1.0 版

EL Matrix 互換 API ユーザマニュアル

有限会社リビッグ

http://www.ribig.co.jp/el

*マニュアルの内容は予告無く変更されることがあります。

*付属ソフトウェアーはマニュアルの内容と一致しないことがあります。

ユーザコード

評価版のユーザコードは 1234 です。

著作権表示

目次

はじめに： ソフトウェア保護へのガイドライン *5*

***1. EL** のセキュリティ* *7*

1.1 EL の暗号技術 7

2. EL の機種 *9*

***3. EL API** やドライバのインストールや配布方法* *10*

3.1 EL API とドライバのアーキテクチャ 10

3.2 Linux & Mac OSX 10

3.3 プロテクトプログラムの配布 12

4. 動作確認 *13*

5. Matrix 互換 API 用のユーティリティ *14*

6. EL ソフトウェア開発キット *15*

7. EL API リファレンス *16*

7.1 静的ライブラリ・動的ライブラリ 16

7.2 Matrix 互換基本 API サマリ 17

7.3 EL 用 Matrix 互換拡張 API サマリ 19

7.4 API エラーコード 21

7.5 戻り値、引数の型 26

7.6 EL 用 Matrix 互換 API 27

Init_MatrixAPI 27

Release_MatrixAPI 27

GetVersionAPI 28

GetVersionDRV_USB 29

SetW95Access 30

GetPortAddr 30

PausePrinterActivity (Windows OS のみ) 31

ResumePrinterActivity (Windows OS のみ) 31

Dongle_Find 32

Dongle_FindEx 33

Dongle_Count 34

Dongle_MemSize 35

Dongle_Model 36

Dongle_Version 37

Dongle_ReadData 38

Dongle_ReadDataEx 39
Dongle_WriteData 40
Dongle_WriteDataEx 41
Dongle_ReadSerNr 42
Dongle_WriteKey 43
Dongle_GetKeyFlag 44
Dongle_EncryptData 45
Dongle_DecryptData 46
Dongle_SetDriverFlag 47
Dongle_GetDriverFlag 48
SetConfig_MatrixNet 49
GetConfig_MatrixNet 50
Login_MatrixNet 51
LogOut_MatrixNet 52
Dongle_SetLED 53
Dongle_GetRand 54
Dongle_GetTime 55
Dongle_ReadGUSN 56
Dongle_MemSize2 57
Dongle_ReadData2 58
Dongle_WriteData2 59
Dongle_LockData 60
Dongle_LockData2 61
Dongle_SetTimer 62
Dongle_StartTimer 63
Dongle_GetTimer 64
Dongle_StopTimer 65
Dongle_CreateRSAKeyPair 66
Dongle_LockRSAKeyPair 67
Dongle_GetRSAPubKey 68
Dongle_EncryptDataRSA 69
Dongle_DecryptDataRSA 70
Dongle_WriteKeyTDES 71
Dongle_EncryptDataTDES 72
Dongle_DecryptDataTDES 73
Dongle_WriteKeyHMACSHA1 74
Dongle_HMACSHA1 75

はじめに：　ソフトウェア保護へのガイドライン

どんなプロテクションでも破られる。

破られないソフトウェア保護はあるのか？　答えは NO です。どんなプロテクションでも見破られ、除去される可能性はあります。絶対に安全な方法というものはありません。問題はいかにプロテクトを破るのを難しくするかです。プログラムのプロテクトをする際、この点は必ず頭にとめて置いてください。

用意されたツール類を理解し、うまく利用する。

不正コピーからプログラムを有効的にプロテクトするには、プロテクト手段の限界を理解していなければなりません。ELを使ったようなソフトウェア保護では、ハードウェア的にはセキュリティはかなり徹底していて、そこに手を入れられる隙は少ないはずです。この方式の弱い点は API をプログラムに組み込むところにあります。ハッカーはここを突いてきます。プログラムコード内からプロテクトAPIを探し出して、ELを接続していなくても、プログラムが実行するように、コードを操作してきます。

EL-Crypt を使用したり、想像力や独創性を生かしてプロテクションコードをプログラムに埋め込むと、プロテクションを破る作業を難しくできます。同じツールを使っていても、戦略的に配置された方が効果的であることは言うまでもありません。

プロテクトの解読

プロテクトの解読にはデバッグプログラムが使われます。DEBUG.EXEなど簡単なものが使われることもありますが、命令毎の実行を可能とするような、もっと洗練されたツールが使われると考えてください。アセンブラー命令を分析して、ELへの呼び出しを見つけ出し、プログラムの実行がELの接続に依存しないように変更してくるはずです。ELへの呼び出しを１回以上、プログラムの異なる場所から行うと、解読作業を難しくできます。

プログラム開始時に呼び出す。

プログラム開始時に、必ず期待するELが接続されているかどうか、確認するための呼び出しを、必ず行ってください。

頻繁に呼び出す。

プログラム開始時にELを呼び出すだけでは十分ではありません。その部分のコードを見つけられて、いじられたら破られてしまいます。また、プログラム起動後にELを抜き取ってもプログラムは動きつづけます。複数のPCにプログラムをインストールして、１つのELで順に複数のプログラムを起動されてしまうかもしれません。これは、プログラムの実行中も、頻繁にELへの呼び出しを行うことで回避できます。

呼び出しの分散化

プログラムコードの全体に渡ってELの呼び出されるように工夫をしてください。

a. プログラムの構造上、もっとも下位部分に呼び出しを含める。
b. 呼び出しを１つの関数にまとめて、その関数経由で呼び出すようなことはしない。
c. ソースコードの全体に呼び出しがあるようにする。

ELメモリの利用

ELのメモリの利用を考えてください。例えば、顧客番号、シリアル番号、プログラム変数値などをメモリに保存しておき、プログラムの実行中にチェックします。プログラムの実行に必要なデータをELメモリに保存してあれば、プログラムをどんなにいじったところで、ELが接続されていなければ、プログラムは稼動不能です（プログラムを解読するために、ELが必要になってきます)。使っていないデータフィールド（メモリ）に、任意の値を書き込んでおいて、そのデータフィールドの値をプログラム実行中に確認するといった手法もあります。

暗号機能の利用

ELの暗号・復号化機能を利用して、プログラムで使用するデータを暗号化・復号化することもできます。MxPRGやAPIを使ってデータは前もって暗号化し、プログラムの変数に暗号化した値を設定します。プログラム実行中に、ELに復号させて利用できるデータに戻します。これでELが接続していなければプログラムが正常に稼動しないようにできます。

プログラミング

呼び出しを隠すために、ソースコードを「汚く」する手法は時間の無駄です。多くのコンパイラは最適化の過程において、「きれい」なアッセンブラーコードを生成してしまいます。
プログラムに隠しスイッチを設けて、プロテクトコードの有効・無効を選択するような仕組みは避けてください。スイッチ変数を操作されたら、プロテクトコードがすべて無効となってしまう可能性があります。
配布するプログラムにはデバッグ情報が含まれないようにしてください。

攻撃への対処

プログラムが攻撃を受けたり、変更されたことを検出したら、プログラムに作動に制限を加えたり不能にする方法はいくつかあります。対策用コードは、攻撃を検出するコードとは切り離して別のものなるようにしてください。

- **攻撃検出と作動不可に時間差をつける。**
  攻撃を検出したら、すぐに作動しなくなるようにするのではなく、数秒から数日間の時間を遅らせて攻撃に対応するようにします。

- **因果関係を隠す。**
  プログラムのある部分が変更されたら、他の複数の部分が連鎖的に変更されるようにして、間接的に変更されたうち、１つが攻撃に対応するコードを有効にするようにします。

- **誤った結果を返す。**

攻撃を見つけてもすぐに終了するのではなく、作動しながらも誤った結果を返すようにします

- **プログラム機能の制限**
  プログラムを作動不能にするのではなく、いくつかの機能（例えば 保存や印刷など）に制限を設けるだけにとどめる手法もあります。ELが接続していなくてもプログラムが作動しつづけているのをみて、成功したものと思わせることができます。

