VRmUsbCam API v2

重要: このドキュメントは、読者が十分な経験のあるソフトウェア開発者で、最新の開発プラットフォームに習熟していることを前提としています!

目次

1 概要

VRmUsbCam API v2は、VRmagicカメラとアナログビデオコンバーターにすばやく簡単にアクセスできるようにしたものです。このインターフェースの主な機能は、フレームの取り込みとデバイスのコンフィギュレーションです。対応しているホストコントローラーは、デバイスのタイプに依存します。

1.1.1 Streaming

Streamingデバイスは、USB 2.0(EHCI)ホストコントローラーを必要とします。

- VRmC-3+(i)(/BW)
- VRmC-4(/BW)
- VRmC-4+(/BW)
- VRmC-8
- VRmC-8　1つの外部VRmS-8センサー付き
- VRmC-8+
- VRmC-9/BW
- VRmC-9　1つの外部VRmS-9センサー付き
- VRmC-9+/BW
- VRmC-12(/BW)
- VRmC-12(/BW)　1つの外部VRmMS-12センサー付き
- VRmC-12(/BW)　1つの外部VRmS-12センサー付き
- VRmC-12+(/BW)
- VRmC-14(/BW)
- VRmC-14(/BW)　1つの外部VRmS-14センサー付き
- VRmAVC-1
- VRmAVC-1+S
- VRmAVC-1+I

1.1.2 Smart

Smartデバイスは、USB 1.x (UHCI/OHCI)およびUSB 2.0 (EHCI)ホストコントローラーに対応しています。

- VRmFC-22(/BW)
- VRmFC-42(/BW)
- VRmMFC　4つまでの外部VRmMS-12センサー用VRmMSA2フロントエンド
- VRmMFC　2つまでのREセンサー用VRmMSARE1フロントエンド

1.1.3 Intelligent

Intelligentデバイスは、デバイス自体のローカルインターフェースと、TCPとUDPを使用するイーサネット上のストリーミングに対応しています。

- VRmDC-8
- VRmDC-9/BW
- VRmDC-12(/BW)
- VRmDC-12(/BW)　1つの外部VRmS-12センサー付き
- VRmDC-14(/BW)
- VRmDC-14(/BW)　1つの外部VRmS-14センサー付き
- VRmDFC-22(/BW)

- VRmDFC-42(/BW)
- VRmDMFC　4つまでの外部VRmMSC12センサー用VRmMSA2フロントエンド
- VRmDC-X-Eカメラ　1枚の外部センサーボード付き

2 VRmUsbCam C API v2

2.1 プロジェクトの扱い

最も扱いやすい方法は、`vrmusbcam2.h`をインクルードして、`vrmusbcam2.lib`をリンクすることです。

Microsoft Visual C++では、以下の手順になります:

- “リンカ→入力→追加の依存ファイル”に“vrmusbcam2.lib”を追加。
- “リンカ→全般→追加のライブラリディレクトリ”に以下のパスを追加。
  “<InstallDir>¥VRmUsbCam Library¥lib”、<InstallDir>はVRmagic USBカメラ開発キットをインストールしたフォルダで、通常は“C:¥Program Files¥VRmagic”になります。
- “C/C++→全般→追加のインクルードディレクトリ”に以下のパスを追加。
  “<InstallDir>¥VRmUsbCam Library¥include”、 <InstallDir>はVRmagic USBカメラ開発キットをインストールしたフォルダで、通常は“C:¥Program Files¥VRmagic”になります。

2.2 エラーの扱い

どのAPIファンクションでも、タイプ`VRmRetVal`の返り値で成功または失敗を示します。ファンクションが`VRM_FAILED`を返す場合は、`VRmUsbCamGetLastError()`を使用して、エラーの説明をCストリングとして取得できます。カスタマサポートの場合は、`VRmUsbCamEnableLogging()`を呼び出すことによってライブラリ内蔵のログ機能を有効にすることもできます。その他の状況でログ機能を使用することは推奨しません。

2.3 デバイスの管理

2つの基本的なステップが必要です:

- **デバイスのスキャン**
  カメラデバイスを使用する前に、`VRmUsbCamUpdateDeviceKeyList()`を使用して、VRmagicデバイス用のUSBバスをライブラリがスキャンするようにしなければなりません。この後で、`VRmUsbCamGetDeviceKeyListSize()`と`VRmUsbCamGetDeviceKeyListEntry()`を使用して、見つかった各デバイスについて`VRmDeviceKey`を取得できます。
  使用後のキーは、`VRmUsbCamFreeDeviceKey()`を使用して解放しなければなりません。

- **デバイスの開/閉**
  `VRmUsbCamDevice`はシングルデバイスへのハンドルです。`VRmUsbCamOpenDevice()`を`VRmUsbCamGetDeviceKeyListEntry()`によって返される`VRmDeviceKey`とともに使用して、特定のデバイス用の有効なハンドルを取得します。この操作が成功すると、ハンドルは`VRmUsbCamCloseDevice()`を使用して閉じるまで使用することができます。
  備考: *開くことができるデバイスはビジーでないデバイスのみです*(`VRmDeviceKey`の`!m_busy`でチェックしてください)。

2.4 画像の扱い

APIは、複数のカラーフォーマット(`VRmColorFormat`)と画像変更フラグ(`VRmImageModifier`)に対応しています。`VRmImageFormat`は、サイズ(幅と高さ)、カラーフォーマット、画像変更フラグのビットの組み合わせを結合します。`VRmImage`は`VRmImageFormat`ストラクト、画像バッファー、ピッチ、タイムスタンプで構成されたストラクトです。
さらに、フレームグラバーAPIによる画像のフレームカウンターを取得するのに`VRmUsbCamGetFrameCounter()`を使用することができます。

画像は以下のファンクションのどれかを使用して作成することができます:

- `VRmUsbCamNewImage()`
  所定のフォーマットの新規画像を割り当てます。
- `VRmUsbCamCopyImage()`
  既存の画像のコピーとして新規画像を作成します。
- `VRmUsbCamCropImage()`
  既存の画像の四角形の部分となる画像を作成します。元の画像が有効なまま残っていないと、返される画像は有効でないので注意してください。
- `VRmUsbCamSetImage()`
  フォーマットとバッファーによって表される既存の画像用のコンテナを作成します。

これらのファンクションのどれかによって得られた画像は`VRmUsbCamFreeImage()`を使用して解放されなければなりません。

様々なカラーフォーマットに対応しています:

- ARGBカラーフォーマット(32ビット/ピクセル)
- BGRカラーフォーマット(24ビット/ピクセル)
- RGB565カラーフォーマット(16ビット/ピクセル)
- GRAY8フォーマット(8ビット/ピクセル)
- YUYVカラーフォーマット(16ビット/ピクセル)
- GRAY10フォーマット(16ビット/ピクセル)　(S)
- BAYER8カラーフォーマット(8ビット/ピクセル)　(S)
- BAYER10カラーフォーマット(16ビット/ピクセル)　(S)

さらに圧縮画像フォーマットもあります:

- GRAY8/RLE　(S)
  GRAY8フォーマット(8ビット/ピクセル)、バイトワイズ損失なしRLE (下を参照)
- BGR/RLE　(S)
  BGRカラーフォーマット(24ビット/ピクセル)、バイトワイズ損失なしRLE (下を参照)

