

EL Matrix 互換 API ユーザマニュアル

©RiBiG Inc

有限会社リビグ

<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

日本語マニュアル、上記 URL オンラインマニュアルに含まれる内容は、とくに断りのない限り、（有）リビグが著作権を有しており、その扱いは日本の著作権法に従います。

私的利用・引用・その他法律が認めた範囲をこえて、本マニュアルやオンラインマニュアルの内容の全体もしくは一部を、（有）リビグに無断で本・CD-ROM・WWW ページその他の媒体に複製し、頒布あるいは閲覧させる等の行為を一切禁じます。

目次

はじめに： ソフトウェア保護へのガイドライン.....	5
2. EL の機種.....	8
3. EL API やドライバのインストールや配布方法.....	9
3.1 EL API とドライバのアーキテクチャ.....	9
3.2 Linux & Mac OSX.....	9
3.3 プロテクトプログラムの配布.....	10
7. EL API リファレンス.....	11
7.1 静的ライブラリ・動的ライブラリ.....	11
7.2 基本 API サマリ.....	12
7.3 拡張 API サマリ.....	13
7.4 API エラーコード.....	15
7.6 EL 用 Genji API	20
Init_MatrixAPI.....	20
Release_MatrixAPI	20
GetVersionAPI	21
Dongle_Count	22
Dongle_MemSize	23
Dongle_Model	24
Dongle_Version	25
Dongle_ReadData.....	26
Dongle_ReadDataEx.....	27
Dongle_WriteData.....	28
Dongle_WriteDataEx.....	29
Dongle_ReadSerNr	30
Dongle_WriteKey.....	31
Dongle_GetKeyFlag	32
Dongle_EncryptData	33
Dongle_DecryptData	34
Dongle_SetDriverFlag	35
Dongle_SetLED	36
Dongle_GetRand.....	37
Dongle_GetTime	38
Dongle_ReadGUSN.....	39
Dongle_MemSize2.....	40
Dongle_ReadData2.....	41

Dongle_WriteData2	42
Dongle_LockData	43
Dongle_LockData2	44
Dongle_SetTimer	45
Dongle_StartTimer	46
Dongle_GetTimer	47
Dongle_StopTimer	48
Dongle_CreateRSAKeyPair	49
Dongle_LockRSAKeyPair	50
Dongle_GetRSAPubKey	51
Dongle_EncryptDataRSA	52
Dongle_DecryptDataRSA	53
Dongle_WriteKeyTDES	54
Dongle_EncryptDataTDES	55
Dongle_DecryptDataTDES	56
Dongle_WriteKeyHMACSHA1	57
Dongle_HMACSHA1	58

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る、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が、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くと、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。
- d.

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に変更されたら、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。

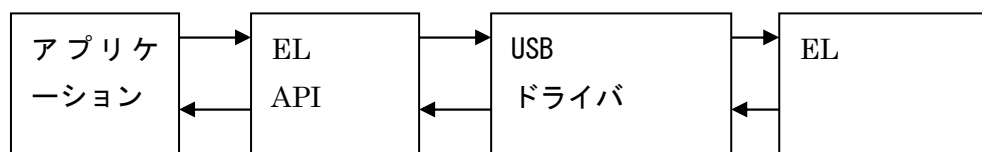
2. EL の機種

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

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 dongle 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。

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。

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

Linux

動的ライブラリ

`libgenji.so`

静的ライブラリ

`libgenji.a`

OSX

動的ライブラリ

`libgenji.dylib`

静的ライブラリ

`libgenji.a`

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、`genji.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、動的ライブラリを配布してください。

Linux

API ファイルに同梱の `readme.txt` をご覧ください

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。

7.2 基本APIサマリ

Init_MatrixAPI	Genji API を初期化する。
Release_MatrixAPI	Genji API を開放する。
GetVersionAPI	Genji API のバージョン番号を返す。
Dongle_Count	指定 LPT/USB ポートに接続されている Genji 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	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 Genji 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 Genji に暗号化させる。
Dongle_DecryptData	8 バイトのデータブロックを Genji 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