1. EL のセキュリティ

1.1 ELの暗号技術

EL は、CPU, RAM, EEPROM を内蔵したスマートカードをベースにしたドングルです。ユーザは EL 内部で実行するプログラムを開発して EL に転送します。コンピュータ側のユーザプログラムは、必要に応じて EL 側のプログラムとやり取りします。これで EL が接続していなければ、コンピュータ側のプログラムは正常に実行されない仕組みが実現されます。

ユーザ開発プログラムを EL 転送することをコードポートと呼びます。コードポートされたプログラムは EL 内部で実行され、EL の OS 提供関数を利用して各種機能を実現します。EL 内部にはファイルを作成することができます。これらのファイルには外部からは直接アクセス（読み書き）することはできません。EL 内部のプログラムを経由してのみデータを読み書きしなければなりません。また、EL には TDES, RSA 暗号用コプロセッサが搭載されていて高速に処理が行われます。

OSX 対応 EL ドングルでは、コードポートを利用して EL を Matrix 互換 API で操作できるようにしたソリューションです。EL にポート済の専用プログラムをコンピュータ側の API が呼び出します。API は Matrix API 互換ですので、EL ドングルをあたかも Matrix ドングルであるかのように操作できます。また、EL の機能をフルに利用するために、拡張 API が用意されています。拡張 API で TDES/RSA 暗号/復号、タイマー、リアルタイムクロック機能を用意に利用できます。Matrix 同様に、暗号鍵は EL の仕組みによって内部に安全に保管され、処理は EL 内部の CPU が実行します。

開発者は、API を使って 自由に任意の秘密鍵を EL に設定します。一度 EL に書き込んだ秘密鍵は、書き込みを行った開発者も、秘密鍵が設定された EL を配布されたユーザも、EL から読み取ることは不可能です（上書きはできます）。

EL ハードウェアは、書き込まれた暗号鍵を使って、TEA/RSA/TDES 暗号アルゴリズムの暗号機能を提供します。これは、EL ハードウェアにデータを与えると、ハード内部で暗号化が行われ、暗号化後のデータが返されるというものです。 逆に、暗号化されたデータを与えると、復号化後のデータが返されます。　この方法では、秘密鍵が通信されることはなく、秘密鍵で暗号化されたデータがやり取りされますので、たとえデータが盗聴されたとしても、秘密鍵は安全です。

EL はアプリケーションプログラムが利用できるメモリフィールドがあります。機種によってサイズは異なりますが、開発者が任意の数値を読み書きするために利用できます。

コンピュータ側 ソフトウェア	EL 側 ソフトウェア	ハードウェア
API (ドライバ)	コードポート プログラム	EL ハードウェア OS 提供 SES 関数

OSX 対応 EL ドングルにはコードポート済 EL と Windows/OSX/Linux 対応 Matrix 互換 API が提供されます。

現在、EL に秘密鍵やメモリに値を書き込むユーティリティは付属していません。 EL ドングルの操作はすべて API で行ってください。

2. EL の機種

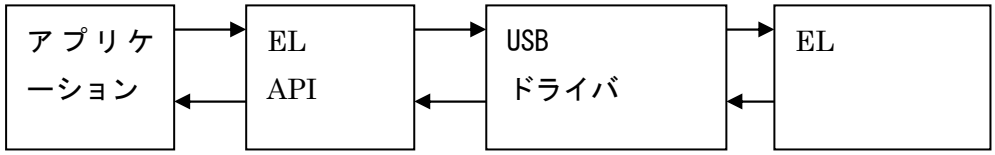
USB タイプ	
Genji	超小型タイプ
EL RTC	リアルタイムクロック内臓

EL RTC 内臓のリアルタイムクロックの時間は正確ではありません。また、時間の変更はできません。しばらくすると、時間はずれてきます。このためプログラム側はこれらを考慮して、時間の調整を行う必要があります。

3. EL API やドライバのインストールや配布方法

3.1 EL API とドライバのアーキテクチャ

ＰＣと EL は、ドライバを経由して通信します。 アプリケーションプログラムとドライバは、API 関数（API 関数を含む DLL）を経由して通信します。



32 ビット/64 ビット、どちらのタイプのプログラムでも EL を利用できます。

- **USB インターフェース**

Windows では、HID モードと専用ドライバモードの２つのモードのどちらかで動作します。 モードは API で設定できます。

> HID モードでは EL は OS (Windows/OSX/Linux)のドライバにより認識されます。別途ドライバを導入する必要はありません。EL USB はプラグアンドプレイ対応なので、接続するとドライバが自動的にＯＳによってロードされます。HID モードで動作するドングルには複数のプログラムが同時にアクセスすることはできません。
>
> 専用ドライバーモードでは EL 専用ドライバで動作します。ドライバは別途導入しなければなりません。ドライバは、複数プログラムからの同時アクセス制御を行います。

Mac OSX/Linux では HID モードでのみ動作します。

3.2 Linux & Mac OSX

Mac OSX/Linux は HID モードの EL キーを自動認識します。

Linux では OS レベルでは EL ドングルは自動認識されますが、そのままでは一般ユーザ権限のプログラムは EL ドングルを認識することはできません。一般ユーザ権限で EL ドングルにアクセスできるようにするには udev ルールを追加しなければなりません。配布ディスクの Linux フォルダ内の udev フォルダに 99-el.rules があります。このファイルを /etc/udev/rules.d ディレクトリにコピーしてください。コピー後, udevadm コマンドでルールファイルを再読み込みさせてください（またはコンピュータを再起動）

```
# udevadm control –reload-rules
```

3.2 API ファイル

Windows

API フォルダ内の 32bit フォルダと 64 ビットフォルダに Visual Studio 用のスタティックライブラリがあります。DLL フォルダ内の 32bit フォルダと 64 ビットフォルダに API DLL とインポートライブラリがあります。

互換 API を呼び出す C/C++プログラムではヘッダファイル matri32rtc.h をインクルードしてください。

OSX

互換 API を利用するには、３つのファイルが必要です。

動的ライブラリ

libELDongle.dylib

libsenseEIV.so

libsenseCrypt.so

３つのファイルを /usr/lib にコピーしてください。

静的ライブラリ

libELDongle.a

libsenseEIV.a

libsenseCrypt.a

３つのファイルをメインプログラムとリンクしてください。

Linux

互換 API を利用するには、３つのファイルが必要です。

動的ライブラリ

libELDongle.so

libsenseEIV.so

libsenseCrypt.so

３つのファイルを ライブラリディレクトリにコピーしてください。

静的ライブラリ

libELDongle.a

libsenseEIV.a

libsenseCrypt.a

３つのファイルをメインプログラムとリンクしてください。

3.3 プロテクトプログラムの配布

プロテクトしたプログラムを配布には以下の関連ファイルを必要に応じて添付してください。プログラムの実行に必要なファイルだけを配布します。

Windows　32/64 ビットアプリケーション

ダイナミックリンクした場合には、matrix32rtc.dll が必要です。 プログラムと同じバージョン 32bit / 64bit DLL を使ってください。DLL はアプリケーションが見つけられるフォルダに配置してください。USB キーが HID モードになっている場合、ドライバを配布する必要はありません。また、C/C++などで静的リンク(static link)ライブラリをつかって生成した実行ファイルは DLL を必要としません。

OSX

動的ライブラリとリンクした場合、動的ライブラリ３つを配布してください。

Linux

udev ファイルと動的ライブラリとリンクした場合、動的ライブラリ３つを配布してください。

4. 動作確認

OSX

OSX フォルダ内に、testELDongle.app という Cocoa プログラム（バンドル）と、testELDongle.app を作成する xcode プロジェクトがあります。

Sample フォルダにはほとんどの互換 API を使った C++のコマンドラインプログラムのソースがあります

Windows

OSX/Linux のコマンドラインプログラムサンプルと同じプログラムが ELDongleTest フォルダ内に Visual Studio 2008 プロジェクトとして用意されています。

