HT シリーズ
PA シリーズ
RH シリーズ
プログラミングマニュアル

2015.1

日本語版　V1.29

ユニテック・ジャパン（株）

目　　次

1. 概要

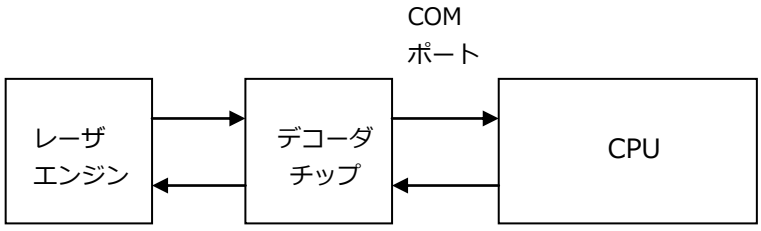
本マニュアルは Window Mobile/Windows CE 用に提供されている各種の開発キットやツールについて、そしていろいろなプログラミングインタフェースの概要について説明しています。Windows Mobile/Windows CE のアプリケーションプログラムは、Windows Mobile/Windows CE デバイスに合った開発、エミュレート、そしてリモート・デバッグをするために標準の Microsoft 開発環境 IDE - Embedded Visual C++ 4.0 と Micosoft Visual Studio .NET 2005 または 2008 Professional Edition で開発することができます。

2. SDK とは?

Microsoft は異なる Windows CE プラットフォーム用に SDK を提供しています。これはサポートされるモジュール、GUI そして各プラットフォームのコンポーネントに違いがあるからです。ユーザがアプリケーションのコンパイルに不適当な SDK を使用すると予期できない問題が発生することがあります。ユーザはアプリケーションプログラムを開発する場合にターゲットプラットフォームに合った SDK を選択しなければなりません。
Unitech のターミナルについても各プラットフォームにあった SDK を提供しております。SDK の入手については、本書中の URL を参照、もしくはユニテック・ジャパンまたは代理店にお問い合わせ下さい。
Windows Mobile および Embedded Handheld OS については、Microsoft の提供する各バージョンの Windows Mobile 用 SDK と当社の提供する Windows Mobile PDA デバイスに内蔵されている USI.DLL を使用して開発を行います。

3. スキャナからデータをダウンロードする方法

HT シリーズ(HT630 を除く)/PA シリーズ(PA700 を除く)/RH シリーズと標準の HPC/Pocket PC/Windows Mobile の大きな違いはバーコード入力ができるかどうかです。Windows Mobile/Windows CE のリファレンスマニュアルにはバーコード入力についての情報は含まれていません。このセクションはバーコードサブシステムのプログラミング構造と HT シリーズ/PA シリーズ/RH シリーズのプログラミング・ユーティリティ・ライブラリを紹介しています。 HT シリーズ/PA シリーズ/RH シリーズの内部には、SE900/SE950 レーザエンジンのコントロールとバーコードのデコードを行うための高機能なデコーダチップがあります。以下は HT シリーズ/PA シリーズ/RH シリーズバーコードのシステム図です:



上記の図にあるように、HT シリーズ/PA シリーズ/RH シリーズはシリアル ポート COMx 経由でデコーダチップと通信します。その通信パラメータは、38400,N,8,1 に固定されています。一般にデコーダチップは、COMx がオープンしていないときはスリープ状態です。デコーダチップは COMx がオープンすると作動し、トリガー キーが押されると、レーザエンジンからのバーコード“ 信号” がデコードされます。デコードの後は、バーコード データとそのバーコードのタイプが直接 HT シリーズ/PA シリーズ/RH シリーズに送られます。
プログラマの皆様が、特に、バーコードやシリアルポートのコントロールに慣れていない場合は、プログラム言語だけでデコーダチップをコントロールすることが難しいことに気づかれるでしょう。このため、ユニテック社はデコーダチップをコントロールするためにユーザあるいはアプリケーションのプログラマの皆さんに以下のユーティリティライブラリとプログラムを提供しています。

1. Scan2Key

アプリケーションプログラム “Scan2Key.exe” はレーザースキャナから入力データを読み、そして HT シリーズ/PA シリーズ/RH シリーズのキーボードバッファに直接データを入力する便利なアプリケーションプログラムです。 Scan2Key.exe はバーコードデータの入力を簡単にし、そして COM ポートのプログラミングに慣れていないプログラマに特に役に立ちます。ユーザプログラムは単にキーボードからバーコードデータを読みます。バーコードシンボルの設定については、サポートされているシンボルと区切り文字のすべてを定義するため、そして**コントロールパネル**から **Scanner Setting** プログラムを実行することができます。

2. ユーティリティライブラリ

プログラミングコントロールのために、HT シリーズ/PA シリーズ/RH シリーズはスキャナ入力のコントロール、シンボルの設定、そしてプロファイルのコントロールをユーザができるように USI.DLL を提供しています。詳細な API リストをご覧下さい。

USI.DLL は、各デバイスに実装されているユニテックの新しいスキャナ関数ライブラリです。以前の製品と互換性をとるために、ユニテック は、既存の PT930/PT930SA ユーザがこれらのソフトウエアをデバイスに実装できるように Scanner3.DLL と ScanKey3.DLL を提供していますが、Scanner3.DLL と ScanKey3.DLL のいくつかの API は、他のデバイスからすでに除かれています。

4. HT シリーズ/PA シリーズ/RH シリーズの COM ポート

一般:

COM 1	本体の RS232 ポート (ActiveSync)
COM 2	スキャナ (1D デコーダ)または RFID(RH767)
COM 3	IrCOMM
COM 4	USB クライアント
COM 5	IrDA または Bluetooth
COM 6	予約
COM 7	Bluetooth プリンタ
COM 8	Bluetooth モデム
COM 9	Bluetooth ActiveSync

PA968*:*

COM 0	USB モデム
COM 1	Bluetooth
COM 2	スキャナ(1D デコーダ) または RFID リーダ
COM 3	予約
COM 4	予約
COM 5	GPS
COM 6	予約
COM 7	Bluetooth プリンタ
COM 8	Bluetooth モデム
COM 9	Bluetooth ActiveSync

HT680:

COM 1	Bluetooth
COM 2	スキャナ (1D デコーダ)
COM 3	予約
COM 4	USB クライアント
COM 5	予約
COM 6	GPRS
COM 7	Bluetooth プリンタ
COM 8	Bluetooth モデム
COM 9	Bluetooth ActiveSync

HT660e:

COM 1	USB モデム
COM 2	スキャナ (1D デコーダ)
COM 3	予約
COM 4	Bluetooth
COM 5	USB Client
COM 6	予約
COM 7	Bluetooth プリンタ
COM 8	Bluetooth モデム
COM 9	Bluetooth ActiveSync

PA600：

Mobile バージョン

COM 0	USB シリアル変換
COM 1	予約
COM 2	Bluetooth
COM 3	IrDAComm
COM 4	スキャナ (1D デコーダ)
COM 5	Bluetooth モデム
COM 6	USB クライアント
COM 7	予約
COM 8	予約
COM 9	RawIR / RFID

CE バージョン:

COM 0	USB シリアル変換
COM 1	Bluetooth
COM 2	RFID/スキャナ(1D デコーダ)
COM 3	IrDAComm
COM 4	USB クライアント
COM 5	RawIR
COM 6	予約
COM 7	BT モデム
COM 8	BT プリンタ
COM 9	BT ActiveSync

PA968II/PA690/PA692：

COM 0	スキャナ (1D デコーダ)
COM 1	USB シリアル変換
COM 2	Bluetooth
COM 3	予約
COM 4	GPS(仮想ポート)
COM 5	BT モデム (PA692 BT DUN)
COM 6	USB クライアント
COM 7	Bluetooth BTHATCI サーバ
COM 8	予約(PA690/PA692 HF ポート)
COM 9	予約