7.3 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 executable 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_INVAIABLE_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_CRYPT0_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
```

7.6 EL用 Genji 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 int16_t Init_MatrixAPI()

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 int16_t Release_MatrixAPI()

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 `int32_t 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

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 `int16_t Dongle_Count(int16_t 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¹。 <0 はエラー

対象

¹

Dongle_MemSize		
説明	<p>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。</p> <p>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。<i>Dongle_MemSize2</i> でサイズを取得できます。</p>	
呼出し	int16_t Dongle_MemSize(int16_t DngNr, int16_t PortNr)	
引数	DngNr	EL の番号 ²
	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	int32_t Dongle_Model(int16_t DngNr, int16_t PortNr)	
引数	DngNr	EL の番号 ³
	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	int32_t Dongle_Version(int16_t DngNr, int16_t PortNr)	
引数	DngNr	EL の番号 ⁴
	PortNr	U' (Ascii 85)
戻り値	EL のバージョン番号 ⁵ <0 でエラー 例 v2.4.4 であれば 0x00020404.	
対象		

4

5

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。 6	
呼出し	int16_t Dongle_ReadData(int32_t UserCode, int32_t *Data, int16_t Count, int16_t DngNr, int16_t Port Nr)	
引数	UserCode	割り当てられたユーザコード ⁷
	*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 ⁸
	Count	読み込むデータフィールド数
	DngNr	EL の番号 ⁹
	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		

。

⁶ 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。

⁷ EL 内のユーザコードと一致しなければなりません。

⁸ int16_t Count で指定する数以上のサイズがなければなりません。

⁹ 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。 ¹⁰	
呼出し	int16_t Dongle_ReadDataEx(int32_t UserCode, int32_t *Data, int16_t Fpos, int16_t Count, int16_t DngNr, int16_t Port Nr)	
引数	UserCode	割り当てられたユーザコード ¹¹
	*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 ¹²
	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 ¹³
	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		

¹⁰ 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。

¹¹ EL 内のユーザコードと一致しなければなりません。

¹² int16_t Count で指定する数以上のサイズがなければなりません。

¹³ 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。 ¹⁴	
呼出し	int16_t Dongle_WriteData(int32_t UserCode, int32_t *Data, int16_t Count, int16_t DngNr, int16_t Port Nr)	
引数	UserCode	割り当てられたユーザコード ¹⁵
	*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 ¹⁶
	Count	書き込むデータフィールド数
	DngNr	EL の番号 ¹⁷
	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		

¹⁴ 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。

¹⁵ EL 内のユーザコードと一致しなければなりません。

¹⁶ int16_t Count で指定する数以上のサイズがなければなりません。

¹⁷ 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。 ¹⁸												
呼出し	int16_t Dongle_WriteDataEx(int32_t UserCode, int32_t *Data, int16_t Fpos, int16_t Count, int16_t DngNr, int16_t Port Nr)												
引数	<table> <tr> <td>UserCode</td><td>割り当てられたユーザコード¹⁹</td></tr> <tr> <td>*Data</td><td>データフィールドに書き込むデータをセットした配列²⁰</td></tr> <tr> <td>Fpos</td><td>書き込みを開始するデータフィールド番号</td></tr> <tr> <td>Count</td><td>書き込むデータフィールド数</td></tr> <tr> <td>DngNr</td><td>EL の番号²¹</td></tr> <tr> <td>PortNr</td><td>'U' (Ascii 85)</td></tr> </table>	UserCode	割り当てられたユーザコード ¹⁹	*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 ²⁰	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 ²¹	PortNr	'U' (Ascii 85)
UserCode	割り当てられたユーザコード ¹⁹												
*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 ²⁰												
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 ²¹												
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													

¹⁸ 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。

¹⁹ EL 内のユーザコードと一致しなければなりません。

²⁰ int16_t Count で指定する数以上のサイズがなければなりません。