Linux

sample フォルダにほとんどの互換 API を使った C++のコマンドラインプログラムのソースがあります。

5. Matrix 互換 API 用のユーティリティ

EL 設定、実行ファイル暗号化、ネットワークドングル関連ユーティリティは付属しません。

API を組み込んだプログラムを解析されて、プロテクト処理部分を無効化されることを防ぐためには、実行ファイルを暗号化する必要があります。

実行ファイル暗号化のための専用プログラムは別途ご用意ください。プログラムの開発言語(.NET 系かネイティブプログラムか)や、32bit / 64bit プログラムなどかによって各種暗号化プログラムが存在します。お問い合わせいただければ暗号化プログラムを推奨します。

6. EL ソフトウェア開発キット

OSX 対応 EL ドングルには既に Matrix 互換 API 用プログラムがコードポートされています。この専用プログラムは EL ソフトウェアキットを使って作成されました。

Matrix 互換 API を使わずに自社で EL ドングルにコードポートするソフトウェアを開発することもできます。Elite EL_vX.X.X というフォルダ内に EL ソフトウェア開発キットが含まれています。このキットと Keil uVision を使えば、EL にコードポートするプログラムを IDE で開発できます。

OSX 対応 EL ドングルは Matrix 互換 API 専用に設定されているため、別のプログラムをコードポートするには EL をリセットする必要があります。弊社まで返送いただければリセットして返送します。事前に連絡をいただければ、Matrix 互換 API 用プログラムがコードポートされていない EL ドングルを最初から納品します。

また、Elite EL_vX.X.X には EL ドングル用のドライバや API ファイルが含まれています。Matrix 互換 API は、これらの API を使っています。互換 API と一緒にここから取り出した API が含まれていますが、もし、Linux ディストリビューションなどで、互換 API と一緒の EL API がうまく動作しない場合は、開発キット内に対象 OS 用の API があるかどうかご確認、お試しください。

Elite EL_vX.X.X フォルダにはユーティリティ類、豊富なサンプルが含まれています。

EL プログラム開発の詳細は「EL プログラム開発編」マニュアルをご参照ください。
また、EL OS 提供関数(SES 関数)、PC 側から EL を操作する API の詳細は、「EL API 編」マニュアルをご参照ください。

7. EL API リファレンス

7.1 静的ライブラリ・動的ライブラリ

EL API を組み込んだプログラムは、EL API を含んだライブラリをリンクするか、呼び出さなければなりません。スタティックリンクとダイナミックリンクの２つの方法があります。 OS プラットフォームに合った方法でご利用ください。

Windows / OSX / Linux で共通の API を利用できます (Linux では LED のブリンクがサポートされません)

7.2 Matrix互換基本APIサマリ

Init_MatrixAPI	Matrix API を初期化する。
Release_MatrixAPI	Matrix API を開放する。
GetVersionAPI	Matrix API のバージョン番号を返す。
GetVersionDRV	LPT ドライバのバージョン番号を返す。
GetVersionDRV_USB	USB ドライバのバージョン番号を返す。
SetW95Access	Windows9x 系 OS で Matrix と通信を行う際、VXD ドライバを経由するかどうかを選択する。
GetPortAdr	LPT ポートのアドレスを返す。
PausePrinterActivity	Windows プリントスプーラを停止する。
ResumePrinterActivity	Windows プリントスプーラを再開する。
Dongle_Find	Matrix が接続されている LPT./USB ポートを返す。
Dongle_FindEx	指定する LPT ポートに接続されている Matrix を全て捜して、結果をバッファにセットする。
Dongle_Count	指定 LPT/USB ポートに接続されている Matrix 数を返す。
Dongle_MemSize	メモリサイズを返す。（バイト）
Dongle_Model	モデル番号を返す。
Dongle_Version	Matrix 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 Matrix 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 Matrix に暗号化させる。
Dongle_DecryptData	8 バイトのデータブロックを Matrix に復号化させる。
Dongle_SetDriverFlag	動作モード（HID/ドライバモード）を設定する
Dongle_GetDriverFlag	動作モード（HID/ドライバモード）を取得する
SetConfig_MatrixNet	ネットワークアクセスを有効・無効にする。
GetConfig_MatrixNet	MxNet サーバプログラムで設定したパラメータ値を取得する。
LogIn_MatrixNet	ネットワークにログオンして、サーバファイルのユーザスロットを獲得する。
LogOut_MatrixNet	ネットワークからログアウトして、ユーザスロットを解放する。

7.3 EL用 Matrix互換拡張APIサマリ

Dongle_SetLED	LED 点灯、点滅、消灯
Dongle_GetRand	EL ハードウェアによる乱数生成
Dongle_GetTime	内臓リアルタイムクロックの時間取得
Dongle_CreateRSAKeyPair	EL 内部の指定場所(idx)にキーペアを生成．３つのキーペアを EL 内部に保存可能
Dongle_LockRSAKeyPair	EL 内部のキーペアの書込ロック
Dongle_GetRSAPubKey	EL 内部の指定場所の公開鍵取得
Dongle_EncryptDataRSA	EL 内部の指定公開鍵で暗号化(API が公開鍵を取得して処理)
Dongle_DecryptDataRSA	EL 内部の指定秘密鍵で復号化（処理は EL 内部で実行)
Dongle_DecryptDataTDES	トリプル DES 復号化
Dongle_EncryptDataTDES	トリプル DES 暗号化
Dongle_WriteKeyTDES	トリプル DES の暗号鍵の設定
Dongle_WriteKeyHMACSHA1	HMAC(SHA1)のパスワード設定
Dongle_HMACSHA1	HMAC(SHA1)ハッシュ値の取得
Dongle_MemSize2	第 2 メモリ領域のサイズ取得
Dongle_WriteData2	第 2 メモリ領域への書込
Dongle_ReadData2	第 2 メモリ領域からの読込
Dongle_LockData	メモリ領域への書込みロック
Dongle_LockData2	第 2 メモリ領域への書込みロック

Dongle_ReadGUSN	EL の固有 ID 取得(Globally Unique Serial Number)を書き込む。
Dongle_SetTimer	タイマーの動作モード設定
Dongle_StartTimer	タイマー開始
Dongle_StopTimer	タイマー停止
Dongle_GetTimer	タイマーカウント取得

7.4 API エラーコード

EL 互換 API では、エラーの発生源は３つあります。１つは PC 側が EL 側にコマンドを与えるときに発生するエラーです。２つ目は、EL 内部のプログラムが返すエラー。３つ目は、暗号関連関連で発生するエラーです。

1 番目のエラーのタイプは ELDongle::ERROR_TYPE::S4 。 2 番目は ELDongle::ERROR_TYPE::SES、そして 3 番目は ELDongle::ERROR_TYPE::CRYPTO で表されます。

それぞれのタイプのエラーコードは以下の通りになります。

１．ELDongle::ERROR_TYPE::S4 -1 から -199 番のエラコード
２．ELDongle::ERROR_TYPE::SES -200 番台エラコード
３．ELDongle::ERROR_TYPE::CRYPTO -300 番台エラコード