PA550：

COM 0	スキャナ (1D デコーダ)
COM 1	USB シリアル変換
COM 2	Bluetooth
COM 3	GPS
COM 4	GPS(仮想ポート)
COM 5	BT モデム
COM 6	USB クライアント
COM 7	Bluetooth BTHATCI サーバ
COM 8	予約
COM 9	予約

HT682:

COM 0	スキャナ (1D デコーダ)
COM 1	予約
COM 2	Bluetooth
COM 3	予約
COM 4	予約
COM 5	USB クライアント
COM 6	予約
COM 7	Bluetooth プリンタ
COM 8	Bluetooth モデム
COM 9	Bluetooth ActiveSync

PA500II:

COM 0	スキャナ (1D スキャナ)
COM 1	USB シリアル変換
COM 2	Bluetooth
COM 3	予約
COM 4	予約
COM 5	Bluetooth モデム
COM 6	USB クライアント
COM 7	Bluetooth BTHATCI サーバ
COM 8	予約
COM 9	予約

5. サンプルプログラムと SDK

VC, C# と VB.NET のサンプルプログラムを以下からダウンロードすることが出来ます。SDK は、Windows CE バージョンにのみ提供されます。Windows Mobile, Windows Embedded Handheld OS では、SDK は使用しません。

SDK のダウンロード

以下の URL から SDK をダウンロードすることができます。

HT660	SDK	http://w3.tw.ute.com/pub/cs/sdk/ht660/HT660SDK.zip
HT680	SDK	http://w3.tw.ute.com/pub/cs/sdk/HT680/HT680SDK.zip
HT660II/660e	SDK	http://w3.tw.ute.com/pub/cs/SDK/HT660II/HT660eSDK.zip
HT660IICore	SDK	http://w3.tw.ute.com/pub/cs/SDK/HT660II/HT660IICore.zip
PA962/963	SDK	http://w3.tw.ute.com/pub/cs/sdk/pa962/pa962sdk.zip
PA966/967	SDK	http://w3.tw.ute.com/pub/cs/sdk/pa966/pa966sdk.zip
PA968	SDK	http://w3.tw.ute.com/pub/cs/sdk/pa968/pa968sdk.zip
PA982	SDK	http://w3.tw.ute.com/pub/cs/sdk/pa982/Pa982SDK.zip
RH767	SDK	http://w3.tw.ute.com/pub/cs/sdk/RH767/RH767_CE5_SDK.zip
PA600	SDK	http://w3.tw.ute.com/pub/cs/SDK/PA600/PA600_CE5_SDK.zip
PA692	SDK	http://w3.tw.ute.com/pub/cs/SDK/PA692/PA692_CE_SDK.zip

6. USI.DLL – Unitech スキャナインターフェース DLL

以下のリンクには、USI のサンプルプログラムと SDK があります。
http://w3.tw.ute.com/pub/cs/SDK/USI/USISDK.zip

注：プログラミングについては、Unitech のビルトイン DLL を使用して動的に DLL を読み込む必要があります。(Unitech は、Windows Mobile OS 用に *.H と *.LIB を提供しておりません。プログラミングについては、第 9 章をご覧下さい。

6.1 USI DLL にアプリケーションを登録

関数の説明:

アプリケーションを USI.DLL に登録するので、DLL はアプリケーションと通信することができます。スキャナポート(例えば、COM2)をオープンし、初期化します。そして、スキャナを動作モードにセットします。アプリケーションは、スキャナの終了後に DLL から登録解除のために USI_Unregister をコールしなければなりません。

関数コール:

BOOL USI_Register(HWND hwnd, UINT msgID);

パラメータ: (入力)

hwnd: USI DLL がエラーメッセージ、スキャンデータレディ等を含むすべての動作をレポートするためのメッセージを送信する Window のハンドル。

msgID: 送られるメッセージを指定します。 DLL は PostMessage(hwnd, msgID, msg, param) を呼ぶことによってメッセージを送ります。

Window の処理は以下の一つである msgID と wPARAM パラメータについてのカスタムメッセージを受け取ります。

SM_ERROR_SYS: システムエラーを示します。これはシステム関数を呼ぶことによって起こります。Param は GetLastEror()からのエラーコードを含んでいます。

SM_ERROR エラーを示します。Param は以下に示すエラーの理由を含んでいます。

SERR_INVALID_HWND: 無効な window ハンドル。

SERR_INVALID_MSGID: msgID は 0 にすることはできません。

SERR_OPEN_SCANNER: スキャナポートのオープンまたは初期化異常。

SERR_CHECKSUM: 受信したパケットのチェックサムエラー。

SERR_DATALOST: データバッファが空でないので、新しいスキャナデータが失われました。

SERR_BUFFEROVERFLOW: データバッファオーバーフロー。標準サイズは 4K バイト。

SM_REPLY 返答を受信したことを示します。スキャンデータ以外のスキャナからのすべての応答はこのメッセージによって通知されます。

SM_DATAREADY　　スキャンデータは正しくデコードされ、読み込む準備ができていることを示します。

SM_ACK　　ACK を受信したことを示します。

SM_NAK　　NAK を受信したことを示します。

SM_NOREAD　　未読パケットを受信したことを示します。

注: スキャナポート設定は以下に説明するレジストリで定義されます。

[HKEY_LOCAL_MACHINE¥SOFTWARE¥Unitech America Inc.¥Scanner¥Settings]
"COMPORT"="COM2:"
"BAUDRATE"="38400"
"STOPBITS"="1"
"PARITY"="None"
"CHECKPARITY"="1"

戻り値:

BOOL:　　TRUE　: OK
　　FALSE　: エラー

6.2 USI.DLL からアプリケーションの登録を解除する

関数の説明:

DLL からアプリケーションの登録を解除します。スキャナポートを閉じ、標準では、スキャナを使用できないようにします。

関数コール:

void USI_Unregister();

戻りコード:

なし

6.3 スキャナを有効 / 無効にする

関数の説明:

USI 関数をスタートまたはストップします。この関数は、キーパッド入力のみが必要、もしくは入力バッファをクリアしておきたい場合一時的にスキャナ機能を停止するアプリケーションに便利です。

関数コール:

BOOL USI_EnableScan(BOOL bStatus);

パラメータ: (入力)

bStatus: TRUE ：スキャナを有効にする。
FALSE ：スキャナを無効にする。

戻りコード:

BOOL： TRUE ：OK
FALSE ：エラー

6.4 スキャナをリセット

関数の説明:

スキャナを動作モードにセットします。そして、通信コントロールをリセットします。

関数コール:

BOOL USI_Reset();

戻り値:

常に TRUE

6.5 エラーコードを得る

関数の説明:

エラーコード (SERR_***) を戻します。

関数コール:

DWORD USI_GetError();

戻り値:

エラーコード (SERR_***)を戻します。これは USI_Register 関数で説明した値です。

6.6 システムエラーコードを戻す

関数の説明:

システムエラーコードを戻します。これは GetLastError によって戻されます。 また NULL 以外のバッファ中のエラーの説明も戻ります。

関数コール:

DWORD USI_GetLastSysError(LPTSTR buffer, int len);

パラメータ: (出力)

buffer:	LPTSTR:	エラーメッセージのデータバッファ
len:	int:	エラーメッセージの長さ

戻り値:

システム関数 GetLastError によって戻されたシステムエラーコードを戻します。 また、システム関数 FormatMessage によって得られたバッファ中のエラーの説明も NULL でなければ戻ります。

エラーコードの完全なリストは、SDK ヘッダファイル WINERROR.H をご覧下さい。

6.7 スキャンデータを得る

関数の説明:

スキャンデータをバッファに取り込みます。文字長を戻します。バーコードタイプを NULL でなければ戻します。戻り値 0 はバッファがデータを保持するのに短すぎることを意味しています。USI_GetData は SM_DATAREADY メッセージを受け取ったときに呼ばれなければなりません。あるいはデータを廃棄するために USI_ResetData を呼びます。これらの両方共にデータバッファをリセットするので、次のスキャンデータを入力することができます。

