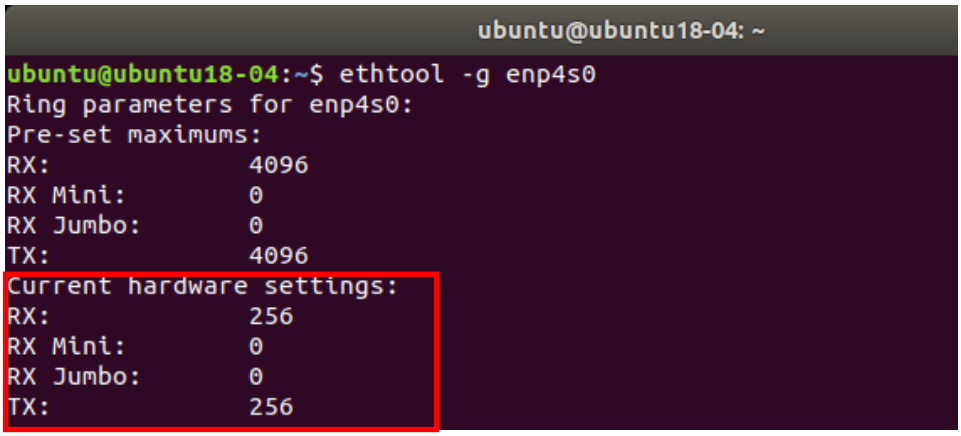
例: sysctl -w net.core.rmem_max=33554432 net.core.wmem_max=33554432
net.core.rmem_default=33554432 net.core.wmem_default=33554432



• Network Interface Cards (NICs) のパケット設定

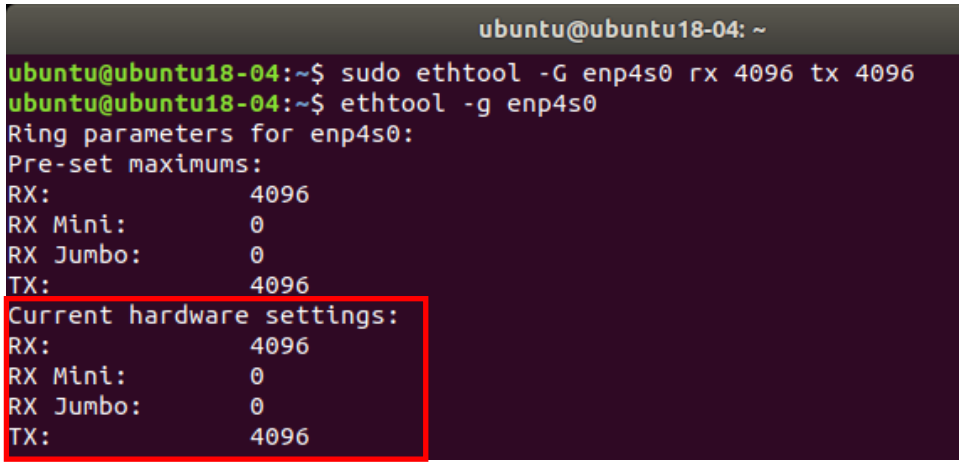
1. ターミナル(gnome-terminal)を起動します。
2. デフォルトの設定値を確認します。

   例：ethtool -g [interface]



3. RX(受信)および TX(送信)の設定を変更します。

   例：ethtool -G [interface] rx 4096 tx 4096



• HUB の QoS 設定

ご使用している HUB ポートの優先度を最高値に設定してください。
(設定方法はご使用中の HUB のマニュアルをご覧ください。またご使用の HUB によっては、QoS の設定ができない場合があります。)

 www.toshiba-teli.co.jp

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* 列挙子]

メンバ	内容
CAM_TYPE_UNKNOWN	不明なタイプ。 Sys_Initialize() 関数では指定しないでください。
CAM_TYPE_U3V	USB3 Vision カメラ。
CAM_TYPE_GEV	GigE Vision カメラ。
CAM_TYPE_ALL	すべてのタイプ。

[戻り値]

実行結果を返します。　戻り値は、5.8. 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 dll 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 dll 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。

[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、5.8. 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
psSysInfo	[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;
```

メンバ		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;
```

メンバ		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;
```

メンバ		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、5.8. 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.sys  version : %s\n", sSysInfo.sU3vInfo.szDriverVersion);
printf("  TeliU3vApi2.dll    version : %s\n", sSysInfo.sU3vInfo.szDllVersion);
printf("  TeliU3vCamApi.dll  version : %s\n", sSysInfo.sU3vInfo.szDllExVersion);
printf("  TeliGevDriver.sys  version : %s\n", sSysInfo.sGevInfo.szDriverVersion);
printf("  TeliGevApi2.dll    version : %s\n", sSysInfo.sGevInfo.szDllVersion);
printf("  TeliGevCamApi.dll  version : %s\n", sSysInfo.sGevInfo.szDllExVersion);
printf("  TeliCamAPI.dll     version : %s\n", sSysInfo.szDllVersion);

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

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

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

5.1.4. Sys_GetNumOfCameras

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
puiNum	[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、5.8. ステータスコード を参照してください。

[備考]

検出されたカメラには `0` から (`*puiNum-1`) までの値のインデックスカメラ特定用に付与されます。

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]

For Windows

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

For Linux

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

[パラメータ]

パラメータ		内　容
phHandle	[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、5.8. 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。

Windows 版では、CreateEvent() を使用してハンドルを作成しても構いません。

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]

For Windows

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

For Linux

```
CAM_API_STATUS  Sys_CloseSignal (
  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、5.8. 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。

Windows 版では、CloseHandle() を使用してシグナルオブジェクトを閉じても構いません。

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か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]

For Windows

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

For Linux

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

[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。
uiMilliseconds	[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。

Windows 版では、WaitForSingleObject() を使用してシグナルオブジェクトの状態を確認しても構いません。

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]

For Windows

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

For Linux

```
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、5.8. 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。

Windows 版では、ResetEvent() を使用してシグナルオブジェクトを非シグナル状態にリセットしても構いません。

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
hCam	[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。
uiCamIdx	[in]	0 で始まるカメラのインデックスです。 hCam 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。
psCamInfo	[out]	カメラ情報を格納する CAM_INFO 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;
```

メンバ		内　容
eCamType	[out]	カメラのタイプです。
szManufacturer	[out]	メーカー名です。
szModelName	[out]	モデル名です。
szSerialNumber	[out]	シリアル番号です。
szUserDefinedName	[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。
sU3vCamInfo	[out]	USB3 Vision カメラに関する情報です。
sGevCamInfo	[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;
```

メンバ		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。 Strm_Open() 、Strm_OpenSimple()で *uiMaxPacketSize* 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;
```

メンバ		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 以外のとき対応です。
cSupportIP_DHCP	[out]	カメラの DHCP 対応状況です。 0 の時非対応、0 以外のとき対応です。
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 以外のとき対応です。
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 以外の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 以外の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 以外の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、5.8. 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。
Windows 版のみ、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。

[構文]

For Windows

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

For Linux

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

[パラメータ]

パラメータ		内　容
uiCamIdx	[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、Sys_GetNumOfCameras() で取得したカメラ台数-1 です。
phCam	[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。
hRmv	[in]	カメラの取り外し通知用のイベント（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なります。 Windows 版のイベント（シグナル）オブジェクトは、Sys_CreateSignal() または Win32API の CreateEvent() で作成します。 Linux 版のイベント（シグナル）オブジェクトは、Sys_CreateSignal() 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。
bUseGenICam	[in]	GenICam アクセスの有効／無効です。

パラメータ	内　容
	`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。
pvXml [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。
eAccessMode [in]	カメラのアクセスモード（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。

[*CAM_ACCESS_MODE* 列挙子]

メンバ	内容
`CAM_ACCESS_MODE_EXCLUSIVE`	特別アクセスモード。 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`	コントロールアクセスモード。 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`	オープンアクセスモード。 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、5.8. 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。ハード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（カメラとの交信が継続していることの確認。一定時間交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、イベント（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するときは、XML ファイルのロード処理に時間がかかる場合があります。

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、カメライベント高水準関数、一部の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 was successful.\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();
```

5.2.3. 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。
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。

[構文]

For Windows

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

For Linux

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

[パラメータ]

パラメータ		内　容
pszSerialNo	[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。
pszModelName	[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。
pszUserDefinedName	[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。
phCam	[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。

パラメータ		内　容
hRmv	[in]	カメラの取り外しを通知するイベント（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なります。 Windows 版のイベント（シグナル）オブジェクトは、 Sys_CreateSignal() または Win32API の CreateEvent() で作成します。 Linux 版のイベント（シグナル）オブジェクトは、 Sys_CreateSignal() 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。
bUseGenICam	[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。
pvXml	[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。
eAccessMode	[in]	カメラのアクセスモード（制御チャネル特権）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。

[戻り値]

実行結果を返します。　戻り値は、5.8. 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。ハード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（カメラとの交信が継続していることの確認。一定時間交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、イベント（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するときは、XML ファイルのロード処理に時間がかかる場合があります。

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、カメライベント高水準関数、一部の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", NULL, &hCam);
if (uiStatus != CAM_API_STS_SUCCESS)
    return -1;

printf("Cam_OpenFromInfo was successful.\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、5.8. 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
hCam	[in]	カメラのカメラハンドルです。
ullAdrs	[in]	取得するレジスタの開始アドレスです。
uiSizeQuadlet	[in]	取得するデータサイズです。（Quadlet 単位。1～）
pvData	[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、5.8. 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;
bool8_t          bSupportIIDC2;
CAM_HANDLE       hCam;

// Initialize system.
```

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

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

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

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

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

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

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

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

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

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

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

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


5.2.6. Cam_WriteReg

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
hCam	[in]	カメラのカメラハンドルです。
ullAdrs	[in]	書き込むレジスタの開始アドレスです。
uiSizeQuadlet	[in]	書き込むデータサイズです。（Quadlet 単位。1～）
pvData	[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、5.8. 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）単位ごとに 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、5.8. 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検出イベント *hRmv* がシグナル状態になります。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通知（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;
CAM_HANDLE       hCam = (CAM_HANDLE)NULL;
SIGNAL_HANDLE    hRemoveEvt = (SIGNAL_HANDLE)NULL;

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

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

```
uiNumOld = uiNum;

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

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

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

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

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

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

    // Re-open

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

        if (uiNum == uiNumOld)
            break;

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

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

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

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


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

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

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

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


5.2.8. Cam_GetHeartbeat

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
hCam	[in]	カメラのカメラハンドルです。
pbEnable	[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。
puiHbTimeout	[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、5.8. 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
hCam	[in]	カメラのカメラハンドルです。
bEnable	[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。
uiHbTimeout	[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、5.8. 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。

尚、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。

5.2.10.Cam_GetMulticast

カメラのマルチキャスト設定を取得します。

[構文]

```
CAM_API_STATUS  Cam_GetMulticast (
  CAM_HANDLE        hCam,
  bool8_t           *pbEnable,
  uint32_t          *puiMulticastIP,
  uint16_t          *pushSCP,
  uint16_t          *pushMCP
);
```

[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。
pbEnable	[out]	マルチキャスト設定（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。
puiMulticastIP	[out]	マルチキャスト IP アドレスを格納する変数へのポインタです。 例えば、読み出した値が 0x0A0101EF のときマルチキャスト IP アドレスは「239.1.1.10」になります。
pushSCP	[out]	マルチキャストで使用するストリームチャンネル（ストリームインターフェース）のポート番号を格納する変数へのポインタです。 取得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。
pushMCP	[out]	マルチキャストで使用するメッセージチャンネル（イベントインターフェース）のポート番号を格納する変数へのポインタです。 取得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。

[戻り値]

実行結果を返します。　戻り値は、5.8. 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。

TeliCamAPI.h をインクルードする必要があります。

5.2.11.Cam_SetMulticast

カメラのマルチキャスト設定を変更します。

[構文]

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

[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。
bEnable	[in]	マルチキャストの設定（有効／無効） です。 true のとき有効、false のとき無効です。 デフォルトは false になっています。 true が指定された場合、*uiMulticastIP* で指定されたマルチキャストグループに参加します。
uiMulticastIP	[in]	マルチキャスト IP アドレスです。 マルチキャスト IP アドレスは Class D アドレス（224.0.0.0 ～ 239.255.255.255）を指定する必要があります。 オープンアクセスモード（Open 特権）でカメラをオープンしている場合、0 を指定することができます。 0 が指定された場合、API はカメラに設定されている SCDA（Stream Channel Destination Address）レジスタ を読み出し、その値をマルチキャスト IP アドレスとして設定します。 0x0A0101EF を指定したとき、マルチキャスト IP アドレスは「239.1.1.10」になります。
ushSCP	[in]	マルチキャストで使用するストリームチャンネル（ストリームインターフェース）のポート番号です。 0 を設定すると、API が適切な値を設定します。 特別な理由がない場合は、0 を指定してください。 省略した場合は、0 が指定されます。
ushMCP	[in]	マルチキャストで使用するメッセージチャンネル（イベントインターフェース）のポート番号です。 0 を設定すると、API が適切な値を設定します。 特別な理由がない場合およびカメライベント通知（メッセージ）を使用しない場合は、0 を指定してください。 省略した場合は、0 が指定されます。

[戻り値]

実行結果を返します。　戻り値は、5.8. ステータスコード を参照してください。

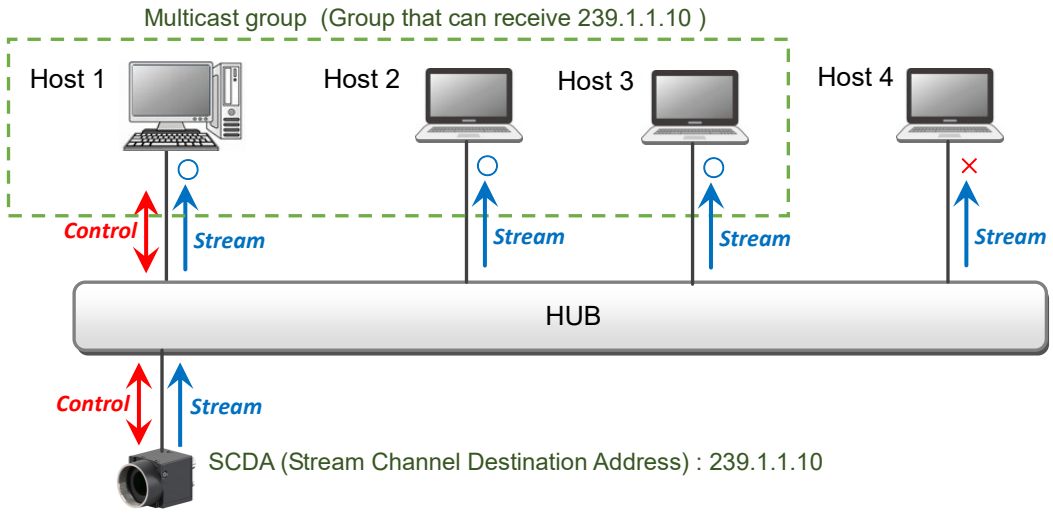
[備考]

本関数は、GigE Vision カメラのみ有効です。

ストリームをオープンする前に本館数を実行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。

1 台のカメラの映像信号を 2 台以上のホスト（PC）で取得する場合には、マルチキャストを使用する必要があります。
カメラをコントロールするホスト（PC）は 1 台のみとなり、コントロールアクセスモード（Control 特権）でカメラオープンする必要があります。
モニターとして使用するホスト（PC）はオープンアクセスモード（Open 特権）でカメラオープンする必要があります。



カメラをコントロールするホスト（PC）がストリームインターフェースをオープンする前に、モニターとして使用するホスト（PC）がストリームインターフェースをオープンする場合は、それぞれのアプリケーションは *uiMulticastIP* 、*ushSCP* および *ushMCP* に有効な値を指定する必要があります。

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。

Windows 版は、複数のアプリケーションが同一のカメラを同時にオープンすることができますが、ストリームインターフェースは複数のアプリケーションの同時使用はできません。

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]

For Windows

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

For Linux

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


[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。
phStrm	[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。
puiMaxPayloadSize	[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。
hCmpEvt	[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イベント（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。 Windows 版のイベント（シグナル）オブジェクトは、 Sys_CreateSignal() または Win32API の CreateEvent() で作成します。 Linux 版のイベント（シグナル）オブジェクトは、 Sys_CreateSignal() 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。
uiApiBufferCount	[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、1 から 128 です。 省略した場合は、DEFAULT_API_BUFFER_CNT (8) が指定されます。
uiMaxPacketSize	[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、5.8. ステータスコード を参照してください。

[備考]

他のアプリケーションが使用しているストリームを開こうとした場合、ストリームのオープンに失敗し、本関数は CAM_API_STS_NOT_AVAILABLE を返します。

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、画像受信イベント *hCmpEvt* をシグナル状態に設定し、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、画像受信イベント *hCmpEvt* の状態値変化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画像受信イベント *hCmpEvt* がリセットされる前に受信した画像に対して、画像受信イベント *hCmpEvt* を再度シグナル状態に設定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;
CAM_HANDLE           hCam = (CAM_HANDLE)NULL;
CAM_STRM_HANDLE      hStrm = (CAM_STRM_HANDLE)NULL;
SIGNAL_HANDLE        hStrmCmpEvt = (SIGNAL_HANDLE)NULL;
void*                pvPayloadBuf = NULL;
CAM_IMAGE_INFO       sImageInfo;


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

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

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

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

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

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

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

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

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

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



```
            break;

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

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

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

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

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

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

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

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

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

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

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

5.3.1.2. Strm_ReadCurrentImage

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
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。
pvBuf	[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。
puiSize	[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）
psImageInfo	[out]	画像付随情報を格納する CAM_IMAGE_INFO 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;
```

メンバ		内　容
ullTimestamp	[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 1 ナノ秒 です。 GigE Vision Camera のとき、単位は 16 ナノ秒 です。
uiPixelFormat	[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。
uiSizeX	[out]	カメラが出力する映像データの水平有効画素数です。
uiSizeY	[out]	カメラが出力する映像データの垂直有効画素数です。
uiOffsetX	[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。
uiOffsetY	[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。

メンバ		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、5.8. 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
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。
puiBufferIndex	[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 Strm_OpenSimple() の *uiApiBufferCount 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、5.8. 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++

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


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

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

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

do
```

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

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

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

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

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

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

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

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

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

    pvPayloadBuf = sImageInfo.pvBuf;

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

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

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

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

    // Close stream interface.
    uiStatus = Strm_Close(hStrm);
```

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

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

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

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


5.3.1.4. Strm_LockBuffer

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
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 Strm_OpenSimple()の *uiApiBufferCount 引数*で指定した値 - 1 までの値が指定できます。
psImageInfo	[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CAM_IMAGE_INFO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。

[戻り値]

実行結果を返します。　戻り値は、5.8. 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_UnlockBuffer (
  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 Strm_OpenSimple()の *uiApiBufferCount 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、5.8. 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
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。
pvContext	[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。
pFunc	[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
hRcvCam	[out]	受信画像を送信したカメラのカメラハンドルです。
hRcvStrm	[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。
psImageInfo	[out]	カメラから受信した画像とその付随情報です。 CAM_IMAGE_INFO 構造体の説明は Strm_ReadCurrentImage() をご覧ください。
uiBufferIndex	[out]	psImageInfo 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。
pvRcvContext	[out]	*pvContext* 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、5.8. 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 CALLBACK CallbackImageAcquired(
    CAM_HANDLE       hRcvCam,
    CAM_STRM_HANDLE  hRcvStrm,
    CAM_IMAGE_INFO   *psImageInfo,
    uint32_t         uiBufferIndex,
    void             *pvContext)
{
    void             *pImageBuf = NULL;
    uint32_t         uiSize, uiAve, i;

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

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

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

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

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

uint32_t OpenStream()
{
    CAM_API_STATUS   uiStatus;
    uint32_t         uiPyldSize;

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

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

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

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

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

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

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

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

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

    return 0;
}
```

5.3.1.7. Strm_SetCallbackImageError

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
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。
pvContext	[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。
pFunc	[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
hRcvCam	[out]	受信画像を送信したカメラのカメラハンドルです。
hRcvStrm	[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。
uiErrorStatus	[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。
uiBufferIndex	[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。
pvRcvContext	[out]	*pvContext* 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、5.8. 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]

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
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。
pvContext	[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。
pFunc	[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
hRcvCam	[out]	受信画像を送信したカメラのカメラハンドルです。
hRcvStrm	[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。
uiBufferIndex	[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。
pvRcvContext	[out]	*pvContext* 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、5.8. 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]

For Windows

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

For Linux

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

[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。
hCmpEvt	[in]	ストリームを受信し、ストリームリクエストが完了キューに移動されたことを通知するイベント（シグナル）オブジェクトのハンドルです。 Windows 版のイベント（シグナル）オブジェクトは、Sys_CreateSignal() または Win32API の CreateEvent() で作成します。 Linux 版のイベント（シグナル）オブジェクトは、Sys_CreateSignal() 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。
puiMaxPayloadSize	[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

パラメータ	内　容
	わります。 0 を指定した場合は、GetCamStreamPayloadSize()で得られる値が使用されます。 通常は、0 または GetCamStreamPayloadSize() で取得した値を指定してください。
phStrm [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。
uiMaxPacketSize [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、5.8. ステータスコード を参照してください。

[備考]

他のアプリケーションが使用しているストリームを開こうとした場合、ストリームのオープンに失敗し、本関数は CAM_API_STS_ALREADY_OPENED を返します。

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、画像受信イベント *hCmpEvt* をシグナル状態に設定します。

TeliCamAPI.h をインクルードする必要があります。

[コード例]

C++

```
const int STRM_REQUEST_NUM = 5;

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


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

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

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

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

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

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

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

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

        if (uiStatus != CAM_API_STS_SUCCESS)
            break;

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



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

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

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

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

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

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

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

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

        // TODO: add your handling code here.

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

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

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

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

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

```
        if (uiStatus == CAM_API_STS_EMPTY_COMPLETE_QUEUE)
            break;
    }

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

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

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

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

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

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

5.3.2.2. Strm_CreateRequest

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                 uiPayloaSize,
  CAM_STRM_REQUEST_HANDLE  *phStrmRequest
);
```

[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。
pvPayloadBuf	[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、uiPayloaSize 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 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、Strm_Open() をコールした時に *puiMaxPayloadSize* に指定（取得）した値と同じ値を指定してください。
phStrmRequest	[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、5.8. 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
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。
hStrmRequest	[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、5.8. 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
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。
hStrmRequest	[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、5.8. 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
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。
phStrmRequest	[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。
ppvPayloadBuf	[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。
puiPayloadSize	[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、5.8. 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
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。

[戻り値]

実行結果を返します。　戻り値は、5.8. 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
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。
hStrmRequest	[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。
psStrmReqInfo	[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;
```

メンバ		内　容
sU3vInfo	[out]	USB3 Vision Camera のとき、ストリームリクエストの情報です。
sGevInfo	[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;
```

メンバ		内　容
pvLeader	[out]	ストリームのリーダへのポインタです。
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。
uiPayloadSize	[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;
```

メンバ		内　容
pvLeader	[out]	ストリームのリーダへのポインタです。
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、5.8. 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、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
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。

[戻り値]

実行結果を返します。　戻り値は、5.8. 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
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。
eAcqMode	[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 列挙子]

メンバ	内容
CAM_ACQ_MODE_CONTINUOUS （連続映像ストリーム 転送モード）	Strm_Stop()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。通常は、このモードが使用されます。 カメラの AcquisitionMode レジスタに ”Continuous” を設定します。
CAM_ACQ_MODE_SINGLE_FRAME （シングルフレーム映像 ストリーム転送モード）	1 枚の画像を撮像・転送するモードです。 本関数を実行すると AcquisitionStart コマンドが実行され、画像が取得されます。 新しい画像を取得するときは、本関数または ExecuteCamAcquisitionStart() を実行してください。 本関数を実行すると AcquisitionFrameCount レジスタの値は 1 に書き換わります。
CAM_ACQ_MODE_MULTI_FRAME （マルチフレーム映像 ストリーム転送モード）	SetCamAcquisitionFrameCount()で設定された枚数の画像を撮像・転送するモードです。 本関数を実行すると AcquisitionStart コマンドが実行され、画像が取得されます。 新しい画像を取得するときは、本関数または ExecuteCamAcquisitionStart() を実行してください。
CAM_ACQ_MODE_IMAGE_BUFFER_READ （カメライメージバッファ 転送モード）	Strm_Stop()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。 ExecuteCamImageBufferRead()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 5.5.9.ImageBuffer を参照してください。

[戻り値]

実行結果を返します。　戻り値は、5.8. ステータスコード を参照してください。

[備考]

使用可能な画像取得モードに関しては使用するカメラの取扱説明書で確認してください。

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、それ以外の場合は 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 TLParamsLocked 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

画像ストリームの転送停止をカメラに要求します。
この関数が実行されると、カメラに AcquisitionStop コマンドが発行されます。

画像データ転送途中の場合、そのフレームを転送し終わってから画像取り込みを停止します。

[構文]

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

[パラメータ]

パラメータ		内　容
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。

[戻り値]

実行結果を返します。　戻り値は、5.8. 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 TLParamsLocked が 0 に設定されます。

Linux 版では、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は Strm_Stop 関数を Strm_Abort 関数に変更して、ご使用してください。

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。

5.3.3.4. Strm_Abort

画像ストリームの転送中断をカメラに要求します。
この関数が実行されると、カメラに AcquisitionAbort コマンドが発行されます。

この関数が実行されると、画像データの転送は即座に中断されます。
このため、転送中の画像データはフレームを完了することなく取り込みが終了します。

[構文]

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

[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。

[戻り値]

実行結果を返します。　戻り値は、5.8. ステータスコード を参照してください。

[備考]

中断されたフレームにはエラーのステータスが付与されます。

フレームが完了せずに中断された場合は、`Strm_SetCallbackImageError()` により登録されたコールバック関数がコールされます。

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 TLParamsLocked 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。

5.4. カメライベント通知（メッセージ）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.カメライベント通知（メッセージ）コントロールを参照してください。

カメライベント通知を受信した順序で取得する必要がある場合、またはフリーランモードやバルクトリガモードで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、低水準関数の使用を推奨します。

Windows 版は、複数のアプリケーションが同一のカメラを同時にオープンすることはできますが、イベントインターフェースを複数のアプリケーションで同時にオープンすることはできません。

5.4.1. 高水準関数

高水準関数はカメライベント通知受信処理の大半を 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。

ただし、ユーザアプリケーションが `WaitForSingleObject() / Sys_WaitForSignal()` や `ResetEvent() / Sys_ResetSignal()` などを使用してイベント（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前に同じカメライベント通知を受信した場合、イベント（シグナル）オブジェクト `hCmpEvt` がリセット状態になった後に再度シグナル状態にはなりませんのでご注意ください。

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通知（メッセージ）の取得 を参照してください。

5.4.1.1. Evt_OpenSimple

カメライベント通知（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、カメライベント通知（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
hCam	[in]	カメラのカメラハンドルです。
phEvt	[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。
uiApiBufferCount	[in]	TeliCamAPI 内部に作成するイベントリクエストの数です。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、5.8. ステータスコード を参照してください。

[備考]

他のアプリケーションが使用しているカメライベントインターフェースを開こうとした場合、インターフェースのオープンに失敗し、本関数は CAM_API_STS_ALREADY_OPENED を返します。

本関数は、TeliCamAPI 内部にカメライベント通知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通知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、カメライベント通知（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。

GenICAM 関数を使用して通知されたカメライベント通知（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通知を設定した場合、複数のカメライベント通知が短時間に発生します。　そのため、使用環境やアプリケーションにより処理が間に合わなくなる場合があります。　この時、受信したカメライベント通知は何もされずに破棄されます。

カメライベント通知を受信した順序で取得する必要がある場合、またはフリーランモードやバルクトリガモードで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、低水準関数の使用を推奨します。

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;


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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

        // Get Timestamp.
        uiStatus = GenApi_GetIntValue(hCam, "EventFrameTriggerTimestamp", &llVal);
```

```
        if (uiStatus != CAM_API_STS_SUCCESS)
            break;
        printf("Timestamp%d = %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)
            break;
    }
} while(false);

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

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

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

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

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

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

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

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

5.4.2. 低水準関数

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

5.4.2.1. Evt_Open

カメライベント通知（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]

For Windows

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

For Linux

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

[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。
phEvt	[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。
puiMaxPayloadSize	[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 `puiMaxPayloadSize` に格納します。通常は `0` を指定してください。
hCmpEvt	[in]	カメライベント通知（メッセージ）を受信したことを通知するイベント（シグナル）オブジェクトのハンドルです。 Windows 版のイベント（シグナル）オブジェクトは、`Sys_CreateSignal()` または Win32API の `CreateEvent()` で作成します。 Linux 版のイベント（シグナル）オブジェクトは、`Sys_CreateSignal()` 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、5.8. ステータスコード を参照してください。

[備考]

他のアプリケーションが使用しているカメライベントインターフェースを開こうとした場合、インターフェースのオープンに失敗し、本関数は CAM_API_STS_ALREADY_OPENED を返します。

カメライベント通知（メッセージ）を受信すると、TeliCamAPI はイベント通知受信待機キューからのイベントリクエストの取り出します。　受信したデータをイベントリクエストに設定し、イベントリクエストをイベント通知受信完了キューに格納させた後にイベント（シグナル）オブジェクト *hCmpEvt* をシグナル状態に設定します。

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

Evt_FlushRequestQueue()を実行してイベントリクエストをイベント通知受信完了キューに移動したときはイベント（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++

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


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

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

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

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

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

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

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

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

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

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

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

        printf("Evt_Activate Success.\n");