特に明記されていなければ、API の返すエラーコード（負数）は、この３つのタイプのいずれかになります。

API が返すエラー番号の意味は、各行の一番右のエラー番号に対応するエラー文字と下のコメント行を参照してください。

```
{ELDongle::ERROR_TYPE::S4, S4_COMM_ERROR,                -1},
/** communication error*/
{ELDongle::ERROR_TYPE::S4, S4_PROTOCOL_ERROR,    -1},
 /** communication protocol error*/
{ELDongle::ERROR_TYPE::SES, ERR_USERCODE,        -2},
// invalid user code
{ELDongle::ERROR_TYPE::SES, ERR_DONGLE_LOCKED,  -4},
// invalid user code
{ELDongle::ERROR_TYPE::S4, S4_UNSUPPORTED,       -7},
/** the function isn't supported*/
{ELDongle::ERROR_TYPE::S4, S4_DEVICE_UNSUPPORTED,       -7},
/** the request can't be supported by the device*/
{ELDongle::ERROR_TYPE::SES, ERR_LOCKED,          -9},
/** Key pair lock */
{ELDongle::ERROR_TYPE::SES, SES_REAL_TIME ,      -10},
/* read clock module error                       */
{ELDongle::ERROR_TYPE::S4, S4_NO_LIST,           -25},
```

```
/** find no device*/
{ELDongle::ERROR_TYPE::S4, S4_UNPOWERED,          -50},
/** the device has been powered off*/
{ELDongle::ERROR_TYPE::S4, S4_INVALID_PARAMETER ,  -51},
/** invalid parameter*/
{ELDongle::ERROR_TYPE::S4, S4_DEVICE_BUSY,        -52},
/** the device is busy*/
{ELDongle::ERROR_TYPE::S4, S4_KEY_REMOVED,        -53},
/** the device has been removed */
{ELDongle::ERROR_TYPE::S4, S4_INSUFFICIENT_BUFFER, -54},
/** the input buffer is insufficient*/
{ELDongle::ERROR_TYPE::S4, S4_GENERAL_ERROR,              -55},
/** general error, commonly indicates not enough memory*/
{ELDongle::ERROR_TYPE::S4, S4_DEVICE_TYPE_MISMATCH,     -56},
/** the device type doesn't match*/
{ELDongle::ERROR_TYPE::S4, S4_FILE_SIZE_CROSS_7FFF,       -57},
/** the execuable file crosses address 0x7FFF*/
{ELDongle::ERROR_TYPE::S4, S4_CURRENT_DF_ISNOT_MF,                -58},
/** a net module must be child directory of the root directory*/
{ELDongle::ERROR_TYPE::S4, S4_INVAILABLE_MODULE_DF,     -59},
/** the current directory is not a module*/
{ELDongle::ERROR_TYPE::S4, S4_FILE_SIZE_TOO_LARGE,                -60},
/** the file size is beyond address 0x7FFF*/
{ELDongle::ERROR_TYPE::S4, S4_DF_SIZE,                    -61},
/** the specified directory size is too small*/
{ELDongle::ERROR_TYPE::S4, S4_FILE_NOT_FOUND,             -62},
/** the specified file or directory can't be found */
{ELDongle::ERROR_TYPE::S4, S4_INSUFFICIENT_SECU_STATE,  -63},
/** the security state doesn't match*/
{ELDongle::ERROR_TYPE::S4, S4_DIRECTORY_EXIST,            -64},
/** the specified directory has already existed*/
{ELDongle::ERROR_TYPE::S4, S4_FILE_EXIST,                 -65},
/** the specified file or directory has already existed*/
{ELDongle::ERROR_TYPE::S4, S4_INSUFFICIENT_SPACE,         -66},
/** the space is insufficient*/
{ELDongle::ERROR_TYPE::S4, S4_OFFSET_BEYOND,              -67},
/** the offset is beyond the file size*/
{ELDongle::ERROR_TYPE::S4, S4_PIN_BLOCK,                  -68},
/** the specified pin or key has been locked*/
{ELDongle::ERROR_TYPE::S4, S4_FILE_TYPE_MISMATCH,                 -69},
```

```
/** the file type doesn't match*/
{ELDongle::ERROR_TYPE::S4, S4_CRYPTO_KEY_NOT_FOUND,    -70},
/** the specified pin or key cann't be found*/
{ELDongle::ERROR_TYPE::S4, S4_APPLICATION_TEMP_BLOCK,  -71},
/** the directory has been temporarily locked*/
{ELDongle::ERROR_TYPE::S4, S4_APPLICATION_PERM_BLOCK,  -72},
/** the directory has been locked*/
{ELDongle::ERROR_TYPE::S4, S4_DATA_BUFFER_LENGTH_ERROR,       -73},
/** invalid data length*/
{ELDongle::ERROR_TYPE::S4, S4_CODE_RANGE,              -74},
/** the PC register of the virtual machine is out of range*/
{ELDongle::ERROR_TYPE::S4, S4_CODE_RESERVED_INST,             -75},
/** invalid instruction*/
{ELDongle::ERROR_TYPE::S4, S4_CODE_RAM_RANGE,          -76},
/** internal ram address is out of range*/
{ELDongle::ERROR_TYPE::S4, S4_CODE_BIT_RANGE,          -77},
/** bit address is out of range*/
{ELDongle::ERROR_TYPE::S4, S4_CODE_SFR_RANGE,          -78},
/** SFR address is out of range*/
{ELDongle::ERROR_TYPE::S4, S4_CODE_XRAM_RANGE,         -79},
/** external ram address is out of range*/
{ELDongle::ERROR_TYPE::S4, S4_ERROR_UNKNOWN,           -80},
/** unknown error*/

{ELDongle::ERROR_TYPE::SES, SES_PARA,              -200},
/* invalid parameter
{ELDongle::ERROR_TYPE::SES, SES_EEPROM,            -201},
        /* write eeprom failed                          */
{ELDongle::ERROR_TYPE::SES, SES_RAM,               -202},
        /* ram out of range                                 */
{ELDongle::ERROR_TYPE::SES, SES_XCOS,              -203},
        /* unknown error                                    */
{ELDongle::ERROR_TYPE::SES, SES_FILEID,            -204},
        /* file not found                               */
{ELDongle::ERROR_TYPE::SES, SES_FILE_ACCESS,       -205},
        /* access file failed                           */
{ELDongle::ERROR_TYPE::SES, SES_FILE_SELECT,       -206},
/* select file error                                 */
{ELDongle::ERROR_TYPE::SES, SES_HANDLE,            -207},
        /* invalid file handle                          */
```

```
{ELDongle::ERROR_TYPE::SES, SES_RANGE,              -208},
        /* out of file range                                    */
{ELDongle::ERROR_TYPE::SES, SES_FILE_SPACE,         -209},
        /* SES no enough space to create new file*/
{ELDongle::ERROR_TYPE::SES, SES_FILE_EXISTING,      -210},
        /* SES file ID has been used                 */
{ELDongle::ERROR_TYPE::SES, SES_KEYID,              -211},
        /* key not found
*/
{ELDongle::ERROR_TYPE::SES, SES_KEY_ACCESS,         -212},
        /* access key failed                                    */
{ELDongle::ERROR_TYPE::SES, SES_SHA1,               -213},
        /* hash failed                                                  */
{ELDongle::ERROR_TYPE::SES, SES_RAND,               -214},
        /* get random data failed
*/
{ELDongle::ERROR_TYPE::SES, SES_RSA,                -215},
                /* RSA calculation failed                           */
{ELDongle::ERROR_TYPE::SES, SES_RSAVERIFY,-216},
/* digest signature verification failed      */
{ELDongle::ERROR_TYPE::SES, SES_INVALID_POINTER, -217},
        /* invalid pointer                                          */
{ELDongle::ERROR_TYPE::SES, SES_INVALID_SIZE,       -218},
        /* invalid size                                             */
{ELDongle::ERROR_TYPE::SES, SES_REAL_TIME_POWER,        -220},

/* the clock module has been power down */
{ELDongle::ERROR_TYPE::SES, ERR_INVALID_PARAMETER,      -221},

// invalid parameter
{ELDongle::ERROR_TYPE::SES, ERR_NOKEY,              -222},

{ELDongle::ERROR_TYPE::CRYPTO, RE_CONTENT_ENCODING,    -300 },
{ELDongle::ERROR_TYPE::CRYPTO, RE_DATA,                -301 },
{ELDongle::ERROR_TYPE::CRYPTO, RE_DIGEST_ALGORITHM,    -302 },
{ELDongle::ERROR_TYPE::CRYPTO, RE_ENCODING,            -303 },
{ELDongle::ERROR_TYPE::CRYPTO, RE_KEY,                 -304 },
{ELDongle::ERROR_TYPE::CRYPTO, RE_KEY_ENCODING,        -305 },
{ELDongle::ERROR_TYPE::CRYPTO, RE_LEN,                 -306 },
{ELDongle::ERROR_TYPE::CRYPTO, RE_MODULUS_LEN,         -307 },
```

```
{ELDongle::ERROR_TYPE::CRYPTO, RE_NEED_RANDOM,          -308 },
{ELDongle::ERROR_TYPE::CRYPTO, RE_PRIVATE_KEY,          -309 },
{ELDongle::ERROR_TYPE::CRYPTO, RE_PUBLIC_KEY,           -310 },
{ELDongle::ERROR_TYPE::CRYPTO, RE_SIGNATURE,            -311 },
{ELDongle::ERROR_TYPE::CRYPTO, RE_SIGNATURE_ENCODING, -312 },
{ELDongle::ERROR_TYPE::CRYPTO, RE_ENCRYPTION_ALGORITHM,
-313 },
{ELDongle::ERROR_TYPE::CRYPTO, RE_PARAMETER,            -314 },
        // error parameter
{ELDongle::ERROR_TYPE::CRYPTO, RE_MEMORY,               -315 },
        // alloc memory failed
```