もしデータバッファが空ではなく、新しいスキャンデータが発生した場合、これは棄却され SERR_DATALOST のコードを持つエラーメッセージ SM_ERROR が送られます。

関数コール:

UINT USI_GetData(LPBYTE buffer, UINT len, UINT* type);

パラメータ: (入力)

len：UINT：Len はバッファの最大長を指定します。

パラメータ: (出力)

buffer ：LPBYTE：スキャンしたデータを保存するデータバッファ。

type ：UINT ：USI.H で定義されたバーコードタイプ。以下のリストを参照。

```
BCT_CODE_39                 // Code 39
BCT_CODABAR                 // CodaBar
BCT_CODE_128                // Code 128
BCT_INTERLEAVED_2OF5        // Interleaves 2 of 5
BCT_CODE_93                 // Code 93
BCT_UPC_A                   // UPC A
BCT_UPC_A_2SUPPS            // UPC A with 2 Supps
BCT_UPC_A_5SUPPS            // UPC A with 5 Supps
```

BCT_UPC_E0	// UPC E
BCT_UPC_E0_2SUPPS	// UPC E with 2 Supps
BCT_UPC_E0_5SUPPS	// UPC E with 5 Supps
BCT_EAN_8	// EAN 8
BCT_EAN_8_2SUPPS	// EAN 8 with 2 Supps
BCT_EAN_8_5SUPPS	// EAN 8 with 5 Supps
BCT_EAN_13	// EAN 13
BCT_EAN_13_2SUPPS	// EAN 13 with 2 Supps
BCT_EAN_13_5SUPPS	// EAN 13 with 5 Supps
BCT_MSI_PLESSEY	// MSI Plessey
BCT_EAN_128	// EAN 128
BCT_UPC_E1	// UPC E1
BCT_UPC_E1_2SUPPS	// UPC E1 with 2 Supps
BCT_UPC_E1_5SUPPS	// UPC E1 with 5 Supps
BCT_TRIOPTIC_CODE_39	// TRIOPTIC CODE 39
BCT_BOOKLAND_EAN	// Bookland EAN
BCT_COUPON_CODE	// Coupon Code
BCT_STANDARD_2OF5	// Standard 2 of 5
BCT_CODE_11_TELPEN	// Code 11 Telpen
BCT_CODE_32	// Code 32
BCT_DELTA_CODE	// Delta Code
BCT_LABEL_CODE	// Label Code IV & V
BCT_PLESSEY_CODE	// Plessey Code
BCT_TOSHIBA_CODE	// Toshiba Code China Postal Code/Matrix 2 of 5

戻り値:

UINT　：データ長

6.8 スキャンしたデータの長さを得る

関数の説明:

スキャンデータのデータ長を戻します。スキャンデータを保持するためにメモリ割り付け時に、文字列終端子に対して最小 1 バイトを追加します。

関数コール:

UINT USI_GetDataLength();

戻り値:

UNIT : データ長

6.9 シンボル名を得る

関数の説明:

バーコードタイプ名を戻します。

関数コール:

LPCTSTR USI_GetBarcodeName(UINT type, LPBYTE buffer, UINT len);

パラメータ: (入力)

type : UINT : バーコードタイプ (タイプ定義については 6.7 を参照)。

buffer : LPBYTE : 以下の表を参照。

Type	Buffer
BCT_CODE_39	Code 39
BCT_CODABAR	Codabar
BCT_CODE_128	Code 128
BCT_INTERLEAVED_2OF5	Interleaved 2 of 5
BCT_CODE_93	Code 93
BCT_UPC_A	UPC A
BCT_UPC_A_2SUPPS	UPC A with 2 Supps.
BCT_UPC_A_5SUPPS	UPC A with 5 Supps.
BCT_UPC_E0	UPC E
BCT_UPC_E0_2SUPPS	UPC E with 2 Supps.
BCT_UPC_E0_5SUPPS	UPC E with 5 Supps.
BCT_EAN_8	EAN 8
BCT_EAN_8_2SUPPS	EAN 8 with 2 Supps.
BCT_EAN_8_5SUPPS	EAN 8 with 5 Supps.
BCT_EAN_13	EAN 13
BCT_EAN_13_2SUPPS	EAN 13 with 2 Supps.
BCT_EAN_13_5SUPPS	EAN 13 with 5 Supps.
BCT_MSI_PLESSEY	MSI Plessey
BCT_EAN_128	EAN 128
BCT_TRIOPTIC_CODE_39	Trioptic Code 39
BCT_BOOKLAND_EAN	Bookland EAN
BCT_COUPON_CODE	Coupon Code
BCT_STANDARD_2OF5	Standard 2 of 5
BCT_CODE_11_TELPEN	Code 11 or Telpen
BCT_CODE_32	Code 32 (Pharmacy Code)
BCT_DELTA_CODE	Delta Code
BCT_LABEL_CODE	Label Code IV & V
BCT_PLESSEY_CODE	Plessey Code
BCT_TOSHIBA_CODE	Toshiba Code (China Postal Code), Matrix 25

len : UINT : 2 番目のバッファパラメータのストリング長

戻り値:

TRUE　: バーコードタイプの名前があった場合。
FALSE : ない場合 (タイプが間違っている可能性あり)

6.10　スキャンデータのシステムバッファをクリア

関数の説明:

データバッファをリセットするので、次の新しいデータを入力することができます。

関数コール:

void USI_ResetData();

6.11　読み取り表示

関数の説明:

スキャンデータを受け取ったことを知らせます。これは音 (wave ファイル scanok.wav)を鳴らしそして LED を 1 秒間点灯します。

関数コール:

void USI_ReadOK();

注:
USI は LED をオン・オフするために、以下で説明する(例、UPI300.DLL)レジストリで定義された DLL によってエクスポートされた関数 GoodReadLEDOn をコールします。DLL が定義されていないか、関数がなかった場合、USI は GoodReadLEDOn のコールをバイパスします。

[HKEY_LOCAL_MACHINE¥SOFTWARE¥Unitech America Inc.¥Scanner¥Settings]
"DLLLEDCONTROL"="UPI300.DLL"

GoodReadLEDOn の関数プロトタイプ:
VOID WINAPI GoodReadLEDOn(BOOL fon);
fon が TRUE の時オンにし、fon が FALSE の時オフにします。

6.12　最後に送ったコマンドの承認を待つ

関数の説明:

最後に送ったコマンドの承認をタイムアウトになるまで待ちます。一連のコマンドを一度に送る必要がある場合に便利です。USI_SendCommand をコールする前に、以前のコマンドが終了していることを確認するために USI_WaitForSendEchoTO をコールします。

関数コール:

BOOL USI_WaitForSendEchoTO(DWORD timeout);

パラメータ: (入力)

timeout: DWORD　　: タイムアウトをミリ秒で指定します。

戻り値:

タイムアウトの場合 FALSE を戻します。

6.13 プロファイルに設定を保存

関数の説明:

現在のスキャナ設定を保存するので、設定はユニットの電源を切り、そして再度オンにしても保持されています。

関数コール:

BOOL USI_SaveCurrentSettings();

戻り値：

TRUE: 正常な場合、
FALSE: エラー

6.14 スキャナ設定を指定したファイルに保存

関数の説明:

現在の設定をファイルに保存します。ファイルは拡張子として “*.USI” が付きます。

関数コール:

BOOL USI_SaveSettingsToFile(LPCTSTR filename)

パラメータ: (入力)

filename： LPCTSTR: 設定プロファイルのファイル名

戻り値:

TRUE = 成功
FALSE = エラー

6.15 指定した設定プロファイルからスキャナ設定を変更

関数の説明:

ファイルから設定を読んで有効にします。

関数コール:

BOOL USI_LoadSettingsFromFile(LPCTSTR filename, BOOL formulaOnly);

パラメータ: (入力)

filename: LPCTSTR： スキャナ設定プロファイル (*.USI) の名前。
formulaOnly: BOOL: TRUE の場合、データ編集式のみが読み込まれます。 他の設定は変更されないまま残ります。

