Ver. 1.0.7

2019/7

USB メモリライセンス認証

ライセンス書込

DLL / LIB 使用方法

API

有限会社リビッグ

〒233-0002 横浜市港南区上大岡西 1-12-2

Tel: 045-843-7122 Fax: 045-843-7142

http://www.ribig.co.jp

内容

評価版について

製品版では、ライセンスを発行するためには必ずライセンス管理 USB キーを接続しなければなりません。ライセンス管理 USB キーによってライセンス発行数が管理されます。

評価版ではライセンス管理 USB キーの接続は不要です。評価版には発行可能なライセンス数に制限はありません。ただし、ユーザ固有データは生成されません。SDK 付属の固有データしか使えません。評価版を利用するすべてのユーザの USB メモリドングルのデータは他のユーザによって読み込み、解読可能です。評価版のセキュリティは確保されません。

評価版は USB メモリドングルを使用するために必要な一連の作業とアプリケーションプログラムでの API 呼び出しを確認するために提供されるとご理解ください。

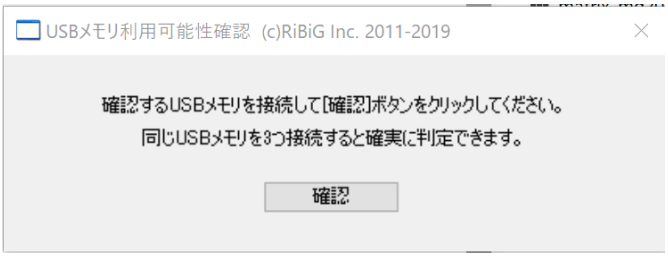
Ⅰ. USB メモリライセンス認証

アプリケーションプログラムから USB ライセンス認証 API を利用する前に必要な作業について説明します。

1. 適切なUSBメモリ選択

すべての USB メモリをドングルとして利用できるわけではありません。固有 ID が割り当てられた USB メモリだけが利用可能です。固有 ID を持たない USB メモリも出回っています。まず最初に検討している USB メモリが固有 ID が割り当てられているタイプかどうか確認しなければなりません。

SDK の[tool]フォルダの check_usbmem.exe は USB メモリキーの ID を確認するツールです。実行すると次のウィンドウが表示されます。



確認する USB メモリキーを１つ接続してから[確認]ボタンをクリックしてください。USB メモリの ID が表示されます。複数キーを接続してから[確認]ボタンをクリックすると、それぞれの ID を比較、すべて異なっていると固有 ID を持っていると判定します。USB キーによっては、同じ機種の２－３つのキーが異なる ID を持っていたとしても、本当にキー固有の ID なのか判定が難しいことがあります（　製造バッチで異なる ID が割り当てられることがあるようです　）。できるだけ多くのキーをつかって判定することで確実な結果を得ることができます。

USB キー各個体で異なる ID を設定するのは、すべての個体で同一 ID を設定するのと比べ、コストがかかるはずです。一般的に超小型にもかかわらず安価な USB メモリは固有 ID を持っていない傾向にあるようです。通常サイズの USB メモリは安価なものでも固有 ID を持っているケースがあります。

2. USBメモリにライセンスを書込む（USBメモリのドングル化）

選択した USB キーにライセンス情報を書き込みドングル化します。詳細は「**Ⅲ. USB メモリのドングル化**」に記載されています。

3. API DLLテンプレートから実DLLを生成する

ライセンスはお客様側で自動生成される秘密鍵やデータ(以降固有データ)をもとに作成されます。ライセンスは固有データを使わなければ解読できません。しかし、API ライブラリは固有データを事前に持つことはできません。そのままではライセンスを解読できません。

API ライブラリのファイルには固有データを埋め込む領域が用意されています。そこに固有データを埋め込むことで、固有データをもとに作成されたライセンスを解読できるようになります。

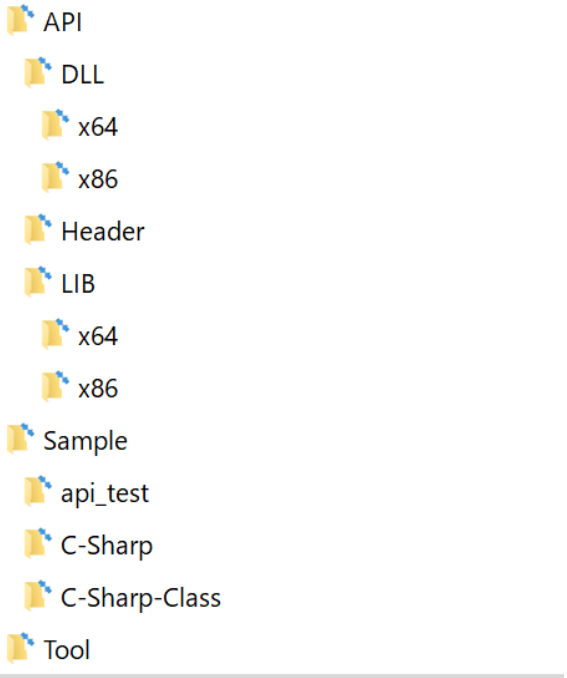
SDK 付属の API DLL/LIB はそのままでは利用できません。これら DLL/LIB に固有データを埋め込むことで初めて正常に動作するようになります。