²¹ 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	int32_t Dongle_ReadSerNr (int32_t UserCode, int16_t DngNr, int16_t PortNr)	
引数	UserCode	割り当てられたユーザコード ²²
	DngNr	EL の番号 ²³
	PortNr	'U' (Ascii 85)
戻り値	指定 EL のシリアル番号	
対象		

²² EL 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 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 `int16_t Dongle_WriteKey(int32_t UserCode, int32_t* KeyData, int16_t DngNr, int16_t PortNr)`

引数	UserCode	割り当てられたユーザコード ²⁴
	* 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 ²⁵
	PortNr	'U' (Ascii 85)

戻り値 1 で書き込み成功。1 以外でエラー。

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_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	int16_t Dongle_GetKeyFlag(int32_t UserCode, int16_t DngNr, int16_t PortNr)	
引数	UserCode	割り当てられたユーザコード ²⁶
	DngNr	EL の番号 ²⁷
	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。

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_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	int16_t Dongle_EncryptData(int32_t UserCode, int32_t* DataBlock, int16_t DngNr, int16_t PortNr)
引数	<div> <div> <div>UserCode</div> <div>割り当てられたユーザコード²⁸</div> </div> <div> <div>*</div> <div>暗号化する 8 バイトのデータブロックへのポインタ</div> </div> <div> <div>DataBlock</div> <div></div> </div> <div> <div>DngNr</div> <div>EL の番号²⁹</div> </div> <div> <div>PortNr</div> <div>'U' (Ascii 85)</div> </div> </div>
戻り値	1 で成功、1 以外でエラー
対象	

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_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	int16_t Dongle_DecryptData(int32_t UserCode, int32_t* DataBlock, int16_t DngNr, int16_t PortNr)	
引数	UserCode	割り当てられたユーザコード ³⁰
	*DataBlock	復号化する 8 バイトのデータブロックへのポインタ
	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_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	int16_t Dongle_SetDriverFlag(int32_t UserCode, int16_t Mode, int16_t DngNr, int16_t PortNr)								
引数	<table> <tr> <td>UserCode</td><td>割り当てられたユーザコード</td></tr> <tr> <td>Mode</td><td>0 又は 1 0 - ドライバーモード 1 - HID モード</td></tr> <tr> <td>DngNr</td><td>EL の番号</td></tr> <tr> <td>PortNr</td><td>'U' (Ascii 85)を指定</td></tr> </table>	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
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									

Dongle_SetLED

説明 LED を点灯、点滅、消灯します

呼出し `int16_t Dongle_SetLED(int16_t mode, int16_t dNr, int16_t 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 `int16_t WINAPI Dongle_GetRand(int32_t UserCode, int16_t siz, unsigned char* pRand, int16_t dNr, int16_t 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を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

呼出し int16_t Dongle_GetTime(int32_t UserCode,struct tm* stime, int16_t dNr, int16_t 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 `int16_t Dongle_ReadGUSN(int32_t UserCode, unsigned char* gSN, int16_t dNr, int16_t 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 `int16_t Dongle_MemSize2(int16_t dNr, int16_t 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 `int16_t Dongle_ReadData2(int32_t UserCode, unsigned char* data, uint16_t pos, uint16_t len, int16_t dNr, int16_t 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	<code>UserCode</code>	ユーザーコード
	<code>data</code>	データを受け取るバッファ。
	<code>pos</code>	読み込む位置
	<code>len</code>	読み込む長さ
	<code>dNr</code>	dongle 番号
	<code>PortNr</code>	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 `int16_t WINAPI Dongle_WriteData2(int32_t UserCode, unsigned char* data, uint16_t pos, uint16_t len, int16_t dNr, int16_t 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	<code>UserCode</code>	ユーザーコード
	<code>data</code>	書き込むデータが入ったバッファ。
	<code>pos</code>	書き込む位置
	<code>len</code>	書き込む長さ
	<code>dNr</code>	dongle 番号
	<code>PortNr</code>	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	<pre>int16_t Dongle_LockData(int32_t UserCode, BOOL bLock, uint32_t lockKey, int16_t dNr, int16_t PortNr)</pre> <p>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くなります。</p>										