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

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

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

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

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

    // Start stream.
    uiStatus = Strm_Start(hStrm);
```

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

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

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

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

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

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

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

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

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

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

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

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

    uiStatus = Evt_Deactivate(hEvent, "ExposureStart");
    if (uiStatus != CAM_API_STS_SUCCESS)
```

```
        printf("Evt_Deactivate error! (0x%x)\n", uiStatus);

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

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

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

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

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

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

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

// Terminate system.
Sys_Terminate();

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

5.4.2.2. Evt_CreateRequest

カメライベント通知（メッセージ）データ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                 uiPayloaSize,
  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通知（メッセージ）の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、*uiPayloaSize* 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通知（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、Evt_Open()実行時に *puiMaxPayloadSize* 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、5.8. 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通知受信待機キューに投入してください。

カメライベント通知を受信すると、TeliCamAPI はイベント通知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通知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通知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。

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。*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. Evt_Flash_WaitQueue() を実行し、イベント通知受信待機キューに投入されているすべてのイベントリクエストをイベント通知受信完了キューへ移動。
3. イベント通知受信完了キューが空になるまで Evt_DequeueRequest() を繰り返し実行し、イベント通知受信完了キューからイベントリクエストを取り出し。
4. Evt_ReleaseRequest()を実行し、イベントリクエストを解放。
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]

Evt_Open() のコード例を参照してください。

5.4.2.3. Evt_ReleaseRequest

カメライベント通知（メッセージ）データ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
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、5.8. 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() で作成したイベントリクエストは、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、イベント通知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、5.8. 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() の備考欄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]

Evt_Open()　のコード例を参照してください。

5.4.2.5. Evt_DequeueRequest

イベント通知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。

[*EVT_REQUEST_INFO* 構造体]

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

メンバ		内　容
sU3vInfo	[out]	USB3 Vision カメラ使用時のイベントリクエスト情報です。
sGevInfo	[out]	GigE Vision カメラ使用時のイベントリクエスト情報です。

[*U3V_EVT_REQUEST_INFO* 構造体]

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

メンバ		内　容
ushRequestId	[out]	リクエスト ID
ushEventId	[out]	イベント ID
ullTimestamp	[out]	タイムスタンプ
pvPayload	[out]	イベントペイロードデータの先頭ポインタ

[*GEV_EVT_REQUEST_INFO* 構造体]

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

メンバ		内　容
ushRequestId	[out]	リクエスト ID
ushEventId	[out]	イベント ID
ullTimestamp	[out]	タイムスタンプ
pvPayload	[out]	イベントペイロードデータの先頭ポインタ
uiReserved1	[out]	予約済み 1（未使用）
uiReserved2	[out]	予約済み 2（未使用）

[戻り値]

実行結果を返します。　戻り値は、5.8. 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。

本関数はイベント通知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。

イベント通知受信完了キューが空のとき、本関数は CAM_API_STS_EMPTY_COMPLETE_QUEUE を戻します。

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、イベント通知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通知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、5.8. 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、イベント通知受信待機キューとイベント通知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通知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通知受信完了キューに移動させた後、イベント通知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通知受信完了キューから取り出してください。

Evt_DequeueRequest()で取り出したストリームリクエストが本関数により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]

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

カメライベント通知（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、5.8. 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。

5.4.3.2. Evt_Activate

指定されたカメライベント通知を有効に設定します。

[構文]

For Windows

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

For Linux

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

[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]	カメライベント通知（メッセージ）のノード名（レジスタ名）です。
hCmpEvt	[in]	カメライベント通知（メッセージ）を受信したことを通知するイベント（シグナル）オブジェクトのハンドルです。 Windows 版のイベント（シグナル）オブジェクトは、Sys_CreateSignal() または Win32API の CreateEvent() で作成します。 Linux 版のイベント（シグナル）オブジェクトは、Sys_CreateSignal() 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。 Evt_OpenSimple()でオープンした場合のみ有効です。 Evt_Open()でオープンされた場合は、本引数は使用されません。

[戻り値]

実行結果を返します。　戻り値は、5.8. ステータスコード を参照してください。

[備考]

Evt_OpenSimple() でオープンした場合

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

TeliCamAPI はカメライベント通知を受信するたびに本関数で指定されたイベント（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 WaitForSingleObject() / Sys_WaitForSignal() や ResetEvent() / Sys_ResetSignal() などによりイベント（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前に同じカメライベント通知を受信した場合、イベント（シ

グナル）オブジェクト hCmpEvt がリセット状態になった後に再度シグナル状態にはなりませんのでご注意ください。

以下の表に *pszEvtName* に指定できるカメライベント通知のノード名（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通知に関する情報は使用するカメラの取扱説明書をご覧ください。

カメライベント通知のノード名	内　　容	付随情報のノード名
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
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、*hCmpEvt* イベントがシグナル状態にセットされます。その際は、ResetEvent()関数等で *hCmpEvt* イベントを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。

Evt_Open() でオープンした場合

カメラに対し、指定されたカメライベントの有効化のみを行います。
pszEvtName に指定できるカメライベント通知のノード名は以下の通りです。

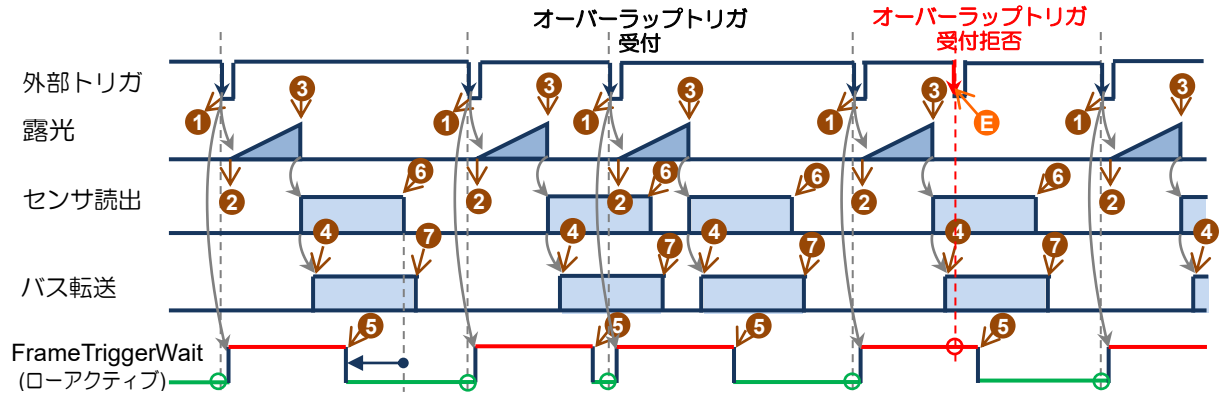
カメライベント通知のノード名	内　　容
FrameTrigger	トリガ受付
FrameTriggerError	トリガエラー
FrameTriggerWait	トリガ受付待ち開始
FrameTransferStart	映像転送開始
FrameTransferEnd	映像転送終了
ExposureStart	露光開始
ExposureEnd	露光終了
Timer0Start	Timer0 開始

カメライベント通知のノード名	内　容
Timer0End	Timer0 終了
ALCLatestInformation	ALC 動作更新時の値
ALCConverged	ALC 動作収束時の値

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。

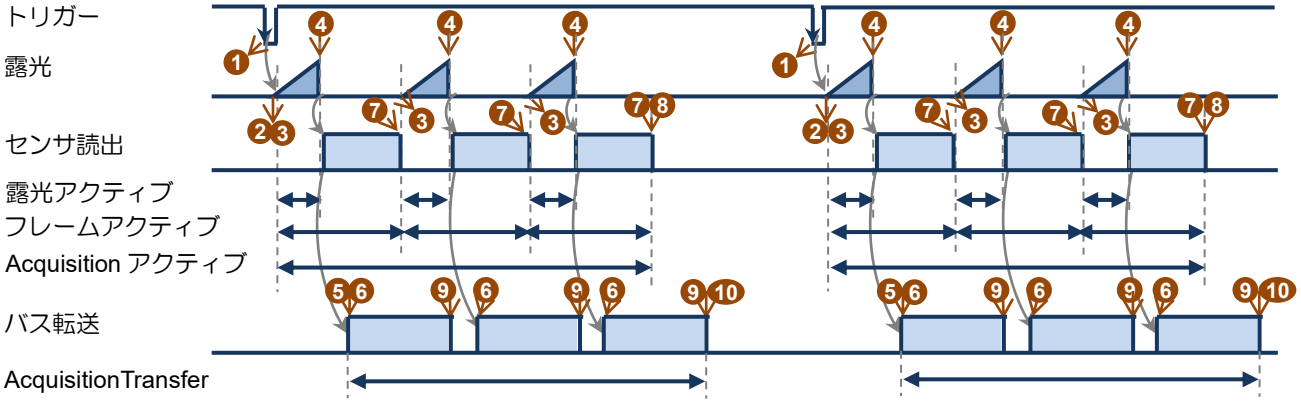
以下に各カメライベント通知のタイミングを示します。



FrameTriggerWait は FrameTrigger イベントでオフになり, センサ読み出し終了タイミングよりおおよそ露光時間だけ前のタイミングにオンになります。.

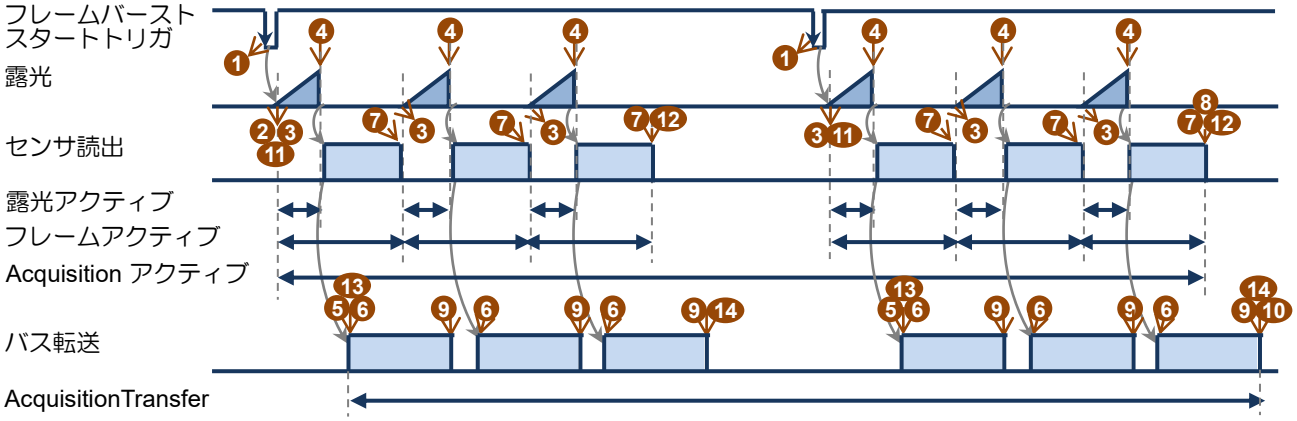
(1) FrameTrigger
(2) ExposureStart
(3) ExposureEnd
(4) FrameTransferStart
(5) FrameTriggerWait
(6) FrameEnd
(7) FrameTransferEnd
(E) FrameTriggerError

図１外部トリガーモード



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

図 ２バルクトリガーモード。 ３フレーム／トリガー



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

図３バースト信号

[コード例]

Evt_OpenSimple()のコード例を参照してください。

5.4.3.3. Evt_Deactivate

指定されたカメライベント通知（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
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]	解除するカメライベント通知（メッセージ）のノード名（レジスタ名）です。

[戻り値]

実行結果を返します。　戻り値は、5.8. ステータスコード を参照してください。

[備考]

Evt_Activate() 関数で登録したカメライベント通知（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。

5.5. 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。

IIDC2 規格に準拠しているカメラでは、処理パフォーマンスを高めるために GenICam GenApi ライブラリを使用せず、レジスタアクセスで処理されます。（一部の関数を除く）

IIDC2 規格に準拠していないカメラでは、GenICam GenApi ライブラリが使用されます。 カメラオープン時に GenApi モジュール無効を指定した場合は、本章の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。TeliCamApi 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（Format0 ～ Format2）の制御を行います。
映像フォーマットの機能はカメラまたはカメラのファームウェアバージョンにより異なります。

カメラの ImageFormatControl 機能に関する詳しい説明は、使用する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
hCam	[in]	カメラのカメラハンドルです。
peFormat	[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。

[*CAM_IMAGE_FORMAT_SELECTOR_TYPE* 列挙子]

メンバ	内容
IMAGE_FORMAT0	Format0
IMAGE_FORMAT1	Format1
IMAGE_FORMAT2	Format2

[戻り値]

実行結果を返します。　戻り値は、5.8. ステータスコード を参照してください。

[備考]

ImageFormatSelector レジスタ（またはノード）が実装されていないカメラで実行するとエラーステータスがリターンされます。

カメラによって機能が異なる場合があります。
カメラの ImageFormatControl 機能に関する詳しい説明は、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。

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
hCam	[in]	カメラのカメラハンドルです。
eFormat	[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、5.8. ステータスコード を参照してください。

[備考]

ImageFormatSelector レジスタ（またはノード）が実装されていないカメラで実行するとエラーステータスがリターンされます。

映像ストリーム出力中は設定を変更することはできません。

カメラによって機能が異なる場合があります。
カメラの ImageFormatControl 機能に関する詳しい説明は、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。

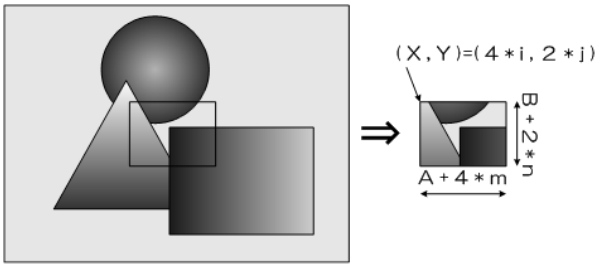
5.5.2. Scalable

カメラのスケーラブル機能の設定を行います。

スケーラブル読み出しは、最大映像出力有効画素領域のうち任意の矩形領域のみを読み出し、出力する方法です。　垂直方向(縦方向)の不要な領域を高速で読み飛ばすことでフレームレートを向上させることができます。
選択できる形状は連続したユニット単位の矩形形状のみで、凸や凹のような選択はできません。また選択できるウィンド数は 1 個です。

・ウィンドのサイズ　　：{A + 4 × m (H)} × {B + 2 × n (V)}
※　A, B はそれぞれの最小ユニットサイズ
※　m, n は整数、但しウィンドが最大ユニットサイズの全画面からはみ出さないこと

・ウィンドの開始位置　　：{4 x i (H)} x {2 x j (V)}
※　i, j は整数、但しウィンドが最大ユニットサイズの全画面からはみ出さないこと



カメラのスケーラブル機能に関する詳しい説明は、使用する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
hCam	[in]	カメラのカメラハンドルです。
puiSensorWidth	[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、5.8. 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
hCam	[in]	カメラのカメラハンドルです。
puiSensorHeight	[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、5.8. 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
hCam	[in]	カメラのカメラハンドルです。
puiWidth	[out]	取得した映像の幅を格納する変数へのポインタです。
puiHeight	[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。
puiOffsetX	[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。
puiOffsetY	[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、5.8. 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
hCam	[in]	カメラのカメラハンドルです。
uiWidth	[in]	設定する映像の幅です。
uiHeight	[in]	設定する映像の高さです。
uiOffsetX	[in]	設定する映像の水平方向開始位置です。
uiOffsetY	[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、5.8. ステータスコード を参照してください。

[備考]

映像ストリーム出力中は設定を変更することはできません。
ただし、映像ストリーム出力中でも OffsetX,OffsetY のみ変更することができるカメラがあります。
カメラのスケーラブル機能に関する詳しい説明は、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。

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
hCam	[in]	カメラのカメラハンドルです。
puiWidthMin	[out]	取得した最小値を格納する変数へのポインタです。
puiWidthMax	[out]	取得した最大値を格納する変数へのポインタです。
puiWidthInc	[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、5.8. 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
hCam	[in]	カメラのカメラハンドルです。
puiWidth	[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、5.8. 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
hCam	[in]	カメラのカメラハンドルです。
uiWidth	[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、5.8. ステータスコード を参照してください。

[備考]

映像ストリーム出力中は設定を変更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。

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
hCam	[in]	カメラのカメラハンドルです。
puiHeightMin	[out]	取得した最小値を格納する変数へのポインタです。
puiHeightMax	[out]	取得した最大値を格納する変数へのポインタです。
puiHeightInc	[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、5.8. 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
hCam	[in]	カメラのカメラハンドルです。
puiHeight	[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、5.8. 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
hCam	[in]	カメラのカメラハンドルです。
uiHeight	[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、5.8. ステータスコード を参照してください。

[備考]

映像ストリーム出力中は設定を変更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。

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
hCam	[in]	カメラのカメラハンドルです。
puiOffsetXMin	[out]	取得した最小値を格納する変数へのポインタです。
puiOffsetXMax	[out]	取得した最大値を格納する変数へのポインタです。
puiOffsetXInc	[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、5.8. 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
hCam	[in]	カメラのカメラハンドルです。
puiOffsetX	[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、5.8. 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
hCam	[in]	カメラのカメラハンドルです。
uiOffsetX	[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、5.8. ステータスコード を参照してください。

[備考]

映像ストリーム出力中に設定できるかどうかはカメラにより異なります。
カメラのスケーラブル機能に関する詳しい説明は、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。

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
hCam	[in]	カメラのカメラハンドルです。
puiOffsetYMin	[out]	取得した最小値を格納する変数へのポインタです。
puiOffsetYMax	[out]	取得した最大値を格納する変数へのポインタです。
puiOffsetYInc	[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、5.8. 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
hCam	[in]	カメラのカメラハンドルです。
puiOffsetY	[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、5.8. 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          uiOffsetY
);
```

[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。
uiOffsetY	[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、5.8. ステータスコード を参照してください。

[備考]

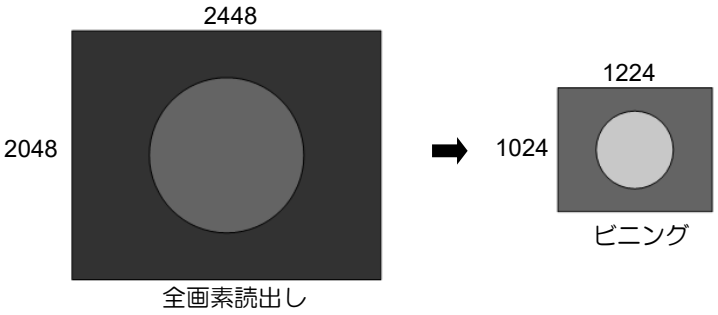
映像ストリーム出力中に設定できるかどうかはカメラにより異なります。
カメラのスケーラブル機能に関する詳しい説明は、使用するカメラの取扱説明書をご覧ください。

TeliCamAPI.h をインクルードする必要があります。

5.5.3. Binning

カメラのビニング機能の制御を行います。

ビニング読出しでは隣接する画素を加算することで、画素単位の感度が向上します。
さらにインターフェース帯域幅の占有帯域の軽減とフレームレートを向上させることができます。



ビニング動作のイメージ（2x2 ビニング）

カメラのビニング機能に関する詳しい説明は、使用する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
hCam	[in]	カメラのカメラハンドルです。
puiMin	[out]	取得した最小値を格納する変数へのポインタです。
puiMax	[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、5.8. ステータスコード を参照してください。

[備考]

BinningHorizontal レジスタ（またはノード）が実装されていないカメラで実行するとエラーステータスがリターンされます。

ビニング機能のパラメータを取得・設定できる条件はカメラにより異なります。
カメラのビニング機能に関する詳しい説明は、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。

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
hCam	[in]	カメラのカメラハンドルです。
puiValue	[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、5.8. ステータスコード を参照してください。

[備考]

BinningHorizontal レジスタ（またはノード）が実装されていないカメラで実行するとエラーステータスがリターンされます。

ビニング機能のパラメータを取得・設定できる条件はカメラにより異なります。
カメラのビニング機能に関する詳しい説明は、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。

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
hCam	[in]	カメラのカメラハンドルです。
uiValue	[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、5.8. ステータスコード を参照してください。

[備考]

BinningHorizontal レジスタ（またはノード）が実装されていないカメラで実行するとエラーステータスがリターンされます。

映像ストリーム出力中は設定を変更することはできません。

ビニング機能のパラメータを取得・設定できる条件はカメラにより異なります。
カメラのビニング機能に関する詳しい説明は、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。

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
hCam	[in]	カメラのカメラハンドルです。
puiMin	[out]	取得した最小値を格納する変数へのポインタです。
puiMax	[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、5.8. ステータスコード を参照してください。

[備考]

BinningVertical レジスタ（またはノード）が実装されていないカメラで実行するとエラーステータスがリターンされます。

ビニング機能のパラメータを取得・設定できる条件はカメラにより異なります。
カメラのビニング機能に関する詳しい説明は、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。

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
hCam	[in]	カメラのカメラハンドルです。
puiValue	[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、5.8. ステータスコード を参照してください。

[備考]

BinningVertical レジスタ（またはノード）が実装されていないカメラで実行するとエラーステータスがリターンされます。

ビニング機能のパラメータを取得・設定できる条件はカメラにより異なります。
カメラのビニング機能に関する詳しい説明は、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。

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
hCam	[in]	カメラのカメラハンドルです。
uiValue	[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、5.8. ステータスコード を参照してください。

[備考]

BinningVertical レジスタ（またはノード）が実装されていないカメラで実行するとエラーステータスがリターンされます。

映像ストリーム出力中は設定を変更することはできません。

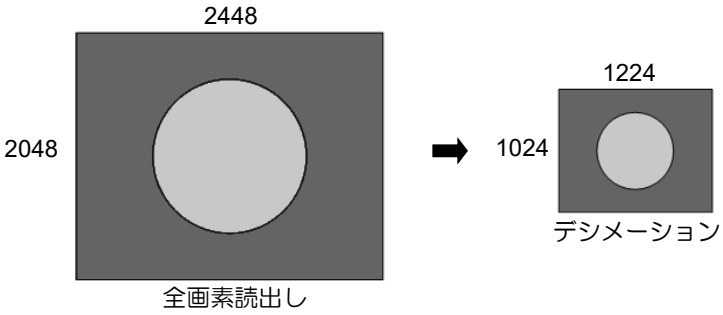
ビニング機能のパラメータを取得・設定できる条件はカメラにより異なります。
カメラのビニング機能に関する詳しい説明は、使用するカメラの取扱説明書をご覧ください。

TeliCamAPI.h をインクルードする必要があります。

5.5.4. Decimation

カメラのデシメーション機能の制御を行います。

デシメーション機能は読み出しラインを間引くことにより全有効エリアを高速で読み出し、
インターフェース帯域幅の占有帯域の軽減とフレームレートを向上させることができます。



デシメーション動作のイメージ（2x2 デシメーション）

カメラのデシメーション機能に関する詳しい説明は、使用する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
hCam	[in]	カメラのカメラハンドルです。
puiMin	[out]	取得した最小値を格納する変数へのポインタです。
puiMax	[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、5.8. ステータスコード を参照してください。

[備考]

DecimationHorizontal レジスタ（またはノード）が実装されていないカメラで実行するとエラーステータスがリターンされます。

デシメーション機能のパラメータを取得・設定できる条件はカメラにより異なります。
カメラのデシメーション機能に関する詳しい説明は、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。

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
hCam	[in]	カメラのカメラハンドルです。
puiValue	[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、5.8. ステータスコード を参照してください。

[備考]

DecimationHorizontal レジスタ（またはノード）が実装されていないカメラで実行するとエラーステータスがリターンされます。

デシメーション機能のパラメータを取得・設定できる条件はカメラにより異なります。
カメラのデシメーション機能に関する詳しい説明は、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。

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
hCam	[in]	カメラのカメラハンドルです。
uiValue	[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、5.8. ステータスコード を参照してください。

[備考]

DecimationHorizontal レジスタ（またはノード）が実装されていないカメラで実行するとエラーステータスがリターンされます。

映像ストリーム出力中は設定を変更することはできません。

デシメーション機能のパラメータを取得・設定できる条件はカメラにより異なります。
カメラのデシメーション機能に関する詳しい説明は、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。

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
hCam	[in]	カメラのカメラハンドルです。
puiMin	[out]	取得した最小値を格納する変数へのポインタです。
puiMax	[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、5.8. ステータスコード を参照してください。

[備考]

DecimationVertical レジスタ（またはノード）が実装されていないカメラで実行するとエラーステータスがリターンされます。

デシメーション機能のパラメータを取得・設定できる条件はカメラにより異なります。
カメラのデシメーション機能に関する詳しい説明は、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。

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
hCam	[in]	カメラのカメラハンドルです。
puiValue	[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、5.8. ステータスコード を参照してください。

[備考]

DecimationVertical レジスタ（またはノード）が実装されていないカメラで実行するとエラーステータスがリターンされます。

デシメーション機能のパラメータを取得・設定できる条件はカメラにより異なります。
カメラのデシメーション機能に関する詳しい説明は、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。

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
hCam	[in]	カメラのカメラハンドルです。
uiValue	[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、5.8. ステータスコード を参照してください。

[備考]

DecimationVertical レジスタ（またはノード）が実装されていないカメラで実行するとエラーステータスがリターンされます。

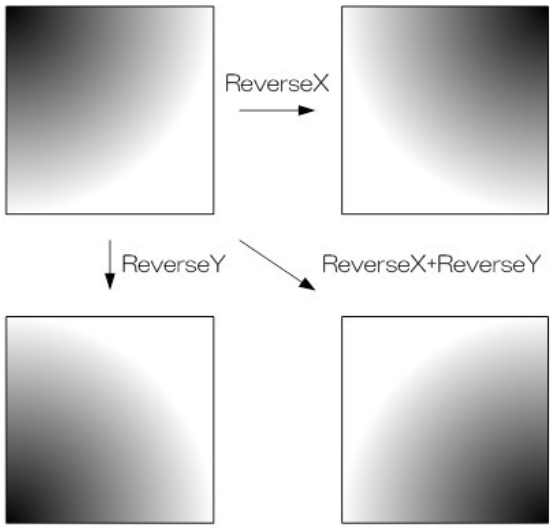
映像ストリーム出力中は設定を変更することはできません。

デシメーション機能のパラメータを取得・設定できる条件はカメラにより異なります。
カメラのデシメーション機能に関する詳しい説明は、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。

5.5.5. Reverse

カメラの映像反転機能の制御を行います。



カメラの映像反転機能に関する詳しい説明は、使用する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,
  bool8_t           *pbValue
);
```

[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。
pbValue	[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。 false の場合 反転 OFF、true の場合 反転 ON です。

[戻り値]

実行結果を返します。　戻り値は、5.8. ステータスコード を参照してください。

[備考]

ReverseX レジスタ（またはノード）が実装されていないカメラで実行するとエラーステータス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。

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
hCam	[in]	カメラのカメラハンドルです。
bValue	[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、5.8. ステータスコード を参照してください。

[備考]

ReverseX レジスタ（またはノード）が実装されていないカメラで実行するとエラーステータスがリターンされます。

映像ストリーム出力中は設定を変更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。

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
hCam	[in]	カメラのカメラハンドルです。
pbValue	[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、5.8. ステータスコード を参照してください。

[備考]

ReverseY レジスタ（またはノード）が実装されていないカメラで実行するとエラーステータス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。

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
hCam	[in]	カメラのカメラハンドルです。
bValue	[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、5.8. ステータスコード を参照してください。

[備考]

ReverseY レジスタ（またはノード）が実装されていないカメラで実行するとエラーステータスがリターンされます。

映像ストリーム出力中は設定を変更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。

5.5.6. PixelFormat

カメラの映像ストリームのピクセルフォーマットの制御を行います。
カメラの映像ストリームのピクセルフォーマットに関する詳しい説明は、使用する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
hCam	[in]	カメラのカメラハンドルです。
puiPixelFormat	[out]	取得したピクセルフォーマットを格納する変数へのポインタです。

[*CAM_PIXEL_FORMAT* 列挙子]

メンバ	内容
PXL_FMT_Unknown	不明なフォーマット
PXL_FMT_Mono8	Mono8 フォーマット
PXL_FMT_Mono10	Mono10 フォーマット
PXL_FMT_Mono12	Mono12 フォーマット
PXL_FMT_Mono16	Mono16 フォーマット
PXL_FMT_BayerGR8	BayerGR8 フォーマット
PXL_FMT_BayerGR10	BayerGR10 フォーマット
PXL_FMT_BayerGR12	BayerGR12 フォーマット
PXL_FMT_BayerRG8	BayerRG8 フォーマット
PXL_FMT_BayerRG10	BayerRG10 フォーマット
PXL_FMT_BayerRG12	BayerRG12 フォーマット
PXL_FMT_BayerGB8	BayerGB8 フォーマット
PXL_FMT_BayerGB10	BayerGB10 フォーマット
PXL_FMT_BayerGB12	BayerGB12 フォーマット
PXL_FMT_BayerBG8	BayerBG8 フォーマット
PXL_FMT_BayerBG10	BayerBG10 フォーマット
PXL_FMT_BayerBG12	BayerBG12 フォーマット
PXL_FMT_RGB8	RGB8 フォーマット
PXL_FMT_BGR8	BGR8 フォーマット
PXL_FMT_BGR10	BGR10 フォーマット
PXL_FMT_BGR12	BGR12 フォーマット
PXL_FMT_YUV411_8	YUV411_8 フォーマット
PXL_FMT_YUV422_8	YUV422_8 フォーマット
PXL_FMT_YUV8	YUV8 フォーマット

[戻り値]
実行結果を返します。　戻り値は、5.8. ステータスコード を参照してください。

[備考]
PixelFormat 、または PixelCoding と PixelSize レジスタ（またはノード）が実装されていないカメラで実行するとエラーステータス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。

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
hCam	[in]	カメラのカメラハンドルです。
uiPixelFormat	[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、5.8. ステータスコード を参照してください。

[備考]
PixelFormat 、または PixelCoding と PixelSize レジスタ（またはノード）が実装されていないカメラで実行するとエラーステータスがリターンされます。

映像ストリーム出力中は設定を変更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。

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。

カメラのテストパターン機能に関する詳しい説明は、使用する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
hCam	[in]	カメラのカメラハンドルです。
peTestPattern	[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。

[*CAM_TEST_PATTERN_TYPE* 列挙子]

メンバ	内容
CAM_TEST_PATTERN_OFF	テストパターン Off、通常映像
CAM_TEST_PATTERN_BLACK	全てのピクセルが 0
CAM_TEST_PATTERN_WHITE	全てのピクセルが 255 @Mono8
CAM_TEST_PATTERN_GREY_A	全てのピクセルが 170 @Mono8
CAM_TEST_PATTERN_GREY_B	全てのピクセルが 85 @Mono8
CAM_TEST_PATTERN_GREY_HORIZONTAL_RAMP	水平方向ランプ
CAM_TEST_PATTERN_GREY_SCALE	グレースケール
CAM_TEST_PATTERN_COLOR_BAR	カラーバー
CAM_TEST_PATTERN_GREY_VERTICAL_RAMP	垂直方向ランプ

[戻り値]

実行結果を返します。　戻り値は、5.8. ステータスコード を参照してください。

[備考]

TestPattern または TestImageSelector レジスタ（またはノード）が実装されていないカメラで実行するとエラーステータス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。

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
hCam	[in]	カメラのカメラハンドルです。
eTestPattern	[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、5.8. ステータスコード を参照してください。

[備考]

TestPattern または TestImageSelector レジスタ（またはノード）が実装されていないカメラで実行するとエラーステータス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。

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。
カメラの AcquisitionControl 機能に関する詳しい説明は、使用する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
hCam	[in]	カメラのカメラハンドルです。
puiPayloadSize	[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、5.8. 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,
  bool8_t           *pbEnable
);
```

[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。
pbEnable	[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。 false のとき無効、true のとき有効 です。

[戻り値]

実行結果を返します。　戻り値は、5.8. 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

“AcquisitionFrameCount” の最小値と最大値を取得します。
“AcquisitionFrameCount” は、マルチフレーム映像ストリーム転送モード時およびカメライメージバッファ転送モード時の映像ストリーム転送枚数です。

[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
hCam	[in]	カメラのカメラハンドルです。
puiAcqFrameCountMin	[out]	取得した最小値を格納する変数へのポインタです。
puiAcqFrameCountMax	[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、5.8. ステータスコード を参照してください。

[備考]

AcquisitionFrameCount レジスタ（またはノード）が実装されていないカメラで実行するとエラーステータス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。

5.5.8.4. GetCamAcquisitionFrameCount

“AcquisitionFrameCount” の値を取得します。

マルチフレーム映像ストリーム転送モード時は、転送したフレームの枚数が “AcquisitionFrameCount” の値に達すると、カメラは画像取得を停止します。