固有データの埋め込み方法詳細は「**Ⅳ. 実 DLL 生成」**に記載されています。

評価版の制限

評価版では固有データは生成されません。評価版に付属する固有データを基にライセンスは発行されます。

Ⅱ. ディスクのファイルについて



API – DLL – x86/64

matrix32mem.dll	アプリケーション拡張
matrix32mem.lib	Object File Library
matrix32memClass.dll	アプリケーション拡張

API - Header

matrix32mem	C言語ヘッダファイル
mxtypes	C言語ヘッダファイル

API - LIB – x86/64

matrix_md2015.lib	Object File Library
matrix_md2017.lib	Object File Library
matrix_md2019.lib	Object File Library
matrix2015.lib	Object File Library
matrix2017.lib	Object File Library
matrix2019.lib	Object File Library

TOOL

Sample

C/C++ サンプル

C# Interop Service を使った サンプル（）

ネイティブ matrix32mem.dll 呼び出し

C# API クラスを使った　サンプル

.NET クラス　matrix32memClass.dll 呼び出し

Ⅲ. USB メモリのドングル化

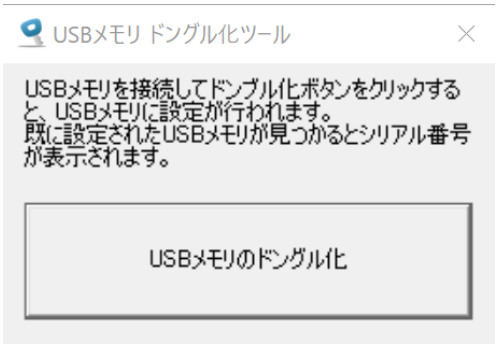
ドングル化プログラム　usbmem_mxapi.exe

USB メモリにライセンスを書き込むプログラムは　**usbmem_mxapi.exe**　です。このプログラムの実行には以下要件を満たす必要があります。

1. ライセンス管理 USB セキュリティキーを接続しなければなりません。ライセンス管理 USB セキュリティキー保有者でなければライセンスは発行できません。ライセンス管理 USB セキュリティキーにより発行可能ライセンス数、発行済ライセンス数が管理されます。
   評価版ではライセンス管理 USB セキュリティキーの接続は不要です。
2. 同じフォルダに matrix.uc ファイルが存在しなければなりません。matrix.uc ファイルはライセンス管理キーに付属するものを使用してください。
   評価版では matrix.uc ファイルは不要です。
3. 同じフォルダに usbmem_mxapi.ini 設定ファイルがあればシリアル番号の初期値を設定できます。SERNR で次回発行するライセンスのシリアル番号を指定できます。

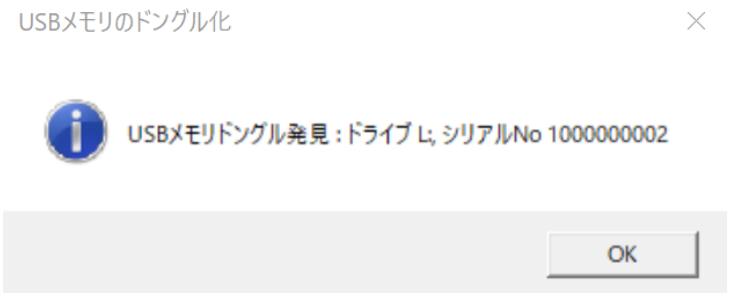
   usbmem_mxapi.ini
   [OPTIONS]
   SERNR=1000000004

usbmem_mxapi.exe　を起動すると以下ウィンドウが表示します。

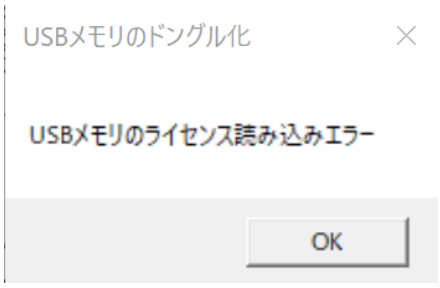


ドングル化する USB メモリを接続後、[USB メモリのドングル化]をクリックします。

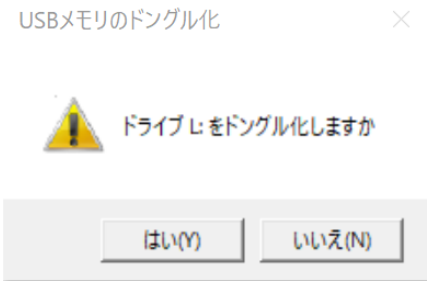
■USB メモリにライセンスが既に書き込まれていて、固有データ生成済みで同じフォルダにあればドライブ名とシリアル番号を表示します。