戻り値:

TRUE = 成功
FALSE = エラー

6.16 トリガキーを押すとスキャナビームを自動的に有効にする

関数の説明:

自動スキャンの開始。スキャンエンジンは自動的にトリガオンになります。

関数コール:

BOOL USI_StartAutoScan(DWORD interval);

パラメータ: (入力)

interval： DWORD: 間隔をミリ秒で指定します。

戻り値:

TRUE = 成功
FALSE = エラー

注: USI はスキャナのスタートとストップをするために以下(例、UPI300.DLL)で説明されたレジストリで定義された DLL によってエクスポートされた SetScannerOn 関数をコールします。DLL が定義されていないか、あるいは関数がなかった場合、自動スキャンは有効にはなりません。

[HKEY_LOCAL_MACHINE¥SOFTWARE¥Unitech America Inc.¥Scanner¥Settings]
"DLLSCANNERCONTROL"="UPI300.DLL"

SetScannerOn の関数プロトタイプ:
VOID WINAPI SetScannerOn(BOOL fon);
fon が TRUE の時スタート、そして fon が FALSE の時ストップ。

6.17 自動スキャン機能を停止

関数の説明:

自動スキャンを停止します。

関数コール:

void USI_StopAutoScan();

6.18 自動スキャンが有効であることをチェック

関数の説明:

自動スキャン機能が有効かどうかをチェックします。

関数コール:

BOOL USI_IsAutoScanning()

戻り値:

BOOL: 自動スキャンが働いているときは TRUE を戻します。
自動スキャンが無効の場合は FALSE を戻します。

6.19 Scan2Key.exe プログラムが実行しているかどうかをチェック

関数の説明:

Scan2Key アプリケーションがバックグラウンドで実行しているかどうかをテストします。(これは Scan2Key がスキャナ入力をキーボードに転送していることを意味していません。転送機能が有効かどうかをチェックするには S2K_IsEnabled() をコールして下さい。)

関数コール:

HWND S2K_IsLoaded();

戻り値:

NULL：Scan2Key は実行していません。
NULL 以外：scan2key は実行していることを示します。scan2key について Window ハンドルを戻しますが、内部使用です-メッセージ送信。

6.20 Scan2Key が有効であることをチェック

関数の説明:

Scan2Key 有効かどうかをチェックします。 Scan2Key はスキャナからのスキャン入力をキーパッドバッファに転送するので、バーコードデータはキーパッドからのキー入力の様に入力されます。

関数コール:

BOOL S2K_IsEnabled();

戻り値:

TRUE = 有効
FALSE = 無効

6.21 Scan2Key.exe のロード/アンロード

関数の説明:

Scan2Key をロード/アンロードします。

関数コール:

BOOL S2K_Load(BOOL load, DWORD timeout);

パラメータ: (入力)

load: BOOL: TRUE = Scan2Key をロード。
FALSE = Scan2Key をアンロード。
timeout: DWORD: Scan2Key をアンロードしたとき、Scan2Key がメモリから消されるか、このパラメータによって指定されたタイムアウトになるまで待ちます。

戻り値:

TRUE = ロード成功。

6.22 Scan2Key を有効または無効にする

関数の説明:

スキャンしたデータを標準のキーボードバッファに入れるために Scan2Key を有効または無効にします。Scan2Key は標準では有効です。

関数コール:

BOOL S2K_Enable(BOOL enable, DWORD timeout);

パラメータ: (入力)

enable: BOOL: TRUE = スキャンしたデータをキーパッドバッファへ入れることが可能。
FALSE = スキャンしたデータをキーボードへ入れることができない。

timeout:DWORD: Scan2Key を有効または無効にした場合、Scan2Key がメモリから消されるか、あるいはこのパラメータで指定したタイムアウトになるまで待ちます。

戻り値:

TRUE = 有効にされました。
その他は FALSE

6.23 スキャナコマンドをデコーダチップへ送る

関数の説明:

スキャナコマンドをデコーダチップに送ります。このコマンドは以下のようにバイト列をデコーダチップへ送ります。 (Esc と BCC は計算され、自動的に追加されます。)

Esc, high-length, low-length, command-ID, operation, set, BCC

セクション 7 のコマンドレファレンスを参照して下さい。
BOOL HAM_SendCommand(BYTE highlen, BYTE lowlen, BYTE cmdID, BYTE op, BYTE set);

パラメータ: (入力)

highlen: BYTE: コマンド長の上位バイト
lowlen: BYTE: コマンド長の下位バイト
cmdID: BYTE: コマンド ID
op: BYTE: このコマンドの動作モード
set: BYTE: このコマンドの動作

戻り値:

TRUE = コマンドが出力のキューに正常に送られたことを示します。

6.24 デコーダチップに一つのコマンドだけを送信

関数の説明:

コマンドをデコーダチップに送ります。これはコマンド HAM_SendCommand の一種です。以下のコマンドを 1D デコーダ に送ります: (注, BCC なし 2 バイトのみ)。
Esc, 0x80+cmd

関数コール:

BOOL HAM_SendCommand1(BYTE cmd);

パラメータ: (入力)

cmd: BYTE: コマンド

戻り値:

TRUE =コマンドが出力のキューに正常に送られたことを示します。

6.25 デコーダチップにコマンドを送信

関数の説明:

コマンドをデコーダチップに送ります。これはコマンド HAM_SendCommand の一種です。これは多数のパラメータとパケットを以下のフォーマットで読み、デコーダチップに送信します。
Esc, パラメータ 1, paramter2, …, BCC

全パラメータ数は最初のパラメータの数字で指定されます。

関数コール:

BOOL HAM_SendCommand2(BYTE num, BYTE パラメータ 1, …);

パラメータ: (入力)

num: BYTE: パラメータの合計数
パラメータ x BYTE: パラメータ

戻り値:

TRUE = コマンドが出力のキューに正常に送られたことを示します。

6.26 デコーダチップにスキャナコマンドセットを送る

関数の説明:

この関数コールは、長さがシングル WORD パラメータとタイムアウトパラメータがあることを除いて、 HAM_SendCommand と同じ機能です。これは同期された機能であり、コマンドが送信され、そしてスキャナから応答を得たときに戻ります。USI_WaitForSendEchoTO は、次の連続する送信コマンドコールの前に必要はありません。

文字列を送信するには、 HAM_SendCommand_SetString を使用します。

BOOL HAM_SendCommand_Set(WORD len, BYTE cmdID, BYTE op, BYTE set, DWORD timeout)

パラメータ(入力)

len:	BYTE	: コマンド長を指定、実際には常に 4
cmdID:	BYTE	: コマンド ID
op:	BYTE	: コマンドの動作モード
set:	BYTE	: このコマンドのオペランド
timeout:	DWORD	: タイムアウトをミリ秒で指定します

戻り値:

TRUE = 設定が正しくセットされたことを示します。

6.27 デコーダチップからスキャナコマンドを得る

関数の説明:

この関数コールは、スキャナから設定を読み出すことを除いて HAM_SendCommand_Set と似た関数です。これは同期された機能であり、コマンドが送信され、そしてスキャナから応答を得たときに戻ります。 USI_WaitForSendEchoTO は、次の連続する送信コマンドコールの前に必要はありません。

BOOL HAM_SendCommand_Get(WORD len, BYTE cmdID, BYTE op, BYTE* get, DWORD timeout)

パラメータ(入力)

len:	BYTE	: コマンド長を指定、実際には常に 4.
cmdID:	BYTE	: コマンド ID
op:	BYTE	: コマンドの動作モード
get:	BYTE*	: スキャナから読み込んだ設定を保持するバイトのポインタ
timeout:	DWORD	: タイムアウトをミリ秒で指定

戻り値:

TRUE = 設定が正しく読み出されたことを示します.