カメライメージバッファ転送モード時は、ExecuteImageBufferRead() が実行されるたびに、カメラはイメージバッファから “AcquisitionFrameCount” で設定された枚数のフレームを転送します。

[構文]

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

[パラメータ]

パラメータ		内　容
hCam	[in]	カメラのカメラハンドルです。
puiAcqFrameCount	[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、5.8. ステータスコード を参照してください。

[備考]

AcquisitionFrameCount レジスタ（またはノード）が実装されていないカメラで実行するとエラーステータス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。

5.5.8.5. SetCamAcquisitionFrameCount

“AcquisitionFrameCount” の値を設定します。

マルチフレーム映像ストリーム転送モード時は、転送したフレームの枚数が “AcquisitionFrameCount” の値に達すると、カメラは画像取得を停止します。

カメライメージバッファ転送モード時は、ExecuteImageBufferRead() が実行されるたびに、カメラはイメージバッファから “AcquisitionFrameCount” で設定された枚数のフレームを転送します。

[構文]

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

[パラメータ]

パラメータ		内　容
hCam	[in]	カメラのカメラハンドルです。
uiAcqFrameCount	[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、5.8. ステータスコード を参照してください。

[備考]

AcquisitionFrameCount レジスタ（またはノード）が実装されていないカメラで実行するとエラーステータスがリターンされます。

映像ストリーム出力中は設定を変更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。

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
hCam	[in]	カメラのカメラハンドルです。
peFrameRateControl	[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。

[*CAM_ACQ_FRAME_RATE_CTRL_TYPE* 列挙子]

メンバ	内容
CAM_ACQ_FRAME_RATE_CTRL_NO_SPECIFY	ExposureTime の設定値優先
CAM_ACQ_FRAME_RATE_CTRL_MANUAL	AcquisitionFrameRate の設定値優先

[戻り値]

実行結果を返します。　戻り値は、5.8. ステータスコード を参照してください。

[備考]

AcquisitionFrameRate レジスタ（または AcquisitionFrameRateControl / AcquisitionFrameRateEnable ノード）が実装されていないカメラで実行するとエラーステータス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。

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
hCam	[in]	カメラのカメラハンドルです。
eFrameRateControl	[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、5.8. ステータスコード を参照してください。

[備考]

AcquisitionFrameRate レジスタ（または AcquisitionFrameRateControl / AcquisitionFrameRateEnable ノード）が実装されていないカメラで実行するとエラーステータスがリターンされます。

NoSpecify（CAM_ACQ_FRAME_RATE_CTRL_NO_SPECIFY）を設定した場合、AcquisitionFrameRate レジスタ（またはノード）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。

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
hCam	[in]	カメラのカメラハンドルです。
pdAcqFrameRateMin	[out]	取得した最小値を格納する変数へのポインタです。
pdAcqFrameRateMax	[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、5.8. ステータスコード を参照してください。

[備考]

AcquisitionFrameRate レジスタ（または AcquisitionFrameRateControl / AcquisitionFrameRateEnable ノード）が実装されていないカメラで実行するとエラーステータス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。

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
hCam	[in]	カメラのカメラハンドルです。
pdAcqFrameRate	[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、5.8. ステータスコード を参照してください。

[備考]

AcquisitionFrameRate レジスタ（または AcquisitionFrameRateControl / AcquisitionFrameRateEnable ノード）が実装されていないカメラで実行するとエラーステータス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。

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
hCam	[in]	カメラのカメラハンドルです。
dAcqFrameRate	[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、5.8. ステータスコード を参照してください。

[備考]

AcquisitionFrameRate レジスタ（または AcquisitionFrameRateControl / AcquisitionFrameRateEnable ノード）が実装されていないカメラで実行するとエラーステータスがリターンされます。

映像のフレームレート設定が NoSpecify（CAM_ACQ_FRAME_RATE_CTRL_NO_SPECIFY）に設定されている場合、この関数はエラーをリターンするか、成功してもすぐに他の値に書き換えられる場合があります。

映像ストリーム出力中に設定できるかどうかは使用するカメラにより異なります。
カメラの AcquisitionControl 機能に関する詳しい説明は、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。

5.5.8.11. ExecuteCamAcquisitionStart

AcquisitionStart コマンドを実行します。

シングルフレーム映像ストリーム転送モードおよびマルチフレーム映像ストリーム転送モード時に再度画像を取得するときに実行します。

[構文]

```
CAM_API_STATUS  ExecuteCamAcquisitionStart (
  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、5.8. ステータスコード を参照してください。

[備考]

AcquisitionCommand レジスタ（またはノード）が実装されていないカメラで実行するとエラーステータスがリターンされます。

シングルフレーム映像ストリーム転送モードまたはマルチフレーム映像ストリーム転送モードでストリーミングを実行している時以外は本関数を実行しないでください。

TeliCamAPI.h をインクルードする必要があります。

5.5.8.12. GetCamHighFramerateMode

カメラの高フレームレートモード（HighFramerateMode）を取得します。

一部のカラーモデルのカメラは高フレームレートモード（HighFramerateMode）を有しています。 高フレームレートモードを利用することにより、フレームレートを向上させることができます。

[構文]

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

[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。
pbValue	[out]	取得した高フレームレートモードを格納する変数へのポインタです。 false の場合 OFF 、true の場合 ON です。

[戻り値]

実行結果を返します。　戻り値は、5.8. ステータスコード を参照してください。

[備考]

HighFramerateMode レジスタ（またはノード）が実装されていないカメラで実行するとエラーステータスがリターンされます。

カメラの高フレームレートモードに関する詳しい説明は、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。

5.5.8.13. SetCamHighFramerateMode

カメラの高フレームレートモード（HighFramerateMode）を設定します。

一部のカラーモデルのカメラは高フレームレートモード（HighFramerateMode）を有しています。 高フレームレートモードを利用することにより、フレームレートを向上させることができます。

[構文]

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

[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。
bValue	[in]	設定する高フレームレートモードです。 false の場合 OFF 、true の場合 ON です。

[戻り値]

実行結果を返します。　戻り値は、5.8. ステータスコード を参照してください。

[備考]

HighFramerateMode レジスタ（またはノード）が実装されていないカメラで実行するとエラーステータスがリターンされます。

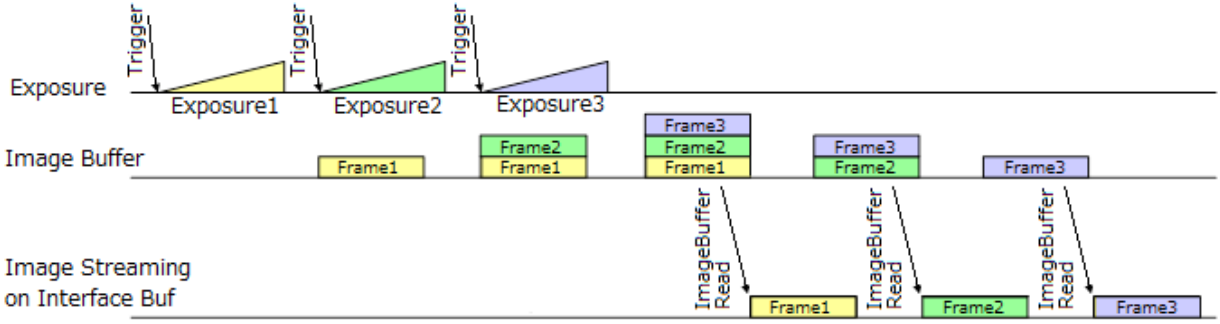
カメラの高フレームレートモードに関する詳しい説明は、使用するカメラの取扱説明書をご覧ください。

TeliCamAPI.h をインクルードする必要があります。

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。

ImageBuffer はイメージバッファに画像を取り込んでおき、任意のタイミングで読み出しを行うことができます。
この機能はノーマルシャッタモードでも動作しますが、通常ランダムトリガモードにて使用します。



カメラのイメージバッファ機能に関する詳しい説明は、使用する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
hCam	[in]	カメラのカメラハンドルです。
peMode	[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。

[*CAM_IMAGE_BUFFER_MODE_TYPE* 列挙子]

メンバ	内容
CAM_IMAGE_BUFFER_MODE_OFF	イメージバッファモード OFF
CAM_IMAGE_BUFFER_MODE_ON	イメージバッファモード ON

[戻り値]

実行結果を返します。　戻り値は、5.8. ステータスコード を参照してください。

[備考]

ImageBufferMode レジスタ（またはノード）が実装されていないカメラで実行するとエラーステータス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。

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
hCam	[in]	カメラのカメラハンドルです。
eMode	[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、5.8. ステータスコード を参照してください。

[備考]

ImageBufferMode レジスタ（またはノード）が実装されていないカメラで実行するとエラーステータスがリターンされます。

映像ストリーム出力中は設定を変更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。

5.5.9.3. GetCamImageBufferFrameCount

カメラのイメージバッファに取り込まれたフレーム数を取得します。

[構文]

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

[パラメータ]

パラメータ		内　容
hCam	[in]	カメラのカメラハンドルです。
puiCount	[out]	取得したイメージバッファに取り込まれたフレーム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、5.8. ステータスコード を参照してください。

[備考]

ImageBufferFrameCount レジスタ（またはノード）が実装されていないカメラで実行するとエラーステータス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。

5.5.9.4. ExecuteCamImageBufferRead

カメラのイメージバッファの画像を読み出します。

[構文]

```
CAM_API_STATUS  ExecuteCamImageBufferRead (
  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、5.8. ステータスコード を参照してください。

[備考]

AcquisitionCommand レジスタ（または ImageBufferRead ノード）が実装されていないカメラで実行するとエラーステータスがリターンされます。

カメラの AcquisitionMode レジスタに ”ImageBufferRead” を設定します。

1 回のコマンド実行でカメラが出力するフレーム数は、`SetCamAcquisitionFrameCount()` で設定できます。

カメラから出力されたフレームは、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機能に関する詳しい説明は、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。

5.5.10.TriggerControl

カメラのトリガ機能の制御を行います。

カメラの露光動作には、フリーランで動作するノーマルシャッタモードと外部からのトリガにより任意のタイミングで動作するランダムトリガシャッタモードの 2 種類があります。

カメラのトリガ機能に関する詳しい説明は、使用するカメラの取扱説明書をご覧ください。

5.5.10.1. GetCamTriggerMode

カメラのトリガ動作モード（TriggerMode）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
hCam	[in]	カメラのカメラハンドルです。
pbValue	[out]	取得したトリガ動作モードを格納する変数へのポインタです。 false の場合 トリガ動作モード OFF 、true の場合 トリガ動作モード ON です。

[戻り値]

実行結果を返します。　戻り値は、5.8. ステータスコード を参照してください。

[備考]

TriggerMode レジスタ（またはノード）が実装されていないカメラで実行するとエラーステータスがリターンされます。

トリガ動作モードが OFF の場合、ノーマルシャッタモードで動作します。 フリーラン（内部同期）モードが同期モードとして使用され、露光時間は ExposureTime レジスタで制御されます。

トリガ動作モードが ON の場合、ランダムシャッタモードで動作します。　ハードウェアトリガモードまたはソフトウェアトリガモードが同期モードとして使用され、露光時間は TriggerSequence 等の他のレジスタによって設定される様々なモードで制御されます。

カメラのトリガ機能に関する詳しい説明は、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。

5.5.10.2. SetCamTriggerMode

カメラのトリガ動作モード（TriggerMode）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
hCam	[in]	カメラのカメラハンドルです。
bValue	[in]	設定するトリガ動作モードです。 false の場合 トリガ動作モード OFF 、true の場合 トリガ動作モード ON です。

[戻り値]

実行結果を返します。　戻り値は、5.8. ステータスコード を参照してください。

[備考]

TriggerMode レジスタ（またはノード）が実装されていないカメラで実行するとエラーステータスがリターンされます。

映像ストリーム出力中に設定できるかどうかはカメラにより異なります。
カメラのトリガ機能に関する詳しい説明は、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。

5.5.10.3. GetCamTriggerSequence

カメラのランダムトリガシャッタの露光時間制御モード（TriggerSequence）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
hCam [in]	カメラのカメラハンドルです。
peTriggerSequence [out]	取得した露光時間制御モードを格納する変数へのポインタです。

[*CAM_TRIGGER_SEQUENCE_TYPE* 列挙子]

メンバ	内容
CAM_TRIGGER_SEQUENCE0	Edge モード。　露光時間は電子シャッタの設定値 IIDC2 規格に準拠したカメラ : TriggerSequence レジスタ = TriggerSequence0 IIDC2 規格に準拠していないカメラ : ExposureMode レジスタ = Timed TriggerSelector レジスタ = FrameStart
CAM_TRIGGER_SEQUENCE1	Level モード。　露光時間はトリガ信号のパルス幅 IIDC2 規格に準拠したカメラ : TriggerSequence レジスタ = TriggerSequence1 IIDC2 規格に準拠していないカメラ : ExposureMode レジスタ = TriggerWidth TriggerSelector レジスタ = FrameStart
CAM_TRIGGER_SEQUENCE6	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。 IIDC2 規格に準拠したカメラ : TriggerSequence レジスタ = TriggerSequence6 IIDC2 規格に準拠していないカメラ : ExposureMode レジスタ = Timed TriggerSelector レジスタ = FrameBurstStart

[戻り値]

実行結果を返します。　戻り値は、5.8. ステータスコード を参照してください。

[備考]

TriggerSequence、または ExposureMode と TriggerSelector レジスタ（またはノード）が実装されていないカメラで実行するとエラーステータスがリターンされます。

カメラによって、読み出すレジスタが異なります。
カメラのトリガ機能に関する詳しい説明は、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。

5.5.10.4. SetCamTriggerSequence

カメラのランダムトリガシャッタの露光時間制御モード（TriggerSequence）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
hCam	[in]	カメラのカメラハンドルです。
eTriggerSequence	[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、5.8. ステータスコード を参照してください。

[備考]

TriggerSequence、または ExposureMode と TriggerSelector レジスタ（またはノード）が実装されていないカメラで実行するとエラーステータスがリターンされます。

カメラによって、設定するレジスタが異なります。
カメラのトリガ機能に関する詳しい説明は、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。

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（TriggerSource）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
hCam	[in]	カメラのカメラハンドルです。
peTriggerSource	[out]	取得したトリガソースを格納する変数へのポインタです。

[*CAM_TRIGGER_SOURCE_TYPE* 列挙子]

メンバ	内容
CAM_TRIGGER_LINE0	ハードウェアトリガ Line0
CAM_TRIGGER_LINE1	ハードウェアトリガ Line1
CAM_TRIGGER_LINE2	ハードウェアトリガ Line2
CAM_TRIGGER_SOFTWARE	ソフトウェアトリガ

[戻り値]

実行結果を返します。　戻り値は、5.8. ステータスコード を参照してください。

[備考]

TriggerSource レジスタ（またはノード）が実装されていないカメラで実行するとエラーステータス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。

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（TriggerSource）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
hCam	[in]	カメラのカメラハンドルです。
eTriggerSource	[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、5.8. ステータスコード を参照してください。

[備考]

TriggerSource レジスタ（またはノード）が実装されていないカメラで実行するとエラーステータス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。

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
hCam	[in]	カメラのカメラハンドルです。
puiAdditionalParameterMin	[out]	取得した最小値を格納する変数へのポインタです。
puiAdditionalParameterMax	[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、5.8. ステータスコード を参照してください。

[備考]

TriggerAdditionalParameter または AcquisitionBurstFrameCount レジスタ（またはノード）が実装されていないカメラで実行するとエラーステータスがリターンされます。

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。

本関数は、TriggerAdditionalParameter または AcquisitionBurstFrameCount レジスタに設定できる最小値と最大値を取得します。
カメラのトリガ機能に関する詳しい説明は、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。

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
hCam	[in]	カメラのカメラハンドルです。
puiAdditionalParameter	[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、5.8. ステータスコード を参照してください。

[備考]

TriggerAdditionalParameter または AcquisitionBurstFrameCount レジスタ（またはノード）が実装されていないカメラで実行するとエラーステータスがリターンされます。

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。

本関数は、TriggerAdditionalParameter または AcquisitionBurstFrameCount レジスタに設定されている値を取得します。
カメラのトリガ機能に関する詳しい説明は、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。

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
hCam	[in]	カメラのカメラハンドルです。
uiAdditionalParameter	[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、5.8. ステータスコード を参照してください。

[備考]

TriggerAdditionalParameter または AcquisitionBurstFrameCount レジスタ（またはノード）が実装されていないカメラで実行するとエラーステータスがリターンされます。

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。

本関数は、TriggerAdditionalParameter または AcquisitionBurstFrameCount レジスタに値を設定します。
カメラのトリガ機能に関する詳しい説明は、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。

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         *pdDelayUsMin,
  float64_t         *pdDelayUsMax
);
```

[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。
pdDelayUsMin	[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：マイクロ秒）
pdDelayUsMax	[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：マイクロ秒）

[戻り値]

実行結果を返します。　戻り値は、5.8. ステータスコード を参照してください。

[備考]

TriggerDelay レジスタ（またはノード）が実装されていないカメラで実行するとエラーステータス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。

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         *pdDelayUs
);
```

[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。
pdDelayUs	[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：マイクロ秒）。

[戻り値]

実行結果を返します。　戻り値は、5.8. ステータスコード を参照してください。

[備考]

TriggerDelay レジスタ（またはノード）が実装されていないカメラで実行するとエラーステータス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。

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         dDelayUs
);
```

[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。
dDelayUs	[in]	設定する遅延量です。（単位：マイクロ秒）。

[戻り値]

実行結果を返します。　戻り値は、5.8. ステータスコード を参照してください。

[備考]

TriggerDelay レジスタ（またはノード）が実装されていないカメラで実行するとエラーステータス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。

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
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、5.8. ステータスコード を参照してください。

[備考]

SoftwareTrigger レジスタ（またはノード）が実装されていないカメラで実行するとエラーステータスがリターンされます。

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ません。

TeliCamAPI.h をインクルードする必要があります。

5.5.10.14. GetCamTriggerActivation

カメラのハードウェアトリガの有効エッジ（TriggerActivation）を取得します。

[構文]

```
CAM_API_STATUS  GetCamTriggerActivation (
  CAM_HANDLE                  hCam,
  CAM_TRIGGER_ACTIVATION_TYPE  *peTriggerActivation
);
```

[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。
peTriggerActivation	[out]	取得した有効エッジの値を格納する変数へのポインタです。

[*CAM_TRIGGER_ACTIVATION_TYPE* 列挙子]

メンバ	内容
CAM_TRIGGER_FALLING_EDGE	Falling Edge モード
CAM_TRIGGER_RISING_EDGE	Rising Edge モード

[戻り値]

実行結果を返します。　戻り値は、5.8. ステータスコード を参照してください。

[備考]

TriggerActivation または LineInverterAll レジスタ（またはノード）が実装されていないカメラで実行するとエラーステータスがリターンされます。

取得される値は、現在設定されているランダムトリガシャッタのトリガソース設定（TriggerSource）に対する有効エッジです。　本関数を実行する前に、SetCamTriggerSource() によりランダムトリガシャッタのトリガソースを設定してください。

LineInverterAll レジスタが存在するカメラでは、本関数を実行すると LineInverterAll レジスタの値を読み出し、対象のラインの値を取得します。

カメラのトリガ機能に関する詳しい説明は、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。

5.5.10.15. SetCamTriggerActivation

カメラのハードウェアトリガの有効エッジ（TriggerActivation）を設定します。

[構文]

```
CAM_API_STATUS  SetCamTriggerActivation (
  CAM_HANDLE                  hCam,
  CAM_TRIGGER_ACTIVATION_TYPE  eTriggerActivation
);
```

[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。
eTriggerActivation	[in]	設定する有効エッジです。

[戻り値]

実行結果を返します。　戻り値は、5.8. ステータスコード を参照してください。

[備考]

TriggerActivation または LineInverterAll レジスタ（またはノード）が実装されていないカメラで実行するとエラーステータスがリターンされます。

設定する値は、現在設定されているランダムトリガシャッタのトリガソース設定（TriggerSource）に対する有効エッジです。　本関数を実行する前に、SetCamTriggerSource() によりランダムトリガシャッタのトリガソースを設定してください。

LineInverterAll レジスタが存在するカメラでは、LineInverterAll レジスタの値を読み出し、対象のラインの値のみを変更して LineInverterAll レジスタに値を設定します。　SetCamLineInverterAll() または SetCamLineInverter() を実行すると、本関数で設定した有効エッジが変更される場合があります。

カメラのトリガ機能に関する詳しい説明は、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。

5.5.11.ExposureTime

カメラの露光時間の制御を行います。

カメラの露光時間制御機能に関する詳しい説明は、使用する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
hCam	[in]	カメラのカメラハンドルです。
peExpControl	[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。

[*CAM_EXPOSURE_TIME_CONTROL_TYPE* 列挙子]

メンバ	内容
CAM_EXPOSURE_TIME_CONTROL_AUTO	自動露光時間制御 IIDC2 規格に準拠したカメラ : ExposureTimeControl レジスタ = Auto IIDC2 規格に準拠していないカメラ : ExposureAuto レジスタ = Continuous
CAM_EXPOSURE_TIME_CONTROL_MANUAL	ExposureTime の設定値優先 IIDC2 規格に準拠したカメラ : ExposureTimeControl レジスタ = Manual IIDC2 規格に準拠していないカメラ : ExposureAuto レジスタ = Off
CAM_EXPOSURE_TIME_CONTROL_NO_SPECIFY	AcquisitionFrameRate の設定値優先 IIDC2 規格に準拠したカメラ : ExposureTimeControl レジスタ = NoSpecify IIDC2 規格に準拠していないカメラ : 利用不可

[戻り値]

実行結果を返します。　戻り値は、5.8. ステータスコード を参照してください。

[備考]

ExposureTimeControl または ExposureAuto レジスタ（またはノード）が実装されていないカメラで実行するとエラーステータスがリターンされます。

カメラによって機能およびアクセスするレジスタが異なる場合があります。
カメラの露光時間制御機能に関する詳しい説明は、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。

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
hCam	[in]	カメラのカメラハンドルです。
eExpControl	[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、5.8. ステータスコード を参照してください。

[備考]

ExposureTimeControl または ExposureAuto レジスタ（またはノード）が実装されていないカメラで実行するとエラーステータスがリターンされます。

カメラによって機能およびアクセスするレジスタが異なる場合があります。
カメラの露光時間制御機能に関する詳しい説明は、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。

5.5.11.3. GetCamExposureTimeMinMax

カメラの露光時間制御モードが Manual に設定されているときの、露光時間の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
hCam	[in]	カメラのカメラハンドルです。
pdExpTimeUsMin	[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：マイクロ秒）
pdExpTimeUsMax	[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：マイクロ秒）

[戻り値]

実行結果を返します。　戻り値は、5.8. ステータスコード を参照してください。

[備考]

ExposureTime レジスタ（またはノード）が実装されていないカメラで実行するとエラーステータス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。

5.5.11.4. GetCamExposureTime

カメラの露光時間制御モードが Manual に設定されているときの、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
hCam	[in]	カメラのカメラハンドルです。
pdExpTimeUs	[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：マイクロ秒）

[戻り値]

実行結果を返します。　戻り値は、5.8. ステータスコード を参照してください。

[備考]

ExposureTime レジスタ（またはノード）が実装されていないカメラで実行するとエラーステータス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。

5.5.11.5. SetCamExposureTime

カメラの露光時間制御モードが Manual に設定されているときの、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
hCam	[in]	カメラのカメラハンドルです。
dExpTimeUs	[in]	設定する露光時間です。（単位：マイクロ秒）

[戻り値]

実行結果を返します。　戻り値は、5.8. ステータスコード を参照してください。

[備考]

ExposureTime レジスタ（またはノード）が実装されていないカメラで実行するとエラーステータス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。

5.5.11.6. GetCamShortExposureMode

カメラの超短時間露光モード（ShortExposureMode）を取得します。

一部のカラーモデルのカメラは超短時間露光モード（ShortExposureMode）を有しています。　超短時間露光モードを ON に設定すると、マニュアル露光時間制御（Manual）時に高速露光時間設定が可能となります。

[構文]

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

[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。
pbValue	[out]	取得した超短時間露光モードを格納する変数へのポインタです。 false の場合 OFF 、true の場合 ON です。

[戻り値]

実行結果を返します。　戻り値は、5.8. ステータスコード を参照してください。

[備考]

ShortExposureMode レジスタ（またはノード）が実装されていないカメラで実行するとエラーステータスがリターンされます。

カメラの超短時間露光モードに関する詳しい説明は、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。

5.5.11.7. SetCamShortExposureMode

カメラの超短時間露光モード（ShortExposureMode）を設定します。

一部のカラーモデルのカメラは超短時間露光モード（ShortExposureMode）を有しています。　超短時間露光モードを ON に設定すると、マニュアル露光時間制御（Manual）時に高速露光時間設定が可能となります。

[構文]

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

[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。
bValue	[in]	設定する超短時間露光モードです。 false の場合 OFF 、true の場合 ON です。

[戻り値]

実行結果を返します。　戻り値は、5.8. ステータスコード を参照してください。

[備考]

ShortExposureMode レジスタ（またはノード）が実装されていないカメラで実行するとエラーステータスがリターンされます。

カメラの超短時間露光モードに関する詳しい説明は、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。

5.5.12.DigitalIoControl

カメラのデジタル I/O の制御を行います。
カメラのデジタル I/O に関する詳しい説明は、使用するカメラの取扱説明書をご覧ください。

5.5.12.1. GetCamLineModeAll

カメラの全ラインに対する入出力設定値を取得します。

[構文]

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

[パラメータ]

パラメータ		内　容
hCam	[in]	カメラのカメラハンドルです。
puiValue	[out]	取得した入出力設定値を格納する変数へのポインタです。 各 bit が各ラインに対応しています。(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。 （`puiValue = 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、5.8. ステータスコード を参照してください。

[備考]

LineModeAll 、または LineSelector と LineMode レジスタ（またはノード）が実装されていないカメラで実行するとエラーステータスがリターンされます。

LineModeAll レジスタが存在しないカメラでは、各ラインの入出力設定値を読み出し、合成して出力します。

カメラのデジタル I/O に関する詳しい説明は、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。

5.5.12.2. SetCamLineModeAll

カメラの全ラインに対する入出力を設定します。

[構文]

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

[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。
uiValue	[in]	設定する入出力の値です。 各 bit が各ラインに対応しています。（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。 （`uiValue = 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、5.8. ステータスコード を参照してください。

[備考]

LineModeAll 、または LineSelector と LineMode レジスタ（またはノード）が実装されていないカメラで実行するとエラーステータスがリターンされます。

LineModeAll レジスタが存在しないカメラでは、各ライン毎にライン設定を行います。

カメラによって機能および書き込み可能な Line が異なる場合があります。
デジタル I/O 制御機能の説明は、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。

5.5.12.3. GetCamLineMode

カメラのラインの入出力を取得します。

[構文]

```
CAM_API_STATUS  GetCamLineMode (
  CAM_HANDLE                hCam,
  CAM_LINE_SELECTOR_TYPE    eLineSelector,
  CAM_LINE_MODE_TYPE        *peLineMode
);
```

[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。
eLineSelector	[in]	取得するラインです。
peLineMode	[out]	取得した入出力設定値を格納する変数へのポインタです。

[*CAM_LINE_SELECTOR_TYPE* 列挙子]

メンバ	内容
CAM_LINE_SELECTOR_LINE0	Line0
CAM_LINE_SELECTOR_LINE1	Line1
CAM_LINE_SELECTOR_LINE2	Line2

[*CAM_LINE_MODE_TYPE* 列挙子]

メンバ	内容
CAM_LINE_MODE_INPUT	入力
CAM_LINE_MODE_OUTPUT	出力

[戻り値]

実行結果を返します。　戻り値は、5.8. ステータスコード を参照してください。

[備考]

LineModeAll 、または LineSelector と LineMode レジスタ（またはノード）が実装されていないカメラで実行するとエラーステータスがリターンされます。

LineModeAll レジスタが存在するカメラでは、LineModeAll レジスタの値を読み出し、対象のラインの値を取得します。

デジタル I/O 制御機能の説明は、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。

5.5.12.4. SetCamLineMode

カメラのラインの入出力を設定します。

[構文]

```
CAM_API_STATUS  SetCamLineMode (
  CAM_HANDLE                hCam,
  CAM_LINE_SELECTOR_TYPE    eLineSelector,
  CAM_LINE_MODE_TYPE        eLineMode
);
```

[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。
eLineSelector	[in]	設定するラインです。
eLineMode	[in]	設定する入出力です。

[戻り値]

実行結果を返します。　戻り値は、5.8. ステータスコード を参照してください。

[備考]

LineModeAll 、または LineSelector と LineMode レジスタ（またはノード）が実装されていないカメラで実行するとエラーステータスがリターンされます。

LineModeAll レジスタが存在するカメラでは、LineModeAll レジスタの値を読み出し、対象のラインの値のみを変更して LineModeAll レジスタに値を設定します。

カメラによって機能および書き込み可能な Line が異なる場合があります。
デジタル I/O 制御機能の説明は、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。

5.5.12.5. GetCamLineInverterAll

カメラの全ラインに対する極性設定値を取得します。

[構文]

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

[パラメータ]

パラメータ		内　容
hCam	[in]	カメラのカメラハンドルです。
puiValue	[out]	取得した極性設定値を格納する変数へのポインタです。 各 bit が各ラインに対応しています。（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。 （puiValue = 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、5.8. ステータスコード を参照してください。

[備考]

LineInverterAll 、または LineSelector と LineInverter レジスタ（またはノード）が実装されていないカメラで実行するとエラーステータスがリターンされます。

LineInverterAll レジスタが存在しないカメラでは、各ラインの極性設定値を読み出し、合成して出力します。

カメラのデジタル I/O に関する詳しい説明は、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。

5.5.12.6. SetCamLineInverterAll

カメラの全ラインに対する極性を設定します。

[構文]

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

[パラメータ]

パラメータ		内　容
hCam	[in]	カメラのカメラハンドルです。
uiValue	[in]	設定する極性の値です。 各 bit が各ラインに対応しています。（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。 （`puiValue = 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、5.8. ステータスコード を参照してください。

[備考]

LineInverterAll 、または LineSelector と LineInverter レジスタ（またはノード）が実装されていないカメラで実行するとエラーステータスがリターンされます。

LineInverterAll レジスタが存在しないカメラでは、各ライン毎に極性設定を行います。

カメラによって機能および書き込み可能な Line が異なる場合があります。
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。
デジタル I/O 制御機能の説明は、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。

5.5.12.7. GetCamLineInverter

カメラのラインの極性を取得します。

[構文]