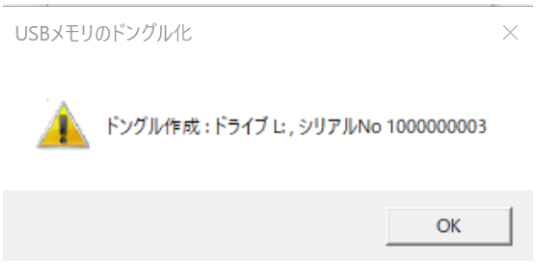
ライセンス発行時の固有データが同じフォルダになければライセンスを読み込むことはできません。エラーになります。



■USB メモリにライセンスが書き込まれておらず **usbmem_mxapi.exe** と同じフォルダに固有データが存在しなければ、USB メモリのドライブを表示してドングル化するかどうか確認を求めます。



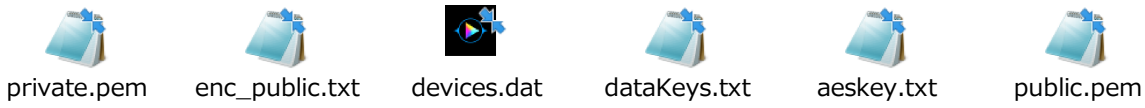
ドングル化に成功するとドライブ名とシリアル番号を表示します。シリアル番号は USB メモリの固有 ID ではありません。ドングル化プログラムが割り当てたライセンスのシリアル番号です。



発行するライセンスのシリアル番号は、**usbmem_mxapi.exe** と同じフォルダに usbmem_mxapi.ini に記録されます。初期値は自由書き換えて構いません。

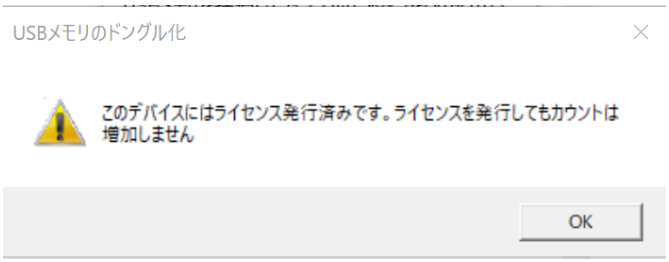
usbmem_mxapi.ini
[OPTIONS]
SERNR=1000000004

ライセンス発行されると、同じフォルダに以下ファイル（固有データ）が作成されます。評**価版では固有データは作成されません。SDK 付属の固有データが使われます**



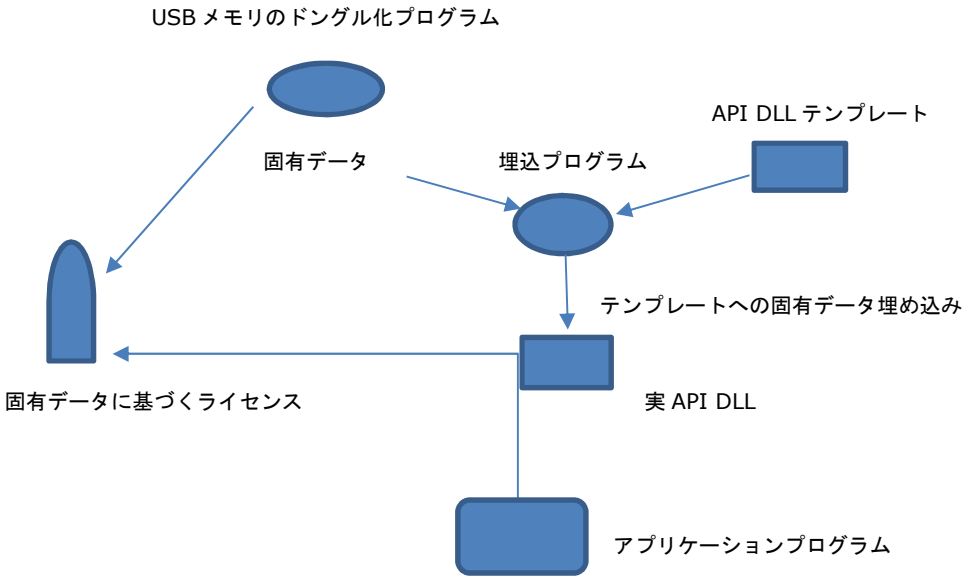
これら固有データファイルは絶対に変更や削除はしないでください。

ライセンスを発行した USB メモリの固有 ID は “device.dat” に保存されます。USB メモリー上のライセンスファイルを誤って削除してしまっても、device.dat に固有 ID の記録があれば、ライセンス数を増加させずにライセンスを再書き込みできます。ライセンスを書き込もうとする USB メモリの固有 ID が device.dat に見つかれば、以下メッセージが表示されます。