7.5 戻り値、引数の型

互換 API では、引数、戻り値の型は C の組み込み型ではなく mxtypes.h に定義されている型を使っています.

_mxINT16	16 ビット符号有数	(Windows では _mxINT16)
_mxUINT16	16 ビット符号無し数	(Windows では unsigned _mxINT16)
_mxINT32	32 ビット符号有数	(Windows では _mxINT32)
_mxUINT32	32 ビット符号無し数	(Windows では _mxUINT32)

クロスプラットフォーム開発を行うケースでは、組み込み型_mxINT32, _mxUINT32 のビット長が異なるため、そのまま移植すると誤作動の原因となりかねません。

しかし、例えば、_mxINT32 型を指定しておけば移植先でも 32 ビット符号有数として処理されるため､組み込み型のビット長の違いから起きる問題を回避できます｡Linux OS 32bit と 64bit の２つの環境で開発するケースや Windows プログラムを移植する際に mxtypes.h の型は重宝します。

7.6 EL用 Matrix互換 API

Init_MatrixAPI	
説明	EL API を初期化します。 EL API を呼び出す前に、必ず呼び出してください。
呼出し	_mxINT16 Init_ELAPI()
引数	なし
戻り値	0 で成功、0 以外はエラー
対象	

Release_MatrixAPI	
説明	EL API を開放します。 EL API を使い終わったら、必ず呼び出してください。
呼出し	_mxINT16 Release_ELAPI()
引数	なし
戻り値	なし
対象	

GetVersionAPI	
説明	EL API のバージョン番号を返します。
呼出し	_mxINT32 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
対象	

GetVersionDRV_USB	
説明	USB ドライバのバージョン番号を返します。
呼出し	_mxINT32 GetVersionDRV_USB()
引数	なし
戻り値	LPT ドライバのバージョン番号 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
対象	USB

SetW95Access		
説明	何もしません（互換のため存在）	
呼出し	void SetW95Access　(_mxINT16 Mode)	
引数	Mode	1 2
戻り値	なし	
対象	LPT	

GetPortAddr		
説明	何もしません（互換のため存在）	
呼出し	_mxINT16 GetPortAddr(_mxINT16 LptNr);	
引数	LptNr	ポート番号[1]
戻り値	常に０。	
対象		

[1]

PausePrinterActivity (Windows OSのみ)	
説明	何もしません（互換のため存在）
呼出し	_mxINT16 PausePrinterActivity()
引数	なし
戻り値	常に-14（ プリントスプーラ停止失敗 ）
対象	

ResumePrinterActivity (Windows OSのみ)	
説明	何もしません（互換のため存在）
呼出し	_mxINT16 ResumePrinterActivity()
引数	なし
戻り値	常に-14 (　　プリンタスプーラ再開失敗)
対象	

Dongle_Find	
説明	EL が接続されているポートを探して、見つかった最初のポートを返します。
呼出し	_mxINT16 Dongle_Find()
引数	なし
戻り値	EL が見つかれば 85(USB) .。85 以外はエラー
対象	

Dongle_FindEx	
説明	何もしません（互換のため存在）
呼出し	_mxINT16 Dongle_FindEx(DNGINFO *xBuffer)

引数	DNGINFO *xBuffer[2]
	typedef struct {
	int LPT_Nr; // LPT ポート番号
	int LPT_Adr; // LPT ポートアドレス
	int DNG_Cnt; // 装着されている EL 数
	} DNGINFO

戻り値	常に 0 0 LPT ポートが存在しない
対象	

[2] xBuffer は DNGINFO 配列へのポインタです。配列の最大サイズは 3（LPT1-LPT3）

Dongle_Count		
説明	引数で指定されたポートに装着された EL 数を返します。	
呼出し	_mxINT16 Dongle_Count(_mxINT16 PortNr);	
引数	PortNr	USB　'U' (Acii 85)
戻り値	0 以上の指定ポートに装着された EL 数[3] 。<0 はエラー	
対象		

[3]

Dongle_MemSize		
説明	バイト単位で EL の既定メモリ領域のメモリサイズを返します。 EL に既定メモリ領域とは別に第 2 メモリ領域があります。***Dongle_MemSize2*** でサイズを取得できます。	
呼出し	_mxINT16 Dongle_MemSize(_mxINT16 DngNr, _mxINT16 PortNr)	
引数	DngNr	EL の番号[4]
	PortNr	'U' (Ascii 85)固定
戻り値	0 以上でバイト単位のメモリサイズ。<0 でエラー	
対象		

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。

Dongle_Model		
説明	EL ハードウェアのモデル番号を返します。	
呼出し	_mxINT32 Dongle_Model(_mxINT16 DngNr, _mxINT16 PortNr)	
引数	DngNr	EL の番号[5]
	PortNr	'U' (Ascii 85)固定
戻り値	EL ハードウェアのバージョン	
対象		

Dongle_Version		
説明	EL ソフトウェアのバージョン番号を返します。	
呼出し	_mxINT32 Dongle_Version(_mxINT16 DngNr, _mxINT16 PortNr)	
引数	DngNr	EL の番号[6]
	PortNr	U’ (Ascii 85)
戻り値	EL のバージョン番号[7] <0 でエラー 例 v2.4.4 であれば *0x00020404*.	
対象		

6

7

Dongle_ReadData		
説明	EL 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。[8]	
呼出し	_mxINT16 Dongle_ReadData(_mxINT32 UserCode, _mxINT32 *Data, _mxINT16 Count, _mxINT16 DngNr, _mxINT16 Port Nr)	
引数	UserCode	割り当てられたユーザコード[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]
	Count	読み込むデータフィールド数
	DngNr	EL の番号[11]
	Port Nr	'U' (Ascii 85)
戻り値	読み込まれたデータフィールド数。<0 でエラー	
対象		

。

[8] 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。
[9] EL 内のユーザコードと一致しなければなりません。
[10] _mxINT16 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 EL が装着できるため、ポート番号に加えて、この引数で EL の番号を指定する必要があります。

Dongle_ReadDataEx		
説明	EL 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。[12]	
呼出し	_mxINT16 Dongle_ReadDataEx(_mxINT32 UserCode, _mxINT32 *Data, _mxINT16 Fpos, _mxINT16 Count, _mxINT16 DngNr, _mxINT16 Port Nr)	
引数	UserCode	割り当てられたユーザコー[13]
	*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[14]
	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	EL の番号[15]
	PortNr	'U' (Ascii 85)
戻り値	読み込まれたデータフィールド数. <0 でエラー	
対象		

[12] 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。

[13] EL 内のユーザコードと一致しなければなりません。

[14] _mxINT16 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 EL が装着できるため、ポート番号に加えて、この引数で EL の番号を指定する必要があります。