```
CAM_API_STATUS  GetCamLineInverter (
  CAM_HANDLE                hCam,
  CAM_LINE_SELECTOR_TYPE    eLineSelector,
  bool8_t                   *pbInvert
);
```

[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。
eLineSelector	[in]	取得するラインです。
pbInvert	[out]	取得した極性設定値を格納する変数へのポインタです。 false の場合 Invert なし、true の場合 invert あり です。

[戻り値]

実行結果を返します。　戻り値は、5.8. ステータスコード を参照してください。

[備考]

LineInverterAll 、または LineSelector と LineInverter レジスタ（またはノード）が実装されていないカメラで実行するとエラーステータスがリターンされます。

LineInverterAll レジスタが存在するカメラでは、LineInverterAll レジスタの値を読み出し、対象のラインの値を取得します。

デジタル I/O 制御機能の説明は、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。

5.5.12.8. SetCamLineInverter

カメラのラインの極性を設定します。

[構文]

```
CAM_API_STATUS  SetCamLineInverter (
  CAM_HANDLE                hCam,
  CAM_LINE_SELECTOR_TYPE    eLineSelector,
  bool8_t                   bInvert
);
```

[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。
eLineSelector	[in]	設定するラインです。
bInvert	[in]	設定する極性の値です。 false の場合 Invert なし、true の場合 invert あり です。

[戻り値]

実行結果を返します。　戻り値は、5.8. ステータスコード を参照してください。

[備考]

LineInverterAll 、または LineSelector と LineInverter レジスタ（またはノード）が実装されていないカメラで実行するとエラーステータスがリターンされます。

LineInverterAll レジスタが存在するカメラでは、LineInverterAll レジスタの値を読み出し、対象のラインの値のみを変更して LineInverterAll レジスタに値を設定します。

カメラによって機能および書き込み可能な Line が異なる場合があります。
デジタル I/O 制御機能の説明は、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。

5.5.12.9. GetCamLineStatusAll

カメラの全ラインに対する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
hCam	[in]	カメラのカメラハンドルです。
puiValue	[out]	取得した現在の状態を表す値を格納する変数へのポインタです。 各 bit が各ラインに対応しています。（Line0 : Bit0 , Line1 : Bit1 , Line2 : Bit2 , ... ） bit データが 0 のラインは Low、1 のラインは High です。 （puiValue = 0x02 のとき、Line0: Low, Line1 : High , Line2 :Low , ...）

[戻り値]

実行結果を返します。　戻り値は、5.8. ステータスコード を参照してください。

[備考]

LineStatusAll 、または LineSelector と LineStatus レジスタ（またはノード）が実装されていないカメラで実行するとエラーステータス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。

5.5.12.10. GetCamLineStatus

カメラのラインの現在の状態を取得します。

[構文]

```
CAM_API_STATUS  GetCamLineStatus (
  CAM_HANDLE                hCam,
  CAM_LINE_SELECTOR_TYPE    eLineSelector,
  bool8_t                   *pbValue
);
```

[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。
eLineSelector	[in]	取得するラインです。
pbValue	[out]	取得した現在の状態を表す値を格納する変数へのポインタです。 false の場合 Low、true の場合 High です。

[戻り値]

実行結果を返します。　戻り値は、5.8. ステータスコード を参照してください。

[備考]

LineStatusAll 、または LineSelector と LineStatus レジスタ（またはノード）が実装されていないカメラで実行するとエラーステータス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。

5.5.12.11. GetCamUserOutputValueAll

カメラの全ラインに対する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
hCam	[in]	カメラのカメラハンドルです。
puiValue	[out]	取得した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 が各ラインに対応しています。（Line0：Bit0，Line1：Bit1，Line2：Bit2 , ... ） bit データが 0 のラインは Low、1 のラインは High です。

[戻り値]

実行結果を返します。　戻り値は、5.8. ステータスコード を参照してください。

[備考]

UserOutputValueAll レジスタ（またはノード）が実装されていないカメラで実行するとエラーステータス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。

5.5.12.12. SetCamUserOutputValueAll

カメラの全ラインに対するユーザ出力値を設定します。

[構文]

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

[パラメータ]

パラメータ		内　容
hCam	[in]	カメラのカメラハンドルです。
uiValue	[in]	設定するユーザ出力設定値です。 各 bit が各ラインに対応しています。（Line0：Bit0，Line1：Bit1，Line2：Bit2 , ... ） bit データが 0 のラインは Low、1 のラインは High です。

[戻り値]

実行結果を返します。　戻り値は、5.8. ステータスコード を参照してください。

[備考]

UserOutputValueAll レジスタ（またはノード）が実装されていないカメラで実行するとエラーステータスがリターンされます。

ラインの信号種類（LineSource）が UserOutput（CAM_LINE_SOURCE_USER_OUTPUT）に設定されているライン以外の bit（出力設定値）は無視されます。

TeliCamAPI.h をインクルードする必要があります。

5.5.12.13. GetCamUserOutputValue

カメラの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,
  CAM_LINE_SELECTOR_TYPE    eLineSelector,
  bool8_t                   *pbValue
);
```

[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。
eLineSelector	[in]	取得するラインです。
pbValue	[out]	取得したユーザ出力設定値を格納する変数へのポインタです。 false の場合 Low、true の場合 High です。

[戻り値]

実行結果を返します。　戻り値は、5.8. ステータスコード を参照してください。

[備考]

UserOutputValueAll レジスタ（またはノード）が実装されていないカメラで実行するとエラーステータス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。

5.5.12.14. SetCamUserOutputValue

カメラのラインに対するユーザ出力値を設定します。

[構文]

```
CAM_API_STATUS  SetCamUserOutputValue (
  CAM_HANDLE                hCam,
  CAM_LINE_SELECTOR_TYPE    eLineSelector,
  bool8_t                   bValue
);
```

[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。
eLineSelector	[in]	設定するラインです。
bValue	[in]	設定するユーザ出力設定値です。 false の場合 Low、true の場合 High です。

[戻り値]

実行結果を返します。　戻り値は、5.8. ステータスコード を参照してください。

[備考]

UserOutputValueAll レジスタ（またはノード）が実装されていないカメラで実行するとエラーステータス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。

5.5.12.15. GetCamLineSource

カメラのラインの信号種類を取得します。

[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
hCam	[in]	カメラのカメラハンドルです。
eLineSelector	[in]	取得するラインです。
peLineSource	[out]	取得した信号種類を格納する変数へのポインタです。

[*CAM_LINE_SOURCE_TYPE* 列挙子]

メンバ	内容
CAM_LINE_SOURCE_OFF	汎用出力は無効です。
CAM_LINE_SOURCE_VD	VD 同期信号です。 (GigE Vision カメラのみ)
CAM_LINE_SOURCE_USER_OUTPUT	UserOutputValue にて設定した値を出力します。
CAM_LINE_SOURCE_TIMER0_ACTIIVE	ストロボ制御用信号として使用できます。トリガ入力からの遅延量と幅を設定できます。
CAM_LINE_SOURCE_ACQUISITION_ACTIVE	AcquisitionStart 状態であることを示す信号です。
CAM_LINE_SOURCE_FRAME_TRIGGER_WAIT	ランダムトリガシャッタ時に、トリガ待ち受け期間であることを示す信号です。
CAM_LINE_SOURCE_FRAME_ACTIVE	露光開始から CMOS 転送完了までの期間です。
CAM_LINE_SOURCE_FRAME_TRANSFER_ACTIVE	映像をバスに転送している期間です。
CAM_LINE_SOURCE_EXPOSURE_ACTIVE	露光開始から露光終了までの期間です。

[戻り値]

実行結果を返します。　戻り値は、5.8. ステータスコード を参照してください。

[備考]

LineSelector と LineSource レジスタ（またはノード）が実装されていないカメラで実行するとエラーステータス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。

5.5.12.16. 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
hCam	[in]	カメラのカメラハンドルです。
eLineSelector	[in]	設定するラインです。
eLineSource	[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、5.8. ステータスコード を参照してください。

[備考]

LineSelector と LineSource レジスタ（またはノード）が実装されていないカメラで実行するとエラーステータスがリターンされます。

カメラにより、設定できる信号種類が異なります。
カメラのデジタル I/O に関する詳しい説明は、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。

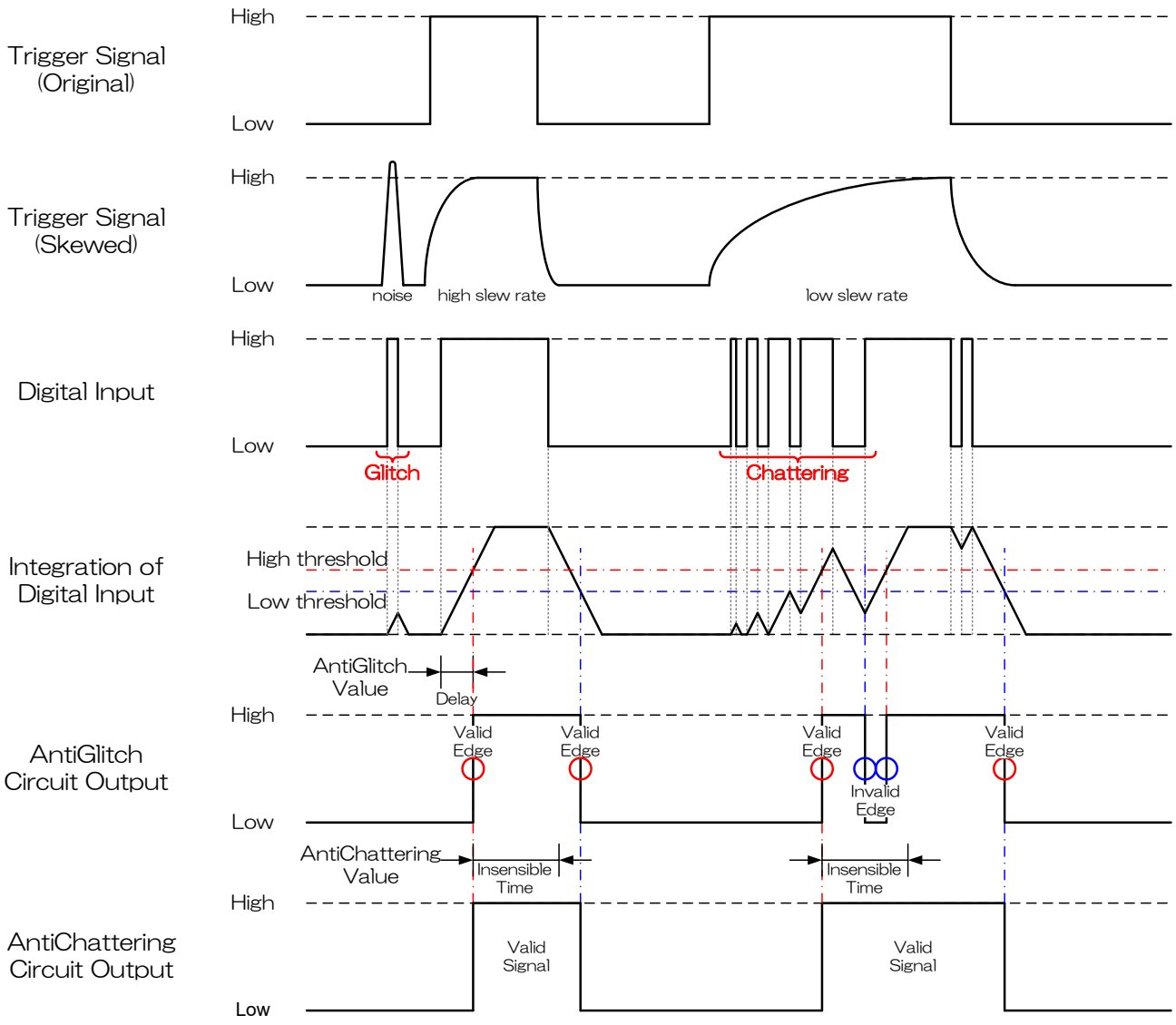
5.5.13.AntiGlitch / AntiChattering

カメラのアンチグリッチとアンチチャタリングの制御を行います。

アンチグリッチとアンチチャタリングはノイズや不安定なデジタル入力（トリガ信号）にフィルタをかける機能です。

アンチグリッチ回路は、トリガ信号のデジタル積分を行います。インパルス性ノイズを取り除くことに有効です。

アンチチャタリング回路は、トリガの誤動作を防止するためにエッジを受け付けない時間を設定します。不安定な論理状態やスイッチチャタリングを取り除くことに有効です。



カメラのアンチグリッチとアンチチャタリングに関する詳しい説明は、使用するカメラの取扱説明書をご覧ください。

5.5.13.1. GetCamAntiGlitchMinMax

カメラのデジタル入力信号の積分時間（絶対値）の最小値と最大値を取得します。

[構文]

```
CAM_API_STATUS  GetCamAntiGlitchMinMax (
  CAM_HANDLE        hCam,
  float64_t         *pdTimeUsMin,
  float64_t         *pdTimeUsMax
);
```

[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。
pdTimeUsMin	[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：マイクロ秒）
pdTimeUsMax	[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：マイクロ秒）

[戻り値]

実行結果を返します。　戻り値は、5.8. ステータスコード を参照してください。

[備考]

AntiGlitch レジスタ（またはノード）が実装されていないカメラで実行するとエラーステータスがリターンされます。

本関数では単位が マイクロ秒になっていることに注意してください。

TeliCamAPI.h をインクルードする必要があります。

5.5.13.2. GetCamAntiGlitch

カメラのデジタル入力信号の積分時間（絶対値）設定値を取得します。

[構文]

```
CAM_API_STATUS  GetCamAntiGlitch (
  CAM_HANDLE        hCam,
  float64_t         *pdTimeUs
);
```

[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。
pdTimeUs	[out]	取得したデジタル入力信号の積分時間（絶対値）を格納する変数へのポインタです。（単位：マイクロ秒）

[戻り値]

実行結果を返します。　戻り値は、5.8. ステータスコード を参照してください。

[備考]

AntiGlitch レジスタ（またはノード）が実装されていないカメラで実行するとエラーステータスがリターンされます。
本関数では単位が マイクロ秒になっていることに注意してください。

TeliCamAPI.h をインクルードする必要があります。

5.5.13.3. SetCamAntiGlitch

カメラのデジタル入力信号の積分時間（絶対値）を設定します。

[構文]

```
CAM_API_STATUS  SetCamAntiGlitch (
  CAM_HANDLE        hCam,
  float64_t         dTimeUs
);
```

[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。
dTimeUs	[in]	設定するデジタル入力信号の積分時間（絶対値）です。（単位：マイクロ秒）

[戻り値]

実行結果を返します。　戻り値は、5.8. ステータスコード を参照してください。

[備考]

AntiGlitch レジスタ（またはノード）が実装されていないカメラで実行するとエラーステータスがリターンされます。
本関数では単位が マイクロ秒になっていることに注意してください。

TeliCamAPI.h をインクルードする必要があります。

5.5.13.4. GetCamAntiChatteringMinMax

カメラのデジタル入力信号のエッジを受け付けない時間（絶対値）の最小値と最大値を取得します。

[構文]

```
CAM_API_STATUS  GetCamAntiChatteringMinMax (
  CAM_HANDLE        hCam,
  float64_t         *pdTimeUsMin,
  float64_t         *pdTimeUsMax
);
```

[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。
pdTimeUsMin	[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：マイクロ秒）
pdTimeUsMax	[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：マイクロ秒）

[戻り値]

実行結果を返します。　戻り値は、5.8. ステータスコード を参照してください。

[備考]

AntiChattering レジスタ（またはノード）が実装されていないカメラで実行するとエラーステータスがリターンされます。

本関数では単位が マイクロ秒になっていることに注意してください。

TeliCamAPI.h をインクルードする必要があります。

5.5.13.5. GetCamAntiChattering

カメラのデジタル入力信号のエッジを受け付けない時間（絶対値）設定値を取得します。

[構文]

```
CAM_API_STATUS  GetCamAntiChattering (
  CAM_HANDLE        hCam,
  float64_t         *pdTimeUs
);
```

[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。
pdTimeUs	[out]	取得したデジタル入力信号のエッジを受け付けない時間（絶対値）を格納する変数へのポインタです。（単位：マイクロ秒）。

[戻り値]

実行結果を返します。　戻り値は、5.8. ステータスコード を参照してください。

[備考]

AntiChattering レジスタ（またはノード）が実装されていないカメラで実行するとエラーステータスがリターンされます。
本関数では単位が マイクロ秒になっていることに注意してください。

TeliCamAPI.h をインクルードする必要があります。

5.5.13.6. SetCamAntiChattering

カメラのデジタル入力信号のエッジを受け付けない時間（絶対値）を設定します。

[構文]

```
CAM_API_STATUS  SetCamAntiChattering (
  CAM_HANDLE        hCam,
  float64_t         dTimeUs
);
```

[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。
dTimeUs	[in]	設定するデジタル入力信号のエッジを受け付けない時間（絶対値）です。（単位：マイクロ秒）。

[戻り値]

実行結果を返します。　戻り値は、5.8. ステータスコード を参照してください。

[備考]

AntiChattering レジスタ（またはノード）が実装されていないカメラで実行するとエラーステータスがリターンされます。
本関数では単位が マイクロ秒になっていることに注意してください。
TeliCamAPI.h をインクルードする必要があります。

5.5.14.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のでこの章の関数の制御対象は Timer0Active 信号に固定されます。

カメラの TimerConrtol 機能（Timer0Active 信号）に関する詳しい説明は、使用するカメラの取扱説明書をご覧ください。

5.5.14.1. 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
hCam	[in]	カメラのカメラハンドルです。
pdTimerDurationMin	[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：マイクロ秒）
pdTimerDurationMax	[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：マイクロ秒）

[戻り値]

実行結果を返します。　戻り値は、5.8. ステータスコード を参照してください。

[備考]

TimerDuration レジスタ（またはノード）が実装されていないカメラで実行するとエラーステータス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。

5.5.14.2. GetCamTimerDuration

カメラの Timer0Active 信号の幅を取得します。

[構文]

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

[パラメータ]

パラメータ		内　容
hCam	[in]	カメラのカメラハンドルです。
pdTimerDuration	[out]	取得した Timer0Active 信号の幅を格納する変数へのポインタです。（単位：マイクロ秒）

[戻り値]

実行結果を返します。　戻り値は、5.8. ステータスコード を参照してください。

[備考]

TimerDuration レジスタ（またはノード）が実装されていないカメラで実行するとエラーステータス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。

5.5.14.3. 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
hCam	[in]	カメラのカメラハンドルです。
dTimerDuration	[in]	設定する Timer0Active 信号の幅です。（単位：マイクロ秒）

[戻り値]

実行結果を返します。　戻り値は、5.8. ステータスコード を参照してください。

[備考]

TimerDuration レジスタ（またはノード）が実装されていないカメラで実行するとエラーステータス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。

5.5.14.4. 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
hCam	[in]	カメラのカメラハンドルです。
pdTimerDelayMin	[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：マイクロ秒）
pdTimerDelayMax	[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：マイクロ秒）

[戻り値]

実行結果を返します。　戻り値は、5.8. ステータスコード を参照してください。

[備考]

TimerDelay レジスタ（またはノード）が実装されていないカメラで実行するとエラーステータス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。

5.5.14.5. GetCamTimerDelay

カメラの Timer0Active 信号の遅延量を取得します。

[構文]

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

[パラメータ]

パラメータ		内　容
hCam	[in]	カメラのカメラハンドルです。
pdTimerDelay	[out]	取得した Timer0Active 信号の遅延量を格納する変数へのポインタです。（単位：マイクロ秒）

[戻り値]

実行結果を返します。　戻り値は、5.8. ステータスコード を参照してください。

[備考]

TimerDelay レジスタ（またはノード）が実装されていないカメラで実行するとエラーステータス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。

5.5.14.6. 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
hCam	[in]	カメラのカメラハンドルです。
dTimerDelay	[in]	設定する Timer0Active 信号の遅延量です。 （単位：マイクロ秒）

[戻り値]

実行結果を返します。　戻り値は、5.8. ステータスコード を参照してください。

[備考]

TimerDelay レジスタ（またはノード）が実装されていないカメラで実行するとエラーステータス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。

5.5.14.7. GetCamTimerTriggerSource

カメラの Timer0Active 信号の基準信号を取得します。

[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
hCam	[in]	カメラのカメラハンドルです。
peTimerTriggerSource	[out]	取得した Timer0Active 信号の基準信号を格納する変数へのポインタです。

[*CAM_TIMER_TRIGGER_SOURCE_TYPE* 列挙子]

メンバ	内容
CAM_TIMER_TRIGGER_SOURCE_OFF	Timer 出力は無効です。
CAM_TIMER_TRIGGER_SOURCE_LINE0_ACTIVE	Line0 入力より Timer がスタートします。
CAM_TIMER_TRIGGER_SOURCE_FRAME_TRIGGER	トリガ受付より Timer がスタートします。
CAM_TIMER_TRIGGER_SOURCE_EXPOSURE_START	露光開始より Timer がスタートします。
CAM_TIMER_TRIGGER_SOURCE_EXPOSURE_END	露光終了より Timer がスタートします。

[戻り値]

実行結果を返します。　戻り値は、5.8. ステータスコード を参照してください。

[備考]

TimerTriggerSource レジスタ（またはノード）が実装されていないカメラで実行するとエラーステータス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。

5.5.14.8. 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
hCam	[in]	カメラのカメラハンドルです。
eTimerTriggerSource	[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、5.8. ステータスコード を参照してください。

[備考]

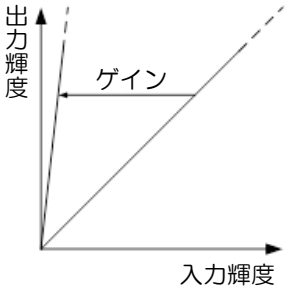
TimerTriggerSource レジスタ（またはノード）が実装されていないカメラで実行するとエラーステータスがリターンされます。

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 TimerConrtol 機能に関する詳しい説明は、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。

5.5.15.Gain

カメラのゲイン調整機能の制御を行います。



ゲインを設定することで、映像輝度の倍率を変更することができます。制御方式としてマニュアルゲイン（MANUAL）と自動ゲイン制御（AGC）が利用可能です。AGC では被写体の明るさに応じてゲインを自動で調整します。

カメラのゲイン調整機能に関する詳しい説明は、使用するカメラの取扱説明書をご覧ください。

5.5.15.1. 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
hCam	[in]	カメラのカメラハンドルです。
pdGainMin	[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 または 倍）
pdGainMax	[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 または 倍）

[戻り値]

実行結果を返します。　戻り値は、5.8. ステータスコード を参照してください。

[備考]

Gain レジスタ（またはノード）が実装されていないカメラで実行するとエラーステータス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。

5.5.15.2. GetCamGain

カメラのゲインを取得します。

[構文]

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

[パラメータ]

パラメータ		内　容
hCam	[in]	カメラのカメラハンドルです。
pdGain	[out]	取得したゲインを格納する変数へのポインタです。（単位：dB または 倍）

[戻り値]

実行結果を返します。　戻り値は、5.8. ステータスコード を参照してください。

[備考]

Gain レジスタ（またはノード）が実装されていないカメラで実行するとエラーステータス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。

5.5.15.3. 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
hCam	[in]	カメラのカメラハンドルです。
dGain	[in]	設定するゲインです。（単位：dB または 倍）

[戻り値]

実行結果を返します。　戻り値は、5.8. ステータスコード を参照してください。

[備考]

Gain レジスタ（またはノード）が実装されていないカメラで実行するとエラーステータス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。

5.5.15.4. GetCamGainAuto

カメラの AGC（Automatic gain control） 動作モードを取得します。

[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
hCam	[in]	カメラのカメラハンドルです。
peGainAuto	[out]	取得した AGC 動作モードを格納する変数へのポインタです。

[*CAM_GAIN_AUTO_TYPE* 列挙子]

メンバ	内容
CAM_GAIN_AUTO_OFF	マニュアルゲイン制御（MANUAL）
CAM_GAIN_AUTO_AUTO	自動ゲイン制御（AGC）

[戻り値]

実行結果を返します。　戻り値は、5.8. ステータスコード を参照してください。

[備考]

Gain または GainAuto レジスタ（またはノード）が実装されていないカメラで実行するとエラーステータス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。

5.5.15.5. SetCamGainAuto

カメラの AGC（Automatic gain control） 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
hCam	[in]	カメラのカメラハンドルです。
eGainAuto	[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、5.8. ステータスコード を参照してください。

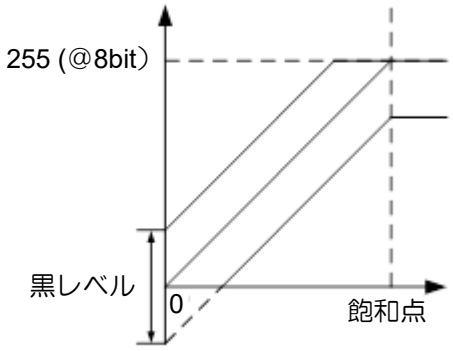
[備考]

Gain または GainAuto レジスタ（またはノード）が実装されていないカメラで実行するとエラーステータス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。

5.5.16.BlackLevel

カメラの黒レベル調整機能の制御を行います。



カメラの黒レベル調整機能に関する詳しい説明は、使用するカメラの取扱説明書をご覧ください。

5.5.16.1. 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、5.8. ステータスコード を参照してください。

[備考]

BlackLevel レジスタ（またはノード）が実装されていないカメラで実行するとエラーステータス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。

5.5.16.2. GetCamBlackLevel

カメラの黒レベルを取得します。

[構文]

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

[パラメータ]

パラメータ		内　容
hCam	[in]	カメラのカメラハンドルです。
pdBlackLevel	[out]	取得した黒レベル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、5.8. ステータスコード を参照してください。

[備考]

BlackLevel レジスタ（またはノード）が実装されていないカメラで実行するとエラーステータス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。

5.5.16.3. 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
hCam	[in]	カメラのカメラハンドルです。
dBlackLevel	[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、5.8. ステータスコード を参照してください。

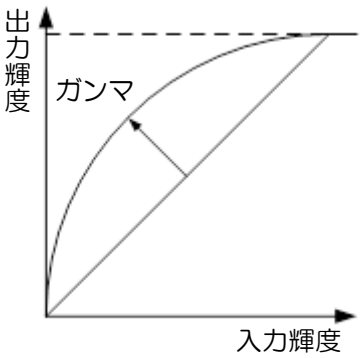
[備考]

BlackLevel レジスタ（またはノード）が実装されていないカメラで実行するとエラーステータス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。

5.5.17.Gamma

カメラのガンマ補正機能の制御を行います。



カメラのガンマ補正機能に関する詳しい説明は、使用するカメラの取扱説明書をご覧ください。

5.5.17.1. GetCamGammaMinMax

カメラのガンマ補正値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
hCam	[in]	カメラのカメラハンドルです。
pdGammaMin	[out]	取得した最小値を格納する変数へのポインタです。
pdGammaMax	[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、5.8. ステータスコード を参照してください。

[備考]

Gamma レジスタ（またはノード）が実装されていないカメラで実行するとエラーステータス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。

5.5.17.2. GetCamGamma

カメラのガンマ補正値を取得します。

[構文]

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

[パラメータ]

パラメータ		内　容
hCam	[in]	カメラのカメラハンドルです。
pdGamma	[out]	取得したガンマ補正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、5.8. ステータスコード を参照してください。

[備考]

Gamma レジスタ（またはノード）が実装されていないカメラで実行するとエラーステータス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。

5.5.17.3. 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
hCam	[in]	カメラのカメラハンドルです。
dGamma	[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、5.8. ステータスコード を参照してください。

[備考]

Gamma レジスタ（またはノード）が実装されていないカメラで実行するとエラーステータスがリターンされます。

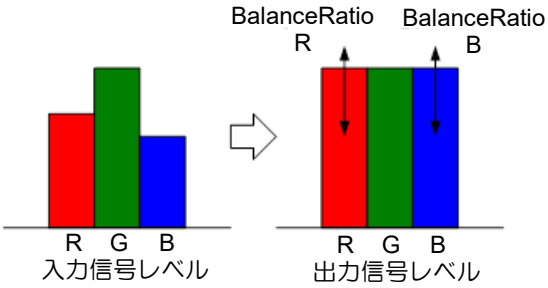
TeliCamAPI.h をインクルードする必要があります。

5.5.18.WhiteBalance (BalanceRatio / BalanceWhiteAuto)

カメラのホワイトバランス調整機能の制御を行います。

この章のメソッド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。



カメラのホワイトバランス調整機能に関する詳しい説明は、使用するカメラの取扱説明書をご覧ください。

5.5.18.1. 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
hCam	[in]	カメラのカメラハンドルです。
eSelector	[in]	ホワイトバランスゲイン設定の対象となる要素です。
pdBalanceRatioMin	[out]	取得した最小値を格納する変数へのポインタです。
pdBalanceRatioMax	[out]	取得した最大値を格納する変数へのポインタです。

[*CAM_BALANCE_RATIO_SELECTOR_TYPE* 列挙子]

メンバ	内容
CAM_BALANCE_RATIO_SELECTOR_RED	BalanceRatio = Red Gain
CAM_BALANCE_RATIO_SELECTOR_BLUE	BalanceRatio = Blue Gain

[戻り値]
実行結果を返します。　戻り値は、5.8. ステータスコード を参照してください。

[備考]

WhiteBalance* レジスタ（または BalanceRatioSelector / BalanceRatio ノード）が実装されていないカメラで実行するとエラーステータス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。

5.5.18.2. GetCamBalanceRatio

カメラのホワイトバランスゲイン（倍率）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
hCam	[in]	カメラのカメラハンドルです。
eSelector	[in]	ホワイトバランスゲイン設定の対象となる要素です。
pdBalanceRatio	[out]	取得したホワイトバランスゲイン（倍率）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、5.8. ステータスコード を参照してください。

[備考]
WhiteBalance* レジスタ（または BalanceRatioSelector / BalanceRatio ノード）が実装されていないカメラで実行するとエラーステータス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。

5.5.18.3. 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
hCam	[in]	カメラのカメラハンドルです。
eSelector	[in]	ホワイトバランスゲイン設定の対象となる要素です。
dBalanceRatio	[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、5.8. ステータスコード を参照してください。

[備考]

WhiteBalance* レジスタ（または BalanceRatioSelector / BalanceRatio ノード）が実装されていないカメラで実行するとエラーステータス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。

5.5.18.4. GetCamBalanceWhiteAuto

カメラのホワイトバランスゲイン自動調整モードを取得します。

[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
hCam	[in]	カメラのカメラハンドルです。
peBalanceWhiteAuto	[out]	取得したホワイトバランスゲイン自動調整モードを格納する変数へのポインタです。

[*CAM_BALANCE_WHITE_AUTO_TYPE* 列挙子]

メンバ	内容
CAM_BALANCE_WHITE_AUTO_OFF	待機状態
CAM_BALANCE_WHITE_AUTO_CONTINUOUS	ホワイトバランスゲインを自動調整しつづけます。
CAM_BALANCE_WHITE_AUTO_ONCE	一度だけホワイトバランスゲインを自動調整します。

[戻り値]

実行結果を返します。　戻り値は、5.8. ステータスコード を参照してください。

[備考]

WhiteBalance* レジスタ（または BalanceWhiteAuto ノード）が実装されていないカメラで実行するとエラーステータス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。

5.5.18.5. 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
hCam	[in]	カメラのカメラハンドルです。
BalanceWhiteAuto	[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、5.8. ステータスコード を参照してください。

[備考]