引数	<table> <tr> <td>UserCode</td><td>ユーザーコード</td></tr> <tr> <td>bLock</td><td> ロックモード TRUE ロック FALSE ロック解除 </td></tr> <tr> <td>lockKey</td><td> 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。 </td></tr> <tr> <td>dNr</td><td> dongle番号</td></tr> <tr> <td>PortNr</td><td>ポート番号(85 固定)</td></tr> </table>	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	dongle番号	PortNr	ポート番号(85 固定)
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	dongle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 メモリ領域への書き込みロック

呼出し `int16_t Dongle_LockData2(int32_t UserCode, BOOL bLock, uint32_t lockKey, int16_t dNr, int16_t PortNr)`

書き込みロックすることで、メモリ領域への書き込みができなくなります。

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、書き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 int16_t WINAPI Dongle_SetTimer(int32_t UserCode, int16_t mode, int16_t dNr, int16_t 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ー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ーフローすると 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 int16_t WINAPI Dongle_StartTimer(int32_t UserCode, int16_t dNr, int16_t PortNr)

引数	UserCode	ユーザーコード
	dNr	dong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。

$$1 * 64 / 24000000 \approx 2.7 \mu$$

カウンタが 0 から DWORD の最大値 0xffffffff になるのは (DWORD 値がオーバーフローするのは) 約 3.21 時間後です。

$$0xffffffff * 2.7 / 1000000 / 3600 \approx 3.21 \text{ hours}$$

呼出し int16_t WINAPI Dongle_GetTimer(int32_t UserCode, uint32_t* dwCount, int16_t dNr, int16_t 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	dongle 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 `int16_t Dongle_StopTimer(int32_t UserCode, int16_t dNr, int16_t 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 int16_tDongle_CreateRSAKeyPair(int32_t UserCode, int16_t idx, int16_t dNr, int16_t 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 `int16_t Dongle_LockRSAKeyPair(int32_t UserCode, int16_t idx, BOOL bLock, uint32_t lockKey, int16_t dNr, int16_t 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 `int16_t Dongle_GetRSAPubKey(int32_t UserCode, int16_t idx, unsigned char* modulus, int16_t* modulus_len, unsigned char* exponent, int16_t* exponent_len, int16_t dNr, int16_t 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	dong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 `int16_t WINAPI Dongle_EncryptDataRSA(int32_t UserCode, int16_t idx, unsigned char* plainText, int16_t plainTextLen, unsigned char* cipher, int16_t* cipherLen, int16_t dNr, int16_t 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	dongle 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	int16_t WINAPI Dongle_DecryptDataRSA(int32_t UserCode, int16_t idx, unsigned char* cipher, int16_t cipherLen, unsigned char* plainText, int16_t* plainTextLen, int16_t dNr, int16_t 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	dong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 int16_t WINAPI Dongle_WriteKeyTDES(int32_t UserCode, unsigned char* key, int16_t dNr, int16_t PortNr)

引数	UserCode	ユーザーコード
	key	暗号鍵 (16バイト固定)
	dNr	dongle 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 `int16_t WINAPI Dongle_EncryptDataTDES(int32_t UserCode, unsigned char* iv, char*plainText, int16_t plainLen, char*cipher, int16_t* cipherLen, int16_t dNr, int16_t 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	dong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 `int16_t WINAPI Dongle_DecryptDataTDES(int32_t UserCode, unsigned char* iv, char*cipher, int16_t cipherLen, char*plain, int16_t* plainLen, int16_t dNr, int16_t 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	int16_t WINAPI Dongle_WriteKeyHMACSHA1(int32_t UserCode, unsigned char* key, int16_t dNr, int16_t 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 int16_t WINAPI Dongle_HMACSHA1(int32_t UserCode, char*plainText, int16_t plainTextLen, char*hash, int16_t* hashLen, int16_t dNr, int16_t 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	dongle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