Dongle_WriteData		
説明	EL 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。[16]	
呼出し	_mxINT16 Dongle_WriteData(_mxINT32 UserCode, _mxINT32 *Data, _mxINT16 Count, _mxINT16 DngNr, _mxINT16 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]
	Count	書き込むデータフィールド数
	DngNr	EL 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。<0 でエラー	
対象		

[16] 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。
[17] EL 内のユーザコードと一致しなければなりません。
[18] _mxINT16 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 EL が装着できるため、ポート番号に加えて、この引数で EL の番号を指定する必要があります。

Dongle_WriteDataEx		
説明	EL 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。[20]	
呼出し	_mxINT16 Dongle_WriteDataEx(_mxINT32 UserCode, _mxINT32 *Data, _mxINT16 Fpos, _mxINT16 Count, _mxINT16 DngNr, _mxINT16 Port Nr)	
引数	UserCode	割り当てられたユーザコード[21]
	*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[22]
	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	EL の番号[23]
	PortNr	'U' (Ascii 85)
戻り値	書き込まれたデータフィールド数。<0 でエラー	
対象		

[20] 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。

[21] EL 内のユーザコードと一致しなければなりません。

[22] _mxINT16 Count で指定する数以上のサイズがなければなりません。

[23] 1つのポートに複数の EL が装着できるため、ポート番号に加えて、この引数で EL の番号を指定する必要があります。

Dongle_ReadSerNr		
説明	EL のシリアル番号を読み込みます。	
呼出し	_mxINT32 Dongle_ReadSerNr (_mxINT32 UserCode, _mxINT16 DngNr, _mxINT16 PortNr)	
引数	UserCode	割り当てられたユーザコード[24]
	DngNr	EL の番号[25]
	PortNr	'U' (Ascii 85)
戻り値	指定 EL のシリアル番号	
対象		

[24] EL 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 EL が装着できるため、ポート番号に加えて、この引数で EL 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	_mxINT16 Dongle_WriteKey(_mxINT32 UserCode, _mxINT32* KeyData, _mxINT16 DngNr, _mxINT16 PortNr)	
引数	UserCode	割り当てられたユーザコード[26]
	* 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	EL の番号[27]
	PortNr	'U' (Ascii 85)
戻り値	1 で書き込み成功。1 以外でエラー。	
対象		

[26] EL 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 EL が装着できるため、ポート番号に加えて、この引数で EL 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 EL に書き込まれているか確認します	
呼出し	_mxINT16 Dongle_GetKeyFlag(_mxINT32 UserCode, _mxINT16 DngNr, _mxINT16 PortNr)	
引数	UserCode	割り当てられたユーザコード[28]
	DngNr	EL の番号[29]
	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。 <0 でエラー	
対象		

注）128 ビット TEA キーは EL 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。

[28] EL 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 EL が装着できるため、ポート番号に加えて、この引数で EL 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 EL に暗号化させます。	
呼出し	_mxINT16 Dongle_EncryptData(_mxINT32 UserCode, _mxINT32* DataBlock, _mxINT16 DngNr, _mxINT16 PortNr)	
引数	UserCode	割り当てられたユーザコード[30]
	* 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	EL の番号[31]
	PortNr	'U' (Ascii 85)
戻り値	1 で成功、1 以外でエラー	
対象		

[30] EL 内のユーザコードと一致しなければなりません。

[31] 1つのポートに複数の EL が装着できるため、ポート番号に加えて、この引数で EL 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 EL に復号化させます。	
呼出し	_mxINT16 Dongle_DecryptData(_mxINT32 UserCode, _mxINT32* DataBlock, _mxINT16 DngNr, _mxINT16 PortNr)	
引数	UserCode	割り当てられたユーザコード[32]
	*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	EL の番号[33]
	PortNr	'U' (Ascii 85)
戻り値	1 で成功、1 以外でエラー	
対象		

[32] EL 内のユーザコードと一致しなければなりません。
[33] 1つのポートに複数の EL が装着できるため、ポート番号に加えて、この引数で EL の番号を指定する必要があります。

Dongle_SetDriverFlag		
説明	動作モードを HID モードかドライバーモードに設定します。	
呼出し	_mxINT16 Dongle_SetDriverFlag(_mxINT32 UserCode, _mxINT16 Mode, _mxINT16 DngNr, _mxINT16 PortNr)	
引数	UserCode	割り当てられたユーザコード
	Mode	0 又は 1 0 – ドライバーモード 1 - HID モード
	DngNr	EL の番号
	PortNr	'U' (Ascii 85)を指定
戻り値	1 で成功、1 以外でエラー	
対象		

- Windows のみ。 Linux/OSX では常に HID で動作させます

Dongle_GetDriverFlag		
説明	現在の動作モードを （HID モードかドライバーモード）を取得します。	
呼出し	_mxINT16 Dongle_GetDriverFlag(_mxINT32 UserCode, _mxINT16 DngNr, _mxINT16 PortNr)	
引数	UserCode	割り当てられたユーザコード
	DngNr	EL の番号
	PortNr	’U’ (Ascii 85)を指定
戻り値	0 又は 1、0,.1 以外でエラー。 1 現在の動作モードは HID モード 0 現在の動作モードはドライバモード	
対象		

SetConfig_MatrixNet		
説明	何もしません（互換目的でのみ存在）	
呼出し	_mxINT16 SetConfig_ELNet(_mxINT16 nAccess, char* nFile)	
引数	nAccess	0 ＝ ネットアクセス無効 （ローカル接続された EL を使用） 1 ＝ ネットアクセス有効
	nFile	MxNet ネットワークアクセス用サーバファイルの名前[34]
戻り値	常に 0 0 ネットアクセスが無効化された。	
対象	Net	

GetConfig_MatrixNet		
説明	何もしません（互換目的でのみ存在）	
呼出し	_mxINT32 GetConfig_ELNet(_mxINT16 Category)	
引数	Category	取得するパラメータを指定します。 0 - サーバファイルのバージョン番号[35] 1 - 設定リフレッシュ時間 （分単位） 2 - 最後のファイルリフレッシュからの経過時間 3 - 設定ユーザタイムアウト（分単位）
戻り値	常に 0	
対象		

Login_MatrixNet		
説明	何もしません（互換目的でのみ存在）	
呼出し	_mxINT16 LogIn_ELNet(_mxINT32 UserCode, _mxINT16 AppSlot, _mxINT16 DngNr)	
引数	UserCode	割り当てられたユーザコード[36]
	AppSlot	ライセンス数を定義した EL 内のデータフィールド番号
	DngNr	EL の番号[37]
戻り値	常に -35	
	-35	MxNet サーバプログラムが実行されていない
対象		

[36] EL 内のユーザコードと一致しなければなりません。
[37] 1つのポートに複数の EL が装着できるため、ポート番号に加えて、この引数で EL の番号を指定する必要があります。

LogOut_MatrixNet		
説明	何もしません（互換目的でのみ存在）	
呼出し	_mxINT16 LogOut_ELNet(_mxINT32 UserCode, _mxINT16 AppSlot, _mxINT16 DngNr)	
引数	UserCode	割り当てられたユーザコード[38]
	AppSlot	ライセンス数を定義した EL 内のデータフィールド番号
	DngNr	EL の番号[39]
戻り値	常に -34 -34　サーバファイルにアクセス中にエラーが発生	
対象		

[38] EL 内のユーザコードと一致しなければなりません。
[39] 1つのポートに複数の EL が装着できるため、ポート番号に加えて、この引数で EL の番号を指定する必要があります。

Dongle_SetLED		
説明	LED を点灯、点滅、消灯します	
呼出し	_mxINT16 Dongle_SetLED(_mxINT16 mode, _mxINT16 dNr, _mxINT16 PortNr);	
引数	Mode	LED モード <0 点灯 ==0 消灯 >0 点滅 (一秒間の点滅回数を指定)
	dNr	ドングル番号
	PortNr	ポート番号(85 固定)
戻り値	1 で成功。1 以外はエラー	
対象		