6.28 スキャナコマンドセット文字列をデコーダチップに送る

関数の説明:

この関数コールは、スキャなに 1 文字ではなく文字列を送ることを除いて HAM_SendCommand_Set と似た関数です。

BOOL HAM_SendCommand_SetString(WORD len, BYTE cmdID, BYTE op, LPCSTR sets, int slen, DWORD timeout)

パラメータ(入カ)

len:	BYTE	: コマンドの長さを指定。これは自動的に計算され調整されます。
cmdID:	BYTE	: コマンド ID
op:	BYTE	: このコマンドの動作モード
sets:	LPCSTR	: 文字列データを指定
slen:	int	: 文字列データの長さを指定。自動的に計算するには -1 をセット
timeout:	DWORD	: タイムアウトをミリ秒で指定

戻り値:

TRUE = 設定が正しく設定されたことを示します。

6.29 デコーダチップからスキャナコマンドセット文字列を得る

関数の説明:

この関数コールは、スキャナからの設定を読み出すことを除いて、HAM_SendCommand_SetString に似た関数です。

BOOL HAM_SendCommand_GetString(WORD len, BYTE cmdID, BYTE op, LPSTR gets, int* slen, DWORD timeout)

パラメータ(入カ)

len:	BYTE	: コマンドの長さを指定します。これは自動的に計算して調整されます。
cmdID:	BYTE	: コマンド ID
op:	BYTE	: このコマンドの動作モード
gets:	LPSTR	: スキャナから読み込んだ設定を保持するバッファ
slen:	int*	: バッファ長を指定し、そしてバッファの中の実際のデータ長を戻します。
timeout:	DWORD	: タイムアウトをミリ秒で指定

戻り値:

TRUE = 設定が正しく読み出されたことを示します。

6.30 スキャナ関連のバージョン情報を得る

関数の説明:

スキャナ関連のバージョン番号を得ます。この関数を使用するために USI_Register をコールする必要はありません。

関数コール:

BOOL USI_GetScannerVersion(LPTSTR model, LPTSTR firmware, LPTSTR sdk, int blen);

パラメータ(出力)

model:	LPTSTR	: スキャナモデル
firmware:	LPTSTR	: ファームウェアバージョン番号
sdk:	LPTSTR	: sdk バージョン番号、もしあれば
blen:	int	: モデル、ファームウェア、SDK のパラメータについての バッファ長を指定

戻り値:

常に True.

6.31 USI から入力要求警告メッセージを有効

関数の説明:

ポップアップウインドウに動作情報をレポートすることを USI に有効にします。

関数コール:

BOOL USI_EnablePromptMessage(BOOL enable);

パラメータ(出力)

enable:	BOOL	: True= 有効, False=無効

戻り値:

常に True.

6.32 スキャナ動作モード (2D モデルのみ有効)

関数の説明:

スキャナエンジンをバーコードデコード/イメージ/プレビュー(mode = SWM_BARCODE) またはイメージキャプチャ (mode = SWM_IMAGE) あるいは 2D スキャナのプレビューとイメージキャプチャ (mode = SWM_IMAGE_PREVIEW) の動作モードにセットします。

関数コール:

BOOL USI_SetWorkingMode(int mode);

パラメータ(出力)

mode: int : mode = SWM_BARCODE – バーコード
mode = SWM_IMAGE – イメージキャプチャ
mode = SWM_IMAGE_PREVIEW – 2D スキャナのプレビューと
イメージキャプチャ

戻り値:

常に True.

6.33 イメージを得る (2D モデルのみ有効)

関数の説明:

ビットマップフォーマットでキャプチャしたイメージを得て、イメージサイズを戻します。

関数コール:

HBITMAP USI_GetImageBitmap(SIZE* imagesize);

パラメータ(出力)

imagesize:	SIZE	: ビットマップイメージサイズ

戻り値:

HBITMAP	: イメージ

6.34 イメージのサイズ変更 (2D モデルのみ有効)

関数の説明:

ビットマップのサイズを変更します。

関数コール:

HBITMAP USI_ResizeBitmap(HBITMAP hbmp, int cx, int cy, BOOL keepratio);

パラメータ(入力)

hbmp:	HBITMAP	: サイズ変更に必要なビットマップハンドル
cx:	int:	: cx は、ビットマップの新しい幅を定義します
cy	int:	: cy は、ビットマップの新しい高さを定義します
keepratio:	BOOL:	: keepratio が True の場合、bmp~cx-cy の形に合わせ、元のイメージの比を保持します。

戻り値:

HBITMAP	: イメージ

6.35 ファイルにイメージを保存 (2D モデルのみ有効)

関数の説明:

キャプチャしたイメージをファイルに保存します。イメージフォーマットは、ファイル拡張子名で定義されます。拡張子は.bmp, .jpg, .jpeg, .tif, .tiff または .raw です。 compressionfactor (圧縮係数)は、 jpeg フォーマットで使用されます。

関数コール:

BOOL USI_SaveImageToFile(LPCTSTR filename, int compressionfactor);

パラメータ(入力)

filename:	LPCTSTR	: ファイル拡張子名がフォーマットを定義、 HHP, SSI は .bmp, .jpg, と .tif をサポート。Tohken は、 .bmp と .jpg をサポート。
compressionfactor:	int	: compressionfactor は、フォーマットが jpg の時に

使用。

戻り値:

常に True.

6.36 ターミネータを得る

関数の説明:

ターミネータを戻します。戻される値は USI_SetTerminator で定義されています。

関数コール:

int USI_GetTerminator();

戻り値:

Int ：ターミネータ

6.37 ターミネータをセットする

関数の説明:

ターミネータをセットします。

関数コール:

BOOL USI_SetTerminator(int terminator);

パラメータ(入力)

terminator:	int	: TERMINATOR_ENTER	: Enter (CR/LF)
TERMINATOR_RETURN		: Return (CR)	
TERMINATOR_LINEFEED		: LineFeed (LF)	
TERMINATOR_NONE		: ターミネータなし	
TERMINATOR_ENTERENTER		: Enter (CR/CR)ふたつ	

戻り値:

常に True.

6.38 読み取り終了サウンドモードとサウンド名を得る

関数の説明:

読み取り終了のサウンドモードとサウンドファイル名を戻します。

関数コール:

int USI_GetGoodReadEcho(LPTSTR buffer, UINT blen);

パラメータ(入力)

buffer:	LPTSTR	: Path を含むサウンドファイル名
blen:	UINT	: バッファ長を定義

戻り値:

Int:	GRE_PLAYSOUND	: プリセットしたサウンドファイルで再生
	GRE_BEEP	: 標準のビープ音を再生
	GRE_NONE	: サウンドなし

6.39 読み取り終了サウンドモードとサウンド名をセット

関数の説明:

読み取り終了のサウンドモードとサウンドファイル名をセットします。

関数コール:

BOOL USI_SetGoodReadEcho(int mode, LPTSTR SoundFileName);

パラメータ(入力)

mode:	int	: GRE_PLAYSOUND : プリセットしたサウンドファイルで再生
GRE_BEEP		: 標準のビープ音を再生
GRE_NONE		: サウンドなし
buffer:	LPTSTR	: Path を含むサウンドファイル名

戻り値:

成功時は True を戻し、モードが無い場合は False を戻します。

6.40 プレビューサイズをセット (2D エンジンのみ有効)

関数の説明:

イメージプレビューのウインドウの大きさを定義します。

関数コール:

void USI_SetPreviewSize (SIZE size);

パラメータ(入力)

size:	SIZE	: イメージプレビューウインドウのサイズ

6.41 プレビューサイズタイムアウトをセット (2D エンジンのみ有効)

関数の説明:

プレビューのタイムアウトを秒単位でセットします。イメージプレビューモードの場合、このタイムアウトは、プレビューを終了しそしてイメージのキャプチャをトリガーします。

関数コール:

void USI_SetPreviewTimeout (DWORD timeout);

パラメータ(入力)

timeout: DWORD ：タイムアウト、秒

6.42 PA966/PA967 の 2D イメージャサポート

PA966/PA967 の 2D サポート API は別の説明書を用意してあります。以下より入手して下さい。
http://w3.tw.ute.com/pub/cs/manual/WinCE_programming_manual/2D_Engine_SDK.pdf