備考: *(S)のマークがあるフォーマットは、ソースフォーマット(デバイスの生のフォーマット)として機能できるだけで、変換ターゲットフォーマットとしては使用されません!*
`VRmUsbCamConvertImage()`は、各ピクセルをターゲット画像フォーマットに変換しながらソースの内容をターゲット画像にコピーします。ターゲット画像フォーマットは、`VRmUsbCamGetTargetFormatListSize[/Entry]()`から返されるフォーマットの1つでなければならず、ソースフォーマットが最初のパラメーターとして与えられます。自前の`VrmImageFormat`ターゲットフォーマットを勝手に作成すると、`VRmUsbCamConvertImage()`を呼び出した場合にエラーが生じることがあります。既存の画像フォーマットで安全に変更してもよい唯一のパラメーターは、画像フォーマット変更の`VRM_HORIZONTAL_MIRRORED`と`VRM_VERTICAL_MIRRORED`です。

備考: *さらにフレームグラバーAPIから`VRmImage`を取得して(`VRmUsbCamLockNextImage()`を参照)、`VRmUsbCamGetTargetFormatListSize[/Entry]Ex()`ファンクションを使用して有効*

な変換プロパティに適用して、`VrmImageFormat`*ターゲットフォーマットを作成できます。*

備考: *「ベイヤー」カラーフォーマットの原画像の変換は、高速の低品質の変換または低速の高品質の変換を使用して実行することができます。初期設定は低品質で、高品質にするには*`VRM_PROPID_CONVERTER_BAYER_HQ_B`*ロパティを使用します。*

標準(四角形)の画像フォーマット

初期設定では、メモリ内の画像データは四角形に整えられています。1つのシングル画像ピクセルに属するバイト数は使用中のカラーフォーマットによります。
`VRmImage`の`mp_buffer`メンバーによって示される画像データバッファーは、画像の各ラインを直線的に格納し、通常はそれらの間にパディングバイトがあります。結果のラインの長さ(すべてのデータバイト+パディングバイト)は画像の *ピッチ*と呼ばれます。
したがって、画像データバッファー内のオフセット#0は、画像の左上ピクセルの最初のバイトを格納します。

備考: *画像データバッファーの左上エッジは画像の「物理的な左上」エッジと異なる場合があり、これは画像変更子VRM_HORIZONTAL_MIRRORED とVRM_VERTICAL_MIRRORED によって示されます。*

座標(x、y)のピクセルに属するデータを見つけるのは簡単です: 画像データバッファー内の対応するオフセットバイトは*(x*pixeldepth+y*pitch)*になり、*pixeldepth*は`VRmUsbCamGetPixelDepthFromColorFormat()`によって画像のカラーフォーマットから決定されます。

備考: *VRmAVC-1 とVRmFAVC-1 のインターレースフルフレームソースフォーマットは、VRM_INTERLACED_FRAME画像変更子が示すように整え方がわずかに異なります: 画像の上半分は第 1 フィールドと、下半分は第2フィールドと呼ばれます。*

ランレングスエンコーディング(RLE)フォーマット

RLEソースフォーマットは、GRAY8またはBGR24ソースフォーマットでスマートカメラ(VRmFC-x)でのみ利用可能です。それらは`VRM_RUN_LENGTH_ENCODED`画像変更子フラグセットによって示されます。
バイトワイズ損失なしRLE圧縮は、ホストコンピュータではなくデバイスによって実行されるので、転送データレートを減少させることになり、適切なタイプの画像ではより高速のフレームレートが可能になります。

備考: *使用されるRLE圧縮アルゴリズムは、非圧縮の生の画像よりも少ないデータを生成することを保証するものではありません。全体でNデータバイトであると、結果のRLE画像は、2*Nデータバイトまで増えることがあります。他方で、均質なカラー画像では合計で2*(N/255+1)データバイトまで減少します。*

一般に、カラー領域の均質な部分が大きい画像は圧縮の結果が良好になりますが、不均質な画像はデータが増えます。圧縮が行われる前に補正フィルターLUTが適用されるので、圧縮ゲインを最大にするために、黒レベル、コントラスト、ガンマ設定を使用することができます。

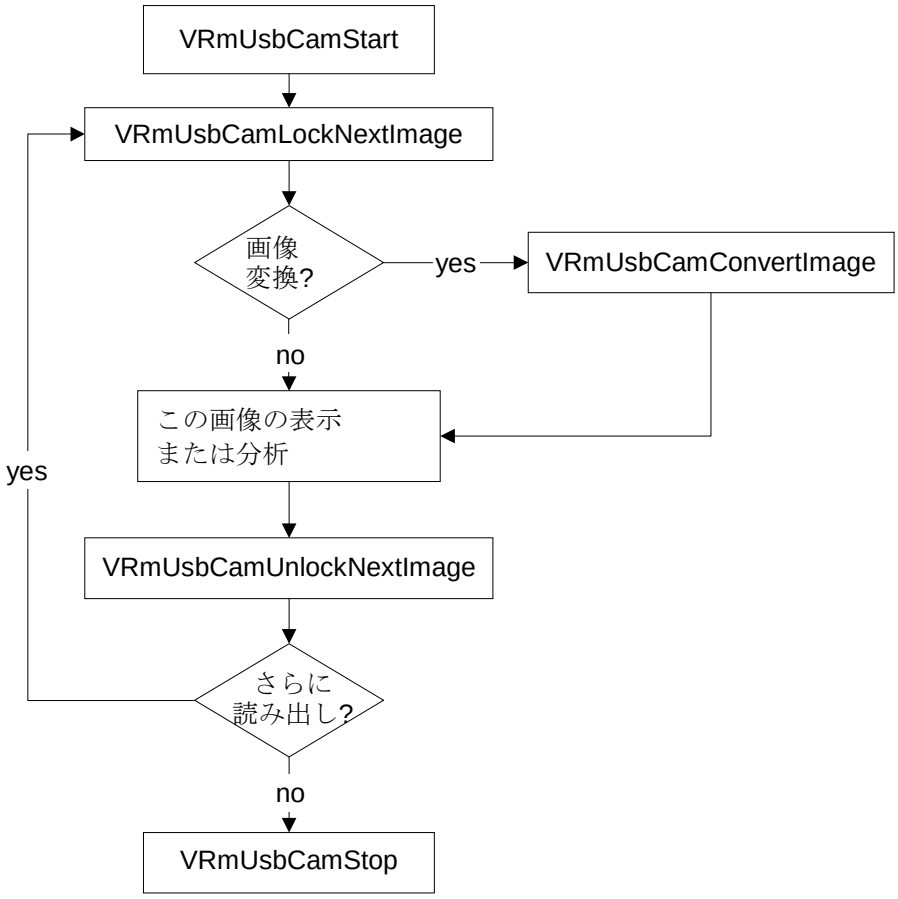
RLE圧縮データは、画像データバッファー内のオフセット#0から開始して、バイトペア(V、L)に配列されます。ピッチ情報は無視されるので、バイトペアは特別なEOF(フレーム終了)マーカーまで直線的に間断なく格納されます。
各バイトペアは「値」Vと「長さの情報」L(値Vの反復数)からなっています。値Vの意味は使用中のカラーフォーマットによります。

備考: *データバイトの総量はVRmUsbCamGetImageBufferSize()で取得することができます。所定の量のデータバイトの後にデコーディングが終了したのを確認することによってRLEデータの整合性をチェックするのにこれを使用することができます。*

2.5 フレームグラバー

VRmUsbCamライブラリは、カメラデバイスから連続した画像を取り込むことができます。取り込みプロセスは、`VRmUsbCamStart()`ファンクションによって開始して、`VRmUsbCamStop()`によって停止します。内部の画像待ち行列(リングバッファー)から画像にアクセスするには`VRmUsbCamLockNextImage()`を使用します。ロックされた画像の処理後に、再びリソースを解放するには`VRmUsbCamUnlockNextImage()`を使用しなければなりません。
内部のリングバッファーのサイズによって、最大で一度にいくつの画像をロックすることができるか決まります。これはカメラの`VRM_PROPID_GRAB_HOST_RINGBUFFER_SIZE_I`のプロパティです(セクション2.7を参照)。スマートデバイスは、`VRM_PROPID_GRAB_DEVICE_RINGBUFFER_SIZE_I`を使用してサイズを変更できるデバイス上のリングバッファーも提供します。