WhiteBalance* レジスタ（または BalanceWhiteAuto ノード）が実装されていないカメラで実行するとエラーステータスがリターンされます。

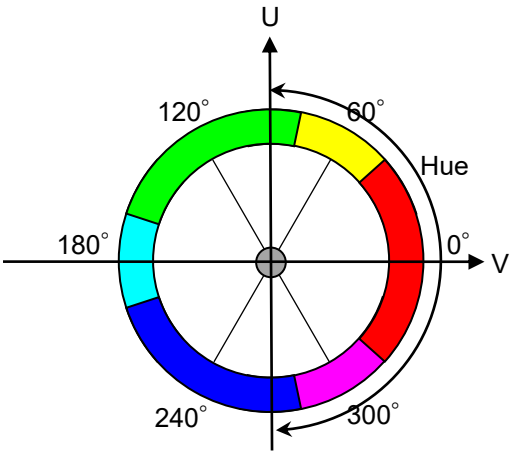
TeliCamAPI.h をインクルードする必要があります。

5.5.19.Hue

カメラの色相調整機能の制御を行います。

この章のメソッドはカラーカメラで使用できます。

カメラの色相調整機能に関する詳しい説明は、使用するカメラの取扱説明書をご覧ください。



5.5.19.1. 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
hCam	[in]	カメラのカメラハンドルです。
pdHueMin	[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：°）
pdHueMax	[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、5.8. ステータスコード を参照してください。

[備考]

Hue レジスタ（またはノード）が実装されていないカメラで実行するとエラーステータス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。

5.5.19.2. 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
hCam	[in]	カメラのカメラハンドルです。
pdHue	[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、5.8. ステータスコード を参照してください。

[備考]

Hue レジスタ（またはノード）が実装されていないカメラで実行するとエラーステータス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。

5.5.19.3. 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
hCam	[in]	カメラのカメラハンドルです。
dHue	[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、5.8. ステータスコード を参照してください。

[備考]

Hue レジスタ（またはノード）が実装されていないカメラで実行するとエラーステータス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。

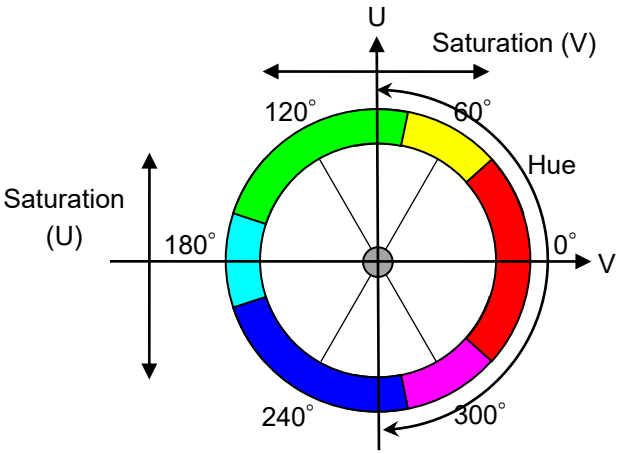
5.5.20.Saturation

カメラの彩度調整機能の制御を行います。

この章のメソッドはカラーカメラで使用できます。

対象となる要素（U/V）を選択して［倍］単位で設定するカメラと、U/V 要素共通で［%］単位で設定するカメラがあります。

カメラの彩度調整機能に関する詳しい説明は、使用するカメラの取扱説明書をご覧ください。



5.5.20.1. GetCamSaturationSelector

カメラの彩度調整の対象となる要素の設定値を取得します。

[構文]

```
CAM_API_STATUS  GetCamSaturationSelector (
  CAM_HANDLE                    hCam,
  CAM_SATURATION_SELECTOR_TYPE *peSelector
);
```

[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。
peSelector	[out]	取得した要素設定値を格納する変数へのポインタです。

[*CAM_SATURATION_SELECTOR_TYPE* 列挙子]

メンバ	内容
CAM_SATURATION_SELECTOR_U	Saturation = U Gain
CAM_SATURATION_SELECTOR_V	Saturation = V Gain

[戻り値]

実行結果を返します。　戻り値は、5.8. ステータスコード を参照してください。

[備考]

SaturationSelector レジスタ（またはノード）が実装されていないカメラで実行するとエラーステータス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。

5.5.20.2. SetCamSaturationSelector

カメラの彩度調整の対象となる要素を設定します。

[構文]

```
CAM_API_STATUS  SetCamSaturationSelector (
  CAM_HANDLE                    hCam,
  CAM_SATURATION_SELECTOR_TYPE  eSelector
);
```

[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。
eTriggerSource	[in]	設定する要素設定値です。

[戻り値]

実行結果を返します。　戻り値は、5.8. ステータスコード を参照してください。

[備考]

SaturationSelector レジスタ（またはノード）が実装されていないカメラで実行するとエラーステータス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。

5.5.20.3. 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
hCam	[in]	カメラのカメラハンドルです。
pdSaturationMin	[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： % または 倍）
pdSaturationMax	[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： % または 倍）

[戻り値]

実行結果を返します。　戻り値は、5.8. ステータスコード を参照してください。

[備考]

Saturation レジスタ（またはノード）が実装されていないカメラで実行するとエラーステータスがリターンされます。

対象となる要素（U/V）を選択して［倍］単位で設定するカメラと、［%］単位で U/V 要素共通で設定するカメラがあります。
対象となる要素（U/V）を選択して設定するカメラの場合は、本関数を実行する前に SetCamSaturationSelector() により要素（U/V）を選択してください。
カメラの彩度調整機能に関する詳しい説明は、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。

5.5.20.4. 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
hCam	[in]	カメラのカメラハンドルです。
pdSaturation	[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： % または 倍）

[戻り値]

実行結果を返します。　戻り値は、5.8. ステータスコード を参照してください。

[備考]

Saturation レジスタ（またはノード）が実装されていないカメラで実行するとエラーステータスがリターンされます。

対象となる要素（U/V）を選択して［倍］単位で設定するカメラと、［%］単位で U/V 要素共通で設定するカメラがあります。
対象となる要素（U/V）を選択して設定するカメラの場合は、本関数を実行する前に SetCamSaturationSelector() により要素（U/V）を選択してください。
カメラの彩度調整機能に関する詳しい説明は、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。

5.5.20.5. 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
hCam	[in]	カメラのカメラハンドルです。
dSaturation	[in]	設定するカメラの彩度です。（単位： % または 倍）

[戻り値]

実行結果を返します。　戻り値は、5.8. ステータスコード を参照してください。

[備考]

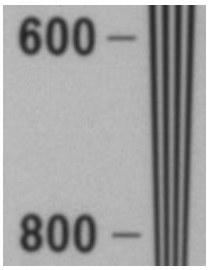
Saturation レジスタ（またはノード）が実装されていないカメラで実行するとエラーステータスがリターンされます。

対象となる要素（U/V）を選択して［倍］単位で設定するカメラと、［%］単位で U/V 要素共通で設定するカメラがあります。
対象となる要素（U/V）を選択して設定するカメラの場合は、本関数を実行する前に SetCamSaturationSelector() により要素（U/V）を選択してください。
カメラの彩度調整機能に関する詳しい説明は、使用するカメラの取扱説明書をご覧ください。

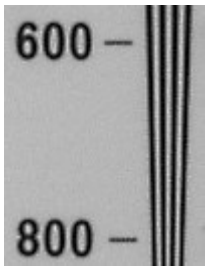
TeliCamAPI.h をインクルードする必要があります。

5.5.21.Sharpness

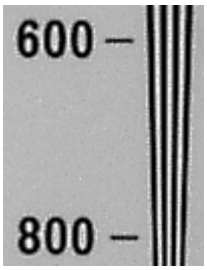
カメラの画像のエッジ強度を調整します。



Sharpness=0



Sharpness=4



Sharpness=7

カメラの画像のエッジ強度の説明は、使用するカメラの取扱説明書をご覧ください。

5.5.21.1. GetCamSharpnessMinMax

カメラの画像のエッジ強度（sharpness）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
hCam	[in]	カメラのカメラハンドルです。
puiSharpnessMin	[out]	取得した最小値を格納する変数へのポインタです。
puiSharpnessMax	[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、5.8. ステータスコード を参照してください。

[備考]

Sharpness レジスタ（またはノード）が実装されていないカメラで実行するとエラーステータス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。

5.5.21.2. GetCamSharpness

カメラの画像のエッジ強度（sharpness）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
hCam	[in]	カメラのカメラハンドルです。
puiSharpness	[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、5.8. ステータスコード を参照してください。

[備考]

Sharpness レジスタ（またはノード）が実装されていないカメラで実行するとエラーステータス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。

5.5.21.3. SetCamSharpness

カメラの画像のエッジ強度（sharpness）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
hCam	[in]	カメラのカメラハンドルです。
uiSharpness	[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、5.8. ステータスコード を参照してください。

[備考]

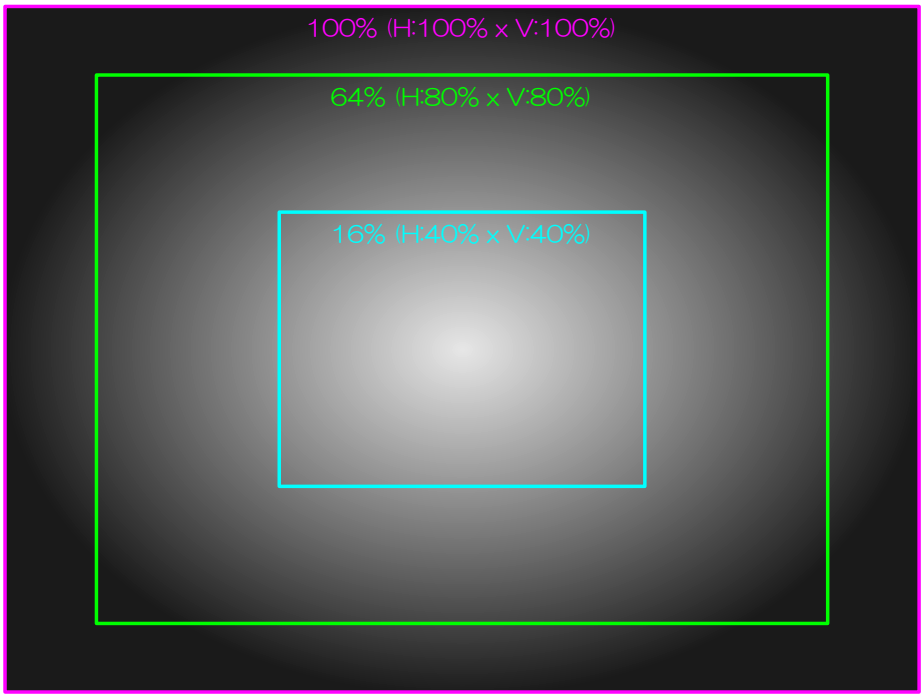
Sharpness レジスタ（またはノード）が実装されていないカメラで実行するとエラーステータスがリターンされます。

TeliCamAPI.h をインクルードする必要があります。

5.5.22.ALCControl

カメラの ALC 動作の制御を行います。

• ALCPhotometricAreaSize は輝度を測定するための測光エリアサイズを定義します。



測光エリアサイズのイメージ（それぞれ 100%、64%、16%で設定した場合）

• ALCExposureValue は収束値の補正値を定義します。
ALC 動作収束補正値設定による最終的な収束値は下記の式により求められます。

$$\text{最終収束値} = \text{８４（基準輝度）} \times 2^{\text{ALCExposureValue}}$$

カメラの ALC 動作に関する詳しい説明は、使用するカメラの取扱説明書をご覧ください。

5.5.22.1. GetCamALCPhotometricAreaSizeMinMax

カメラの ALC 動作の映像輝度を設定するエリアサイズの最小値と最大値を取得します。

[構文]

```
CAM_API_STATUS  GetCamALCPhotometricAreaSizeMinMax (
  CAM_HANDLE        hCam,
  float64_t         *pdPercentMin,
  float64_t         *pdPercentMax
);
```

[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。
pdPercentMin	[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：%）
pdPercentMax	[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、5.8. ステータスコード を参照してください。

[備考]

ALCPhotometricAreaSize レジスタ（またはノード）が実装されていないカメラで実行するとエラーステータス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。

5.5.22.2. GetCamALCPhotometricAreaSize

カメラの ALC 動作の映像輝度を設定するエリアサイズを取得します。

[構文]

```
CAM_API_STATUS  GetCamALCPhotometricAreaSize (
  CAM_HANDLE        hCam,
  float64_t         *pdPercent
);
```

[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。
pdPercent	[out]	取得したエリアサイズ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、5.8. ステータスコード を参照してください。

[備考]

ALCPhotometricAreaSize レジスタ（またはノード）が実装されていないカメラで実行するとエラーステータス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。

5.5.22.3. SetCamALCPhotometricAreaSize

カメラの ALC 動作の映像輝度を設定するエリアサイズを設定します。

[構文]

```
CAM_API_STATUS  SetCamALCPhotometricAreaSize (
  CAM_HANDLE        hCam,
  float64_t         dPercent
);
```

[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。
dPercent	[in]	設定するエリアサイズです。（単位：%）

[戻り値]

実行結果を返します。　戻り値は、5.8. ステータスコード を参照してください。

[備考]

ALCPhotometricAreaSize レジスタ（またはノード）が実装されていないカメラで実行するとエラーステータス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。

5.5.22.4. GetCamALCExposureValueMinMax

カメラの ALC 動作の映像輝度収束補正値の最小値と最大値を取得します。

[構文]

```
CAM_API_STATUS  GetCamALCExposureValueMinMax (
  CAM_HANDLE        hCam,
  float64_t         *pdExposureValueMin,
  float64_t         *pdExposureValueMax
);
```

[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。
pdExposureValueMin	[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：EV）
pdExposureValueMax	[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：EV）

[戻り値]

実行結果を返します。　戻り値は、5.8. ステータスコード を参照してください。

[備考]

ALCExposureValue レジスタ（またはノード）が実装されていないカメラで実行するとエラーステータス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。

5.5.22.5. GetCamALCPhotometricAreaSize

カメラの ALC 動作の映像輝度収束補正値を取得します。

[構文]

```
CAM_API_STATUS  GetCamALCPhotometricAreaSize (
  CAM_HANDLE        hCam,
  float64_t         *pdExposureValue
);
```

[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。
pdExposureValue	[out]	取得した映像輝度収束補正値を格納する変数へのポインタです。 （単位：EV）

[戻り値]

実行結果を返します。　戻り値は、5.8. ステータスコード を参照してください。

[備考]

ALCExposureValue レジスタ（またはノード）が実装されていないカメラで実行するとエラーステータス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。

5.5.22.6. SetCamALCPhotometricAreaSize

カメラの ALC 動作の映像輝度収束補正値を設定します。

[構文]

```
CAM_API_STATUS  SetCamALCPhotometricAreaSize (
  CAM_HANDLE        hCam,
  float64_t         dExposureValue
);
```

[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。
dExposureValue	[in]	設定する映像輝度収束補正値です。（単位：EV）

[戻り値]

実行結果を返します。　戻り値は、5.8. ステータスコード を参照してください。

[備考]

ALCExposureValue レジスタ（またはノード）が実装されていないカメラで実行するとエラーステータス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。

5.5.23.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。

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。

$$\begin{pmatrix} R' \\ G' \\ B' \end{pmatrix} = \begin{pmatrix} 1 & -mask\_rg & -mask\_rb \\ -mask\_gr & 1 & -mask\_gb \\ -mask\_br & -mask\_bg & 1 \end{pmatrix} \begin{pmatrix} R & (G-R) & (B-R) \\ (R-G) & G & (B-G) \\ (R-B) & (G-B) & B \end{pmatrix}$$

$$R' = (1 - mask\_rg - mask\_rb) \times R + mask\_rg \times G + mask\_rb \times B$$

$$G' = mask\_gr \times R + (1 - mask\_gr - mask\_gb) \times G + mask\_gb \times B$$

$$B' = mask\_br \times R + mask\_bg \times G + (1 - mask\_br \hat{} mask\_bg) \times B$$

GenICam GenApi では「SelectoorI」と「SelectorJ」の２個のセレクタを使用して 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
SelectorI = R		mask_rg	mask_rb
SelectorI = G	mask_gr		mask_gb
SelectorI = B	mask_br	mask_bg	

本章の関数は「SelectorI」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。

カメラの色補正マトリックス調整機能に関する詳しい説明は、使用するカメラの取扱説明書をご覧ください。

5.5.23.1. 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
hCam	[in]	カメラのカメラハンドルです。
eType	[in]	取得する色補正マトリクスの要素です。
pdMatrixMin	[out]	取得した最小値を格納する変数へのポインタです。
pdMatrixMax	[out]	取得した最大値を格納する変数へのポインタです。

[*CAM_COLOR_CORRECTION_MATRIX_TYPE* 列挙子]

メンバ	内容
CAM_COLOR_CORRECTION_MATRIX_RG	SelectorI = R , SelectorJ = G
CAM_COLOR_CORRECTION_MATRIX_RB	SelectorI = R , SelectorJ = B
CAM_COLOR_CORRECTION_MATRIX_GR	SelectorI = G , SelectorJ = R
CAM_COLOR_CORRECTION_MATRIX_GB	SelectorI = G , SelectorJ = B
CAM_COLOR_CORRECTION_MATRIX_BR	SelectorI = B , SelectorJ = R
CAM_COLOR_CORRECTION_MATRIX_BG	SelectorI = B , SelectorJ = G

[戻り値]

実行結果を返します。　戻り値は、5.8. ステータスコード を参照してください。

[備考]

Masking* レジスタ、または ColorCorrectionMatrix / ColorCorrectionMatrixSelectorI / ColorCorrectionMatrixSelectorJ レジスタ（またはノード）が実装されていないカメラで実行するとエラーステータス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。

5.5.23.2. GetCamColorCorrectionMatrix

カメラの色補正マトリクスの係数を取得します。

[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
hCam	[in]	カメラのカメラハンドルです。
eType	[in]	取得する色補正マトリクスの要素です。
pdMatrix	[out]	取得した色補正マトリクスの係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、5.8. ステータスコード を参照してください。

[備考]

Masking* レジスタ、または ColorCorrectionMatrix / ColorCorrectionMatrixSelectorI / ColorCorrectionMatrixSelectorJ レジスタ（またはノード）が実装されていないカメラで実行するとエラーステータス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。

5.5.23.3. 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
hCam	[in]	カメラのカメラハンドルです。
eType	[in]	取得する色補正マトリクスの要素です。
dMatrix	[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、5.8. ステータスコード を参照してください。

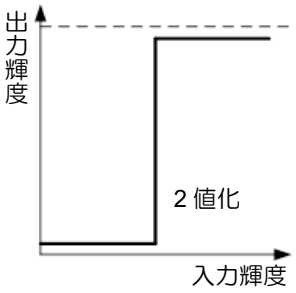
[備考]

Masking* レジスタ、または ColorCorrectionMatrix / ColorCorrectionMatrixSelectorI / ColorCorrectionMatrixSelectorJ レジスタ（またはノード）が実装されていないカメラで実行するとエラーステータス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。

5.5.24.LUT Control

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 の設定例

カメラの LUT 調整機能に関する詳しい説明は、使用するカメラの取扱説明書をご覧ください。

5.5.24.1. 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,
  bool8_t           *pbEnable
);
```

[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。
pbEnable	[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、5.8. ステータスコード を参照してください。

[備考]

LUTEnable レジスタ（またはノード）が実装されていないカメラで実行するとエラーステータス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。

5.5.24.2. 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
hCam	[in]	カメラのカメラハンドルです。
bEnable	[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、5.8. ステータスコード を参照してください。

[備考]

LUTEnable レジスタ（またはノード）が実装されていないカメラで実行するとエラーステータス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。

5.5.24.3. 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
hCam	[in]	カメラのカメラハンドルです。
uiLutIndex	[in]	LUT の入力値です。 （設定範囲：0 ～ 1023 または 0 ～ 4095）
puiLutValue	[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 または 0 ～ 4095）

[戻り値]

実行結果を返します。　戻り値は、5.8. ステータスコード を参照してください。

[備考]

LUTValueAll 、または LUTIndex と LUTValue レジスタ（またはノード）が実装されていないカメラで実行するとエラーステータスがリターンされます。

LUT の入力値と出力値の範囲は、カメラにより異なります。
カメラの LUT 調整機能に関する詳しい説明は、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。

5.5.24.4. 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
hCam	[in]	カメラのカメラハンドルです。
uiLutIndex	[in]	LUT の入力値です。 （設定範囲：0 ～ 1023 または 0 ～ 4095）
puiLutValue	[in]	設定する LUT の出力値です。 （設定範囲：0 ～ 1023 または 0 ～ 4095）

[戻り値]

実行結果を返します。　戻り値は、5.8. ステータスコード を参照してください。

[備考]

LUTValueAll 、または LUTIndex と LUTValue レジスタ（またはノード）が実装されていないカメラで実行するとエラーステータスがリターンされます。

LUT の入力値と出力値の範囲は、カメラにより異なります。
カメラの LUT 調整機能に関する詳しい説明は、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。

5.5.25.UserSetControl

カメラのユーザ設定機能の制御を行います。
カメラのユーザ設定機能に関する詳しい説明は、使用するカメラの取扱説明書をご覧ください。

5.5.25.1. 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
hCam	[in]	カメラのカメラハンドルです。
eSelector	[in]	ロードするユーザ設定チャンネルです。

[*CAM_USER_SET_SELECTOR_TYPE* 列挙子]

メンバ	内容
CAM_USER_SET_SELECTOR_DEFAULT	工場出荷設定
CAM_USER_SET_SELECTOR_USER_SET1	ユーザー設定チャンネル 1
CAM_USER_SET_SELECTOR_USER_SET2	ユーザー設定チャンネル 2
CAM_USER_SET_SELECTOR_USER_SET3	ユーザー設定チャンネル 3
CAM_USER_SET_SELECTOR_USER_SET4	ユーザー設定チャンネル 4
CAM_USER_SET_SELECTOR_USER_SET5	ユーザー設定チャンネル 5
CAM_USER_SET_SELECTOR_USER_SET6	ユーザー設定チャンネル 6
CAM_USER_SET_SELECTOR_USER_SET7	ユーザー設定チャンネル 7
CAM_USER_SET_SELECTOR_USER_SET8	ユーザー設定チャンネル 8
CAM_USER_SET_SELECTOR_USER_SET9	ユーザー設定チャンネル 9
CAM_USER_SET_SELECTOR_USER_SET10	ユーザー設定チャンネル 10
CAM_USER_SET_SELECTOR_USER_SET11	ユーザー設定チャンネル 11
CAM_USER_SET_SELECTOR_USER_SET12	ユーザー設定チャンネル 12
CAM_USER_SET_SELECTOR_USER_SET13	ユーザー設定チャンネル 13
CAM_USER_SET_SELECTOR_USER_SET14	ユーザー設定チャンネル 14
CAM_USER_SET_SELECTOR_USER_SET15	ユーザー設定チャンネル 15

[戻り値]

実行結果を返します。　戻り値は、5.8. ステータスコード を参照してください。

[備考]

UserSetSelector と UserSetCommand、または UserSetLoad レジスタ（またはノード）が実装されていないカメラで実行するとエラーステータスがリターンされます。

ロードされるパラメータは、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。

5.5.25.2. 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
hCam	[in]	カメラのカメラハンドルです。
eSelector	[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、5.8. ステータスコード を参照してください。

[備考]

UserSetSelector と UserSetCommand 、または UserSetSave レジスタ（またはノード）が実装されていないカメラで実行するとエラーステータスがリターンされます。

セーブされるパラメータ、および UserSetSave と UserSetQuickSave の違いは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。

5.5.25.3. ExecuteCamUserSetQuickSave

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揮発性メモリ（ユーザメモリ）にセーブします。

[構文]

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

[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。
eSelector	[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、5.8. ステータスコード を参照してください。

[備考]

UserSetSelector と UserSetCommand 、または UserSetQuickSave レジスタ（またはノード）が実装されていないカメラで実行するとエラーステータスがリターンされます。

ExecuteCamUserSetSave() 関数に比べ高速に処理できますが、揮発性メモリ（カメラ内部 RAM）にセーブされるため、カメラの電源が OFF されるとセーブしたデータは消えてしまいます。

セーブされるパラメータ、および UserSetSave と UserSetQuickSave の違いは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。

5.5.25.4. GetCamUserSetDefault

カメラの起動時にロードするユーザー設定チャンネルを読み出します。

[構文]

```
CAM_API_STATUS  GetCamUserSetDefault (
  CAM_HANDLE                    hCam,
  CAM_USER_SET_SELECTOR_TYPE    *peSelector
);
```

[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。
peSelector	[out]	取得したユーザー設定チャンネ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、5.8. ステータスコード を参照してください。

[備考]

UserSetDefault レジスタ（またはノード）が実装されていないカメラで実行するとエラーステータス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。

5.5.25.5. SetCamUserSetDefault

カメラの起動時にロードするユーザー設定チャンネルを設定します。

[構文]

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

[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。
eSelector	[in]	設定するユーザー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、5.8. ステータスコード を参照してください。

[備考]

UserSetDefault レジスタ（またはノード）が実装されていないカメラで実行するとエラーステータスがリターンされます。

UserSetDefault レジスタに値を設定したとき、カメラのファームウェアバージョンにより不揮発性メモリにセーブされるカメラとセーブされないカメラがあります。　不揮発性メモリにセーブされないカメラの場合は、ExecuteCamUserSetSaveAndSetDefault() 関数を使用してください。

カメラのユーザ設定機能に関する詳しい説明は、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。

5.5.25.6. 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
hCam	[in]	カメラのカメラハンドルです。
eSelector	[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。

[戻り値]

実行結果を返します。　戻り値は、5.8. ステータスコード を参照してください。

[備考]

UserSetDefault と UserSetCommand 、または UserSetSave レジスタ（またはノード）が実装されていないカメラで実行するとエラーステータスがリターンされます。

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。

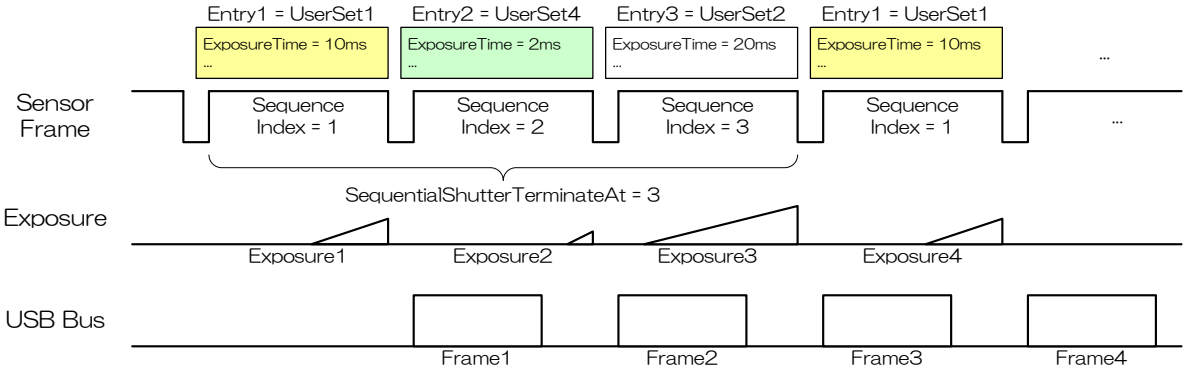
eSelector に CAM_USER_SET_SELECTOR_DEFAULT を指定したときは、工場出荷時設定がロードされ、現在のパラメータのセーブは行われません。カメラ起動時は工場出荷時設定のパラメータで起動されます。

TeliCamAPI.h をインクルードする必要があります。

5.5.26.SequentialShutterControl

カメラのシーケンシャルシャッタ機能の設定を行います。

シーケンシャルシャッタ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。



カメラのシーケンシャルシャッタ機能に関する詳しい説明は、使用するカメラの取扱説明書をご覧ください。

5.5.26.1. GetCamSequentialShutterEnable

カメラのシーケンシャルシャッタモード（有効 / 無効）を取得します。

[構文]