7. デコーダチップのコントロールコマンド(1D デコーダ のみ)

重要: **本章はスキャナコントロール機能の低レベルコマンドについて説明しています。スキャナプログラミングに USI.DLL を使用している場合、本章をお読みになる必要はありません。一般に、スキャナのコントロールに低レベルコマンドを使用することはお勧めいたしません。これはシリアル通信のプログラムでタイミングの問題があり、通信に詳しいことが求められ、また本書では十分には説明しきれないからです。**

ホストがコマンドを 1D デコーダに送る準備をする場合、最初に CTS をチェックしなければなりません。CTS が high の場合、ホストは RTS を high にし、そして 1D デコーダ を起動するために RTS をクリアして low にしなければなりません。

制御のための特殊コマンド		
コマンド	フォーマット	コメント
コントロール	Esc,80H+SOH(01H)	1D デコーダ をスレーブ状態にします。この状態で、1D デコーダ はコマンドを受信し、リリースコマンドを受信するか、タイムアウト(約 10 秒)になるまでこのコマンドを実行します。他の状態では、タイムアウトはコマンド間隔と同じ約 1 秒です。
リリース	Esc,80H+EOT(04H)	1D デコーダ をスレーブ状態から終了させます。
実行/ 照会	Esc,80H+ENQ(05H)	1D デコーダ に以前に保存したコマンドを実行させ、そしてホストに送信するための以前に実行したコマンドの結果があるか 1D デコーダ をチェックします。以前に保存したコマンドがすでに実行され、そして送る結果がない場合、1D デコーダ は結果が出るまで応答しません。ホストが結果を受信したが、BCC が間違っている場合、結果を再送するために ENQ を再送信することができます。
ACK	Esc,80H+ACK(06H)	1D デコーダ からホスト。1D デコーダ がコマンドを受信し、このコマンドがメッセージを返す必要がない場合、1D デコーダ は ACK を応答します。
NAK	Esc,80H+NAK(15H)	1D デコーダ からホスト。1D デコーダ はホストがコマンドを再送信する必要があります。通常、間違った BCC を受信した場合、NAK を送信することができます。1D デコーダ は意味のないコマンドを受信した場合はいつでも NAK を返します。

ホストから 1D デコーダ へのコマンド

コマンドフォーマット:Esc,Lh,Ll,n,m,S1,...,Si,BCC

ここで: Esc はエスケープコード(H'1B)

Lh/Ll は Lh.b7 がゼロの場合コマンド長、Lh は上位バイト、Ll は下位バイト、 n から BCC をカウントします。 Lh.b7=1 が 1 の場合、これは二バイトの特殊コマンドです。

n はコマンド ID です。

m は演算子です: 設定コマンドについては通常、 0 は設定、1 は標準値、2 は現在の設定を読む、3 は特殊動作を意味します。 m=1 または 2 の場合、 S1 はビットまたは 1 文字設定のために 0 でなければなりません。Pre_amble(プリアンブル)の様に設定が文字列の場合、読み取りもしくは標準値コマンドは Si バイトを含んではいけません。コマンドの特別な意味についてはコマンド定義をご覧下さい。

Si は、設定/データ読み込みです。

BCC: BCC の前までの全バイトの XOR です。

表記: S1.bj はバイト S1 の j ビット目を意味します。

1~64:2 の表記は数字が 1 と 64 の間にあり、標準値は 2 であることを意味します。

注: 送信するコマンドの間隔は 1 秒を越えてはいけません。

コマンド	フォーマット	コメント
Initial/ Warm start	Esc,0,2,0,BCC	1D デコーダ は RAM の設定に従ってポートとフラグを初期化します。
Default	Esc,0,2,1,BCC	RAM の設定をリセットして初期化します。
Mpu_idle	Esc,0,4,2,m,S1,BCC	S1 は 0~3:0 はスリープモード、1 はウオッチモード、2 はスタンバイモード。
Beep	Esc,0,4,3,m,S1,BCC	S1 0 なし、1 低、2 中、3 高、4 低/高、5 高/低
block_delay	Esc,0,4,4,m,S1,BCC	S1 は、 0 10ms、1 50ms、2 100ms、3 500ms、4 1s、5 3s
char_delay	Esc,0,4,5,m,S1,BCC	S1 は、 0 なし、1 1ms、2 5ms、3 10ms、4 20ms、5 50ms
Function_code	Esc,0,4,6,m,S1,BCC 内部使用	S1 は、 0 オフ、1 オン(標準値)
Capslock	Esc,0,4,7,m,S1,BCC 内部使用	S1 は、 0 自動トレース(標準値)、1 小文字、2 大文字
Language	Esc,0,4,8,m,S1,BCC 内部使用	S1 は、 0 米語(標準値)、1 英語、2 スイス語、3 スエーデン語、4 スペイン語、5 ノルウェイ語、6 イタリア語、7 ドイツ語,8 フランス語、9 Alt キーモード、A デンマーク語
Baud_rate	Esc,0,4,0D,m,S1,BCC 内部使用	S1 は 0 300、1 600、2 1200、3 2400、4 4800、5 9600、6 19200、7 38400(標準値)
Parity	Esc,0,4,0E,m,S1,BCC 内部使用	S1 は 0 EVEN、1 ODD、2 MARK、3 SPACE、4 NONE(標準値)

Data_bits	Esc,0,4,0F,m,S1,BCC 内部使用	S1 は 0 7、1 8BIT(標準値)
Handshake	Esc,0,4,10,m,S1,BCC 内部使用	S1 は 0 無視(標準値)、1 電源オン時に RTS 有効、2 通信時に RTS 有効
Ack_nak	Esc,0,4,11,m,S1,BCC 内部使用	S1 は 0 OFF(標準値)、1 ON
BCC_char	Esc,0,4,12,m,S1,BCC 内部使用	S1 は 0 OFF(標準値)、1 ON
Data_direction	Esc,0,4,13,m,S1,BCC 内部使用	S1 は =0 ホストへ送信(標準値)、1 ホストとターミナルへ送信 2 ターミナルへ送信
Time_out	Esc,0,4,14,m,S1,BCC 内部使用	S1 は 0 1S(標準値)、1 3S、2 10S、3 制限なし
Terminator	Esc,0,4,15,m,S1,BCC	S1 は B1B0=0 ENTER(CR/LF) (標準値)、1 FIELD EXIT(CR)、2 RETURN(LF)、3 なし
Code_id	Esc,0,4,16,m,S1,BCC	S1 は 0 OFF(標準値)、1 ON
Verification	Esc,0,4,17,m,S1,BCC	S1 は 0: OFF(標準値)、1~7 1 から 7 回確認
Scan_mode	Esc,0,4,18,m,S1,BCC	S1 は 0: トリガモード(標準値)、1: フラッシュモード、2 マルチスキャンモード、3: ワンプレス・ワンスキャン、4~7 予約
Label_type	Esc,0,4,19,m,S1,BCC	S1 は 0: 正(標準値)、1: 正と負
Aim_fuction	Esc,0,4,1a,m,S1,BCC	S1 は 0: 無効(標準値)、1: 有効
Scan_pre_data	Esc,0,L,1b,m,S1,…Si,BCC	Si は 1 から 8 文字
Scan_post_data	Esc,0,L,1c,m,S1,…Si,BCC	Si は 1 から 8 文字
Define_code39f	Esc,0,4,1d,m,S1,BCC	Code 39 フル ASCII ID 定義:ここで S1 は 1 文字
Define_code39s	Esc,0,4,1e,m,S1,BCC	Code 39 標準 ID 定義:ここで S1 は 1 文字
Define_EAN13	Esc,0,4,1f,m,S1,BCC	EAN13 ID 定義:ここで S1 は 1 文字
Define_UPCA	Esc,0,4,20,m,S1,BCC	UPC A ID 定義: ここで S1 は 1 文字
Define_EAN8	Esc,0,4,21,m,S1,BCC	EAN8 ID 定義: ここで S1 は 1 文字
Define_UPCE	Esc,0,4,22,m,S1,BCC	UPC E ID 定義: ここで S1 は 1 文字