簡単なフローチャートで、一度に1枚の画像を取り込む基本的な取り込みプロセスを示します:



取り込み時に、デバイスからのUSB転送が遅れる場合があります。結果として1枚以上の画像が欠落して転送は自動的に再開します。そのようなことが起こった場合にわかるように、ライブラリには`VRmUsbCamLockNextImage()`ファンクションの`fp_frames_dropped`パラメーターがあります。`VRmUsbCamLockNextImage()`によって得られる各画像には、ミリセカンド(`m_time_stamp`)で与えられるタイムスタンプとフレームカウンター(`VRmUsbCamGetFrameCounter`)があり、欠落したフレームと欠落したトリガーイベントも含まれます。

DirectDraw C++デモアプリケーションのソースコードから抽出した以下のコードでプロセスがよくわかります:

```
// prepare image format for locked ddraw surface.
VRmImageFormat target_format;
if(!VRmUsbCamGetTargetFormatListEntryEx(device,0,&target_format))
    LogExit();

// start grabber at first
if(!VRmUsbCamStart(device))
    LogExit();

// and enter the loop
while (!g_quit)
{
    // lock the DirectDraw off-screen buffer to output the image to the screen.
    // The screen_buffer_pitch variable will receive the pitch (byte size of
    // one line) of the buffer.
    VRmDWORD screen_buffer_pitch;
    VRmBYTE* p_screen_buffer=DDrawLockBuffer(screen_buffer_pitch);

    // now, wrap a VRmImage around the locked screen buffer to receive the converted image
    VRmImage* p_target_img=0;
    VRmUsbCamSetImage(&p_target_img,target_format,p_screen_buffer,screen_buffer_pitch);

    // lock next (raw) image for read access, convert it to the desired
    // format and unlock it again, so that grabbing can
    // go on
    VRmImage* p_source_img=0;
    VRmBOOL frames_dropped;
    if(!VRmUsbCamLockNextImage(device,&p_source_img,&frames_dropped))
        LogExit();
    if(!VRmUsbCamConvertImage(p_source_img,p_target_img))
        LogExit();
    if(!VRmUsbCamUnlockNextImage(device,&p_source_img))
        LogExit();

    // see, if we had to drop some frames due to data transfer stalls. if so,
    // output a message
    if (frames_dropped)
        cout << "- frame(s) dropped -" << endl;

    // free the resources of the target image
    if(!VRmUsbCamFreeImage(&p_target_img))
        LogExit();
    // give the off-screen buffer back to DirectDraw
    DDrawUnlockBuffer(p_screen_buffer);

    // and update the screen
    DDrawUpdate();
}
// stop grabber
if(!VRmUsbCamStop(device))
    LogExit();
```

すべてのデバイスは、複数のソース(「生」)の画像フォーマットに対応しています。これらのフォーマットのリストを取得するには`VRmUsbCamGetSourceFormatListSize()`と`VRmUsbCamGetSourceFormatListEntry()`を使用します。グラバーが動作していない間に、`VRmUsbCamSetSourceFormatIndex()`を使用してそれらの1つを有効にします。

アプリケーションによっては、デバイスの原画像の処理で十分ですが、多くの場合は、原画像を周知のカラーフォーマットに変換する必要があります。これはAPIの画像変換機能を使用して行うことができます。
ファンクション`VRmUsbCamGetTargetFormatListSize()`と`VRmUsbCamGetTargetFormatListEntry()`によって所定の原フォーマットを変換できる画像フォーマットのリストにアクセスできます。それから画像変換は`VRmUsbCamConvertImage()`によって実行されます。

対応しているターゲット画像フォーマット:
0: ARGBカラーフォーマット
1: BGRカラーフォーマット
2: RGB565フォーマット
3: グレースケールフォーマット
4: YUYVカラーフォーマット

備考: `VRmUsbCamGetTargetFormatListSize[/Entry]Ex()` *ファンクションを使用する場合、ターゲット画像フォーマットの順序は*`VRM_PROPID_CONVERTER_PREFER_GRAY_OUTPUT_B` *プロパティによります。*

2.6 トリガーファンクション

トリガーの機能によって画像露光の時点をコントロールできます。カメラは2種類のトリガーシグナルを受け入れます:

- ソフトトリガー
  `VRmUsbCamSoftTrigger()`はできるだけ高速に画像を露光します(これにはファームウェアのアップデートが必要なので、詳しくはVRmagicのサポート部に問い合わせてください)。ソフトトリガーは、外部の配線なしで複数のカメラを同期させるのに使用することができます。

- 外部トリガー
  この場合、画像露光は何らかの外部の電気的シグナルによって引き起こされます。
  これは対応する外部コネクタを備えたカメラのみで使用できます。

すべてのデバイスがトリガーファンクションに対応しているわけではありません。どのトリガーモードが利用可能であるかを見るには利用可能な選択肢がわかる`VRM_PROPID_GRAB_MODE_E`プロパティを参照します。1つまたは両方のトリガーモードを有効にするには、プロパティを対応する値に設定します。

外部トリガーファンクションは、`VRM_PROPID_CAM_STROBE_POLARITY_E`プロパティを使用して、異なった極性の「エッジ」または「レベル」のトリガーに設定することができます。

2.7 コンフィギュレーション設定(プロパティ)

デバイスの各機能をコントロールするために様々なパラメーターを変更することができます。よくある例は「露光時間」や「ピクセルクロック」です。v2.6.0.0では、VRmUsbCam APIはこれらをコントロールする一貫した方法としてプロパティインターフェースを提供しています。旧ヴァージョンでは`VRmSettings1`、`VrmSettings2`のようなストラクトのAPI使用が、同じ結果を得るのに必要でした(これは互換性のためにまだ可能になっていますが、*ここに示した高度なメソッドを使用することを強く推奨します*。古い方法を使用すると、コードをデバイスで横断的に使用することが多少妨げられる場合があるからです)。
まず、プロパティインターフェースは現在のデバイスが対応するすべてのパラメーターを、短い記述、初期設定値、(オプションで)許容範囲を含めてリストにすることができます。
また、個別のパラメーターの設定や取得をする機能も提供します。

「プロパティ」は識別子(たとえば「`VRM_PROPID_CAM_EXPOSURE_TIME_F`」)、短い記述(たとえば露光時間[ms])、可能な変更についての情報(たとえばプロパティが書き込み可能か読み出しのみか)、現在の値、初期設定値、値の範囲(たとえば最小値と最大値に関する情報)からなっています。また、いくつかのプロパティは、値の可能な最小の変更量を示すステップ値も含んでいます。たとえば露光時間プロパティのステップ値は0.1[ms]で、これはもちろん露光時間の可能な最小の変更量は0.1msであることを意味しています。

プロパティの様々な情報を使用すると扱いが簡単になります。たとえば、以下のC++デモアプリケーションからの抽出コードが示しているように、露光時間を25msに簡単に変更できます:

```
float value=25.f;
if (!VRmUsbCamSetPropertyValueF(device,VRM_PROPID_CAM_EXPOSURE_TIME_F,&value))
    LogExit();
```