Dongle_GetRand		
説明	EL ハードウェアによる乱数生成。初期化処理は不要です	
呼出し	_mxINT16 WINAPI Dongle_GetRand(_mxINT32 UserCode, _mxINT16 siz, unsigned char* pRand, _mxINT16 dNr, _mxINT16 PortNr)	
引数	UserCode	ユーザーコード
	siz	生成する乱数のサイズ（バイト）
	pRand	乱数を受け取るバッファ。sizで指定した乱数を受け取る大きさでなければなりません。
	dNr	ドングル番号
	PortNr	ポート番号(85 固定)
戻り値	1 で成功。1 以外はエラー	
対象		

Dongle_GetTime		
説明	内臓リアルタイムクロックの時間取得 取得される時間は 標準時（GMT）の 1970 年 1 月 1 日の 00:00:00 から現在までの経過秒数(C 標準関数の *time()* と同じ)	
呼出し	_mxINT16 Dongle_GetTime(_mxINT32 UserCode,_mxUINT32* nTime, _mxINT16 dNr, _mxINT16 PortNr)	
引数	UserCode	ユーザーコード
	nTime	時計の日時を受け取るバッファ。
	dNr	ドングル番号
	PortNr	ポート番号(85 固定)
戻り値	1 で成功。1 以外はエラー	
対象		

Dongle_ReadGUSN		
説明	EL の固有 ID 取得(Globally Unique Serial Number)を読み込む GUSN はハードウェアに組み込まれている ID で EL 間で重複しません。 Dongle_ReadSerNr で取得するシリアル番号は 互換 API が割り当てている番号です。 両者はまったく異なります。固有 ID には GUSN を優先させてください。	
呼出し	_mxINT16 Dongle_ReadGUSN(_mxINT32 UserCode, unsigned char* gSN, _mxINT16 dNr, _mxINT16 PortNr)	
引数	UserCode	ユーザーコード
	gSN	GUSNを受け取る32バイトのバッファ。
	dNr	ドングル番号
	PortNr	ポート番号(85 固定)
戻り値	1 で成功。1 以外はエラー	
対象		

Dongle_MemSize2		
説明	第 2 メモリ領域のサイズ取得 EL には既定のメモリ領域とは別に第２メモリ領域があります。	
呼出し	_mxINT16 Dongle_MemSize2(_mxINT16 dNr, _mxINT16 PortNr)	
引数	dNr	ドングル番号
	PortNr	ポート番号(85 固定)
戻り値	>0 でメモリサイズ。>0 以外はエラー	
対象		

Dongle_ReadData2

説明	第 2 メモリ領域からの読込	
呼出し	_mxINT16 Dongle_ReadData2(_mxINT32 UserCode, unsigned char* data, _mxUINT16 pos, _mxUINT16 len, _mxINT16 dNr, _mxINT16 PortNr) 第２メモリ領域のサイズは Dongle_MemSiz2 で取得できます。その領域の任意の位置から任意の長さのバイトデータを読み込みます。	
引数	UserCode	ユーザーコード
	data	データを受け取るバッファ。
	pos	読み込む位置
	len	読み込む長さ
	dNr	ドングル番号
	PortNr	ポート番号(85 固定)
戻り値	>0 で読み込んだデータ長。>0 以外はエラー	
対象		

Dongle_WriteData2		
説明	第 2 メモリ領域への書込	
呼出し	_mxINT16 WINAPI Dongle_WriteData2(_mxINT32 UserCode, unsigned char* data, _mxUINT16 pos, _mxUINT16 len, _mxINT16 dNr, _mxINT16 PortNr) 第２メモリ領域のサイズは Dongle_MemSiz2 で取得できます。その領域の任意の位置から任意の長さのバイトデータを書き込みます。	
引数	UserCode	ユーザーコード
	data	書き込むデータが入ったバッファ。
	pos	書き込む位置
	len	書き込む長さ
	dNr	ドングル番号
	PortNr	ポート番号(85 固定)
戻り値	>0 で書き込んだデータ長。>0 以外はエラー	
対象		

Dongle_LockData

説明	メモリ領域への書込みロック	
呼出し	_mxINT16 Dongle_LockData(_mxINT32 UserCode, BOOL bLock, _mxUINT32 lockKey, _mxINT16 dNr, _mxINT16 PortNr) 書込みロックすることで、メモリ領域への書込みができないようになります。	
引数	UserCode	ユーザーコード
	bLock	ロックモード TRUE ロック FALSE ロック解除
	lockKey	ロックキー（暗証番号）。 ロック時に使ったロックキーでのみ、書込みロックを解除できます。
	dNr	ドングル番号
	PortNr	ポート番号(85 固定)
戻り値	1 で成功。1 以外はエラー	
対象		

Dongle_LockData2

説明	第 2 メモリ領域への書込みロック	
呼出し	_mxINT16 Dongle_LockData2(_mxINT32 UserCode, BOOL bLock, _mxUINT32 lockKey, _mxINT16 dNr, _mxINT16 PortNr) 書込みロックすることで、メモリ領域への書込みができないようになります。	
引数	UserCode	ユーザーコード
	bLock	ロックモード TRUE　ロック FALSE　ロック解除
	lockKey	ロックキー（暗証番号）。 ロック時に使ったロックキーでのみ、書込みロックを解除できます。
	dNr	ドングル番号
	PortNr	ポート番号(85 固定)
戻り値	1 で成功。1 以外はエラー	
対象		

Dongle_SetTimer

説明	タイマーの動作モード設定	
呼出し	_mxINT16 WINAPI Dongle_SetTimer(_mxINT32 UserCode, _mxINT16 mode, _mxINT16 dNr, _mxINT16 PortNr) タイマーは CPU がカウントする 64 ビットカウンタです。カウンタ値と CPU 周波数から経過時間を取得できます。EL が正常に動作しているときのみ利用可能です。	
引数	UserCode	ユーザーコード
	mode	タイマーモード 0 非循環モード カウンタ値がオーバフローするとタイマーは止まります 1 循環モード カウンタ値がオーバフローすると 0 に戻ります
	dNr	ドングル番号
	PortNr	ポート番号(85 固定)
戻り値	1 で成功。1 以外はエラー	
対象		

Dongle_StartTimer		
説明	タイマーを開始します。	
呼出し	_mxINT16 WINAPI Dongle_StartTimer(_mxINT32 UserCode, _mxINT16 dNr, _mxINT16 PortNr)	
引数	UserCode	ユーザーコード
	dNr	ドングル番号
	PortNr	ポート番号(85 固定)
戻り値	1 で成功。1 以外はエラー	
対象		

Dongle_GetTimer

説明	タイマーのカウンター値を取得します タイマーのカウンターは、64 CPU サイクルで１つ増加します。EL の Ver 2.3.6 の CPU 周波数は 24MHz です。このため、カウンターの１つは約 2.7μ 秒に相当します。 1*64/24000000 ≈ 2.7μ カウンタが 0 から DWOD の最大値 0xffffffff になるのは（DWORD 値がオーバフローするのは）約 3.21 時間後です。 0xffffffff*2.7/1000000/3600 ≈ 3.21 hours
呼出し	_mxINT16 WINAPI Dongle_GetTimer(_mxINT32 UserCode, _mxUINT32* dwCount, _mxINT16 dNr, _mxINT16 PortNr)
引数	UserCode　ユーザーコード dwCount　タイマー値を取得する変数へのポインタ dNr　ドングル番号 PortNr　ポート番号(85 固定)
戻り値	1 で成功。1 以外はエラー
対象	

Dongle_StopTimer		
説明	タイマーを停止します	
呼出し	_mxINT16 Dongle_StopTimer(_mxINT32 UserCode, _mxINT16 dNr, _mxINT16 PortNr)	
引数	UserCode	ユーザーコード
	dNr	ドングル番号
	PortNr	ポート番号(85 固定)
戻り値	1 で成功。1 以外はエラー	
対象		