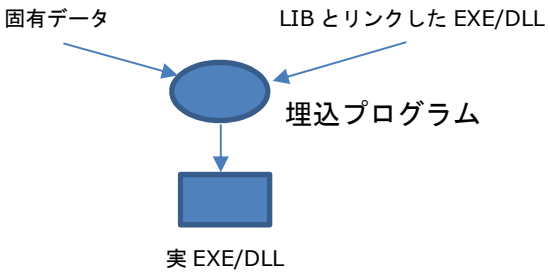
Ⅳ. 実 DLL 生成

お客様固有のデータは、データが存在しないフォルダでライセンス書き込みプログラムを実行すると自動作成されます。ライセンスや USB メモリ上のデータは固有データを使って暗号化されます。API ライブラリは同じ固有データを使わなければライセンス/データを解読できません。API ライブラリ（DLL/LIB）には固有データを“埋め込む“場所が確保されていて、そこに固有データを埋め込む作業が必ず必要です。



アプリケーションプログラムは SDK 付属 DLL ではなく、固有データが埋め込まれた実 API DLL 経由で USB メモリドングルを操作しなければなりません。

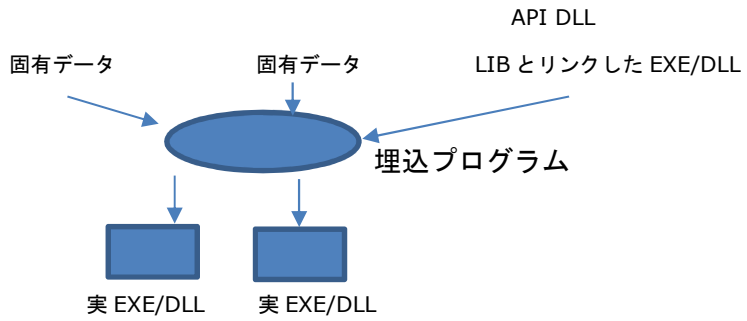
スタティックライブラリ（LIB）には直接固有データを埋め込むことはできません。LIB をリンクしたアプリケーションプログラム（EXE/DLL）に固有データを埋め込みます。



固有データは、データが存在しないフォルダでライセンス書き込みプログラムを実行すると自動作成されます。

1. 固有データ（複数ファイル）を失うと、そのデータに基づくライセンスの新規発行は不可能です。既存ライセンスや実 DLL/EXE はそのまま使えますが、その実 DLL/EXE が解読できる新規ライセンスを発行することはできません。

2. 固有データが存在しないフォルダでライセンス書き込みプログラムを実行すると新規に固有データが自動作成されます。異なる固有データに基づくライセンスで同一プログラムを運用できます。



同一 DLL/Exe から異なるライセンスを解読する実 EXE/DLL 作成可能

3. 公開鍵は埋め込まれますが秘密鍵は埋め込まれません。ユーザが秘密鍵を管理します。

このような仕組みのため、SDK 付属 API ライブラリ（DLL/LIB）は、そのままでは USB メモリに書き込まれたライセンスにアクセスできません。API ライブラリを正常に動作させるには、固有データを埋め込む作業が必要です。

DLL

API ライブラリ DLL には直接固有データを埋め込みます。

LIB

API ライブラリ LIB に対して直接埋め込むことはできません。LIB をリンクしたプログラム本体（EXE/DLL）に固有データを埋め込みます。

固有データ埋め込みプログラム　embedinfo.exe

embedinfo.exe はコンソールプログラムです。コマンドプロンプトで実行してください。

a. embedinfo.exe を起動するにはライセンス管理キーを接続してください。
b. 同じフォルダに固有データファイル / matrix.uc がなければなりません。

embedinfo.exe起動方法
固有データを埋め込む DLL/EXE を引数に指定して起動します。

例：
>embedinfo　..¥matrix32mem.dll

固有データが埋め込まれたファイルは、.embed 拡張子が追加されて埋め込む元ファイルと同じフォルダに作成されます。

例：
..¥matrix32mem.dll　-→ ..¥matrix32mem.dll.embed

matrix32mem.dll.embed が実 DLL ファイルです。 .embed 拡張子を取り除いて使用してください。

SDK付属API　DLL（32ビット版/64ビット版）への固有データ埋め込み
以下２つ付属 API DLL に固有データを埋め込んでください。

- matrix32mem.dll
- matrix32memClass.dll

API LIBをリンクした実行ファイル（EXE/DLL）への固有データ埋め込み
LIB ファイルをリンクしたプログラムはそのままではライセンスを解読できません。固有データを埋め込んでください。

Ⅴ. ライセンス管理 USB キー

ライセンス発行プログラム usbmem_mxapi.exe はライセンス管理 USB キーを使って発行可能ライセンス最大数と発行済みライセンス数を管理します。発行可能ライセンス最大数は購入したライセンス数に設定されます。