プロパティには様々なタイプのパラメーターがあります。パラメータータイプはプロパティの識別子に示されています。*float*のようなインテグラルタイプのパラメーターがあり(したがって露光時間プロパティの識別子は“VRM_PROPID_CAM_EXPOSURE_TIME_F”; “_F”で示されます)、*boolean* (“_B”)、*integer* (“_I”)の値や、もっと複雑なタイプでは*VRmPointI* (“_POINT_I”)、*VRmRectI* (“_RECT_I”)、*VRmSizeI* (“_SIZE_I”)があります。対応しているパラメータータイプの組はVRmPropType列挙にあり、これはAPIヘッダーファイル“vrmusbcam2.h”に含まれています。
どのパラメータータイプも対応するファンクションによって扱われます。たとえば*Integer*プロパティ(“_I”)は、ファンクションVRmUsbCamSetPropertyValueI()によって変更できます。*float*プロパティはVRmUsbCamSetPropertyValueF()が必要です; やはり最後の文字がプロパティタイプを示します。
特別なパラメータータイプは*Enum* (“_E”)タイプです。このパラメータータイプを特徴とするプロパティは、一定の範囲または数値の値を含むことができません。そうではなく、これらの値はシンボリックなものです。したがって、たとえばVRM_PROPID_CAM_VIDEO_STANDARD_Eによって示されるプロパティは有効な値としてVRM_PROPID_CAM_VIDEO_STANDARD_PALまたはVRM_PROPID_CAM_VIDEO_STANDARD_NTSCのみがあります。

特定のプロパティの「名称」は (VRM_PROPID_CAM_EXPOSURE_TIME_Fのように) 識別子によって表されます。プレフィックス“VRM_PROPID_”はC API特有のもので、COMまたは.NET環境では異なります(以下の関連する章を参照)。プロパティ識別子のリストは“vrmusbcam2props.h”.というヘッダーファイルに含まれるVRmPropId列挙にあります。

プロパティ識別子を使用すると、特定のプロパティの現在の値を簡単に呼び出すことができます。これもC++デモアプリケーションにある例で示します:

```
float value;
if (!VRmUsbCamGetPropertyValueF(device,VRM_PROPID_CAM_EXPOSURE_TIME_F,&value))
    LogExit();
```

このようにVRmUsbCamGetPropertyValue<x>()が使用するファンクションで、<x>はもちろんパラメータータイプを示します。タイプから独立しているプロパティ情報を取得するにはVRmUsbCamGetPropertyInfo()を使用します。これが提供するVRmPropInfoというストラクトには、プロパティのIDストリング、タイプ、記述、書き込み可能フラグを含んでいます。所定のプロパティのタイプに依存する情報を見つけたい場合は、VRmUsbCamGetPropertyAttribs<x>()を使用できます。これらは、初期設定値、最小値、最大値、ステップ値を含んでいます。

備考: *どのデバイスでも(ヘッダーファイルにある)すべての可能なプロパティに対応するというわけではありません*!

デバイスの対応しているプロパティを簡単に見つけるには、この配布に含まれている*XmlDeviceInfo*というツールを使用します[1]。これはスタートメニュー"VRmagic VRmUsbCam Library XmlDeviceInfo (VB.NET)"によってアクセス可能で、コンピュータに接続されているデバイスでVRmUsbCam APIによって対応されるデバイスのプロパティのすべてのリストを挙げます。複数のデバイスが検出された場合は、1番目のデバイスのプロパティのみが取得されます。

[1] *XmlDeviceInfo*を使用するには.NET frameworksをインストールする必要があります。

以下のC API呼び出しの使用法を示すためにC++ (democ++deviceinfo)には類似したツールも含まれています: `VRmUsbCamGetPropertyListSize()`、`VRmUsbCamGetPropertyListEntry()`、`VRmUsbCamGetPropertySupported()`、`VRmUsbCamGetPropertyInfo()`。

ほとんどすべての設定(現在のプロパティ値)は、`VRmUsbCamLoadConfig()`と`VRmUsbCamSaveConfig()`をそれぞれ使用してデバイスの不揮発性メモリから呼び出したり逆に保存したりすることができます。もちろん、読み出しのみのプロパティは除外されます。
`VRmUsbCamSaveConfig()`を使用して、9つまでの異なったコンフィギュレーションを作成して、いわゆる*Configuration Id*で識別できます。最初の2つのID、0と1は保留になり、0は「出荷時初期設定」、1は「ユーザー初期設定」です。出荷時初期設定を上書きすることはできません。
起動時にユーザー初期設定は自動的に読み出されます。
特定のコンフィギュレーションが不要になった場合は、`VRmUsbCamDeleteConfig()`を使用して削除して、占めているメモリを解放できます。
各コンフィギュレーションに任意の記述を割り当てることも可能です。プロパティ`VRM_PROPID_GRAB_CONFIG_DESCRIPTION_S`を参照してください。

API version 2.6.1.0では、VRmUsbCam C APIはグラフィカルユーザーインターフェース[2]によってデバイスを設定する便利な方法も提供しています。これはすでにVRmagic CamLabアプリケーションとVRmUsbCamDS取り込みソースの不可欠の部分になっています(第4章を参照)。
ヘッダーファイル“`vrmusbcam2win32.h`”がこれに必要なファンクションを定義します: `VRmUsbCamCreateDevicePropertyPage()`と`VRmUsbCamDestroyDevicePropertyPage()`です。

2.8 コールバック

あるイベントが起こる場合に呼び出される様々なコールバックフックファンクションをVRmUsbCam APIに登録することができます。

現在は2つのグループのコールバックがあります:

- 静的コールバック(VRmUsbCamグローバル)
  `VRmUsbCamRegisterStaticCallback()`と
  `VRmUsbCamUnregisterStaticCallback()`を参照。
- デバイスコールバック(特定のVRmUsbCamデバイスに関連)
  `VRmUsbCamRegisterDeviceCallback()`と
  `VRmUsbCamUnregisterDeviceCallback()`を参照。

`VRmStaticCallbackType`と`VRmDeviceCallbackType`列挙はそれぞれ登録されるすべてのイベントタイプをリストします。コールバックファンクションを登録するのは一度だけです。
静的コールバックは、GUIループつまりWin32ファンクション`GetMessage()`と`DispatchMessage()`を動作させる必要があります。それに対してデバイスコールバックはVRmUsbCam APIの呼び出し内で同時的に呼び出されます。

2.9 ユーザーデータ

状況によっては、デバイスのオペレーションに関連しない特定のデータ(たとえばデバイスの位置、メンテナンス手順、画像など)をデバイス自体に格納するのが便利な場合があります。
この要求を満たすために、どのVRmUsbCamデバイスもそのようなデータを不揮発性メモリに格納できるようにしています。
VRmUsbCam C API内で、この機能は4つの異なったファンクションによってアクセス可能です:

[2] 現在はMicrosoft Windowsでのみ対応しています。

- `VRmUsbCamLoadUserData()`と`VRmUsbCamSaveUserData()`はそれぞれ、デバイスの不揮発性メモリからデータを読み出すのと逆に格納するのに用いられます。
- `VRmUsbCamNewUserData()`と`VRmUsbCamFreeUserData()`は、読み出しと格納のファンクションに使用されるデータの作成や削除に関係するホスト自体のメモリ管理を扱います。

一度に1ユニットのユーザーデータ(たとえば1ファイル)のみしかどのデバイスにも格納することができません。さらにメモリスペースは限られています。古いデバイスには約1KBの自由なユーザースペースがありましたが、現在のデバイスは約25KBが限界です。(ユーザーデータのための)全メモリと空メモリを見るには、読み出しのみのプロパティ`VRM_PROPID_DEVICE_NV_MEM_TOTAL_I`または`VRM_PROPID_DEVICE_NV_MEM_FREE_I`をそれぞれ呼び出します。

重要: *`VRmUsbCamSaveUserData()`によってユーザーデータを保存するには3秒ほどかかる場合があります。保存プロセスが完了する前にホストからデバイスを切断してはいけません!*