```
CAM_API_STATUS  GetCamSequentialShutterEnable (
  CAM_HANDLE        hCam,
  bool8_t           *pbEnable
);
```

[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。
pbEnable	[out]	取得したシーケンシャルシャッタモード（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、5.8. ステータスコード を参照してください。

[備考]

SequentialShutterEnable レジスタ（またはノード）が実装されていないカメラで実行するとエラーステータス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。

5.5.26.2. SetCamSequentialShutterEnable

カメラのシーケンシャルシャッタモード（有効 / 無効）を設定します。

[構文]

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

[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。
bEnable	[in]	設定するシーケンシャルシャッタモード（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、5.8. ステータスコード を参照してください。

[備考]

SequentialShutterEnable レジスタ（またはノード）が実装されていないカメラで実行するとエラーステータス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。

5.5.26.3. GetCamSequentialShutterTerminateAtMinMax

カメラのシーケンシャルシャッタ機能の Sequence の繰り返しを行うインデックス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
hCam	[in]	カメラのカメラハンドルです。
puiMin	[out]	取得した最小値を格納する変数へのポインタです。
puiMax	[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、5.8. ステータスコード を参照してください。

[備考]

SequentialShutterTerminateAt レジスタ（またはノード）が実装されていないカメラで実行するとエラーステータス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。

5.5.26.4. GetCamSequentialShutterTerminateAt

カメラのシーケンシャルシャッタ機能の Sequence の繰り返しを行うインデックス数を取得します。

[構文]

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

[パラメータ]

パラメータ		内　容
hCam	[in]	カメラのカメラハンドルです。
puiValue	[out]	取得した Sequence の繰り返しを行うインデックス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、5.8. ステータスコード を参照してください。

[備考]

SequentialShutterTerminateAt レジスタ（またはノード）が実装されていないカメラで実行するとエラーステータス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。

5.5.26.5. SetCamSequentialShutterTerminateAt

カメラのシーケンシャルシャッタ機能の Sequence の繰り返しを行うインデックス数を設定します。

[構文]

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

[パラメータ]

パラメータ		内　容
hCam	[in]	カメラのカメラハンドルです。
uiValue	[in]	設定する Sequence の繰り返しを行うインデックス数です。

[戻り値]

実行結果を返します。　戻り値は、5.8. ステータスコード を参照してください。

[備考]

SequentialShutterTerminateAt レジスタ（またはノード）が実装されていないカメラで実行するとエラーステータス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。

5.5.26.6. GetCamSequentialShutterIndexMinMax

カメラのシーケンシャルシャッタ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
hCam	[in]	カメラのカメラハンドルです。
puiMin	[out]	取得した最小値を格納する変数へのポインタです。
puiMax	[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、5.8. ステータスコード を参照してください。

[備考]

SequentialShutterSequenceTable または SequentialShutterIndex レジスタ（またはノード）が実装されていないカメラで実行するとエラーステータス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。

5.5.26.7. GetCamSequentialShutterEntryMinMax

カメラのシーケンシャルシャッタ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
hCam	[in]	カメラのカメラハンドルです。
puiMin	[out]	取得した最小値を格納する変数へのポインタです。
puiMax	[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、5.8. ステータスコード を参照してください。

[備考]

SequentialShutterSequenceTable または SequentialShutterEntry レジスタ（またはノード）が実装されていないカメラで実行するとエラーステータス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。

5.5.26.8. GetCamSequentialShutterEntry

カメラのシーケンシャルシャッタ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
hCam	[in]	カメラのカメラハンドルです。
uiIndex	[in]	取得するシーケンス番号です。
puiEntry	[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、5.8. ステータスコード を参照してください。

[備考]

SequentialShutterSequenceTable または SequentialShutterEntry レジスタ（またはノード）が実装されていないカメラで実行するとエラーステータス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。

5.5.26.9. SetCamSequentialShutterEntry

カメラのシーケンシャルシャッタ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
hCam	[in]	カメラのカメラハンドルです。
uiIndex	[in]	登録するシーケンス番号です。
uiEntry	[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、5.8. ステータスコード を参照してください。

[備考]

SequentialShutterSequenceTable または SequentialShutterEntry レジスタ（またはノード）が実装されていないカメラで実行するとエラーステータス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。

5.5.27.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）に関する詳しい説明は、使用するカメラの取扱説明書をご覧ください。

5.5.27.1.　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
hCam	[in]	カメラのカメラハンドルです。
pszName	[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、puiSizeに必要なバッファサイズが格納されます。
puiSize	[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、*pszName* に格納したサイズが格納されます。 *pszName* 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、5.8. ステータスコード を参照してください。

[備考]

UserDefinedName または DeviceUserID レジスタ（またはノード）が実装されていないカメラで実行するとエラーステータスがリターンされます。

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。

5.5.27.2. 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
hCam	[in]	カメラのカメラハンドルです。
pszName	[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、5.8. ステータスコード を参照してください。

[備考]

UserDefinedName または DeviceUserID レジスタ（またはノード）が実装されていないカメラで実行するとエラーステータスがリターンされます。

設定できる文字数は、カメラにより異なります。

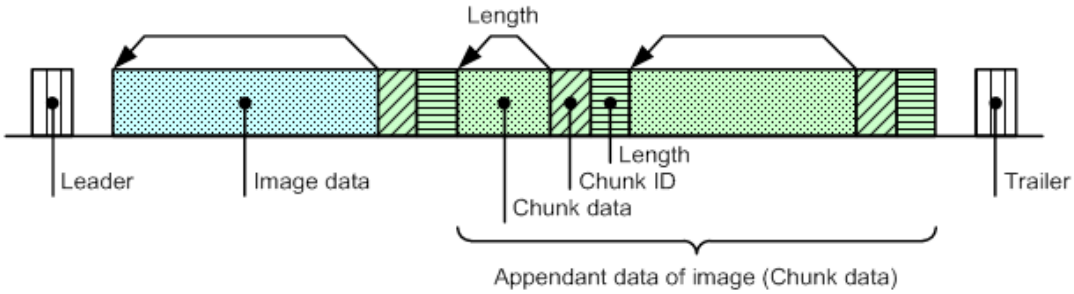
GigE Vision カメラは最大 16 文字(16byte)、USB3 Vision カメラは最大 64 文字(64byte)です。（終端の NULL 文字は 1 文字とカウントされます。）

TeliCamAPI.h をインクルードする必要があります。

5.5.28.Chunk

カメラのチャンク機能の設定を行います。

チャンクデータとは画像データ毎に付加されたタグ情報を指します。
このタグ情報はアプリケーションがデータのペイロードを解析して様々な要素を抽出・識別できるようにするものです。
有効化されたチャンクデータの内容が多くなると、そのフレーム長は長くなります。



カメラのチャンク機能に関する詳しい説明は、使用するカメラの取扱説明書をご覧ください。

GenICam を使用してチャンクデータを取得するときは、ペイロードデータが格納されているバッファを GenICam のチャンクアダプタにアタッチする必要があります。
詳細は、GenApi_ChunkAttachBuffer() の説明をご覧ください。

5.5.28.1. GetCamChunkModeActive

カメラのチャンク機能の有効状態を取得します。

[構文]

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

[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。
pbValue	[out]	取得した有効状態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、5.8. ステータスコード を参照してください。

[備考]

ChunkModeActive レジスタ（またはノード）が実装されていないカメラで実行するとエラーステータス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。

5.5.28.2. SetCamChunkModeActive

カメラのチャンク機能の有効/無効を設定します。

[構文]

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

[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。
bValue	[in]	設定する有効状態の値です。 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、5.8. ステータスコード を参照してください。

[備考]

ChunkModeActive レジスタ（またはノード）が実装されていないカメラで実行するとエラーステータス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。

5.5.28.3. GetCamChunkEnable

カメラのチャンクデータの有効状態を取得します。

[構文]

```
CAM_API_STATUS  GetCamChunkEnable (
  CAM_HANDLE                  hCam,
  CAM_CHUNK_SELECTOR_TYPE     eSelector,
  bool8_t                     *pbEnable
);
```

[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。
eSelector	[in]	設定の対象となるチャンクデータです。
pbEnable	[out]	取得した有効状態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。

[*CAM_CHUNK_SELECTOR_TYPE* 列挙子]

メンバ	内容
CAM_CAM_CHUNK_SELECTOR_BLOCK_ID	BlockID
CAM_CAM_CHUNK_SELECTOR_FRAME_BURST_TRIGGER_COUNT	FrameBurstTriggerCount
CAM_CAM_CHUNK_SELECTOR_SEQUENTIAL_SHUTTER_NUMBER	SequentialShutter number
CAM_CAM_CHUNK_SELECTOR_SEQUENTIAL_SHUTTER_ELEMENT	SequentialShutter element
CAM_CAM_CHUNK_SELECTOR_USER_AREA	User area
CAM_CAM_CHUNK_SELECTOR_EXPOSURE_TIME	Exposure time
CAM_CAM_CHUNK_SELECTOR_GAIN	Gain
CAM_CAM_CHUNK_SELECTOR_WHITE_BALANCE_R	WhiteBalanceR
CAM_CAM_CHUNK_SELECTOR_WHITE_BALANCE_B	WhiteBalanceB
CAM_CAM_CHUNK_SELECTOR_LINE_STATUS_ALL	LineStatusAll

[戻り値]

実行結果を返します。　戻り値は、5.8. ステータスコード を参照してください。

[備考]

ChunkEnableOf* 、または ChunkSelector と ChunkEnable レジスタ（またはノード）が実装されていないカメラで実行するとエラーステータス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。

5.5.28.4. SetCamChunkEnable

カメラのチャンクデータの有効状態を設定します。

[構文]

```
CAM_API_STATUS  SetCamChunkEnable (
  CAM_HANDLE                hCam,
  CAM_CHUNK_SELECTOR_TYPE   eSelector,
  bool8_t                   bEnable
);
```

[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。
eSelector	[in]	設定の対象となるチャンクデータです。
bEnable	[in]	設定する有効状態の値です。 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、5.8. ステータスコード を参照してください。

[備考]

ChunkEnableOf* または ChunkEnable レジスタ（またはノード）が実装されていないカメラで実行するとエラーステータス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。

5.5.28.5. GetCamChunkUserAreaLength

カメラの ChunkUserAreaTable の長さを取得します。

[構文]

```
CAM_API_STATUS  GetCamChunkUserAreaLength (
  CAM_HANDLE        hCam,
  uint32_t          *puiLength
);
```

[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。
puiLength	[out]	取得した ChunkUserAreaTable の長さ（byte 単位）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、5.8. ステータスコード を参照してください。

[備考]

ChunkUserArea または ChunkUserAreaTable レジスタ（またはノード）が実装されていないカメラで実行するとエラーステータス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。

5.5.28.6. GetCamChunkUserAreaTable

カメラの ChunkUserAreaTable に設定されているデータを取得します。

[構文]

```
CAM_API_STATUS  GetCamChunkUserAreaTable (
  CAM_HANDLE        hCam,
  char              *pvBuffer,
  uint32_t          uiOffset,
  uint32_t          uiSize
);
```

[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。
pvBuffer	[out]	ChunkUserAreaTableに設定されているデータを格納するバッファへのポインタです。
uiOffset	[in]	ChunkUserAreaTableのオフセット値です。（単位：byte） ChunkUserAreaTableの先頭からデータを取得する場合は 0 を設定してください。
puiSize	[in]	取得するデータのサイズです。（単位：byte） *pvBuffer*で指定したバッファのサイズより大きい値は設定しないでください。

[戻り値]

実行結果を返します。　戻り値は、5.8. ステータスコード を参照してください。

[備考]

ChunkUserArea または ChunkUserAreaTable レジスタ（またはノード）が実装されていないカメラで実行するとエラーステータス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。

5.5.28.7. SetCamChunkUserAreaTable

カメラの ChunkUserAreaTable にデータを設定します。

[構文]

```
CAM_API_STATUS  SetCamChunkUserAreaTable (
  CAM_HANDLE        hCam,
  char              *pvBuffer,
  uint32_t          uiOffset,
  uint32_t          uiSize
);
```

[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。
pvBuffer	[in]	ChunkUserAreaTableに設定するデータを格納するバッファへのポインタです。
uiOffset	[in]	ChunkUserAreaTableのオフセット値です。（単位：byte） ChunkUserAreaTableの先頭からデータを設定する場合は 0 を設定してください。
puiSize	[in]	設定するデータのサイズです。（単位：byte） *pvBuffer*で指定したバッファのサイズより大きい値は設定しないでください。

[戻り値]

実行結果を返します。　戻り値は、5.8. ステータスコード を参照してください。

[備考]

ChunkUserArea または ChunkUserAreaTable レジスタ（またはノード）が実装されていないカメラで実行するとエラーステータス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。

5.5.29.FrameSynchronization

カメラのフレーム同期の制御を行います。
フレーム同期は、USB3 Vision カメラのみ有効です。

カメラのフレーム同期に関する詳しい説明は、使用するカメラの取扱説明書をご覧ください。

5.5.29.1. GetCamFrameSynchronization

カメラのフレーム同期制御方法を取得します。

[構文]

```
CAM_API_STATUS  GetCamFrameSynchronization (
  CAM_HANDLE                      hCam,
  CAM_FRAME_SYNCHRONIZATION_TYPE  *peSync
);
```

[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。
peSync	[out]	取得したフレーム同期制御方法を格納する変数へのポインタです。

[*CAM_FRAME_SYNCHRONIZATION_TYPE* 列挙子]

メンバ	内容
CAM_FRAME_SYNCHRONIZATION_OFF	内部同期
CAM_FRAME_SYNCHRONIZATION_BUS	バス同期

[戻り値]

実行結果を返します。　戻り値は、5.8. ステータスコード を参照してください。

[備考]

FrameSynchronization レジスタ（またはノード）が実装されていないカメラで実行するとエラーステータス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。

5.5.29.2. SetCamFrameSynchronization

カメラのフレーム同期制御方法を設定します。

[構文]

```
CAM_API_STATUS  SetCamFrameSynchronization (
  CAM_HANDLE                    hCam,
  CAM_FRAME_SYNCHRONIZATION_TYPE  eSync
);
```

[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。
eSync	[in]	設定するフレーム同期制御方法です。

[戻り値]

実行結果を返します。　戻り値は、5.8. ステータスコード を参照してください。

[備考]

FrameSynchronization レジスタ（またはノード）が実装されていないカメラで実行するとエラーステータス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。

5.5.30.その他の関数

5.5.30.1. 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
hCam	[in]	カメラのカメラハンドルです。
puiCamIndex	[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、5.8. 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。

5.5.30.2. 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
hCam	[in]	カメラのカメラハンドルです。
peType	[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、5.8. 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。

5.5.30.3. GetCamSupportIIDC2

カメラが IIDC2 に準拠しているかどうかを取得します。

[構文]

```
CAM_API_STATUS  GetCamSupportIIDC2 (
  CAM_HANDLE        hCam,
  bool8_t           *pbValue
);
```

[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。
pbValue	[out]	取得した値を格納する変数へのポインタです。 true とき、カメラは IIDC 規格に準拠しています。 false のとき、カメラは IIDC 規格に準拠していません。

[戻り値]

実行結果を返します。　戻り値は、5.8. 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。

5.5.30.4. 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
hCam	[in]	カメラのカメラハンドルです。
puiValue	[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、5.8. 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。

5.6. 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. INode 系関数

5.6.1.1. GenApi_GetType

カメラのノード名（機能またはカテゴリの名称）からノードの型を取得します。

[構文]

```
CAM_API_STATUS  GenApi_GetType (
  CAM_HANDLE        hCam,
  const char        *pszNodeName,
  TC_NODE_TYPE      *peNodeType
);
```

[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。
pszNodeName	[in]	NULL で終端されたノード名です。
peNodeType	[out]	取得したノードの型を格納する変数へのポインタです。

[*TC_NODE_TYPE* 列挙子]

メンバ	内容
TC_NODE_TYPE_VALUE	IValue インターフェース
TC_NODE_TYPE_BASE	IBase インターフェース
TC_NODE_TYPE_INTEGER	IInteger インターフェース
TC_NODE_TYPE_BOOLEAN	IBoolean インターフェース
TC_NODE_TYPE_COMMAND	ICommand インターフェース
TC_NODE_TYPE_FLOAT	IFloat インターフェース
TC_NODE_TYPE_STRING	IString インターフェース
TC_NODE_TYPE_REGISTER	IRegister インターフェース
TC_NODE_TYPE_CATEGORY	ICategory インターフェース
TC_NODE_TYPE_ENUMERATION	IEnumeration インターフェース
TC_NODE_TYPE_ENUM_ENTRY	IEnumEntry インターフェース
TC_NODE_TYPE_PORT	IPort インターフェース
TC_NODE_TYPE_UNKNOWN	不明なインターフェース

[戻り値]

実行結果を返します。　戻り値は、5.8. ステータスコード を参照してください。

[備考]

TC_NODE_TYPE 列挙子は、GenApi の Types.h で定義されている EInterfaceType 列挙子に対応しています。

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;
TC_NODE_TYPE              eNodeType = TC_NODE_TYPE_UNKNOWN;
TC_NODE_ACCESS_MODE       eNodeAccessMode = TC_NODE_ACCESS_MODE_UNKNOWN;
TC_NODE_VISIBILITY        eNodeVisibility = TC_NODE_VISIBILITY_UNKNOWN;
TC_NODE_CACHING_MODE      eNodeCachingMode = TC_NODE_CACHING_MODE_UNKNOWN;
TC_NODE_REPRESENTATION    nodeRepresentation = TC_NODE_REPRESENTATION_UNKNOWN;
char                      *pszBuf = NULL;

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

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

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

__try
{
    // Get type.
    uiStatus = GenApi_GetType(hCam, "Gain", &eNodeType);
    if (uiStatus != CAM_API_STS_SUCCESS)
        return -1;
    printf("Type = %d\n", (uint32_t)eNodeType);

    // Get access mode.
    uiStatus = GenApi_GetAccessMode(hCam, "Gain", &eNodeAccessMode);
    if (uiStatus != CAM_API_STS_SUCCESS)
        return -1;
    printf("AccessMode = %d\n", (uint32_t)eNodeAccessMode);

    // Get visibility.
    uiStatus = GenApi_GetVisibility(hCam, "Gain", &eNodeVisibility);
    if (uiStatus != CAM_API_STS_SUCCESS)
        return -1;
    printf("Visibility = %d\n", (uint32_t)eNodeVisibility);

    // Get cachingMode.
    uiStatus = GenApi_GetCachingMode(hCam, "Gain", &eNodeCachingMode);
    if (uiStatus != CAM_API_STS_SUCCESS)
        return -1;
    printf("CachingMode = %d\n", (uint32_t)eNodeCachingMode);

    // Get description.
    uiSize = 0;
    uiStatus = GenApi_GetDescription(hCam, "Gain", NULL, &uiSize);
    if (uiStatus != CAM_API_STS_SUCCESS)
        return -1;
```

```
    // Allocate buffer for description data.
    pszBuf = (char *)VirtualAlloc(NULL,
                                  uiSize,
                                  MEM_RESERVE | MEM_COMMIT,
                                  PAGE_EXECUTE_READWRITE);
    if (pszBuf == NULL) {
        return -1;
    }

    uiStatus = GenApi_GetDescription(hCam, "Gain", pszBuf, &uiSize);
    if (uiStatus != CAM_API_STS_SUCCESS)
        return -1;
    printf("Description = %s\n", pszBuf);

    // Release buffer for description data.
    VirtualFree(pszBuf, 0, MEM_RELEASE);
    pszBuf = NULL;

    // Get tooltip.
    uiSize = 0;
    uiStatus = GenApi_GetToolTip(hCam, "Gain", NULL, &uiSize);
    if (uiStatus != CAM_API_STS_SUCCESS)
        return -1;

    // Allocate buffer for tooltip data.
    pszBuf = (char *)VirtualAlloc(NULL,
                                  uiSize,
                                  MEM_RESERVE | MEM_COMMIT,
                                  PAGE_EXECUTE_READWRITE);
    if (pszBuf == NULL) {
        return -1;
    }

    uiStatus = GenApi_GetToolTip(hCam, "Gain", pszBuf, &uiSize);
    if (uiStatus != CAM_API_STS_SUCCESS)
        return -1;

    printf("ToolTip = %s\n", pszBuf);

    // Release buffer for tooltip data.
    VirtualFree(pszBuf, 0, MEM_RELEASE);
    pszBuf = NULL;

    // Get representation.
    uiStatus = GenApi_GetRepresentation(hCam, "Gain", &nodeRepresentation);
    if (uiStatus != CAM_API_STS_SUCCESS)
        return -1;
    printf("Representation = %d\n", (uint32_t)nodeRepresentation);

    // Get unit.
    uiSize = 0;
    uiStatus = GenApi_GetUnit(hCam, "Gain", NULL, &uiSize);
    if (uiStatus != CAM_API_STS_SUCCESS)
        return -1;

    // Allocate buffer for unit data.
    pszBuf = (char *)VirtualAlloc(NULL,
                                  uiSize,
                                  MEM_RESERVE | MEM_COMMIT,
                                  PAGE_EXECUTE_READWRITE);
    if (pszBuf == NULL) {
        return -1;
    }

    uiStatus = GenApi_GetUnit(hCam, "Gain", pszBuf, &uiSize);
```

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

    printf("Unit = %s\n", pszBuf);

    // Release buffer for unit data.
    VirtualFree(pszBuf, 0, MEM_RELEASE);
    pszBuf = NULL;

    return 0;
}
__finally
{
    if (pszBuf != NULL) {
        free( pszBuf);
    }

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

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

5.6.1.2. GenApi_GetAccessMode

カメラのノード名（機能またはカテゴリの名称）からノードのアクセスモードを取得します。

[構文]

```
CAM_API_STATUS  GenApi_GetAccessMode (
  CAM_HANDLE            hCam,
  const char            *pszNodeName,
  TC_NODE_ACCESS_MODE   *peAccessMode
);
```

[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。
pszNodeName	[in]	NULL で終端されたノード名です。
peAccessMode	[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* 列挙子]

メンバ	内容
TC_NODE_ACCESS_MODE_NI	未実装
TC_NODE_ACCESS_MODE_NA	使用不可
TC_NODE_ACCESS_MODE_WO	書き込み専用
TC_NODE_ACCESS_MODE_RO	読み出し専用
TC_NODE_ACCESS_MODE_RW	読み書き可
TC_NODE_ACCESS_MODE_Undefined	オブジェクト未初期化
TC_NODE_ACCESS_MODE_UNKNOWN	不明なモード

[戻り値]

実行結果を返します。　戻り値は、5.8. ステータスコード を参照してください。

[備考]

TC_NODE_ACCESS_MODE 列挙子は、GenApi の Types.h で定義されている EAccessMode 列挙子に対応しています。

TeliCamAPI.h をインクルードする必要があります。

[コード例]

GenApi_GetType() のコード例を参照してください。

5.6.1.3. GenApi_GetVisibility

カメラのノード名（機能またはカテゴリの名称）からからノードの推奨可視レベルを取得します。

[構文]

```
CAM_API_STATUS  GenApi_GetVisibility (
  CAM_HANDLE            hCam,
  const char            *pszNodeName,
  TC_NODE_VISIBILITY    *peVisibility
);
```

[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。
pszNodeName	[in]	NULL で終端されたノード名です。
peVisibility	[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* 列挙子]

メンバ	内容
TC_NODE_VISIBILITY_BEGINNER	常に可視
TC_NODE_VISIBILITY_EXPERT	熟練者および専門家に可視
TC_NODE_VISIBILITY_GURU	専門家に可視
TC_NODE_VISIBILITY_INVISIBLE	不可視
TC_NODE_VISIBILITY_Undefined	オブジェクト未初期化
TC_NODE_VISIBILITY_UNKNOWN	不明

[戻り値]

実行結果を返します。　戻り値は、5.8. ステータスコード を参照してください。

[備考]

TC_NODE_VISIBILITY 列挙子は、GenApi の Types.h で定義されている EVisibility 列挙子に対応しています。

TeliCamAPI.h をインクルードする必要があります。

[コード例]

GenApi_GetType() のコード例を参照してください。

5.6.1.4. GenApi_GetCachingMode

カメラのノード名（機能またはカテゴリの名称）からノードのキャッシュ設定を取得します。

[構文]

```
CAM_API_STATUS  GenApi_GetCachingMode (
  CAM_HANDLE            hCam,
  const char            *pszNodeName,
  TC_NODE_CACHING_MODE  *peCachingMode
);
```

[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。
pszNodeName	[in]	NULL で終端されたノード名です。
peCachingMode	[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* 列挙子]

メンバ	内容
TC_NODE_CACHING_MODE_NO_CACHE	キャッシュ不可
TC_NODE_CACHING_MODE_WRITE_THROUGH	レジスタ書き込み時にキャッシュへも書き込み
TC_NODE_CACHING_MODE_WRITE_AROUND	読み出し時にキャッシュへ書き込み。 書き込み時はレジスタのみへ書き込み。
TC_NODE_CACHING_MODE_UNDEFINED	オブジェクト未初期化
TC_NODE_CACHING_MODE_UNKNOWN	不明

[戻り値]

実行結果を返します。　戻り値は、5.8. ステータスコード を参照してください。

[備考]

TC_NODE_CACHING_MODE 列挙子は、GenApi の Types.h で定義されている ECachingMode 列挙子に対応しています。
TeliCamAPI.h をインクルードする必要があります。

[コード例]

GenApi_GetType() のコード例を参照してください。

5.6.1.5. GenApi_GetDescription

カメラのノード名（機能またはカテゴリの名称）からノードの説明文字列を取得します。

[構文]

```
CAM_API_STATUS  GenApi_GetDescription (
  CAM_HANDLE          hCam,
  const char          *pszNodeName,
  char                *pszBuf,
  uint32_t            *puiSize
);
```

[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。
pszNodeName	[in]	NULL で終端されたノード名です。
pszBuf	[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、*puiSize* に必要なバッファサイズが格納されます。
puiSize	[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） *pszBuf* 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、5.8. 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]

GenApi_GetType() のコード例を参照してください。

5.6.1.6. GenApi_GetToolTip

カメラのノード名（機能またはカテゴリの名称）からノードのツールチップを取得します。

[構文]

```
CAM_API_STATUS  GenApi_GetToolTip (
  CAM_HANDLE            hCam,
  const char            *pszNodeName,
  char                  *pszBuf,
  uint32_t              *puiSize
);
```

[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。
pszNodeName	[in]	NULL で終端されたノード名です。
pszBuf	[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、*puiSize* に必要なバッファサイズが格納されます。
puiSize	[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） *pszBuf* 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、5.8. 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]

GenApi_GetType() のコード例を参照してください。

5.6.1.7. GenApi_GetRepresentation

カメラのノード名（機能またはカテゴリの名称）からノードの推奨表現を取得します。

[構文]

```
CAM_API_STATUS  GenApi_GetRepresentation (
  CAM_HANDLE                hCam,
  const char                *pszNodeName,
  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。
pszNodeName	[in]	NULL で終端されたノード名です。
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* 列挙子]

メンバ	内容
TC_NODE_REPRESENTATION_LINEAR	リニア表示のスライダ
TC_NODE_REPRESENTATION_LOGARITHMIC	対数表示のスライダ
TC_NODE_REPRESENTATION_BOOLEAN	チェックボックス
TC_NODE_REPRESENTATION_PURE_NUMBER	10 進数表示編集コントロール
TC_NODE_REPRESENTATION_HEX_NUMBER	16 進数表示編集コントロール
TC_NODE_REPRESENTATION_IPV4_ADDRESS	IP アドレス形式表示
TC_NODE_REPRESENTATION_MAC_ADDRESS	MAC アドレス形式表示
TC_NODE_REPRESENTATION_UNDEFINED	オブジェクト未初期化
TC_NODE_REPRESENTATION_UNKNOWN	不明

[戻り値]

実行結果を返します。　戻り値は、5.8. ステータスコード を参照してください。

[備考]

TC_NODE_REPRESENTATION 列挙子は、GenApi の Types.h で定義されている ERepresentation 列挙子に対応しています。
TeliCamAPI.h をインクルードする必要があります。

[コード例]

GenApi_GetType() のコード例を参照してください。

5.6.1.8. GenApi_GetUnit

カメラのノード名（機能またはカテゴリの名称）からノードの単位名を取得します。

[構文]

```
CAM_API_STATUS  GenApi_GetUnit (
  CAM_HANDLE            Cam,
  const char            *pszNodeName,
  char                  *pszBuf,
  uint32_t              *puiSize
);
```

[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。
pszNodeName	[in]	NULL で終端されたノード名です。
pszBuf	[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、*puiSize* に必要なバッファサイズが格納されます。
puiSize	[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） *pszBuf* 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、5.8. 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]

GenApi_GetType() のコード例を参照してください。

5.6.2. ICategory 型　関数

5.6.2.1. GenApi_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  GenApi_GetNumOfFeatures (
  CAM_HANDLE        hCam,
  const char        *pszCategoryName,
  uint32_t          *puiNum
);
```

[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。
pszCategoryName	[in]	NULL で終端された ICategory 型ノードの名称です。
puiNum	[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、5.8. 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ステータス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。

[コード例]

C++

```
CAM_API_STATUS   uiStatus;
uint32_t         uiNum, uiFeatureNum, i;
CAM_HANDLE       hCam = (CAM_HANDLE)NULL;
CAM_NODE_NAME    sFeatureName;

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

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

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

__try
{
    // Get num of features.
    uiStatus = GenApi_GetNumOfFeatures(hCam, "DeviceControl", &uiFeatureNum);
```

```
    if (uiStatus != CAM_API_STS_SUCCESS)
        return -1;
    printf("Num of features = %d\n", uiFeatureNum);

    for (i = 0; i < uiFeatureNum; i++) {
        // Get node name.
        uiStatus = GenApi_GetName(hCam, "DeviceControl", i, &sFeatureName);
        if (uiStatus != CAM_API_STS_SUCCESS)
            return -1;
        printf("No.%d : %s\n", i, sFeatureName.szNodeName);
    }

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

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

5.6.2.2. GenApi_GetFeatureName

ICategory 型ノードの子 Feature ノードの名称を取得します。

[構文]

```
CAM_API_STATUS  GenApi_GetFeatureName (
  CAM_HANDLE        hCam,
  const char        *pszCategoryName,
  uint32_t          uiNodeIndex,
  CAM_NODE_NAME     *peFeatureName
);
```

[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。
pszCategoryName	[in]	NULL で終端された ICategory 型ノードの名称です。
uiNodeIndex	[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 ）
peFeatureName	[out]	取得したフィーチャー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;
```

メンバ		内　容
szNodeName	[out]	ノード名です。

[戻り値]

実行結果を返します。　戻り値は、5.8. 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]

GenApi_GetNumOfFeatures() のコード例を参照してください。

5.6.3. IInteger 型　関数

5.6.3.1. GenApi_GetIntMin

IInteger 型ノードの現在有効な最小値を取得します。

[構文]

```
CAM_API_STATUS  GenApi_GetIntMin (
  CAM_HANDLE        hCam,
  const char        *pszFeatureName,
  int64_t           *pllMin
);
```

[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。
pszFeatureName	[in]	NULL で終端された IInteger 型ノードの名称です。
pllMin	[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、5.8. ステータスコード を参照してください。

[備考]

本関数は、IInteger 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ステータス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。

[コード例]

C++

```
CAM_API_STATUS   uiStatus;
uint32_t         uiNum;
CAM_HANDLE       hCam = (CAM_HANDLE)NULL;
int64_t          llMin, llMax, llInc, llRdValue, llWrValue;

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

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

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

__try
{
    // Get minimum value.
    uiStatus = GenApi_GetIntMin(hCam, "Width", &llMin);
```

```
    if (uiStatus != CAM_API_STS_SUCCESS)
        return -1;
    printf("Width Min   : %d\n", (uint32_t)llMin);

    // Get maximum value.
    uiStatus = GenApi_GetIntMax(hCam, "Width", &llMax);
    if (uiStatus != CAM_API_STS_SUCCESS)
        return -1;
    printf("      Max   : %d\n", (uint32_t)llMax);

    // Get increment value.
    uiStatus = GenApi_GetIntInc(hCam, "Width", &llInc);
    if (uiStatus != CAM_API_STS_SUCCESS)
        return -1;
    printf("      Inc   : %d\n", (uint32_t)llInc);

    // Get value.
    uiStatus = GenApi_GetIntValue(hCam, "Width", &llRdValue);
    if (uiStatus != CAM_API_STS_SUCCESS)
        return -1;
    printf("      Before Value : %d\n", (uint32_t)llRdValue);

    // Set value.
    llWrValue = llMin + llInc;
    uiStatus = GenApi_SetIntValue(hCam, "Width", llWrValue);
    if (uiStatus != CAM_API_STS_SUCCESS)
        return -1;

    // Get value.
    uiStatus = GenApi_GetIntValue(hCam, "Width", &llRdValue);
    if (uiStatus != CAM_API_STS_SUCCESS)
        return -1;
    printf("      After Value : %d\n", (uint32_t)llRdValue);

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

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

5.6.3.2. GenApi_GetIntMax

IInteger 型ノードの現在有効な最大値を取得します。

[構文]

```
CAM_API_STATUS  GenApi_GetIntMax (
  CAM_HANDLE        hCam,
  const char        *pszFeatureName,
  int64_t           *pllMax
);
```

[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。
pszFeatureName	[in]	NULL で終端された IInteger 型ノードの名称です。
pllMax	[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、5.8. ステータスコード を参照してください。

[備考]

本関数は、IInteger 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ステータス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。

[コード例]

GenApi_GetIntMin() のコード例を参照してください。

5.6.3.3. GenApi_GetIntInc

IInteger 型ノードの増加量を取得します。

[構文]

```
CAM_API_STATUS  GenApi_GetIntInc (
  CAM_HANDLE        hCam,
  const char        *pszFeatureName,
  int64_t           *pllInc
);
```

[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。
pszFeatureName	[in]	NULL で終端された IInteger 型ノードの名称です。
pllInc	[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、5.8. ステータスコード を参照してください。

[備考]

本関数は、IInteger 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ステータス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。

[コード例]

GenApi_GetIntMin() のコード例を参照してください。

5.6.3.4. GenApi_GetIntValue

IInteger 型ノードの値を取得します。

[構文]