発行済みライセンス数が発行可能最大数に達すると、ライセンスを新規に発行することはできません。

追加ライセンスを購入後、ライセンス管理 USB キーを弊社まで返送ください。ライセンス管理 USB キーの発行済みライセンス数を 0、購入した追加ライセンス数を発行可能ライセンス最大数に再設定します。また、usbmem_mxapi.exe　も更新されます。発行可能ライセンス最大数の管理は、ライセンス管理 USB キーだけでなく、プログラム側でも処理しているためです。

usbmem_mxapi.exe は更新されても、既存固有データはそのまま利用可能です。

評価版ではライセンス管理 USB キーの接続は不要です。評価版には発行可能なライセンス数に制限はありません。ただし、ユーザ固有データは生成されません。SDK 付属の固有データしか使えません。評価版を利用するすべてのユーザの USB メモリドングルのデータは他のユーザによって読み込み、解読可能です。評価版のセキュリティは確保されません。

Ⅵ. USB メモリドングルの操作

SDK 付属 API　DLL/LIB には以下 API が含まれます。

Init_MatrixAPI	USB メモリドングル API を初期化する。
Release_MatrixAPI	USB メモリドングル API を開放する。
GetVersionAPI	USB メモリドングル API のバージョン番号を返す。
Dongle_Count	USB ポートに接続されている USB メモリドングル数を返す。
Dongle_MemSize	メモリサイズを返す。（バイト）
Dongle_Version	USB メモリドングルのバージョン番号を返す。
Dongle_ReadData	1番目から n 番目までのデータフィールドからデータを読み込む。
Dongle_ReadDataEx	m 番目から n 番目までのデータフィールドからデータを読み込む。
Dongle_WriteData	1 番目からn番目までのデータフィールドにデータを書き込む。
Dongle_WriteDataEx	M 番目から n 番目までのデータフィールドにデータを書き込む。
Dongle_ReadSerNr	シリアル番号を読み込む。
Dongle_WriteKey	128 ビットの TEA 秘密鍵を書き込む。
Dongle_GetKeyFlag	128 ビットの TEA 秘密鍵が USB メモリドングルに書き込まれているか確認する。
Dongle_EncryptData	8 バイトのデータブロックを USB メモリドングルに暗号化させる。
Dongle_DecryptData	8 バイトのデータブロックを USB メモリドングルに復号化させる。

操作手順

もっとも単純な使い方は、正当なライセンスが書き込まれた USB メモリドングルが接続しているかどうかを確認します。

1. 他の API を呼び出す前に Init_MatrixAPI　を呼び出す
2. Dongle_Count で接続している USB メモリドングル数を確認
3. Release_MatrixAPI で API 操作を終了

```
Init_MatrixAPI();
short count = Dongle_Count(85);
Release_MatrixAPI();

if( count <= 0 )
{
        // 未接続
        return;
}
```

API DLL ファイルを認証するには USB メモリドングルに秘密鍵を事前に設定します。

1. 他の API を呼び出す前に Init_MatrixAPI　を呼び出す
2. Dongle_Count で接続している USB メモリドングル数を確認
3. ランダム数を 2 つ生成して　Dongle_EncryptData 呼び出して暗号化
4. 暗号化データをアプリケーションプログラム側で復号化、暗号化前のデータと一致することを確認
5. Release_MatrixAPI で API 操作を終了

秘密鍵書き込み

```
Init_MatrixAPI();
short count = Dongle_Count(85);
if( count <= 0 )
{
        // 未接続
        return;
}

long key[4];
```

```
key[0] = 1111111;
key[1] = 2222222;
key[2] = 3333333;
key[3] = 4444444;
Dongle_WriteKey( UserCode, key, 1, 85 );

Release_MatrixAPI();
```

API DLL 認証

```
#include “mxtea.h”
Init_MatrixAPI();
short count = Dongle_Count(85);
if( count <= 0 )
{
        // 未接続
        return;
}

long data[2], data1[2]
data1[0] = data[0] = GetTickCount64();
data1[1] =   data[1] = time(NULL);
short ret = Dongle_EncryptData(UserCode, data, 1, 85 );
if( ret < 0 ) return;

long key[4];
key[0] = 1111111;
key[1] = 2222222;
key[2] = 3333333;
key[3] = 4444444;
MxApp_DecryptData( key, data );

if( data[0] == data1[0] && data[1] == data1[1] )
        // ok

Release_MatrixAPI();
```

メモリフィールドにデータ（暗号化したもの）を書き込んで確認する方法なども考えられます。
シリアル番号を利用すると USB キードングル事に異なるデータの保管可能になります。

Ⅶ　ネイティブ　API　説明

ネイティブプログラムであるダイナミックライブラリ　Matrix32mem.DLL / スタリックライブラリに含まれる API を説明します。

Init_MatrixAPI	
説明	API を初期化します。 他 API を呼び出す前に、必ず呼び出してください。
呼出し	Short Init_MatrixAPI()
引数	なし
戻り値	0　　成功

Release_MatrixAPI	
説明	API を開放します。 API を使い終わったら、必ず呼び出してください。
呼出し	Short Release_MatrixAPI()
引数	なし
戻り値	なし

GetVersionAPI	
説明	API のバージョン番号を返します。
呼出し	Long GetVersionAPI()
引数	なし
戻り値	バージョン番号の 上位 2 バイト ＝　メジャーバージョン 下位 2 バイト ＝ マイナーバージョン