3 VRmUsbCam COMと.NET API v2

`vrmusbcam2.dll`によって提供されるネイティヴVRmUsbCam C API v2は別として、COM(コンポーネントオブジェクトモデル)クラスはVRmagic USBカメラ開発キットのセットアップウィザードによって登録されています。
これらはCOMオブジェクトを扱えるプログラミング言語内でVRmUsbCam C API v2のすべての機能にアクセスするのに使用できます。COMライブラリ“`VRmagic VRmUsbCam COM API v2`” (`vrmusbcam2.dll`)の`VRmUsbCam`オブジェクトをインスタンス化[3]するだけです。

Microsoft Visual Basic .NET や*Microsoft Visual C#*のような.NETベースの言語でもCOMは可能ですが、この目的で別のAPIが提供されています: *VRmUsbCamNET*アセンブリです。インストール時に.NET Frameworkが検出されると、これはセットアップウィザードによってグローバルアセンブリキャッシュにインストールされます。**VRmUsbCam COM APIではなく.NET言語内で使用することを強く推奨します。**
また、アプリケーション内から簡単にアクセスするために、開発キットのインストールフォルダの"VRmUsbCam¥wrappers¥net"サブディレクトリ内にもアセンブリがあります。

備考: *VRmUsbCamNETアセンブリを使用するには、System.Drawingアセンブリをインポートしなければなりません。*

VRmUsbCam COMと.NET APIのインターフェースは非常に似ています。どちらも前の節で説明したVRmUsbCam C APIのインターフェースに基づいています。したがって、以下の箇所ではCOM/.NET APIとC APIの使用での構造的な違いを指摘するだけです。

3.1 命名規則

C APIのすべてのタイプとストラクト識別子はプレフィックス"VRm"で始まりますが、VRmUsbCam COM/.NET API全体は、VRmUsbCamCOM/VRmUsbCamNETネームスペース内にあってプレフィックスは不要なのでありません。

3.2 クラスとストラクト

VRmUsbCam COM/.NET APIはファンクション(メソッド)とデータを意味の上で整えるので、それぞれのネームスペース内で以下のクラス/オブジェクトを定義しています:

- `DeviceKey`
  VRmagicデバイスの独自の識別子; VRmDeviceKeyに対応。
- `ImageFormat`
  シングル画像のフォーマット情報を保持; VRmImageFormatに対応。
- `Image`
  デバイスから直接読み出されたかデバイスから読み出された別の画像から変換されたシングル画像を保持; VRmImageに対応。
- `Device`
  シングルVRmagicデバイスの表現; VRmUsbCamDeviceに対応。
- `DevicePropertyPage`
  デバイスの一般的なパラメーターのコンフィギュレーションを可能にする簡単なGUIを提供; 親または子ウィンドウから独立可能。
- `Exception` (.NETのみ)
  VRmUsbCam .NETアセンブリ内のすべてのクラス/メソッドによって発せられる例外。

[3] Microsoft Visual Basic 6.0は`VRmUsbCam`を自動的にインスタンス化します。アプリケーションオブジェクトとして宣言されているためです。

- `VRmUsbCam`
  すべての「グローバル」メソッドのコンテナクラス。

備考: *COMでは、ImageFormatとVRmUsbCamは唯一のユーザー作成可能クラスです。ほかのすべてのクラスはVRmUsbCam COM APIのメンバーメソッドで取得できるのみです。*

さらに、ストラクトはC APIに従って定義されます:

- `PropInfo`
  デバイスプロパティに関する一般情報を受け取る; VRmPropInfoに対応。セクション2.7を参照。
- `PropAttribs`
  デバイスプロパティの初期設定値、最小値、最大値、ステップ値を受け取る; VRmPropAttribs<x>ストラクトに対応。セクション2.7を参照。
- `Size, Point, Rectangle` (.COMのみ[4])
  VRmSizeI、VRmPointI、VRmRectIに対応。

3.3 列挙

列挙はC APIに従って定義されるので、すべての列挙値にはVRM_プレフィックスが付きません:

- `ColorFormat`
  VRmColorFormatに対応。
- `ImageModifier`
  VRmImageModifierに対応。
- `PropId`
  VRmPropIdに対応。.NETには列挙定数のPROPID_プレフィックスも付きません。
- `PropType` (.COMのみ[5])
  VRmPropTypeに対応。

3.4 イベント

VRmUsbCam C APIのコールバックの定義はCOM/.NETの統合されたイベントモデルにマッピングされます。以下のクラスはイベントソースです:

- `VRmUsbCam`
  VRmStaticCallbackType列挙に従ってDeviceChangeイベント[6]を提供します。これらのイベントには動作中のGUIイベントループが必要です。
- `Device`
  `VRmDeviceCallbackType`列挙に従って`PropertyListChanged`、`PropertyValueChanged`、`SourceFormatChanged`、`SourceFormatListChanged`、`TargetFormatListChanged`イベントを提供します。
  これらのイベントはVRmUsbCam APIの呼び出し内で同時的に立ち上げられます。

4 .NETはビルトインタイプを使用します。
5 .NETはデバイスのプロパティタイプを識別するのにSystem.Typeを使用します。
6 .NETでは、これらのイベントは静的/共有です。すなわちインスタンスメンバーでなくタイプメンバーです。

3.5 レガシークラス、ストラクト、列挙

VRmUsbCam APIの古いヴァージョンとの互換性を維持するために、COMと.NETインターフェースは、新しいアプリケーションで使用されないクラス、ストラクト、列挙、メソッドをまだ含んでいます。
COMでは、それらは記述に“**OBSOLETE**”と記しています。あいにくVRmUsbCam .NET APIは現在、そのメソッド、クラス、ストラクトの記述/ヘルプストリングに対応していません。けれども代わりに、DeviceとVRmUsbCamクラスの廃棄されたメソッドを含む`_obsolete_Device`と`_obsolete_VRmUsbCam`クラスという2つの追加のクラスがあります。さらに、これらのメソッド/プロパティは"Obsolete"アトリビュートでマーク付けされています。

廃棄された機能の大部分はセクション2.7にあるようにDeviceプロパティに置き換えられています。

3.6 エラーの扱い

C API内のエラーの扱いはエラーコード(`VRmRetVal`)を返して`VRmUsbCamGetLastError()`ファンクションを与えることによって実行されますが、VRmUsbCam .NET APIは例外を利用します。したがって、ほとんどのメソッドは返り値を提供しませんが、`VRmUsbCamNET.Exception`を発する場合があります。
C APIにさらに似て、VRmUsbCam COM APIはCOMエラーコード(HRESULT)を返すことによってエラーを扱います。さらに、すべてのVRmUsbCam COMクラスはエラーの場合に内容の記述を提供するためにIErrorInfoインターフェースを実行しますが、ほとんどのCOM可能のプログラミング言語はこのインターフェースを自動的に使用します。

3.7 プロパティ

COMと.NETはプロパティモデルを提供するので、C APIの`VRmUsbCamGet<xxx>/Set<xxx>`ファンクションは、クラス`DeviceKey`、`ImageFormat`、`Image`、`Device`、`VRmUsbCam`のプロパティの読み出しのみプロパティまたは読み出し書き込みプロパティにマッピングされます。

備考: *これはセクション2.7に示したDeviceプロパティと混ざることはありません!*

- `DeviceKey`クラス: `VRmDeviceKey`ストラクトのメンバーに対応した読み出しのみのプロパティ:
  `Busy`、`Manufacturer`、`Product`、`Serial`
  さらに、C APIの`VRmUsbCamGetProduct/VendorId()`によって得られる`ProductId`と`VendorId`があります。