```
CAM_API_STATUS  GenApi_GetIntValue (
  CAM_HANDLE        hCam,
  const char        *pszFeatureName,
  int64_t           *pllValue,
  bool8_t           bVerify = false,
  bool8_t           bIgnoreCache = false,
);
```

[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。
pszFeatureName	[in]	NULL で終端された IInteger 型ノードの名称です。
pllValue	[out]	取得した値を格納する変数へのポインタです。
bVerify	[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。
bIgnoreCache	[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、5.8. ステータスコード を参照してください。

[備考]

本関数は、IInteger 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ステータス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。

[コード例]

GenApi_GetIntMin() のコード例を参照してください。

5.6.3.5. GenApi_SetIntValue

IInteger 型ノードの値を設定します。

[構文]

```
CAM_API_STATUS  GenApi_SetIntValue (
  CAM_HANDLE        hCam,
  const char        *pszFeatureName,
  int64_t           llValue,
  bool8_t           bVerify = true
);
```

[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。
pszFeatureName	[in]	NULL で終端された IInteger 型ノードの名称です。
llValue	[in]	設定する値です。
bVerify	[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、5.8. ステータスコード を参照してください。

[備考]

本関数は、IInteger 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ステータス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。

[コード例]

GenApi_GetIntMin() のコード例を参照してください。

5.6.4. IFloat 型　関数

5.6.4.1. GenApi_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  GenApi_GetFloatMin (
  CAM_HANDLE        hCam,
  const char        *pszFeatureName,
  float64_t         *pdMin
);
```

[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。
pszFeatureName	[in]	NULL で終端された IFloat 型ノードの名称です。
pdMin	[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、5.8. 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ステータス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。

[コード例]

C++

```
CAM_API_STATUS            uiStatus;
uint32_t                  uiNum;
CAM_HANDLE                hCam = (CAM_HANDLE)NULL;
float64_t                 dMin, dMax, dInc, dRdValue, dWrValue;
bool8_t                   bInc;
TC_NODE_DISPLAY_NOTATION  eDisplayNotation;
int64_t                   llPrecision;

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

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

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

__try
```

```
{
    // Get minimum value.
    uiStatus = GenApi_GetFloatMin(hCam, "Gain", &dMin);
    if (uiStatus != CAM_API_STS_SUCCESS)
        return -1;
    printf("Width Min   : %f\n", dMin);

    // Get maximum value.
    uiStatus = GenApi_GetFloatMax(hCam, "Gain", &dMax);
    if (uiStatus != CAM_API_STS_SUCCESS)
        return -1;
    printf("      Max   : %f\n", dMax);

    // Confirm whether the float node has a constant increment.
    uiStatus = GenApi_GetFloatHasInc(hCam, "Gain", &bInc);
    if (uiStatus != CAM_API_STS_SUCCESS)
        return -1;

    if (bInc) {
        // Get increment value.
        uiStatus = GenApi_GetFloatInc(hCam, "Gain", &dInc);
        if (uiStatus != CAM_API_STS_SUCCESS)
            return -1;
        printf("      Inc   : %f\n", dInc);
    }

    // Get display notation.
    uiStatus = GenApi_GetFloatDisplayNotation(hCam, "Gain", &eDisplayNotation);
    if (uiStatus != CAM_API_STS_SUCCESS)
        return -1;
    printf("      Display Notation : %d\n", eDisplayNotation);

    // Get display precision.
    uiStatus = GenApi_GetFloatDisplayPrecision(hCam, "Gain", &llPrecision);
    if (uiStatus != CAM_API_STS_SUCCESS)
        return -1;
    printf("      Display Precision : %d\n", (uint32_t)llPrecision);

    // Get value.
    uiStatus = GenApi_GetFloatValue(hCam, "Gain", &dRdValue);
    if (uiStatus != CAM_API_STS_SUCCESS)
        return -1;
    printf("      Before Value : %f\n", dRdValue);

    // Set value.
    dWrValue = (dMin + dMax) / 2.0f;
    uiStatus = GenApi_SetFloatValue(hCam, "Gain", dWrValue);
    if (uiStatus != CAM_API_STS_SUCCESS)
        return -1;

    // Get value.
    uiStatus = GenApi_GetFloatValue(hCam, "Gain", &dRdValue);
    if (uiStatus != CAM_API_STS_SUCCESS)
        return -1;
    printf("      After Value : %f\n", dRdValue);

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



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

5.6.4.2. GenApi_GetFloatMax

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  GenApi_GetFloatMax (
  CAM_HANDLE        hCam,
  const char        *pszFeatureName,
  float64_t         *pdMax
);
```

[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。
pszFeatureName	[in]	NULL で終端された IFloat 型ノードの名称です。
dMax	[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、5.8. 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ステータス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。

[コード例]

GenApi_GetFloatMin() のコード例を参照してください。

5.6.4.3. GenApi_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  GenApi_GetFloatHasInc (
  CAM_HANDLE        hCam,
  const char        *pszFeatureName,
  bool8_t           *pbInc
);
```

[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。
pszFeatureName	[in]	NULL で終端された IFloat 型ノードの名称です。
pbInc	[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、5.8. 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ステータス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。

[コード例]

GenApi_GetFloatMin() のコード例を参照してください。

5.6.4.4. GenApi_GetFloatInc

IFloat 型ノードの増加量を取得します。

[構文]

```
CAM_API_STATUS  GenApi_GetFloatInc (
  CAM_HANDLE        hCam,
  const char        *pszFeatureName,
  float64_t         *pdInc
);
```

[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。
pszFeatureName	[in]	NULL で終端された IFloat 型ノードの名称です。
pdInc	[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、5.8. 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ステータスがリターンされます。

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]

GenApi_GetFloatMin() のコード例を参照してください。

5.6.4.5. GenApi_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  GenApi_GetFloatDisplayNotation (
  CAM_HANDLE                hCam,
  const char                *pszFeatureName,
  TC_NODE_DISPLAY_NOTATION  *peDisplayNotation
);
```

[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。
pszFeatureName	[in]	NULL で終端された IFloat 型ノードの名称です。
peDisplayNotation	[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* 列挙子]

メンバ	内容
TC_NODE_DISPLAY_NOTATION_AUTOMATIC	固定小数点と指表記で短いを自動選択。
TC_NODE_DISPLAY_NOTATION_FIXED	固定小数点。 例： 123.4
TC_NODE_DISPLAY_NOTATION_SCIENTIFIC	指数表記。　例： 1.234e2
TC_NODE_DISPLAY_NOTATION_UNDEFINED	オブジェクト未初期化。
TC_NODE_DISPLAY_NOTATION_UNKNOWN	不明。

[戻り値]

実行結果を返します。　戻り値は、5.8. ステータスコード を参照してください。

[備考]

TC_NODE_DISPLAY_NOTATION 列挙子は、GenApi の Types.h で定義されている EDisplayNotation 列挙子に対応しています。

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ステータス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。

[コード例]

GenApi_GetFloatMin() のコード例を参照してください。

5.6.4.6. GenApi_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  GenApi_GetFloatDisplayPrecision (
  CAM_HANDLE        hCam,
  const char        *pszFeatureName,
  int64_t           *pllPrecision
);
```

[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。
pszFeatureName	[in]	NULL で終端された IFloat 型ノードの名称です。
pllPrecision	[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、5.8. 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ステータス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。

[コード例]

GenApi_GetFloatMin() のコード例を参照してください。

5.6.4.7. GenApi_GetFloatValue

IFloat 型ノードの値を取得します。

[構文]

```
CAM_API_STATUS  GenApi_GetFloatValue (
  CAM_HANDLE        hCam,
  const char        *pszFeatureName,
  float64_t         *pdValue,
  bool8_t           bVerify = false,
  bool8_t           bIgnoreCache = false,
);
```

[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。
pszFeatureName	[in]	NULL で終端された IFloat 型ノードの名称です。
pdValue	[out]	取得した値を格納する変数へのポインタです。
bVerify	[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。
bIgnoreCache	[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、5.8. 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ステータス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。

[コード例]

GenApi_GetFloatMin() のコード例を参照してください。

5.6.4.8. GenApi_SetFloatValue

IFloat 型ノードの値を設定します。

[構文]

```
CAM_API_STATUS  GenApi_SetFloatValue (
  CAM_HANDLE        hCam,
  const char        *pszFeatureName,
  float64_t         dValue,
  bool8_t           bVerify = true
);
```

[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。
pszFeatureName	[in]	NULL で終端された IFloat 型ノードの名称です。
dValue	[in]	設定する値です。
bVerify	[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、5.8. 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ステータス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。

[コード例]

GenApi_GetFloatMin() のコード例を参照してください。

5.6.5. IBoolean 型　関数

5.6.5.1. GenApi_GetBoolValue

IBoolean 型ノードの値を取得します。

[構文]

```
CAM_API_STATUS  GenApi_GetBoolValue (
  CAM_HANDLE        hCam,
  const char        *pszFeatureName,
  bool8_t           *pbValue,
  bool8_t           bVerify = false,
  bool8_t           bIgnoreCache = false,
);
```

[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。
pszFeatureName	[in]	NULL で終端された IBoolean 型ノードの名称です。
pbValue	[out]	取得した値を格納する変数へのポインタです。
bVerify	[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。
bIgnoreCache	[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、5.8. 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ステータス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。

[コード例]

C++

```
CAM_API_STATUS       uiStatus;
uint32_t             uiNum;
CAM_HANDLE           hCam = (CAM_HANDLE)NULL;
bool8_t              bRdValue, bWrValue;

// Initialize system.
uiStatus = Sys_Initialize();
if (uiStatus != CAM_API_STS_SUCCESS)
```

```
    return -1;

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

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

__try
{
    // Get value.
    uiStatus = GenApi_GetBoolValue(hCam, "ReverseX", &bRdValue);
    if (uiStatus != CAM_API_STS_SUCCESS)
        return -1;
    printf("      Before Value : %d\n", (uint32_t)bRdValue);

    // Set value.
    bWrValue = !bRdValue;
    uiStatus = GenApi_SetBoolValue(hCam, "ReverseX", bWrValue);
    if (uiStatus != CAM_API_STS_SUCCESS)
        return -1;

    // Get value.
    uiStatus = GenApi_GetBoolValue(hCam, "ReverseX", &bRdValue);
    if (uiStatus != CAM_API_STS_SUCCESS)
        return -1;
    printf("      After Value : %d\n", (uint32_t)bRdValue);

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

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

5.6.5.2. GenApi_SetBoolValue

IBoolean 型ノードの値を設定します。

[構文]

```
CAM_API_STATUS  GenApi_SetBoolValue (
  CAM_HANDLE        hCam,
  const char        *pszFeatureName,
  bool8_t           bValue,
  bool8_t           bVerify = true
);
```

[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。
pszFeatureName	[in]	NULL で終端された IBoolean 型ノードの名称です。
bValue	[in]	設定する値です。
bVerify	[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、5.8. 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ステータス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。

[コード例]

GenApi_GetBoolValue() のコード例を参照してください。

5.6.6. IEnumeration 型、IEnumEntry 型　関数

5.6.6.1. GenApi_GetEnumIntValue

IEnumeration 型ノードの値を整数値で取得します。

[構文]

```
CAM_API_STATUS  GenApi_GetEnumIntValue (
  CAM_HANDLE        hCam,
  const char        *pszFeatureName,
  int64_t           *pllValue,
  bool8_t           bVerify = false,
  bool8_t           bIgnoreCache = false
);
```

[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。
pszFeatureName	[in]	NULL で終端された IEnumeration 型ノードの名称です。
pllValue	[out]	取得した整数値を格納する変数へのポインタです。
bVerify	[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。
bIgnoreCache	[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、5.8. 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ステータス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。

[コード例]

C++

```
CAM_API_STATUS       uiStatus;
uint32_t             uiNum;
CAM_HANDLE           hCam = (CAM_HANDLE)NULL;
int64_t              llRdValue, llWrValue;

// Initialize system.
uiStatus = Sys_Initialize();
if (uiStatus != CAM_API_STS_SUCCESS)
```

```
    return -1;

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

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

__try
{
    // Set int value.
    llWrValue = 64;      // 64 : Software
    uiStatus = GenApi_SetEnumIntValue(hCam, "TriggerSource", llWrValue);
    if (uiStatus != CAM_API_STS_SUCCESS)
        return -1;       // Not impremented, or any other error

    // Readback.
    uiStatus = GenApi_GetEnumIntValue(hCam, "TriggerSource", &llRdValue);
    if (uiStatus != CAM_API_STS_SUCCESS)
        return -1;
    printf("      Read Int value : %d\n", (uint32_t)llRdValue);

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

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

5.6.6.2. GenApi_SetEnumIntValue

IEnumeration 型ノードの値を整数値で設定します。

[構文]

```
CAM_API_STATUS  GenApi_SetEnumIntValue (
CAM_HANDLE          hCam,
  const char        *pszFeatureName,
int64_t             llValue,
bool8_t             bVerify = true
);
```

[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。
pszFeatureName	[in]	NULL で終端された IEnumeration 型ノードの名称です。
llValue	[in]	設定する整数値です。
bVerify	[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、5.8. 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ステータス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。

[コード例]

GenApi_GetEnumIntValue()のコード例を参照してください。

5.6.6.3. GenApi_GetEnumStrValue

IEnumeration 型ノードの値を文字列で取得します。

[構文]

```
CAM_API_STATUS  GenApi_GetEnumStrValue (
  CAM_HANDLE        hCam,
  const char        *pszFeatureName,
  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
hCam	[in]	カメラのカメラハンドルです。
pszFeatureName	[in]	NULL で終端された IEnumeration 型ノードの名称です。
pszBuf	[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、*puiSize* に必要なバッファサイズが格納されます。
puiSize	[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） *pszBuf* 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。
bVerify	[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。
bIgnoreCache	[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、5.8. 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ステータス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。

[コード例]

C++

```
CAM_API_STATUS       uiStatus;
uint32_t             uiNum, uiSize;
CAM_HANDLE           hCam = (CAM_HANDLE)NULL;
char                 szbuf[32];

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

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

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

__try
{
    // Read string.
    uiSize = 32;
    uiStatus = GenApi_GetEnumStrValue(hCam, "TriggerSource", szbuf, &uiSize);
    if (uiStatus != CAM_API_STS_SUCCESS)
        return -1;
    printf("Before read str value : %s\n", szbuf);

    // Set string.
    if (strncmp((const char*)szbuf, "Line0", uiSize) == 0) {
        uiStatus = GenApi_SetEnumStrValue(hCam, "TriggerSource", "Software");
    } else {
        uiStatus = GenApi_SetEnumStrValue(hCam, "TriggerSource", "Line0");
    }
    if (uiStatus != CAM_API_STS_SUCCESS)
        return -1;       // Not impremented, or any other error

    // Readback.
    uiSize = 32;
    uiStatus = GenApi_GetEnumStrValue(hCam, "TriggerSource", szbuf, &uiSize);
    if (uiStatus != CAM_API_STS_SUCCESS)
        return -1;
    printf("After  read str value : %s\n", szbuf);

    return 0;
}
__finally
{
    // Close camera.
    if (hCam != (CAM_HANDLE)NULL) {
        uiStatus = Cam_Close(hCam);
        if (uiStatus != CAM_API_STS_SUCCESS)
            return -1;
    }

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


5.6.6.4. GenApi_SetEnumStrValue

IEnumeration 型ノードの値を文字列で設定します。

[構文]

```
CAM_API_STATUS  GenApi_SetEnumStrValue (
  CAM_HANDLE        hCam,
  const char        *pszFeatureName,
  const char        *pszBuf,
  bool8_t           bVerify = true
);
```

[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。
pszFeatureName	[in]	NULL で終端された IEnumeration 型ノードの名称です。
hNode	[in]	値を設定する IEnumeration 型ノードのノードハンドルです。
pszBuf	[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。
bVerify	[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、5.8. 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ステータス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。

[コード例]

GenApi_GetEnumStrValue()のコード例を参照してください。

5.6.6.5. GenApi_GetNumOfEnumEntries

IEnumeration 型ノードのエントリー数を取得します。

[構文]

```
CAM_API_STATUS  GenApi_GetNumOfEnumEntries (
  CAM_HANDLE        hCam,
  const char        *pszFeatureName,
  uint32_t          *puiNum
);
```

[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。
pszFeatureName	[in]	NULL で終端された IEnumeration 型ノードの名称です。
puiNum	[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、5.8. ステータスコード を参照してください。

[備考]

取得されるエントリ数は、カメラ記述ファイル（XML ファイル）に記述されているエントリノードの数です。　カメラに実装されていないエントリノード及び利用不可のエントリノードの数も含まれますのでご注意ください。

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ステータスがリターンされます。

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++

```
CAM_API_STATUS       uiStatus;
uint32_t             uiNum, uiSize;
CAM_HANDLE           hCam = (CAM_HANDLE)NULL;
uint32_t             uiEntriesNum;
int64_t              llEntryValue;
char                 *pszBuf = NULL;
TC_NODE_ACCESS_MODE  eAccessMode;

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

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

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

```
    return -1;

__try
{
    // Get num of entries.
    uiStatus = GenApi_GetNumOfEnumEntries(hCam, "LineSelector", &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++) {
        // Confirm whether entry value is valid.
        uiStatus = GenApi_GetEnumEntryAccessMode(hCam, "LineSelector", i, &eAccessMode);
        if ((uiStatus == CAM_API_STS_SUCCESS) && (eAccessMode != TC_NODE_ACCESS_MODE_NI)
&& (eAccessMode != TC_NODE_ACCESS_MODE_NA))
        {
            // Get int value.
            uiStatus = GenApi_GetEnumEntryIntValue(hCam, "LineSelector", i,
&llEntryValue);
            if (uiStatus != CAM_API_STS_SUCCESS)
                return -1;

            // Get size of the string.
            uiSize = 0;
            uiStatus = GenApi_GetEnumEntryStrValue(hCam, "LineSelector", i, NULL,
&uiSize);
            if (uiStatus != CAM_API_STS_SUCCESS)
                return -1;

            // Allocate buffer.
            pszBuf = (char *)VirtualAlloc(NULL,
                                          uiSize,
                                          MEM_RESERVE | MEM_COMMIT,
                                          PAGE_EXECUTE_READWRITE);
            if (pszBuf == NULL) {
                return -1;
            }

            // Get string.
            uiStatus = GenApi_GetEnumEntryStrValue(hCam, "LineSelector", i, pszBuf,
&uiSize);
            if (uiStatus != CAM_API_STS_SUCCESS)
                return -1;

            printf(" Enum %d : EnumEntryIntValue = %d, EnumEntryStrValue = %s\n", i,
(int)llEntryValue, pszBuf);

            VirtualFree(pszBuf, 0, MEM_RELEASE);
            pszBuf = NULL;
        }
    }

    return 0;
}
__finally
{
    if (pszBuf != NULL) {
        free( pszBuf);
    }

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

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

5.6.6.6. GenApi_GetEnumEntryAccessMode

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 型ノードのアクセスモードを取得します。

[構文]

```
CAM_API_STATUS   GenApi_GetEnumEntryAccessMode (
  CAM_HANDLE            hCam,
  const char            *pszFeatureName,
  uint32_t              uiEnumIndex,
  TC_NODE_ACCESS_MODE   *peAccessMode
);
```

[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。
pszFeatureName	[in]	NULL で終端された IEnumeration 型ノードの名称です。
uiEnumIndex	[in]	IEnumEntry型ノードリスト内のインデックスです。
peAccessMode	[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。

[戻り値]

実行結果を返します。　戻り値は、5.8. 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ステータス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。

[コード例]

GenApi_GetNumOfEnumEntries() のコード例を参照してください。

5.6.6.7. GenApi_GetEnumEntryIntValue

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 型ノードの値を整数値で取得します。

[構文]

```
CAM_API_STATUS  GenApi_GetEnumEntryIntValue (
  CAM_HANDLE        hCam,
  const char        *pszFeatureName,
  uint32_t          uiEnumIndex,
  int64_t           *pllValue
);
```

[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。
pszFeatureName	[in]	NULL で終端された IEnumeration 型ノードの名称です。
uiEnumIndex	[in]	IEnumEntry型ノードリスト内のインデックスです。
pllValue	[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、5.8. 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ステータス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。

[コード例]

GenApi_GetNumOfEnumEntries() のコード例を参照してください。

5.6.6.8. GenApi_GetEnumEntryStrValue

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 型ノードの値を文字列で取得します。

[構文]

```
CAM_API_STATUS  GenApi_GetEnumEntryStrValue (
  CAM_HANDLE        hCam,
  const char        *pszFeatureName,
  uint32_t          uiEnumIndex,
  char              *pszBuf,
  uint32_t          *puiSize
);
```

[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。
pszFeatureName	[in]	NULL で終端された IEnumeration 型ノードの名称です。
uiEnumIndex	[in]	IEnumEntry型ノードリスト内のインデックスです。
pszBuf	[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、*puiSize* に必要なバッファサイズが格納されます。
puiSize	[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） *pszBuf* 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、5.8. 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ステータス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。

[コード例]

GenApi_GetNumOfEnumEntries() のコード例を参照してください。

5.6.7. ICommand 型　関数

5.6.7.1. GenApi_CmdExecute

ICommand 型ノードのコマンドを実行します。

[構文]

```
CAM_API_STATUS  GenApi_CmdExecute (
  CAM_HANDLE        hCam,
  const char        *pszFeatureName,
  bool8_t           bVerify = true
);
```

[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。
pszFeatureName	[in]	NULL で終端された ICommand 型ノードの名称です。
bVerify	[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、5.8. 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ステータス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。

[コード例]

C++

```
...

bool bDone = false;

// Execute command.
uiStatus = GenApi_CmdExecute(hCam, "TriggerSoftware");
if (uiStatus != CAM_API_STS_SUCCESS)
    return -1;

while(1) {
    // Confirm whether the execution has been accomplished.
    uiStatus = GenApi_GetCmdIsDone(hCam, "TriggerSoftware", &bDone);
    if (uiStatus != CAM_API_STS_SUCCESS)
        return -1;

    if (bDone == true)
        break;

    Sleep(0);
}
...
```

5.6.7.2. GenApi_GetCmdIsDone

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  GenApi_GetCmdIsDone (
  CAM_HANDLE        hCam,
  const char        *pszFeatureName,
  bool8_t           *pbDone,
  bool8_t           bVerify = false
);
```

[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。
pszFeatureName	[in]	NULL で終端された ICommand 型ノードの名称です。
pbDone	[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。
bVerify	[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、5.8. 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ステータス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。

[コード例]

GenApi_CmdExecute() のコード例を参照してください。

5.6.8. IString 型　関数

5.6.8.1. GenApi_GetStrValue

IString 型ノードの値（文字列）を取得します。

[構文]

```
CAM_API_STATUS  GenApi_GetStrValue (
  CAM_HANDLE        hCam,
  const char        *pszFeatureName,
  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
hCam	[in]	カメラのカメラハンドルです。
pszFeatureName	[in]	NULL で終端された IString 型ノードの名称です。
pszBuf	[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、*puiSize*に必要なバッファサイズが格納されます。
puiSize	[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） *pszBuf*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。
bVerify	[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。
bIgnoreCache	[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、5.8. 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ステータス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。

[コード例]

C++

```
...

char            szBuf[128];
uint32_t        uiSize = 128;


// Set string.
uiStatus = GenApi_SetStrValue(hCam, "UserDefinedName", "Test");
if (uiStatus != CAM_API_STS_SUCCESS)
    return -1;

// Get string.
uiStatus = GenApi_GetStrValue(hCam, "UserDefinedName", &szBuf[0], &uiSize);
if (uiStatus != CAM_API_STS_SUCCESS)
    return -1;
printf("  GenApi_GetStrValue : %s\n", szBuf);

...
```


5.6.8.2. GenApi_SetStrValue

IString 型ノードの値（文字列）を設定します。

[構文]

```
CAM_API_STATUS  GenApi_SetStrValue (
  CAM_HANDLE        hCam,
  const char        *pszFeatureName,
  const char        *pszBuf,
  bool8_t           bVerify = true
);
```

[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。
pszFeatureName	[in]	NULL で終端された IString 型ノードの名称です。
pszBuf	[in]	文字列を格納したバッファへのポインタです。
bVerify	[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、5.8. 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ステータス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。

[コード例]

GenApi_GetStrValue() のコード例を参照してください。

5.6.9. Chunk 関数

5.6.9.1. GenApi_ChunkAttachBuffer

GenICam のチャンクアダプタに、バッファをアタッチします。
受信したペイロードデータから GenICam を使用してチャンクデータを取得するとき、ペイロードデータが格納されているバッファを GenICam のチャンクアダプタにアタッチする必要があります。

[構文]

```
CAM_API_STATUS  GenApi_ChunkAttachBuffer (
  CAM_STRM_HANDLE   hStrm,
  void              *pvPayloadBuf,
  uint32_t          uiPayloadSize
);
```

[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。
pvPayloadBuf	[in]	受信したペイロードデータが格納されているバッファへのポインタです。
uiPayloadSize	[in]	受信したペイロードデータのサイズです。

[戻り値]

実行結果を返します。　戻り値は、5.8. ステータスコード を参照してください。

[備考]

ペイロードデータにチャンクデータが付加されていない場合はエラーがリターンされます。
チャンクデータは、GenICam 関数を使用して取得します。

TeliCamAPI.h をインクルードする必要があります。

[コード例]

C++

```
...

CAM_IMAGE_INFO   sImageInfo;
int64_t          lFrameID;
float64_t        dExposureTime;

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

// Attach Buffer
uiStatus = Chunk_AttachBuffer(hStrm, pvPayloadBuf, uiPyldSize);
if (uiStatus != CAM_API_STS_SUCCESS)
    return -1;

// Get ChunkFrameID
uiStatus = GenApi_GetIntValue(hCam, "ChunkFrameID", &lFrameID);
if (uiStatus != CAM_API_STS_SUCCESS)
    return -1;
```

```
printf("ChunkFrameID = %d\n", lFrameID);

// Get ChunkExposureTime
uiStatus = GenApi_GetFloatValue(hCam, "ChunkExposureTime", &dExposureTime);
if (uiStatus != CAM_API_STS_SUCCESS)
    return -1;
printf("ChunkExposureTime = %f\n", dExposureTime);

...
```

5.6.9.2. GenApi_ChunkUpdateBuffer

GenICam のチャンクアダプタにアタッチされているバッファを更新します。

[構文]

```
CAM_API_STATUS  GenApi_ChunkUpdateBuffer (
  CAM_STRM_HANDLE   hStrm,
  void              *pvPayloadBuf
);
```

[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。
pvPayloadBuf	[in]	受信したペイロードデータが格納されてい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、5.8. ステータスコード を参照してください。

[備考]

ペイロードデータにチャンクデータが付加されていない場合はエラーがリターンされます。
チャンクデータは、GenICam 関数を使用して取得します。

GenICam のチャンクアダプタにバッファが一度もアタッチされていない場合、またはチャンクデータのレイアウトが変更されている場合はエラーがリターンされます。

GenApi_ChunkAttachBuffer() より高速に処理できます。

TeliCamAPI.h をインクルードする必要があります。

5.6.9.3. GenApi_ChunkCheckBufferLayout

指定されたバッファに既知の形式のチャンクデータが含まれているかどうか確認します。

[構文]

```
CAM_API_STATUS  GenApi_ChunkCheckBufferLayout (
  CAM_STRM_HANDLE   hStrm,
  void              *pvPayloadBuf,
  uint32_t          uiPayloadSize,
  bool8_t           *pbValue
);
```

[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。
pvPayloadBuf	[in]	受信したペイロードデータが格納されているバッファへのポインタです。
uiPayloadSize	[in]	受信したペイロードデータのサイズです。
pbValue	[out]	取得した値を格納する変数へのポインタです。 true のとき、有効なチャンクデータが含まれています。 false のとき、有効なチャンクデータが含まれていません。

[戻り値]

実行結果を返します。　戻り値は、5.8. 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。

5.6.10.その他の関数

5.6.10.1. GenApi_GetLastError

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  GenApi_GetLastError (
  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;
```

メンバ		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、5.8. 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。

5.6.11.旧 GenICam 関数

以前のバージョンとの互換性のために本章の関数を使用することができますが、下位互換性が必要ない場合は 5.6.1 項から 5.6.10 項の新しい GenICam 関数(GenApi_ *)を使用することをお勧めします。

5.6.11.1. Node 系関数

5.6.11.1.1. 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
hCam	[in]	カメラのカメラハンドルです。
pszName	[in]	NULL で終端されたカメラ機能の機能名（ノード名）です。
phNode	[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、5.8. ステータスコード を参照してください。

[備考]

本関数は、以前のバージョンとの互換性のために残されています。
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。

5.6.11.1.2. 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
hCam	[in]	カメラのカメラハンドルです。
hNode	[in]	タイプを取得するノードのノードハンドルです。
peNodeType	[out]	取得したノード型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、5.8. ステータスコード を参照してください。

[備考]

本関数は、以前のバージョンとの互換性のために残されています。　新しいアプリケーションを作成する場合は、GenApi_GetType() を使用してください。

5.6.11.1.3. 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
hCam	[in]	カメラのカメラハンドルです。
hNode	[in]	ノード名を取得するノードのノードハンドルです。
pszNodeName	[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。

[戻り値]

実行結果を返します。　戻り値は、5.8. ステータスコード を参照してください。

[備考]

本関数は、以前のバージョンとの互換性のために残されています。　新しいアプリケーションを作成する場合は、GenApi_GetFeatureName() を使用してください。

5.6.11.1.4. 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            hCam,
  CAM_NODE_HANDLE       hNode,
  TC_NODE_ACCESS_MODE   *peAccessMode
);
```

[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]	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。
peAccessMode	[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。

[戻り値]

実行結果を返します。　戻り値は、5.8. ステータスコード を参照してください。

[備考]

本関数は、以前のバージョンとの互換性のために残されています。　新しいアプリケーションを作成する場合は、GenApi_GetAccessMode() または GenApi_GetEnumEntryAccessMode()を使用してください。

5.6.11.1.5. 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            hCam,
  CAM_NODE_HANDLE       hNode,
  TC_NODE_VISIBILITY    *peVisibility
);
```

[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]	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。
peVisibility	[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。

[戻り値]

実行結果を返します。　戻り値は、5.8. ステータスコード を参照してください。

[備考]

本関数は、以前のバージョンとの互換性のために残されています。　新しいアプリケーションを作成する場合は、GenApi_GetVisibility() を使用してください。

5.6.11.1.6. 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            hCam,
  CAM_NODE_HANDLE       hNode,
  TC_NODE_CACHING_MODE  *peCachingMode
);
```

[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]	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。
peCachingMode	[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。

[戻り値]

実行結果を返します。　戻り値は、5.8. ステータスコード を参照してください。

[備考]

本関数は、以前のバージョンとの互換性のために残されています。　新しいアプリケーションを作成する場合は、GenApi_GetCachingMode() を使用してください。

5.6.11.1.7. 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            hCam,
  CAM_NODE_HANDLE       hNode,
  char                  *pszBuf,
  uint32_t              *puiSize
);
```

[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]	説明文字列を取得するノードのノードハンドルです。
pszBuf	[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、*puiSize*に必要なバッファサイズが格納されます。
puiSize	[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） *pszBuf*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、5.8. ステータスコード を参照してください。