Dongle_CreateRSAKeyPair		
説明	互換 API では 3 つ(1,2,3)の RSA キーペアを保存できます。指定番号の RSA キーペアを生成、保存します。	
呼出し	_mxINT16Dongle_CreateRSAKeyPair(_mxINT32 UserCode, _mxINT16 idx, _mxINT16 dNr, _mxINT16 PortNr)	
引数	UserCode	ユーザーコード
	idx	キーペア番号(1, 2, 3 を指定)
	dNr	ドングル番号
	PortNr	ポート番号(85 固定)
戻り値	1 で成功。1 以外はエラー	
対象		

Dongle_LockRSAKeyPair

説明	EL 内部のキーペアの書込ロック RSA キーペア番号を指定して、その番号に対してキーペアを生成できないようにロックします。ロック設定時に暗証番号となる 32 ビット数を指定します。ロック解除には同じ暗証番号を指定しなければなりません。	
呼出し	_mxINT16 Dongle_LockRSAKeyPair(_mxINT32 UserCode, _mxINT16 idx, BOOL bLock, _mxUINT32 lockKey, _mxINT16 dNr, _mxINT16 PortNr)	
引数	UserCode	ユーザーコード
	Idx	キーペア番号(1, 2, 3 を指定)
	bLock	ロックモード TRUE ロック FALSE ロック解除
	lockKey	ロックキー（暗証番号）
	dNr	ドングル番号
	PortNr	ポート番号(85 固定)
戻り値	1 で成功。1 以外はエラー	
対象		

Dongle_GetRSAPubKey		
説明	指定番号の RSA キーペアの公開鍵を取得します。	
呼出し	_mxINT16 Dongle_GetRSAPubKey(_mxINT32 UserCode, _mxINT16 idx, unsigned char* modulus, _mxINT16* modulus_len, unsigned char* exponent, _mxINT16* exponent_len, _mxINT16 dNr, _mxINT16 PortNr)	
引数	UserCode	ユーザーコード
	Idx	キーペア番号(1, 2, 3 を指定)
	modulus	Modulus を受け取るバッファ(128バイト)
	modulus_len	modules にセットされたデータ長を受け取る変数へのポインタ
	exponent	Exponentを受け取るバッファ (4バイト)
	exponent_len	exponent にセットされたデータ長を受け取る変数へのポインタ
	dNr	ドングル番号
	PortNr	ポート番号(85 固定)
戻り値	1 で成功。1 以外はエラー	
対象		

Dongle_EncryptDataRSA		
説明	EL 内部の指定公開鍵でデータを暗号化します。この API は、EL から指定番号キーペアの公開鍵を取得してから、コンピュータ側で暗号処理を行います。	
呼出し	_mxINT16 WINAPI Dongle_EncryptDataRSA(_mxINT32 UserCode, _mxINT16 idx, unsigned char* plainText, _mxINT16 plainTextLen, unsigned char* cipher, _mxINT16* cipherLen, _mxINT16 dNr, _mxINT16 PortNr)	
引数	UserCode	ユーザーコード
	idx	キーペア番号(1, 2, 3 を指定)
	plainText	暗号化する平文データ
	plainTextLen	暗号化する平文データ長を受け取る変数へのポインタ
	cipher	暗号化後のデータ
	cipherLen	暗号化後データの長さを受け取る変数へのポインタ
	dNr	ドングル番号
	PortNr	ポート番号(85 固定)
戻り値	1 で成功。1 以外はエラー	
対象		

Dongle_DecryptDataRSA		
説明	EL 内部の指定秘密鍵でデータを復号化します。復号化処理は EL 内部で行われます。	
呼出し	_mxINT16 WINAPI Dongle_DecryptDataRSA(_mxINT32 UserCode, _mxINT16 idx, unsigned char* cipher, _mxINT16 cipherLen, unsigned char* plainText, _mxINT16* plainTextLen, _mxINT16 dNr, _mxINT16 PortNr)	
引数	UserCode	ユーザーコード
	Idx	キーペア番号(1, 2, 3 を指定)
	Cipher	暗号化する平文データ
	cipherLen	暗号化する平文データ長
	plaintext	暗号化後のデータ
	plainTextLen	暗号化後データの長さ
	dNr	ドングル番号
	PortNr	ポート番号(85 固定)
戻り値	1 で成功。1 以外はエラー	
対象		

Dongle_WriteKeyTDES		
説明	トリプル DES の暗号鍵を EL に書き込みます。	
呼出し	_mxINT16 WINAPI Dongle_WriteKeyTDES(_mxINT32 UserCode, unsigned char* key, _mxINT16 dNr, _mxINT16 PortNr)	
引数	UserCode	ユーザーコード
	key	暗号鍵 (16バイト固定)
	dNr	ドングル番号
	PortNr	ポート番号(85 固定)
戻り値	1 で成功。1 以外はエラー	
対象		

Dongle_EncryptDataTDES		
説明	トリプル DES で指定データを暗号化します。処理は EL 内部で行われます。	
呼出し	_mxINT16 WINAPI Dongle_EncryptDataTDES(_mxINT32 UserCode, unsigned char* iv, char*plainText, _mxINT16 plainLen, char*cipher, _mxINT16* cipherLen, _mxINT16 dNr, _mxINT16 PortNr)	
引数	UserCode	ユーザーコード
	iv	初期ベクトル（ 8バイト固定 ）
	plainText	暗号化する平文データ
	plainTextLen	暗号化する平文データ長
	cipher	暗号化後のデータ
	cipherLen	暗号化後データの長さ
	dNr	ドングル番号
	PortNr	ポート番号(85 固定)
戻り値	1 で成功。1 以外はエラー	
対象		

Dongle_DecryptDataTDES		
説明	トリプル DES で指定データを復号化します。処理は EL 内部で行われます。	
呼出し	_mxINT16 WINAPI Dongle_DecryptDataTDES(_mxINT32 UserCode, unsigned char* iv, char*cipher, _mxINT16 cipherLen, char*plain, _mxINT16* plainLen, _mxINT16 dNr, _mxINT16 PortNr)	
引数	UserCode	ユーザーコード
	iv	初期ベクトル（ 8バイト固定 ）
	cipher	復号化するデータを保持するバッファ
	cipherLen	復号化するデータ長を受け取る変数へのポインタ
	plaintext	復号化後のデータを保持するバッファ
	plainTextLen	復号化後データの長さを受け取る変数へのポインタ
	dNr	ドングル番号
	PortNr	ポート番号(85 固定)
戻り値	1 で成功。1 以外はエラー	
対象		

Dongle_WriteKeyHMACSHA1		
説明	HMAC(SHA1)のパスワードを EL に書き込みます	
呼出し	_mxINT16 WINAPI Dongle_WriteKeyHMACSHA1(_mxINT32 UserCode, unsigned char* key, _mxINT16 dNr, _mxINT16 PortNr)	
引数	UserCode	ユーザーコード
	key	暗号鍵（25バイト固定）
	dNr	ドングル番号
	PortNr	ポート番号(85 固定)
戻り値	1 で成功。1 以外はエラー	
対象		

Dongle_HMACSHA1		
説明	HMAC(SHA1)ハッシュ値を計算します。処理は EL 内部で行われます。	
呼出し	_mxINT16 WINAPI Dongle_HMACSHA1(_mxINT32 UserCode, char*plainText, _mxINT16 plainTextLen, char*hash, _mxINT16* hashLen, _mxINT16 dNr, _mxINT16 PortNr)	
引数	UserCode	ユーザーコード
	plainText	HMAC値を計算するデータを保持するバッファ
	plainTextLen	バッファ長を保持する変数へのポインタ
	hash	ハッシュ値を受け取るバッファ
	hashLen	IN　ハッシュ値を受け取るバッファの長さ 計算されたハッシュ値を格納できるだけのサイズを指定してください。計算されたハッシュ値の方が長いと、ハッシュ値はバッファ長までしかセットされません。 OUT ハッシュ値を受け取るバッファにセットされたデータ長.
	dNr	ドングル番号
	PortNr	ポート番号(85 固定)
戻り値	1 で成功。1 以外はエラー	
対象		