- `ImageFormat`クラス:
  - 配列タイププロパティ
    `TargetFormatList`
  - `VRmImageFormat`ストラクトのメンバーに対応するプロパティ:
    `Size`、`ColorFormat`、`ImageModifier`
    ImageModifierの部分的な変更は以下によって可能です。
    `FlipHorizontal()`および`FlipVertical()`

  さらに`PixelDepth`とグラバーソースフォーマットの場合は、短い`Description`も提供されます。
- `Image`クラス: C APIのVRmImageストラクトに従って読み出しのみのプロパティが定義されます:
  `Buffer`、`ImageFormat`、`Pitch`、`TimeStamp`
  `BufferSize`プロパティは、`VRmUsbCamGetImageBufferSize()`によって得られる実際のバッファーサイズを示します。`FrameCounter`もプロパティとして提供されます。画像バッファーへの直接アクセスは、`LockBits()`および`UnlockBits()`メソッドを使用して行うことができます。
- `PropInfo`クラス: VRmPropInfoストラクトに対応して読み出しのみのプロパティが定義されます:
  `Description`、`Id`、`IdString`、`Type`、`Writeable`
- `Device`クラス:
  - 配列タイププロパティ
    `SourceFormatList`、`TargetFormatList`[7]、`PropertyList`
  - 読み出しのみのプロパティ
    `DeviceKey`、`Running`、`PropertySupported`、`PropertyAttribs`、`PropertyInfo`
  - 読み出しと書き込みのプロパティ
    `SourceFormat`、`SourceFormatIndex`、`PropertyValue`
  - イベント
    `PropertyListChanged`、`PropertyValueChanged`、`SourceFormatChanged`、`SourceFormatListChanged`、`TargetFormatListChanged`
- `VRmUsbCam`クラス
  - 配列タイププロパティ
    `DeviceKeyList`
  - 読み出しのみのプロパティ
    `CurrentTime`、`Version`

3.8 VRmUsbCamDSキャプチャソースとの相互作用

VRmUsbCam COM/.NET APIは、設定とVRmUsbCamDSキャプチャソースフィルターのソース構成を選択するのに使用することができます。
したがって`VRmUsbCam`クラスは`AttachToVRmUsbCamDS()`メソッドを提供します。例としてはDirectShowデモアプリケーションを見てください。
`AttachToVRmUsbCamDS()`を呼び出すと、指定されたDirectShowキャプチャソースによって現在開かれているVRmagic Deviceを表す`Device`オブジェクトを取得できます。DirectShowがデバイスを完全にコントロールするので、メソッドのいくつかはこのオブジェクトで使用できないことに注意してください。つまり、以下の`Device`メソッドは禁じられます:
`Start()`、`Stop()`、`IsNextImageReady()`、`LockNextImage()`、`UnlockNextImage()`

終了したら、`Dispose()`メソッドを使用して`Device`オブジェクトを解放するだけです。

[7] ImageFormat.TargetFormatListと対照的に、このリストはDeviceのCONVERTER_xxxプロパティに影響を受けます。

4 DirectShow

VRmUsbCam DevKitは2つのDirectShowフィルターをインストールします: VRmUsbCamDSとVRmImageConverterです。

4.1 VRmUsbCamDS

VRmUsbCamDSは、対応しているデバイスについて一度でインスタンス化できるDirectShowビデオキャプチャソースです。これには取り込みのタイムスタンプを含む異なったフォーマットの画像を供給する`FindPin` "1"を使用する)1つの出力ピンがあります。この出力ピンには、デバイス識別とソースフォーマット選択のためのピンプロパティページがあります。
さらに、フィルターのプロパティページでは様々なデバイスパラメータを調整することができます。

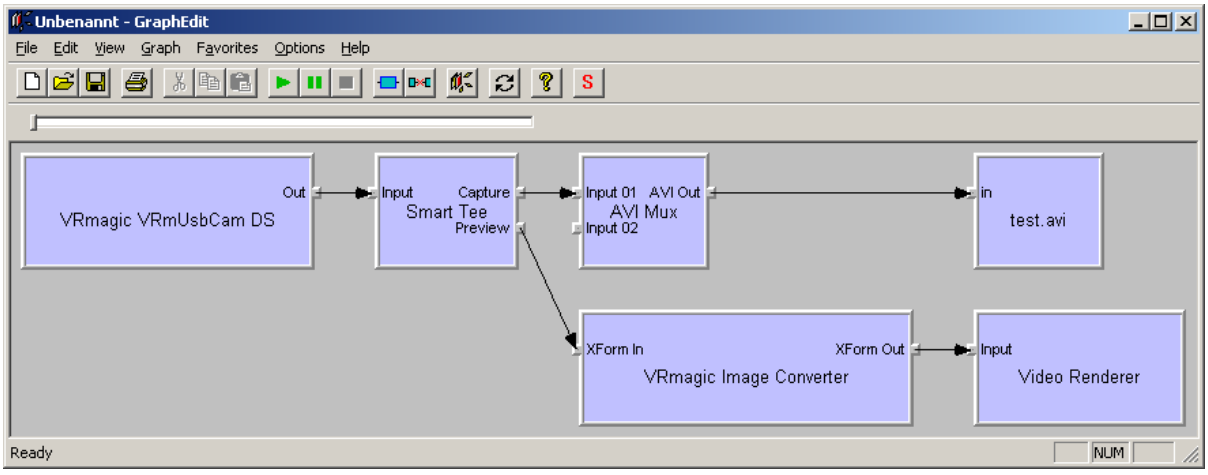
対応している出力フォーマット:
#0 `MEDIASUBTYPE_VRMM` ソースフォーマットと同じ(セクション2.5を参照)
#1 `MEDIASUBTYPE_ARGB32` ターゲットフォーマットARGB32と同じ
#2 `MEDIASUBTYPE_RGB32` ターゲットフォーマットRGB32と同じ
#3 `MEDIASUBTYPE_RGB24` ターゲットフォーマットRGB24と同じ
#4 `MEDIASUBTYPE_RGB565` ターゲットフォーマットRGB565と同じ
#5 `MEDIASUBTYPE_ARGB32` ターゲットフォーマットARGB32と同じ(逆さま*)
#6 `MEDIASUBTYPE_RGB32` ターゲットフォーマットRGB32と同じ(逆さま*)
#7 `MEDIASUBTYPE_RGB24` ターゲットフォーマットRGB24と同じ(逆さま*)
#8 `MEDIASUBTYPE_RGB565` ターゲットフォーマットRGB565と同じ(逆さま*)

**逆さまの画像(高さが負)が好まれるビデオレンダラーもあります。*

白黒カメラの場合は、さらに`MEDIASUBTYPE_RGB8`も利用可能です。これは`CONVERTER_PREFER_GRAY_OUTPUT_B`デバイスプロパティに応じてそれぞれの`ARGB32`の前かそれぞれの`RGB565`フォーマットの後の出力フォーマットのリストに挿入されます。

取り込み中にプレビューを得るには、(DirectXによってインストールされた)Smart Teeフィルターを使用することが考えられます。
取り込みアプリケーションでは、ネイティヴの`MEDIASUBTYPE_VRMM`の取り込みを推奨します。

ビデオプレビューを含むAVI取り込みのサンプルグラフ:



これらのフィルターを使用してC++プログラムを書くには、`vrmdshow.h`をインクルードして、フィルター用に定義されたGUIDとVRMMメディアサブタイプを使用します。

VRmUsbCam COMまたは.NET APIの`AttatchToVRmUsbCamDS()`メソッドを使用してすべてのVRmUsbCamDSインスタンスの`VRmUsbCamDevice`にアクセスすることができます(セクション3.8を参照).