Define_I25	Esc,0,4,23,m,S1,BCC	I25 ID 定義: ここで S1 は 1 文字
Define_CDB	Esc,0,4,24,m,S1,BCC	Codabar ID 定義: ここで S1 は 1 文字
Define_C128	Esc,0,4,25,m,S1,BCC	Code128 ID 定義: ここで S1 は 1 文字
Define_C93	Esc,0,4,26,m,S1,BCC	Code93 ID 定義: ここで S1 は 1 文字
Define_S25	Esc,0,4,27,m,S1,BCC	S25 ID 定義: ここで S1 は 1 文字
Define_MSI	Esc,0,4,28,m,S1,BCC	MSI ID 定義: ここで S1 は 1 文字
Define_C11	Esc,0,4,29,m,S1,BCC	Code11 ID 定義: ここで S1 は 1 文字
Define_C32	Esc,0,4,2a,m,S1,BCC	Code32 ID 定義: ここで S1 は 1 文字
Define_DELTA	Esc,0,4,2b,m,S1,BCC	Delta ID 定義: ここで S1 は 1 文字
Define_LABEL	Esc,0,4,2c,m,S1,BCC	Label code ID 定義: ここで S1 は 1 文字
Define_PLESSEY	Esc,0,4,2d,m,S1,BCC	Plessey ID 定義: ここで S1 は 1 文字
Define_TELEPEN	Esc,0,4,2e,m,S1,BCC	Telepen ID 定義: ここで S1 は 1 文字
Define_TOSHIBA/Matrix 25, China Postal	Esc,0,4,2f,m,S1,BCC	Toshiba ID 定義: ここで S1 は 1 文字
Define_EAN128	Esc,0,4,30,m,S1,BCC	EAN128 ID 定義: ここで S1 は 1 文字;H'FF の場合、"]C1"を使用
Mterminator	Esc,0,4,31,m,S1,BCC 内部使用	ここで S1 は 0: ENTER(標準値)、1: なし
Sentinal	Esc,0,4,32,m,S1,BCC 内部使用	S1 は 0: 送信しない、1: 送信
Track_selection	Esc,0,4,33,m,S1,BCC 内部使用	ここで S1 は =0: 全トラック(標準値)、1: トラック 1 とトラック 2、2: トラック 1 とトラック 3、3: トラック 2 と トラック 3、4: トラック 1、5: トラック 2、6: トラック 3
T2_account_only	Esc,0,4,34,m,S1,BCC 内部使用	S1 は 0:NO(標準値)、1: YES
Separator	Esc,0,4,35,m,S1,BCC 内部使用	S1 は 1 文字

Must_have_data	Esc,0,4,36,m,S1,BCC 内部使用	S1 は 0: YES、1:NO(標準値)
Track1_sequence	Esc,0,L,37,m,S1,…Si,BCC 内部使用	Si は 1 から 16 文字
Track2_sequence	Esc,0,L,38,m,S1,…Si,BCC 内部使用	Si は 1 から 8 文字
Code39_set	Esc,0,4,39,m,S1,BCC	S1.B0 は Code39_enable、S1.B1 は Code39_standard、S1.B3B2 は Code39_cd、S1.B4 Code39_ss
Code39_enable	Esc,0,4,3a,m,S1,BCC	S1 は 0 無効、1 有効(標準値)
Code39_sandard	Esc,0,4,3b,m,S1,BCC	S1 は 0 フル ASCII(標準値)、1 標準
Code39_cd:	Esc,0,4,3c,m,S1,BCC	S1 は 0 計算と送信、1 計算と送信せず、2 計算せず(標準値)
Code39_ss	Esc,0,4,3d,m,S1,BCC	ここで S1 は 0 SS 送信、1 SS 送信せず(標準値)
Code39_min	Esc,0,4,3e,m,S1,BCC	S1 は 0~48:0 (min<=データ長)
Code39_max	Esc,0,4,3f,m,S1,BCC	S1 は 0~48:48 (データ長<=max)
I2of5_set	Esc,0,4,40,m,S1,BCC	S1 は、 S1.B0 は I2of5_enable、S1.B1 は I2of5_fixlength、S1.B3B2 は I2of5_cd、S1.B5B4 は I2of5_ss
I2of5_enable	Esc,0,4,41,m,S1,BCC	S1 は =0 無効、1 有効(標準値)
I2of5_fixlength	Esc,0,4,42,m,S1,BCC	S1 は =0 オン、1 オフ (最初の 3 レコード長を記録) (標準値)
I2of5_cd	Esc,0,4,43,m,S1,BCC	S1 は =0 計算と送信、1 計算と送信せず、2 計算しない
I2of5_ss	Esc,0,4,44,m,S1,BCC	S1 は 0 最初の桁サプレス、1 最後の桁サプレス、2 サプレスしない(標準値)
I25_min	Esc,0,4,45,m,S1,BCC	S1 は 2~64:10 (min<= データ長)
I25_max	Esc,0,4,46,m,S1,BCC	S1 は 2~64:64 (データ長 <=max)
S2of5_set	Esc,0,4,47,m,S1,BCC	S1 は 、S1.b0 は S2of5_enable、S1.b1 は S2of5_fixlength、S1.b3b2 は S2of5_cd
S2of5_enable	Esc,0,4,48,m,S1,BCC	S1 は 0 無効(標準値)、1 有効
S2of5_fixlength	Esc,0,4,49,m,S1,BCC	S1 は 0 オン(標準値)、1 オフ (最初の 3 レコード長を記録)

S2of5_cd	Esc,0,4,4a,m,S1,BCC	S1 is 0 計算して送信、1 計算するが送信しない、2 計算しない(標準値)
S25_min	Esc,0,4,4b,m,S1,BCC	S1 は 1~48:4 (min<= データ長)
S25_max	Esc,0,4,4c,m,S1,BCC	S1 は 1~48:48 (データ長 <=max)
Code32_set	Esc,0,4,4d,m,S1,BCC	S1 は 、S1.b0 は Code32_enable、S1.b1 は Code32_sc、S1.b2 は Code32_lc
Code32_enable	Esc,0,4,4e,m,S1,BCC	S1 は 0 無効、1 有効
Code32_sc	Esc,0,4,4f,m,S1,BCC	S1 は 0 先頭文字送信、1 送信しない
Code32_lc	Esc,0,4,50,m,S1,BCC	S1 は 0 末尾文字送信、1 送信しない
Telepen	Esc,0,4,51,m,S1,BCC	S1 は、S1.b0 は Telepen_enable、S1.b1 は Telepen_charset
Telepen_enable	Esc,0,4,52,m,S1,BCC	S1 は 0 無効(標準値)、1 有効
Telepen_charset	Esc,0,4,53,m,S1,BCC	S1 は 0 標準(標準値)、1 数字
Ean128	Esc,0,4,54,m,S1,BCC	S1 は、 S1.b0 は Ean128_id、 S1.b1 は Ean128_id
Ean128_enable	Esc,0,4,55,m,S1,BCC	S1 は 0 無効、1 有効
Ean128_id	Esc,0,4,56,m,S1,BCC	S1 は 0 ID 無効、1 ID 有効(標準値)
Ean128_func1	Esc,0,4,57,m,S1,BCC	S1 は 1 文字
Code128	Esc,0,4,58,m,S1,BCC	S1 は 0 無効、1 有効(標準値)
Code128_min	Esc,0,4,59,m,S1,BCC	S1 は 1~64:1 (min<=データ長)
Code128_max	Esc,0,4,5a,m,S1,BCC	S1 は 1~64:64 (データ長<=max)
Msi_pleasey	Esc,0,4,5b,m,S1,BCC	S1 は、 S1.b0 は Msi_p_enable、S1.b1 は Msi_pleasey_cd、 S1.b3b2 は Msi_p_cdmode
Msi_p_enable	Esc,0,4,5c,m,S1,BCC	S1 は 0 無効(標準値)、1 有効
Msi_pleasey_cd	Esc,0,4,5d,m,S1,BCC	S1 は 0 チェックデジット送信、1 送信せず(標準値)
Msi_p_cdmode	Esc,0,4,5e,m,S1,BCC	S1 は 0 チェックデジット・ダブルモジュール 10、1 チェックデジット・モジュール 11 プラス 10、2 チェックデジット・シングルモジュール 10
Msi_pleasey_min	Esc,0,4,5f,m,S1,BCC	S1 は 1~64:1 (min<=データ長)