Dongle_Count		
説明	引数で指定されたポートに装着された USB メモリドングル数を返します。返されるのは Init_MatrixAPI 呼出し時点の接続ドングル数です。Init_MatrixAPI 呼び出し以降にドングルを抜き差しすると正しいドングル数は取得できません。	
呼出し	short Dongle_Count(short PortNr);	
引数	PortNr	'U'(Acii 85)
戻り値	指定ポートに装着されたドングル数 0　ドングルが接続されていない	

重要: ドングル操作 API は、Init_MatrixAPI 呼び出し時点の接続ドングルを操作対象とします。Init_MatrixAPI 呼び出し時点でドングルが接続していたら、仮に、その後ドングルを抜き取っても新しい状態は反映されません。Release_MatrixAPI を呼び出し、再度 Init_MatrixAPI を呼び出すとその時点の状態が操作対象となります。ドングル抜き差し検出 API で抜き差しは追跡できます。

Dongle_MemSize		
説明	バイト単位でメモリサイズを返します。	
呼出し	short Dongle_MemSize(Short DngNr, short PortNr)	
引数	DngNr	USB メモリドングルの番号[1]
	PortNr	'U'(Ascii 85)
戻り値	バイト単位のメモリサイズ -1　DngNr にドングルが見つからない -26　ドングルが見つからない	

注)データフィールドサイズは 4 バイト固定のため、データフィールド数は、この関数の戻り値から算出できます。

[1] 1つのポートに複数のドングルが装着できるため、ポート番号に加えて、この引数でドングルの番号を指定しなければなりません。

Dongle_Version		
説明	ドングルソフトウェアのバージョン番号を返します。	
呼出し	long Dongle_Version(short DngNr, short PortNr)	
引数	DngNr	ドングルの番号[2]
	PortNr	'U'(Ascii 85)
戻り値	ドングルソフトウェアのバージョン番号[3]　または、下記参照。 -1　DngNr にドングルが見つからない -26　ドングルが見つからない	

Dongle_ReadData		
説明	ドングルの内臓メモリの***第 1***データフィールドから指定フィールド数分のデータを読み込みます。[4]	
呼出し	short Dongle_ReadData(long UserCode, long *Data, short Count, short DngNr, short Port Nr)	
引数	UserCode	割り当てられたユーザコード[5]
	*Data	データフィールドから読み込んだデータをセットする配列[6]
	Count	読み込むデータフィールド数
	DngNr	ドングルの番号[7]
	Port Nr	'U'(Ascii 85)
戻り値	読み込まれたデータフィールド数 -1　DngNr にドングルが見つからない -2　ユーザコードエラー -26　ドングルが見つからない	

[2] 1つのポートに複数のドングルが装着できるため、ポート番号に加えて、この引数でドングルの番号を指定する必要があります。

[3]上位 2 バイトは、メジャーバージョンを表し、下位 2 バイトはマイナーバージョンを表します。

[4] 例えばデータフィールド数が 3 ならば、第1から第3データフィールドのデータを読み込みます。

[5] ドングル内のユーザコードと一致しなければなりません。

[6] short Count で指定する数以上のサイズがなければなりません。

[7] 1つのポートに複数のドングルが装着できるため、ポート番号に加えて、この引数でドングルの番号を指定する必要があります。

Dongle_ReadDataEx		
説明	ドングルの内臓メモリの***任意の***データフィールドから指定フィールド数分のデータを読み込みます。[8]	
呼出し	short Dongle_ReadDataEx(long UserCode, long *Data, short Fpos, short Count, short DngNr, short Port Nr)	
引数	UserCode	割り当てられたユーザコー[9]
	*Data	データフィールドから読み込んだデータをセットする配列[10]
	Fpos	読み込みを開始するデータフィールド番号
	Count	読み込むデータフィールド数
	DngNr	ドングルの番号[11]
	PortNr	ドングルが装着されているポート番号。LPT では 1～3。 USB は'U'(Ascii 85)
戻り値	読み込まれたデータフィールド数 -1 DngNr にドングルが見つからない -2 ユーザコードエラー -26 ドングルが見つからない	

注)UserCode が一致していないとデータを読み込めません。

[8] 例えば 5 番目のデータフィールドから、3 つのデータフィールド数を読み込むならば、第 5 から第 7 データフィールドのデータを取得できます。
[9] ドングル内のユーザコードと一致しなければなりません。
[10] short Count で指定する数以上のサイズがなければなりません。
[11] 1つのポートに複数のドングルが装着できるため、ポート番号に加えて、この引数でドングルの番号を指定する必要があります。7

Dongle_WriteData		
説明	ドングル内臓メモリの***第 1***データフィールドから指定フィールド数分のフィールドにデータを書き込みます。[12]	
呼出し	short Dongle_WriteData(long UserCode, long *Data, short Count, short DngNr, short Port Nr)	
引数	UserCode	割り当てられたユーザコード[13]
	*Data	データフィールドに書き込むデータをセットした配列[14]
	Count	書き込むデータフィールド数
	DngNr	ドングルの番号[15]
	PortNr	'U'(Ascii 85)
戻り値	書き込まれたデータフィールド数 -1 DngNr にドングルが見つからない -2 ユーザコードエラー -26 ドングルが見つからない	