4.2 VRmImageConverter

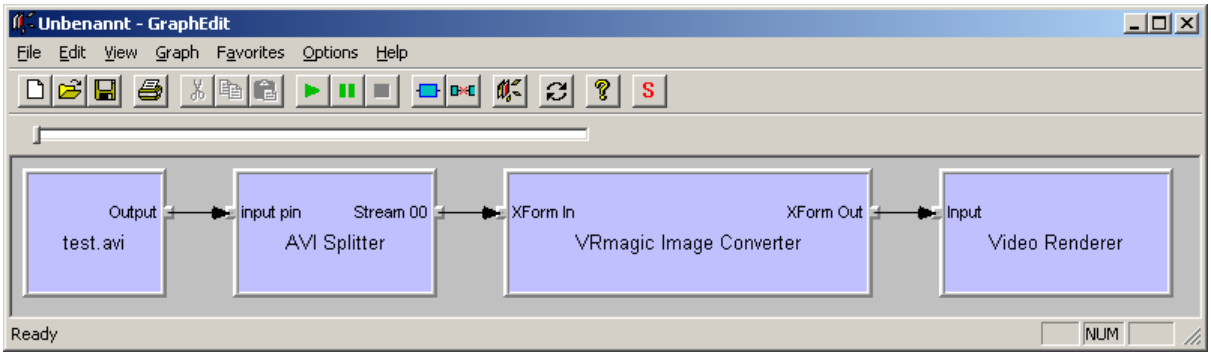
VRmImageConverterはDirectShowビデオ変換フィルターで、`MEDIASUBTYPE_VRMM`入力を様々なRGBフォーマットに変換できます:

対応している出力フォーマット:
#0 `MEDIASUBTYPE_ARGB32`
#1 `MEDIASUBTYPE_RGB32`
#2 `MEDIASUBTYPE_RGB24`
#3 `MEDIASUBTYPE_RGB565`
#4 `MEDIASUBTYPE_ARGB32` (逆さま*)
#5 `MEDIASUBTYPE_RGB32` (逆さま*)
#6 `MEDIASUBTYPE_RGB24` (逆さま*)
#7 `MEDIASUBTYPE_RGB565` (逆さま*)

**逆さまの画像(高さが負)が好まれるビデオレンダラーもあります。*

白黒カメラの場合は、さらに`MEDIASUBTYPE_RGB8`も利用可能です。これは`CONVERTER_PREFER_GRAY_OUTPUT_B`デバイスプロパティに応じてそれぞれの`ARGB32`の前かそれぞれの`RGB565`フォーマットの後の出力フォーマットのリストに挿入されます。

`MEDIASUBTYPE_VRMM`のAVI取り込みを表示するサンプルグラフ:



5 改定履歴

API v2.8.1.x or later	see “VRmUsbCam Devkit Changelog.pdf” for details
API v2.7.2.7	fixed occasional startup problem of VRmAVC-1(+I/+S) with NXP chip (requires firmware version >= v21.99)
API v2.7.2.6	fixed image order and config import of VRmMFC
API v2.7.2.5	added bayer 10bit format to VRmMFC fixed multithreading issue in "UnlockNextImage"
API v2.7.2.4	added support for VRmC-4+ and VRmC-12+ added support for gray 16bit and BGR 48bit target formats improved startup behavior and added support for RLE compressed bayer format to VRmMFC
API v2.7.2.3	improved deserializer of VRmMFC
API v2.7.2.2	added support for VRmAVC-1+I added support for selection of TFF/ BFF formats to VRm(F)AVC-1 (might require a firmware update) fixed sync-freerunning mode of VRmDC-12
API v2.7.2.1	fixed minor documentation issues
API v2.7.2.0	added support for VRmMFC (Smart Multi-Sensor Camera) added support for VRmDC-8, VRmDC-9/BW, VRmDC-12 added support for ethernet receiving added support for VRmAVC-1+S added support for VRmSM-1 added possibility to lock mutiple images of host ring buffer
API v2.6.7.8	fixed non working AttachToVRmUsbCamDS fixed timing changes in freerunning sequential mode fixed trigger timeout for VRmFC-x devices fixed maximal pixel clock for VRmFC-6
API v2.6.7.7	added support for binning modes to VRm(F)C-12/BW added support for overclocking to VRmC-3+ and VRm(F)C-12 added support for external sensor models of VRmFC-12 added auto pixel clock for VRmFC-6
API v2.6.7.6	added support for external sensor models of VRmC-12 added support for TTL trigger/strobe of VRmC-8+/VRmC-9+ Rev 1.1 added channel balance plugin to improve image quality of VRmC-9(+) fixed maximum value of trigger timeout property reduced jitter for VRmFC-x cameras

API v2.6.7.5	added property for roi of auto exposure, with default = center 1/9 of image
API v2.6.7.4	added support for VRmC-9+ fixed incompatibility of VRmUsbCamDS with Adobe Flash Media Encoder
API v2.6.7.3	added support for synchronized free-running grabbing mode (VRmC-12) (might require a firmware update)
API v2.6.7.2	fixed auto-white balance that was accidentally disabled
API v2.6.7.1	added support for VRmC-8+ added subsampling, auto-exposure and optimized 10bit evaluation for VRmFC-x cameras
API v2.6.7.0	built with VS2005 SP1, Visual C++ Runtime 8.0 SP1, .NET Framework 2.0 (no longer supports Framework 1.x) fixed problem of unlit images in free-running and soft/edge triggered modes Device Property Page now runs in its own thread
API v2.6.6.3	added error handling for trigger stalls
API v2.6.6.2	fixed loading/switching between user ROI configs
API v2.6.6.1	added support for VRmFC-6(/BW) updated source format list of VRmFC-4/8/9
API v2.6.6.0	added support for VRmFC-4, VRmFC-8, VRmFC-9, VRmFC-12 added RLE compressed source format for VRmFC-x added device callbacks (C) and events (COM, .NET) fixed user roi of VRmC-3+ and VRmC-12
API v2.6.5.0	added support for VRmFAVC-1 added support for VRmCI enhanced user settings to support 9 different configs
API v2.6.4.0	added VRmUsbCamIsFirmwareCompressionRequired
API v2.6.3.0	added image analysis plugin (chessboard + concentric marker) added defective pixel management (DPM) added property for "images ready in host ringbuffer"
API v2.6.1.0	improved precision of strobe output and trigger timeout (might require a firmware update) added free-running sequential mode added support for VRmC-3+(/BW) added blacklevel property to filter adjustable pixel clock for VRmC-12 (5 to 26.6 MHz) fixed VRmUsbCamReloadUserSettings

API v2.6.0.0	added property based configuration interface added support for VRmC-9/BW and VRmC-12(/BW) added properties for converter (flips, BayerHQ, prefer gray), multi channel filter settings (R/G/B), plugins for auto exposure, auto white balance and auto reset level calibration (for VRmC-3 and VRmC-4) added frame counter information to image fixed negative luminance values
API v2.5.0.0	completed support for VRmC-8pro added shutter config (might require a firmware update) added trigger timeout added user data storage (in eeprom) improved user roi handling added YUYV as target format to image converter
API v2.4.0.0	added optional High-Quality Bayer Filter added basic support for VRmC-8pro
API v2.3.0.0	added VRmUsbCam COM API v2 added soft trigger added COM/.NET interoperability with VRmUsbCamDS Capture Source re-designed .NET API v2 to match COM API v2 added source format to load/save settings
API v2.2.0.0	added support for VRmC-4pro v2 and VRmC-6pro added trigger controls
API v2.1.0.1	added access to timer enhanced target format handling: added support for RGB565, fixed horizontal mirrored, target format handling is now device independent
API v2.0.2.6	added support for VRmC-OEM-1
API v2.0.2.5	initial API v2.x release