[備考]

本関数は、以前のバージョンとの互換性のために残されています。　新しいアプリケーションを作成する場合は、GenApi_GetDescriptio()を使用してください。

5.6.11.1.8. 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            hCam,
  CAM_NODE_HANDLE       hNode,
  char                  *pszBuf,
  uint32_t              *puiSize
);
```

[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]	ツールチップを取得するノードのノードハンドルです。
pszBuf	[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、*puiSize* に必要なバッファサイズが格納されます。
puiSize	[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） *pszBuf* 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、5.8. ステータスコード を参照してください。

[備考]

本関数は、以前のバージョンとの互換性のために残されています。　新しいアプリケーションを作成する場合は、GenApi_GetToolTip() を使用してください。

5.6.11.1.9. 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                hCam,
  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。

[戻り値]

実行結果を返します。　戻り値は、5.8. ステータスコード を参照してください。

[備考]

本関数は、以前のバージョンとの互換性のために残されています。　新しいアプリケーションを作成する場合は、GenApi_GetRepresentation() を使用してください。

5.6.11.1.10.　Nd_GetUnit

ノードの単位名を取得します。

[構文]

```
CAM_API_STATUS  Nd_GetUnit (
  CAM_HANDLE          hCam,
  CAM_NODE_HANDLE     hNode,
  char                *pszBuf,
  uint32_t            *puiSize
);
```

[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]	単位名を取得するノードのノードハンドルです。
pszBuf	[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、*puiSize* に必要なバッファサイズが格納されます。
puiSize	[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） *pszBuf* 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、5.8. ステータスコード を参照してください。

[備考]

本関数は、以前のバージョンとの互換性のために残されています。　新しいアプリケーションを作成する場合は、GenApi_GetUnit() を使用してください。

5.6.11.2. Node 系 ICategory 型　関数

5.6.11.2.1. 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
hCam	[in]	カメラのカメラハンドルです。
hNode	[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。
puiNum	[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、5.8. ステータスコード を参照してください。

[備考]

本関数は、以前のバージョンとの互換性のために残されています。　新しいアプリケーションを作成する場合は、GenApi_GetNumOfFeatures() を使用してください。

5.6.11.2.2. 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
hCam	[in]	カメラのカメラハンドルです。
hNode	[in]	ICategory 型ノードのノードハンドルです。
uiNodeIndex	[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 ）
phNode	[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、5.8. ステータスコード を参照してください。

[備考]

本関数は、以前のバージョンとの互換性のために残されています。

5.6.11.3. Node 系 IInteger 型　関数

5.6.11.3.1. Nd_GetIntMin

IInteger 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
hCam	[in]	カメラのカメラハンドルです。
hNode	[in]	最小値を取得する IInteger 型ノードのノードハンドルです。
pllMin	[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、5.8. ステータスコード を参照してください。

[備考]

本関数は、以前のバージョンとの互換性のために残されています。　新しいアプリケーションを作成する場合は、GenApi_GetIntMin() を使用してください。

5.6.11.3.2. Nd_GetIntMax

IInteger 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
hCam	[in]	カメラのカメラハンドルです。
hNode	[in]	最大値を取得する IInteger 型ノードのノードハンドルです。
pllMax	[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、5.8. ステータスコード を参照してください。

[備考]

本関数は、以前のバージョンとの互換性のために残されています。　新しいアプリケーションを作成する場合は、GenApi_GetIntMax() を使用してください。

5.6.11.3.3. Nd_GetIntInc

IInteger 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
hCam	[in]	カメラのカメラハンドルです。
hNode	[in]	増加量を取得する IInteger 型ノードのノードハンドルです。
pllInc	[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、5.8. ステータスコード を参照してください。

[備考]

本関数は、以前のバージョンとの互換性のために残されています。　新しいアプリケーションを作成する場合は、GenApi_GetIntInc() を使用してください。

5.6.11.3.4. Nd_GetIntValue

IInteger 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
hCam	[in]	カメラのカメラハンドルです。
hNode	[in]	値を取得する IInteger 型ノードのノードハンドルです。
pllValue	[out]	取得した値を格納する変数へのポインタです。
bVerify	[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。
bIgnoreCache	[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、5.8. ステータスコード を参照してください。

[備考]

本関数は、以前のバージョンとの互換性のために残されています。　新しいアプリケーションを作成する場合は、GenApi_GetIntValue() を使用してください。

5.6.11.3.5. Nd_SetIntValue

IInteger 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
hCam	[in]	カメラのカメラハンドルです。
hNode	[in]	値を設定する IInteger 型ノードのノードハンドルです。
llValue	[in]	設定する値です。
bVerify	[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、5.8. ステータスコード を参照してください。

[備考]

本関数は、以前のバージョンとの互換性のために残されています。　新しいアプリケーションを作成する場合は、GenApi_SetIntValue() を使用してください。

5.6.11.4. Node 系 IFloat 型　関数

5.6.11.4.1. 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
hCam	[in]	カメラのカメラハンドルです。
hNode	[in]	最小値を取得する IFloat 型ノードのノードハンドルです。
pdMin	[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、5.8. ステータスコード を参照してください。

[備考]

本関数は、以前のバージョンとの互換性のために残されています。　新しいアプリケーションを作成する場合は、GenApi_GetFloatMin() を使用してください。

5.6.11.4.2. 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
hCam	[in]	カメラのカメラハンドルです。
hNode	[in]	最大値を取得する IFloat 型ノードのノードハンドルです。
dMax	[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、5.8. ステータスコード を参照してください。

[備考]

本関数は、以前のバージョンとの互換性のために残されています。　新しいアプリケーションを作成する場合は、GenApi_GetFloatMax() を使用してください。

5.6.11.4.3. 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
hCam	[in]	カメラのカメラハンドルです。
hNode	[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。
pbInc	[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、5.8. ステータスコード を参照してください。

[備考]

本関数は、以前のバージョンとの互換性のために残されています。　新しいアプリケーションを作成する場合は、GenApi_GetFloatHasInc() を使用してください。

5.6.11.4.4. 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
hCam	[in]	カメラのカメラハンドルです。
hNode	[in]	増加量を取得する IFloat 型ノードのノードハンドルです。
pdInc	[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、5.8. ステータスコード を参照してください。

[備考]

本関数は、以前のバージョンとの互換性のために残されています。　新しいアプリケーションを作成する場合は、GenApi_GetFloatInc() を使用してください。

5.6.11.4.5. 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
hCam	[in]	カメラのカメラハンドルです。
hNode	[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。
peDisplayNotation	[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。

[戻り値]

実行結果を返します。　戻り値は、5.8. ステータスコード を参照してください。

[備考]

本関数は、以前のバージョンとの互換性のために残されています。　新しいアプリケーションを作成する場合は、GenApi_GetFloatDisplayNotation() を使用してください。

5.6.11.4.6. 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
hCam	[in]	カメラのカメラハンドルです。
hNode	[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。
pllPrecision	[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、5.8. ステータスコード を参照してください。

[備考]

本関数は、以前のバージョンとの互換性のために残されています。　新しいアプリケーションを作成する場合は、GenApi_GetFloatDisplayPrecision() を使用してください。

5.6.11.4.7. 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
hCam	[in]	カメラのカメラハンドルです。
hNode	[in]	値を取得する IFloat 型ノードのノードハンドルです。
pdValue	[out]	取得した値を格納する変数へのポインタです。
bVerify	[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。
bIgnoreCache	[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、5.8. ステータスコード を参照してください。

[備考]

本関数は、以前のバージョンとの互換性のために残されています。　新しいアプリケーションを作成する場合は、GenApi_GetFloatValue() を使用してください。

5.6.11.4.8. 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
hCam	[in]	カメラのカメラハンドルです。
hNode	[in]	値を設定する IFloat 型ノードのノードハンドルです。
dValue	[in]	設定する値です。
bVerify	[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、5.8. ステータスコード を参照してください。

[備考]

本関数は、以前のバージョンとの互換性のために残されています。　新しいアプリケーションを作成する場合は、GenApi_SetFloatValue() を使用してください。

5.6.11.5. Node 系 IBoolean 型　関数

5.6.11.5.1. 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
hCam	[in]	カメラのカメラハンドルです。
hNode	[in]	値を取得する IBoolean 型ノードのノードハンドルです。
pbValue	[out]	取得した値を格納する変数へのポインタです。
bVerify	[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。
bIgnoreCache	[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、5.8. ステータスコード を参照してください。

[備考]

本関数は、以前のバージョンとの互換性のために残されています。　新しいアプリケーションを作成する場合は、GenApi_GetBoolValue() を使用してください。

5.6.11.5.2. 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
hCam	[in]	カメラのカメラハンドルです。
hNode	[in]	値を設定する IBoolean 型ノードのノードハンドルです。
bValue	[in]	設定する値です。
bVerify	[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、5.8. ステータスコード を参照してください。

[備考]

本関数は、以前のバージョンとの互換性のために残されています。　新しいアプリケーションを作成する場合は、GenApi_SetBoolValue() を使用してください。

5.6.11.6. Node 系 IEnumeration 型　関数

5.6.11.6.1. 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
hCam	[in]	カメラのカメラハンドルです。
hNode	[in]	値を取得する IEnumeration 型ノードのノードハンドルです。
pllValue	[out]	取得した整数値を格納する変数へのポインタです。
bVerify	[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。
bIgnoreCache	[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、5.8. ステータスコード を参照してください。

[備考]

本関数は、以前のバージョンとの互換性のために残されています。　新しいアプリケーションを作成する場合は、GenApi_GetEnumIntValue() を使用してください。

5.6.11.6.2. 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
hCam	[in]	カメラのカメラハンドルです。
hNode	[in]	値を設定する IEnumeration 型ノードのノードハンドルです。
llValue	[in]	設定する整数値です。
bVerify	[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、5.8. ステータスコード を参照してください。

[備考]

本関数は、以前のバージョンとの互換性のために残されています。　新しいアプリケーションを作成する場合は、GenApi_SetEnumIntValue() を使用してください。

5.6.11.6.3. 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_GetEnumStrValue (
  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
hCam	[in]	カメラのカメラハンドルです。
hNode	[in]	値を取得する IEnumeration 型ノードのノードハンドルです。
pszBuf	[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、*puiSize* に必要なバッファサイズが格納されます。
puiSize	[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） *pszBuf* 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。
bVerify	[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。
bIgnoreCache	[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、5.8. ステータスコード を参照してください。

[備考]

本関数は、以前のバージョンとの互換性のために残されています。　新しいアプリケーションを作成する場合は、GenApi_GetEnumStrValue() を使用してください。

5.6.11.6.4. 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
hCam	[in]	カメラのカメラハンドルです。
hNode	[in]	値を設定する IEnumeration 型ノードのノードハンドルです。
pszBuf	[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。
bVerify	[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、5.8. ステータスコード を参照してください。

[備考]

本関数は、以前のバージョンとの互換性のために残されています。　新しいアプリケーションを作成する場合は、GenApi_SetEnumStrValue() を使用してください。

5.6.11.6.5. 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          *puiNum
);
```

[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]	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。
puiNum	[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、5.8. ステータスコード を参照してください。

[備考]
本関数は、以前のバージョンとの互換性のために残されています。　新しいアプリケーションを作成する場合は、GenApi_GetNumOfEnumEntries() を使用してください。

5.6.11.6.6. 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
hCam	[in]	カメラのカメラハンドルです。
hNode	[in]	IEnumeration 型ノードのノードハンドルです。
uiEnumIdx	[in]	IEnumEntry型ノードリスト内のインデックスです。
phEnumEntryNode	[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、5.8. ステータスコード を参照してください。

[備考]
本関数は、以前のバージョンとの互換性のために残されています。

5.6.11.7. Node 系 IEnumEntry 型　関数

5.6.11.7.1. 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
hCam	[in]	カメラのカメラハンドルです。
hNode	[in]	IEnumEntry 型ノードのノードハンドルです。
pllValue	[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、5.8. ステータスコード を参照してください。

[備考]

本関数は、以前のバージョンとの互換性のために残されています。　新しいアプリケーションを作成する場合は、GenApi_GetEnumEntryIntValue() を使用してください。

5.6.11.7.2. 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
hCam	[in]	カメラのカメラハンドルです。
hNode	[in]	IEnumEntry 型ノードのノードハンドルです。
pszBuf	[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、*puiSize*に必要なバッファサイズが格納されます。
puiSize	[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） *pszBuf* 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、5.8. ステータスコード を参照してください。

[備考]

本関数は、以前のバージョンとの互換性のために残されています。　新しいアプリケーションを作成する場合は、GenApi_GetEnumEntryStrValue() を使用してください。

5.6.11.8. Node 系 ICommand 型　関数

5.6.11.8.1. 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
hCam	[in]	カメラのカメラハンドルです。
hNode	[in]	実行する ICommand 型ノードのノードハンドルです。
bVerify	[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、5.8. ステータスコード を参照してください。

[備考]

本関数は、以前のバージョンとの互換性のために残されています。　新しいアプリケーションを作成する場合は、GenApi_CmdExecute() を使用してください。

5.6.11.8.2. 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,
  bool8_t           *pbDone,
  bool8_t           bVerify = false
);
```

[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]	実行状態を取得する ICommand 型ノードのノードハンドルです。
pbDone	[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。
bVerify	[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、5.8. ステータスコード を参照してください。

[備考]

本関数は、以前のバージョンとの互換性のために残されています。　新しいアプリケーションを作成する場合は、GenApi_GetCmdIsDone() を使用してください。

5.6.11.9. Node 系 IString 型　関数

5.6.11.9.1. Nd_GetStrValue

IString 型ノードの値（文字列）を取得します。

[構文]

```
CAM_API_STATUS  Nd_GetStrValue (
  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
hCam	[in]	カメラのカメラハンドルです。
hNode	[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。
pszBuf	[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、*puiSize* に必要なバッファサイズが格納されます。
puiSize	[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） *pszBuf* 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。
bVerify	[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。
bIgnoreCache	[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、5.8. ステータスコード を参照してください。

[備考]

本関数は、以前のバージョンとの互換性のために残されています。　新しいアプリケーションを作成する場合は、GenApi_GetStrValue() を使用してください。

5.6.11.9.2. 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
hCam	[in]	カメラのカメラハンドルです。
hNode	[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。
pszBuf	[in]	文字列を格納したバッファへのポインタです。
bVerify	[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、5.8. ステータスコード を参照してください。

[備考]

本関数は、以前のバージョンとの互換性のために残されています。　新しいアプリケーションを作成する場合は、GenApi_SetStrValue() を使用してください。

5.6.11.10. Chunk 関数

5.6.11.10.1. Chunk_AttachBuffer

GenICam のチャンクアダプタに、バッファをアタッチします。

[構文]

```
CAM_API_STATUS  Chunk_AttachBuffer (
  CAM_STRM_HANDLE   hStrm,
  void              *pvPayloadBuf,
  uint32_t          uiPayloadSize
);
```

[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。
pvPayloadBuf	[in]	受信したペイロードデータが格納されているバッファへのポインタです。
uiPayloadSize	[in]	受信したペイロードデータのサイズです。

[戻り値]

実行結果を返します。　戻り値は、5.8. ステータスコード を参照してください。

[備考]

本関数は、以前のバージョンとの互換性のために残されています。
GenApi_ChunkAttachBuffer() と機能および引数は同一です。
新しいアプリケーションを作成する場合は、GenApi_ChunkAttachBuffer() を使用してください。

5.6.11.11. その他の関数

5.6.11.11.1. 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。

[戻り値]

実行結果を返します。　戻り値は、5.8. ステータスコード を参照してください。

[備考]

本関数は、以前のバージョンとの互換性のために残されています。
GenApi_GetLastError() と機能および引数は同一です。
新しいアプリケーションを作成する場合は、GenApi_GetLastError() を使用してください。

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、5.8. 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。

TeliCamUtl.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_UYYVYY（(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
puiDstARGB	[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。
pvSrc	[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。
uiWidth	[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。
uiHeight	[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、5.8. ステータスコード を参照してください。

[備考]

TeliCamUtl.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
);
```

[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
ConvMono8ToBGR	Mono8	0x01080001
ConvMono10ToBGR	Mono10	0x01100003
ConvMono12ToBGR	Mono12	0x01100005
ConvMono16ToBGR	Mono16	0x01100007
ConvByrGR8ToBGR	BayerGR8	0x01080008
ConvByrRG8ToBGR	BayerRG8	0x01080009
ConvByrGB8ToBGR	BayerGB8	0x0108000A
ConvByrBG8ToBGR	BayerBG8	0x0108000B
ConvByrGR10ToBGR	BayerGR10	0x0110000C
ConvByrRG10ToBGR	BayerRG10	0x0110000D
ConvByrGB10ToBGR	BayerGB10	0x0110000E
ConvByrBG10ToBGR	BayerBG10	0x0110000F
ConvByrGR12ToBGR	BayerGR12	0x01100010
ConvByrRG12ToBGR	BayerRG12	0x01100011
ConvByrGB12ToBGR	BayerGB12	0x01100012
ConvByrBG12ToBGR	BayerBG12	0x01100013
ConvRGB8PToBGR	RGB8 (RGB8Packed)	0x02180014
ConvBGR8PToBGR	BGR8 (BGR8Packed)	0x02180015
ConvBGR10PToBGR	BGR10 (BGR10Packed)	0x02300019
ConvBGR12PToBGR	BGR12 (BGR12Packed)	0x0230001B
ConvYUV411PToBGR	YUV411_8_UYYVYY ((YUV411Packed)	0x020C001E
ConvYUV422PToBGR	YUV422_8_UYVY (YUV422Packed)	0x0210001F
ConvYUV444PToBGR	YUV8_UYV (YUV444Packed)	0x02180020

[パラメータ]

パラメータ		内　容
puiDstBGR	[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。
pvSrc	[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。
uiWidth	[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。
uiHeight	[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、5.8. ステータスコード を参照してください。

[備考]

TeliCamUtl.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
);
```

[パラメータ]

パラメータ		内　容
eDstFormat	[in]	出力画像フォーマットです。
uiSrcPixelFormat	[in]	原画像データの PixelFormat です。
bBayerConversion	[in]	原画像がベイヤータイプで、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。 原画像がベイヤータイプで、この値が真のときは、原画像をベ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。 原画像がベイヤータイプでない場合、この値は無視されます。
puiDst	[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。
pvSrc	[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。
uiWidth	[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。
uiHeight	[in]	原画像データの高さです。(単位：画素)

[*DST_FORMAT* 列挙子]

メンバ	内容
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

[戻り値]

実行結果を返します。　戻り値は、5.8. ステータスコード を参照してください。

[備考]

TeliCamUtl.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 Soure image and set image size and PixelFormat
uiStatus = GetCamWidth(m_hCam, &uiWidth);
if (uiStatus != CAM_API_STS_SUCCESS)
    return -1;

uiStatus = GetCamHeight(m_hCam, &uiHeight);
if (uiStatus != CAM_API_STS_SUCCESS)
    return -1;

uiStatus = GetCamPixelFormat(m_hCam, &uiPixelFormat);
if (uiStatus != CAM_API_STS_SUCCESS)
    return -1;

// TODO: add your handling code here


// Allocate memory to BGR buffer (uiWidth should be multiple of 4)
pucImgBGR = new uint8_t[uiWidth * uiHeight * 3];
if (pucImgBGR == NULL)
    return -1;

// Convert source image to BGR
uiStatus = ConvImage(DST_FMT_BGR24, uiPixelFormat, true,
             (void *)pucImgBGR, (void *)pvImgSrc, uiWidth, uiHeight);

if (uiStatus != CAM_API_STS_SUCCESS)
    return -1;

// TODO: add your handling code here
```


5.7.2. その他ユーティリティ

5.7.2.1. BitPerPixel

引数の PixekFormat 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]

TeliCamUtl.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]

TeliCamUtl.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]

TeliCamUtl.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]

TeliCamUtl.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
pvTgt	[in]	保存対象画像データの先頭画素のポインタです。
uiWidth	[in]	保存対象画像の幅方向画素数です。
uiHeight	[in]	保存対象画像の高さ方向画素数です。
pszPath	[in]	画像保存先のファイル Path です。 指定されたファイルのフォルダが実在し、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、5.8. ステータスコード を参照してください。

[備考]

TeliCamUtl.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
pvDst	[in]	変換先画像データです。
pvSrc	[in]	変換元の画像データです。
uiPixelFormat	[in]	変換元画像の PixelFormat です。
uiWidth	[in]	変換元画像の幅方向画素数です。
uiHeight	[in]	変換元画像の高さ方向画素数です。
bRevX	[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。
bRevY	[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、5.8. ステータスコード を参照してください。

[備考]

TeliCamAPI.h と TeliCamUtl.h をインクルードする必要があります。

5.8. 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	機能がカメラまたはAPIに実装されていません。 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いた。　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_IO_DEVICE_ERROR	0x00000020	コントローラから I/Oエラーが通知されました。
CAM_API_STS_LOGICAL_PARAM_ERROR	0x00000021	論理エラーが通知されました。
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_NO_SYSTEM_RESOURCES	0x000005AA	システム リソースが不足しているため、要求されたサービスを完了できませんでした。
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_STATUS	値	内　容
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_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_VERIFY_ERR	0x00000809	カメラのレジスタにデータを書き込んだとき、ベリファイエラーが発生しました。
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_REMOVE	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_MAPPING_ERROR	0x00001010	ユーザバッファをシステム空間の仮想アドレスにマッピングする際にエラーが発生しました。 システムリソースが少ないために発生した可能性があ

CAM_API_STATUS	値	内　　容
		ります。
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_INVALID_PARAMETER_FROM_CAM	0x00008002	カメラから無効なパラメータエラーがリターンされました。
CAM_API_STS_SI_PAYLOAD_SIZE_NOT_ALIGNED	0x0000A003	ストリームインターフェースのペイロードサイズ設定にアライメントされていない異常がありました。
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_DATA_OVERRUN	0x0000A101	カメラはすべてのデータを送信することができませんでした。　SIRMレジスタの設定に問題がある可能性があります。
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

サンプルコードを理解するためには GenICam、GigE Vision、USB3 Vision、IIDC2 レジスタマップなどの知識が必要になる場合があります。　これらの規格に関する詳細情報、最新情報はそれぞれの規格提供元のホームページを参照してください。

GenICam	http://www.emva.org/cms/index.php?idcat=47&lang=1
GigE Vision	http://www.visiononline.org/vision-standards-details.cfm?type=5
USB3 Vision	http://www.visiononline.org/vision-standards-details.cfm?type=11
IIDC2	http://jiia.org/standard_dl/ngcp-wg/

6.1. Windows 版のサンプルソース

TeliCamSDK は C++ユーザプリケーション実装の参考として以下の表に記載した 9 個のサンプルソースを同梱しています。.Net 用のサンプルについては、TeliCamDNetAPI Library Manual に記載しています。サンプルソースは順次追加予定です。

サンプル名	言語	UI	機能	備考
Camera_ Information	VC2005	CUI	カメラ情報の表示。	5.1 章、5.2 章の API 使用
Camera_ ControllingFunction	VC2005	CUI	カメラ制御関数を使用したパラメータの取得と設定。	5.1 章、5.2 章、5.5 章の API 使用
Stream_ FreerunCallback	VC2005	CUI	Callback 関数を使用したイメージバッファの画素値表示。(連続取込)	5.1 章、5.2 章、5.3 章の API 使用
Stream_ FreerunLockBuffer	VC2005	CUI	LockBuffer 関数を使用したイメージバッファの画素値表示。(連続取込)	5.1 章、5.2 章、5.3 章の API 使用
Stream_ SWTrgReadCurrentImage	VC2005	CUI	ReadCurrentImage 関数を使用したイメージバッファの画素値表示。(トリガー受付取込)	5.1 章、5.2 章、5.3 章の API 使用
Stream_ Lowlevel	VC2005	CUI	低水準ストリーム関数を使用したイメージバッファの画素値表示。(連続取込)	5.1 章、5.2 章、5.3 章の API 使用
CameraEvent	VC2005	CUI	FrameTrigger イベント取得。	5.1 章、5.2 章、5.3 章、5.4 章の API を使用
MultiCamera	VC2005	GUI	最大４カメラ画像同時表示	5.1 章、5.2 章、5.3 章、5.4 章、 5.5 章、5.5 章の API 使用
MultiCameraPrimitive	VC2005	GUI	最大４カメラ画像同時表示	5.1 章、5.2 章、5.3 章、5.4 章、 5.5 章、5.5 章の API 使用

これらのサンプルソースは TeliCamSDK がインストールされているフォルダの下の Samples フォルダに入っています。 Windows7 以降の OS ではこのフォルダへの書き込みには管理者権限が必要です。サンプルソースをコンパイルする場合は「マイ　ドキュメント」などの書き込み可能なフォルダにコピーしてからコンパイルしてください。

6.2. Linux 版のサンプルソース

TeliCamSDK for Linux はユーザ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
Camera_Information	CUI	カメラ情報の表示。
Camera_ControllingFunction	CUI	カメラ制御関数を使用したパラメータの取得と設定。
Stream_FreerunCallback	CUI	Callback 関数を使用した画像の連像取込。
Stream_FreerunLockBuffer	CUI	LockBuffer 関数を使用した画像の連像取込。
Stream_SWTrgReadCurrentImage	CUI	ReadCurrentImage 関数を使用した画像のトリガー取込。
Stream_LowLevel	CUI	低水準ストリーム関数を使用した画像の連続取込。
CameraEvent	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

6.2.1. 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

6.2.2. 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. 免責事項

このソフトウェアの免責事項は、別途付属の”TeliCamSDK_License_Agreement_J.pdf” または ” 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以下のフォルダにインストールされています。

Windows 版　：　[TeliCamSDK インストールフォルダ]\Licenses
Linux 版　：　/opt/TeliCamSDK/licenses

7.2. 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、7.4 項に記載の弊社お問い合わせ窓口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。

Windows 版　：　[TeliCamSDK インストールフォルダ]\Licenses
Linux 版　：　/opt/TeliCamSDK/licenses

東芝テリー株式会社は、東芝テリー株式会社が定める条件の基で TeliCamSDK の動作を保証します。（以下のドキュメントをご覧ください。

Windows 版　：　”License Agreement TeliCamSDK Jpn.txt”,
“License Agreement TeliCamSDK Sample Jpn.txt”
Linux 版　：　”License Agreement TeliCamSDK for Linux Jpn.txt”,
“License Agreement TeliCamSDK for LinuxSample Jpn.txt”)

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ら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、東芝テリー株式会社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。

Windows 版

対応ソフトウェアモジュール	ライセンス
GenICam GenApi	GenICam License

Linux 版

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

GenICam GenApi は以下のサードパーティソフトウェアを使用しています。

対応ソフトウェアモジュール	ライセンス
MathParser	LGPLv2.1
Log4Cpp	LGPLv2.1
CppUnit	LGPLv2.1
CLSerAll	NI license
xs3p	DSTC license
xxhash	xxhash license
XSLTProc	MIT license
XSDe	Proprietary

TeliCamSDK は、LGPL 適用ソフトウェアのバイナリを再配布しており、これらのソースコードに限っては、 LGPL の定めに従い、入手、改変、再配布する権利をお客様は有します。
ソースコードはご希望のお客さまへは、メディア（CD-ROM 等）に書き込み郵送にてお送りします。　送料等実費にてご提供させていただいておりますので、ご希望の場合は 7.4 項に記載の弊社お問い合わせ窓口までお問い合わせください。　尚、ソースコードは、お客様が入手の権利を有するオープンソースソフトウェアのみ配布いたします。（TeliCamSDK のソースコードは含まれません。）
ソースコードの内容などについてのご質問にはお答えいたしかねますので、あらかじめご了承ください。

Microsoft、Windows、Windows XP、Windows Vista、Windows 7、Windows 8.1、Windows 10 及び、Visual C++は、Microsoft 社の商標もしくは登録商標です。
GigE Vision™ は AIA(Automated Imaging Association)が策定した統一規格です。
USB3 Vision™ は AIA(Automated Imaging Association)が策定した統一規格です。
GenICam™ は EMVA(European Machine Vision association)の商標もしくは登録商標です。
その他、この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。

7.3. 改定履歴

Date	Version	内容
2014/09/11	1.0.0	初版
2014/10/17	1.0.1	・ステータスコード追加 ・関数名の変更 (Cam_PortReset → Cam_ResetPort)
2014/11/06	1.0.2	・Strm_OpenSimple() の uiApiBufferCount デフォルト値を変更（5 → 8） ・Evt_OpenSimple() の uiApiBufferCount デフォルト値を変更（5 → 8） ・関数追加（GetCamUserDefinedName() 、SetCamUserDefinedName() ）
2015/02/13	1.0.3	・「2．構成」「4．ライブラリ」本文差し替え。 ・以下の仕様変更に伴い5.2.2, 5.2.3, 5.3, 5.3.1.1, 5.3.2.1, 5.4, 5.4.1.1, 5.4.2.1 の説明変更 ✓ 他のアプリケーションで使用中のカメラのオープンが可能になった。 ✓ GenICamモジュール不使用条件でカメラを使用するアプリケーションの実行環境ではGenApiのインストールが不要に変更。 ・「5.ライブラリ関数」の記載形式変更、各関数説明文の見直し
2015/06/12	1.0.4	・Windows 8.1 対応の記載
2015/10/21	1.0.5	・カメラ接続可能台数の記載
2016/07/12	2.0.0	・Windows 10 対応の記載 ・Windows XP＆Windows Vista 対応を削除 ・FrameBurst 機能に対応 ・関数追加（SetCamLineModeAll() ）
2016/12/16	2.0.1	・Chunk に関する関数を追加 ・UserSetControl に関する関数を追加
2017/06/13	2.0.2	・Strm_Abort() 関数追加 ・ステータスコード追加 ・各項目の説明を修正
2017/09/05	2.0.3	・Strm_OpenSimple() の引数 uiApiBufferCount で指定できる範囲を変更しました。（最小値：3 → 1 、最大値：30 → 128）
2018/01/25	2.0.4	・新しいカメラ制御関数の説明を追加 ・新しい GenICam 関数の説明を追加 ・ステータスコード追加
2018/06/29	2.0.5	・マルチキャスト対応の記載 ・シングルフレームモード対応の記載 ・関数追加（Cam_GetMulticast(), Cam_SetMulticast, ExecuteCamAcquisitionStart() ） ・ステータスコード追加 ・「6.サンプルソース」を改定
2019/02/07	2.0.6	・「3.動作環境」を改訂 ・Evt_Activate()、Evt_Deactivate() の説明を修正 ・ステータスコード追加
2020/01/08	3.0.0	・Windows 版と Linux 版のライブラリマニュアルを統合 ・カメラの 高フレームレートモード（HighFramerateMode） と 超短時間露光モード（ShortExposureMode）に対応した関数を追加

7.4. お問い合わせ

TeliCamSDK ならびにGigE Vision カメラ、USB3 Vision カメラに関するよくあるご質問とその回答(FAQ)は、下記のWebサイトをご利用ください。

https://secure.toshiba-teli.co.jp/ttfa/web/faq_j/top.html

それでも解決できない場合は、下記Webサイトの「お問い合わせ窓口」までお問い合わせください。

https://www.toshiba-teli.co.jp/support/contact/industrial_j.htm