[12] 例えばデータフィールド数が 3 ならば、第1から第3データフィールドにデータを書き込みます。
[13] ドングル内のユーザコードと一致しなければなりません。
[14] short Count で指定する数以上のサイズがなければなりません。
[15] 1つのポートに複数のドングルが装着できるため、ポート番号に加えて、この引数でドングルの番号を指定する必要があります。

Dongle_WriteDataEx		
説明	ドングル内臓メモリの任意のデータフィールドから指定フィールド数分のフィールドにデータを書き込みます。[16]	
呼出し	short Dongle_WriteDataEx(long UserCode, long *Data, short Fpos, short Count, short DngNr, short Port Nr)	
引数	UserCode	割り当てられたユーザコード[17]
	*Data	データフィールドに書き込むデータをセットした配列[18]
	Fpos	書き込みを開始するデータフィールド番号
	Count	書き込むデータフィールド数
	DngNr	ドングルの番号[19]
	PortNr	'U'(Ascii 85)
戻り値	書き込まれたデータフィールド数 -1 DngNr にドングルが見つからない -2 ユーザコードエラー -26 ドングルが見つからない	

[16] 例えば、書き込みを開始するデータフィールドが3、データフィールド数が 3 ならば、第3から第5データフィールドにデータを書き込みます。
[17] ドングル内のユーザコードと一致しなければなりません。
[18] short Count で指定する数以上のサイズがなければなりません。
[19] 1つのポートに複数のドングルが装着できるため、ポート番号に加えて、この引数でドングルの番号を指定する必要があります。

Dongle_ReadSerNr		
説明	ドングルのシリアル番号を読み込みます。	
呼出し	long Dongle_ReadSerNr (long UserCode, short DngNr, short PortNr)	
引数	UserCode	割り当てられたユーザコード[20]
	DngNr	ドングルの番号[21]
	PortNr	'U' (Ascii 85)
戻り値	指定ドングルのシリアル番号 -1 DngNr にドングルが見つからない -2 ユーザコードエラー -26 ドングルが見つからない	

[20] ドングル内のユーザコードと一致しなければなりません。
[21] 1つのポートに複数のドングルが装着できるため、ポート番号に加えて、この引数でドングルの番号を指定する必要があります。

Dongle_WriteKey		
説明	128 ビットの TEA 秘密鍵を書き込みます。	
呼出し	short Dongle_WriteKey(long UserCode, unsigned long* KeyData, short DngNr, short PortNr)	
引数	UserCode	割り当てられたユーザコード[22]
	* KeyData	書き込むデータをセットしたバッファへのポインタ
	DngNr	ドングルの番号[23]
	PortNr	'U'(Ascii 85)
戻り値	>0 で書き込み成功。 0 鍵を保存できなかった -1 DngNr にドングルが見つからない -2 ユーザコードエラー -26 ドングルが見つからない	

[22] ドングル内のユーザコードと一致しなければなりません。
[23] 1つのポートに複数のドングルが装着できるため、ポート番号に加えて、この引数でドングルの番号を指定する必要があります。

Dongle_GetKeyFlag		
説明	128 ビットの TEA 秘密鍵がドングルに書き込まれているか確認します	
呼出し	short Dongle_GetKeyFlag(long UserCode, short DngNr, short PortNr)	
引数	UserCode	割り当てられたユーザコード[24]
	DngNr	ドングルの番号[25]
	PortNr	'U'(Ascii 85)
戻り値	キーが存在するならば１、存在しなければ 0、または下記参照。 -1 DngNr にドングルが見つからない -2 ユーザコードエラー -26 ドングルが見つからない	

注）128 ビット TEA キーはドングルから読み込むことはできませんが、この関数で存在するかどうかを確認することはできます。すべてのバイトが 0 の TEA キーは、0に設定された有効なキーです。

[24] ドングル内のユーザコードと一致しなければなりません。
[25] 1つのポートに複数のドングルが装着できるため、ポート番号に加えて、この引数でドングルの番号を指定する必要があります。

Dongle_EncryptData		
説明	8 バイトのデータブロックをドングルに暗号化させます。USB メモリはハードウェアで暗号/復号化機能を有していません。API で暗号化します	
呼出し	short Dongle_EncryptData(long UserCode, unsigned long* DataBlock, short DngNr, short PortNr)	
引数	UserCode	割り当てられたユーザコード[26]
	* DataBlock	暗号化する 8 バイトのデータブロックへののポインタ
	DngNr	ドングルの番号[27]
	PortNr	'U'(Ascii 85)
戻り値	>0 で成功、または下記参照。 −1 DngNr にドングルが見つからない −2 ユーザコードエラー −26 ドングルが見つからない	