6 移行のヒント

C++ API v1.x to C API v2.x

- Replace the access via instances of `VRmUsbCam` by the device handle `VRmUsbCamDevice`
- Replace `GetNextImageBGRA` (or `GetNextImageRAW`) by consecutive `VRmUsbCamLockNextImage` and `VRmUsbCamUnlockNextImage`, you may use `VRmUsbCamConvertImage` between these calls to convert this source image to different target formats
- Camera Configuration is now device type dependent: use `VRmUsbCamGetFeatures` to check for accessible camera settings via `VRmUsbCamSetSettingsX` / `VRmUsbCamGetSettingsX`

C API v2.0.x to C API v2.1.x

- Replace `VRmUsbCamGetDevice` in `VRmUsbCamGetTargetFormatListSize` by the active source format given in `VRmUsbCamGetSourceFormat`

C API v2.6.0.0 to C API v2.6.1.0

- Replace VRmUsbCamLastErrorWasTransferTimeout by VRmUsbCamLastErrorWasTriggerTimeout

.NET API v1.x to .NET API v2.x

- VRmUsbCam.tDeviceId class
  - Rename: `VRmUsbCam.tDeviceId` => `DeviceKey`, rename members:
    `name` => `Product`
    `serial` => `Serial`

- VRmUsbCam class
  - The following methods have to be remapped / renamed:
    - `CheckFramesDropped()`: use `framesDropped` parameter of `LockNextImage()`
    - `GetNextImage()`: replace by `LockNextImage()`, `ConvertImage()`, `UnlockNextImage()`
      Example code (VB.NET), where cam is an instance of VRmUsbCam:
      ```
      Dim framesDropped As Boolean
      Dim convimg As New VRmagic.Image(cam.targetFormats(0))
      Dim rawimg As VRmagic.Image =
      cam.LockNextImage(framesDropped)
      cam.ConvertImage(rawimg, convimg)
      cam.UnlockNextImage(rawimg)
      ```
    - `GetTimeStamp()`: use `TimeStamp` property of `VRmagic.Image` class
    - `ScanForDevices()`: use `UpdateDeviceKeyList()` instead and examine the `DeviceKeyList` property
    - `xxxAdjustments()`: rename to `xxxSettings()`
    - `SkipNextImage()`: replace by `LockNextImage()`,`UnlockNextImage()` pair
  - The following properties have to be remapped/renamed:
    - `DefaultROI`: removed without replacement (but selecting a source format by the `SourceFormat` property restores the ROI to its defaults)

- `DeviceId`: rename to `DeviceKey`
- `Red/Green/BlueGain, ExposureTime, PixelBiasVoltage, PixelClockMhz, ResetLevel, ROI, SensorSize`: use Settings1.xxx property instead and rename:
  `ExposureTime` => `ExposureTimeMs`
  `ImageROI` => `ROI`
- `Gamma`: use `Settings3.Gamma` property instead
- `IlluminationIntensity`: use `Settings2.IlluminationIntensity` property instead
- `IsIlluminationAvailable`: check whether `FeaturesType.SETTINGS2_SUPPORTED` flag is set in `Features` property
- `SupportedPixelFormats`: use `TargetFormatList` property instead. The basic `System.Drawing.Imaging.PixelFormat` has been replaced by the more complex `VRmagic.ImageFormat`, but can be retrieved by its `ToPixelFormat()` method.

.NET API v2.0.x to .NET API v2.1.x

- `VRmUsbCam.TargetFormatList`: use `VRmUsbCam.SourceFormat.TargetFormatList` property instead

.NET API v2.1.x to .NET API v2.2.x

- `VRmUsbCam.Bitmap`: removed, use `VRmUsbCam.ToBitmap()` method instead. This reflects the fact that a pitch conversion may be performed

.NET API v2.2.x to .NET API v2.3.x

- The following namespaces have to be remapped/renamed:
  - `VRmagic`: rename to `VRmUsbCamNET`
- The following classes/types have to be remapped/renamed:
  - `(VRmagic.)VRmUsbCam`: rename to `VRmUsbCamNET.Device`, except regarding the following methods and properties that still remain in `VRmUsbCamNET.VRmUsbCam`: `EnableLogging(), UpdateDeviceKeyList(), RestartTimer(), CurrentTime, DeviceKeyList`
  - all `VRmUsbCam.<xxxx>Type`: rename to `VRmUsbCamNET.<xxxx>` (note the missing "Type" suffix)
  - `VRmUsbCamException`: rename to `VRmUsbCamNET.Exception`
- The following methods have to be remapped/renamed:
  - `New ImageFormat(...)` (constructor): removed. Do not create ImageFormats on your own. Instead, only use the formats in the lists `Device.SourceFormatList` and `ImageFormat.TargetFormatList`
  - `New Image(...)` (constructors): use `VRmUsbCam.NewImage()`, `VRmUsbCam.SetImage()` and `Image.Clone()` methods
  - `Image.ConvertImage`: use `VRmUsbCam.ConvertImage()` method instead
  - `New VRmUsbCam(...)` (constructor): use `VRmUsbCam.OpenDevice()` method instead
- The following properties have to be remapped/renamed:
  - `Image.ImageModifier`: is no longer writable, use `FlipHorizontal()` and `FlipVertical()` instead

.NET API v2.6.6.x to .NET API v2.6.7.x

- **The assembly is now based on .NET Framework 2.0 and thus no longer compatible to applications based on earlier versions of the Framework.**

  In order to use VRmagic devices from within .NET applications based on **Framework 1.x**, you can use the **VRmUsbCam COM API via .NET COM Interop** instead, however functionality will then be limited:

  - The `LockBits()` and `UnlockBits()` methods of the Image object cannot be used, so image data cannot be modified.
    Note: read access to the actual image data is provided by the To…Array() methods.
  - The `PropertyAttribs` property of the Device object is inaccessible because of a bug in COM Interop.

  If you do not need these features, you can still use the VRmUsbCam API from your .NET 1.x application by applying the following adaptions to your existing .NET source code:

  - Replace all references to `VRmUsbCamNET` by `VRmUsbCamCOM`
  - Add a reference to `stdole` assembly
  - Statically instantiate an instance of `VRmUsbCamCOM.VRmUsbCam` somewhere in your code and name it `VRmUsbCam`
  - Replace all array types returned by any member method or property of any VRmUsbCam class by `System.Array`
  - Replace occurrences of `System.Drawing.Size`, `Point` or `Rectangle` by their corresponding classes in `VRmUsbCamCOM`, use `New…()` methods of the static `VRmUsbCam` object for creating them
  - Catch `System.Runtime.InteropServices.COMException` instead of `VRmUsbCamNET.Exception` and investigate the `ErrorCode` member instead of the `Number` member
  - Add the `PROPID_` prefix to all property identifiers used
  - Replace calls to `Image.ToBitmap()` and `Image.ConvertToBitmap()` by calls to `Image.ToPicture()`, `ConvertToPicture()`…,
    if required, use the HDC of your target graphics object as argument to each method, finally, use the `Drawing.Bitmap.FromHbitmap()` static method to create a `Drawing.Bitmap` from the Handle of the Picture object.
    This C# example shows how to extract a `Bitmap` from a VRmUsbCam COM API `Image` object (`vrmImage`):

```
stdole.IPictureDisp tmppic = vrmImage.ToPictureNoDC();
Bitmap tmpbitmap = Bitmap.FromHbitmap(new IntPtr(tmppic.Handle));
System.Runtime.InteropServices.Marshal.ReleaseComObject(tmppic);
…do what you want with tmpbitmap here…
tmpbitmap.Dispose();
```