[26] ドングル内のユーザコードと一致しなければなりません。

[27] 1つのポートに複数のドングルが装着できるため、ポート番号に加えて、この引数でドングルの番号を指定する必要があります。

Dongle_DecryptData		
説明	8 バイトのデータブロックを復号化します。USB メモリはハードウェアで暗号/復号化機能を有していません。API で復号化します	
呼出し	short Dongle_DecryptData(long UserCode, unsigned long* DataBlock, short DngNr, short PortNr)	
引数	UserCode	割り当てられたユーザコード[28]
	*DataBlock	復号化する 8 バイトのデータブロックへののポインタ
	DngNr	ドングルの番号[29]
	PortNr	'U'(Ascii 85)
戻り値	>0 で成功、または下記参照。 -1 DngNr にドングルが見つからない -2 ユーザコードエラー -26 ドングルが見つからない	

[28] ドングル内のユーザコードと一致しなければなりません。
[29] 1つのポートに複数のドングルが装着できるため、ポート番号に加えて、この引数でドングルの番号を指定する必要があります。

ドングル抜き差し検出API

ドングル操作は別にドングルの抜き差しを検出する API を提供します。

```
_mxINT16 Init_UsbMemDetect(void (*OnConnect)(void*),void (*OnDisconnect)(void*),void* pArg)
_mxINT16 Release_UsbMemDetect();
```

Init_UsbMemDetect メソッドは、第一引数にドングルが接続されたときに呼び出される（コールバックされる）関数、第二引数に抜き取られたときに呼び出される関数、第三引数にコールバックされる関数に引数として渡されるデータを指定して呼び出します。Init によりドングル抜き差し検出のためのスレッドが開始されます。

OnConnect, OnDisconnect コールバック関数でドングルが抜き差しされたときの処理を行います。

Release_UsbMemDetect は Init_UsbMemDetect が開始したスレッドを終了して、抜き差し検出を停止します。

アプリケーションプログラムは Init_UsbMemDetect を呼び出しても、呼び出し時点でドングルが接続されているかどうかは分かりません。呼び出し以降の抜き差しを検出するのみです。呼び出し時点にドングルが接続しているかどうかは、ドングル操作 API の Dongle_Count で確認してください。

Ⅷ　マネージ API

.NET プログラムは マネージドプログラムである Matrix32memClass.DLL を参照して API クラスを利用できます。

ネームスペース　: matrix32mem

Matrix32mem ネームスペースには２つのクラスが含まれます。

クラス　　: API,　Detect

API クラスは USB メモリドングルを操作するメソッドを公開します。Detect クラスはドングルの抜き差しを検出するために利用します。

APIクラス

ネイティブ API に対応するメソッドを公開します。ポート番号（‘U’又は 85 ）の指定は不要です。

```
short Init();
short Release();
short Count();
long  VersionAPI();
short MemSize(short dngNr);
long  Model(short dngNr);
long  Version(short dngNr);
long  ReadSerNr(int UserCode, int dngNr);
short ReadData(int UserCode, array<int>^ data, short count, short dngNr);
short ReadDataEx(int UserCode, array<int>^ data, short pos, short count, short dngNr);
short WriteData(int UserCode, array<int>^ data, short count, short dngNr);
short WriteDataEx(int UserCode, array<int>^ data, short pos, short count, short dngNr);
short WriteKey(int  UserCode, array<unsigned int>^ key, short dngNr);
short GetKeyFlag(int UserCode, short dngNr);
short EncryptData(int UserCode, array<unsigned int>^ dataBlock, short dngNr);
short DecryptData(int UserCode, array<unsigned int>^ dataBlock, short dngNr);
```

重要： API クラスは、Init 呼び出し時点の接続ドングルを操作対象とします。Init 呼び出し時点でドングルが接続していたら、仮に、その後ドングルを抜き取っても新しい状態は反映されません。Release を呼び出し、再度 Init を呼び出すとその時点の状態が操作対象となります。ドングル抜き差し検出 API で抜き差しは追跡できます。

Detectクラス

このクラスを利用することで、ドングルの抜き差しイベントをアプリケーションプログラムで検出できるようになります。

```
delegate void UsbMemDetectDelegate(IntPtr arg);
short Init(UsbMemDetectDelegate^ onConnect, UsbMemDetectDelegate^ onDisConnect);
short Release();
```

Init メソッドは、第一引数にドングルが接続されたときに呼び出される（コールバックされる）関数を、第二引数に抜き取られたときに呼び出される関数を指定して呼び出します。Init によりドングル抜き差し検出のためのスレッドが開始されます。

OnConnect, OnDisconnect コールバック関数でドングルが抜き差しされたときの処理を行います。

Release メソッドは Init が開始したスレッドを終了します。

このクラスはフォームクラスで使うようにしてください。

アプリケーションプログラムは Init を呼び出しても、呼び出し時点でドングルが接続されているかどうかは分かりません。呼び出し以降の抜き差しを検出するのみです。呼び出し時点にドングルが接続しているかどうかは、API クラスの Count で確認してください。