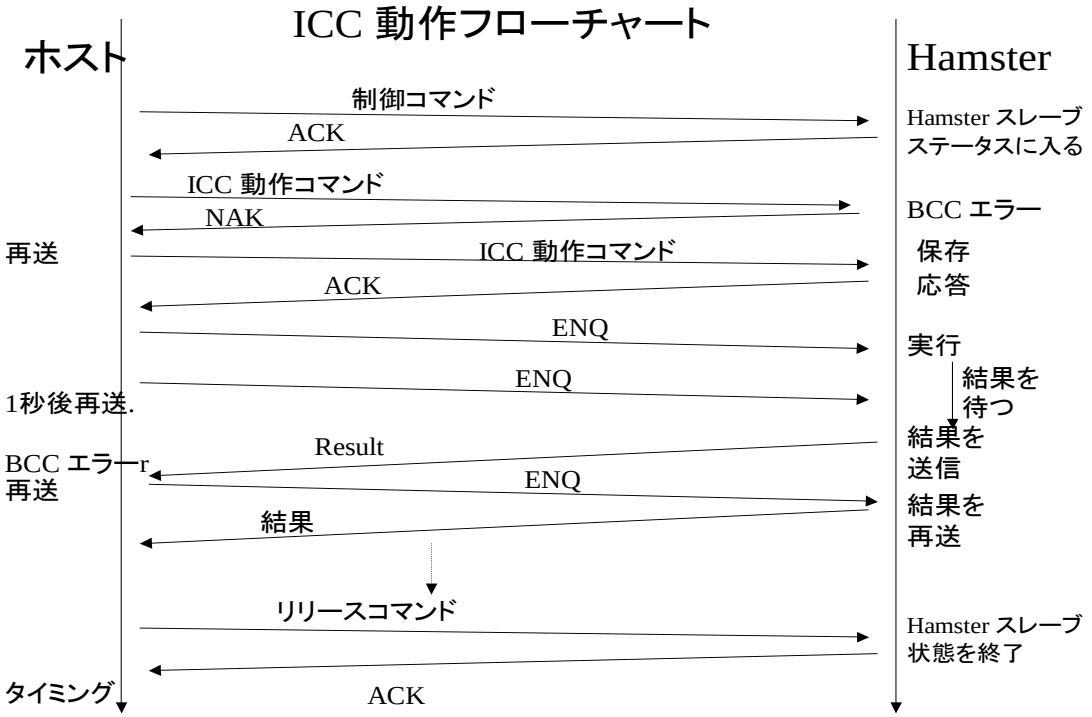
Msi_pleasey_max	Esc,0,4,60,m,S1,BCC	S1 は 1~64:64 (データ長<=max)
Code93	Esc,0,4,61,m,S1,BCC	S1 は 0 無効、1 有効(標準値)
Code93_min	Esc,0,4,62,m,S1,BCC	S1 は 1~48:1 (min<=データ長)
Code93_max	Esc,0,4,63,m,S1,BCC	S1 1~48:48 (データ長<=max)
Code11	Esc,0,4,64,m,S1,BCC	S1 は、S1.b0 は Code11_enable、S1.b1 は Code11_cdnumber、S1.b2 は Code11_cdsend
Code11_enable	Esc,0,4,65,m,S1,BCC	S1 は 0 無効(標準値)、 1 有効
Code11_cdnumber	Esc,0,4,66,m,S1,BCC	S1 は 0 1 チェックデジット、1 2 チェックデジット(標準値)
Code11_cdsend	Esc,0,4,67,m,S1,BCC	S1 は 0 チェックデジット送信、1 送信せず(標準値)
Code11_min	Esc,0,4,68,m,S1,BCC	S1 は 1~48:1 (min<=データ長)
Code11_max	Esc,0,4,69,m,S1,BCC	S1 は 1~48:48 (データ長<=max)
Codabar_set	Esc,0,4,6a,m,S1,BCC	S1 は、 S1.b0 は Codabar_enable、 S1.b1 は Codabar_ss、 S1.b3b2 は Codabar_cd、 S1.b4 は Codabar_CLSI
Codabar_enable	Esc,0,4,6b,m,S1,BCC	S1 は 0 無効(標準値)、1 有効
Codabar_ss	Esc,0,4,6c,m,S1,BCC	S1 は 0 スタートとストップ文字送信、1 送信しない(標準値)
Codabar_cd	Esc,0,4,6d,m,S1,BCC	S1 は 0 チェックデジット計算と送信、1 チェックデジット計算・送信せず、2 チェックデジット計算しない(標準値)
Codabar_CLSI	Esc,0,4,6e,m,S1,BCC	S1 は 0 CLSI フォーマットオン、1 オフ(標準値)
Codabar_min	Esc,0,4,6f,m,S1,BCC	S1 は 3~48:3 (min<=データ長)
Codabar_max	Esc,0,4,70,m,S1,BCC	S1 は 3~48:48
Label_code	Esc,0,4,71,m,S1,BCC	S1 は、 S1.b0 は Label_c_enable、S1.b1 は Label_code_cd
Label_c_enable	Esc,0,4,72,m,S1,BCC	S1 は 0 無効(標準値)、1 有効
Label_code_cd	Esc,0,4,73,m,S1,BCC	S1 は 0 チェックデジット送信、1 送信しない

Upc_a_set	Esc,0,4,74,m,S1,BCC	S1 は、S1.b0 は Upc_a_enable、S1.b1 は Upc_a_ld、S1.b2 は Upc_a_cd
Upc_a_enable	Esc,0,4,75,m,S1,BCC	S1 は 0 無効、1 有効(標準値)
Upc_a_ld	Esc,0,4,76,m,S1,BCC	S1 は 0 先頭桁送信(標準値)、1 送信しない
Upc_a_cd	Esc,0,4,77,m,S1,BCC	S1 は 0 チェックデジット送信(標準値)、1 送信しない
Upc_e_set	Esc,0,4,78,m,S1,BCC	S1 は、 S1.b1 は Upc_e_enable、S1.b2 は Upc_e_ld、S1.b3 は Upc_e_cd、S1.b4 は Upc_e_expand、S1.b0 は Upc_e_nsc
Upc_e_enable	Esc,0,4,79,m,S1,BCC	S1 は 0 無効、1 有効(標準値)
Upc_e_ld	Esc,0,4,7a,m,S1,BCC	S1 は 0 先頭桁送信(標準値)、1 送信せず
Upc_e_cd	Esc,0,4,7b,m,S1,BCC	S1 は 0 チェックデジット送信、1 送信せず(標準値)
Upc_e_expand	Esc,0,4,7c,m,S1,BCC	S1 は 0 ゼロ拡張オン、1 オフ(標準値)
Upc_e_nsc	Esc,0,4,7d,m,S1,BCC	S1 は 0 無効(標準値)、1 有効
Ean_13_set	Esc,0,4,7e,m,S1,BCC	S1 は、 S1.b0 は Ean_13_enable、S1.b1 は Ean_13_ld、S1.b2 は Ean_13_cd、S1.b3 は Ean_13_bookland
Ean_13_enable	Esc,0,4,7f,m,S1,BCC	S1 は 0 無効、1 有効(標準値)
Ean_13_ld	Esc,0,4,80,m,S1,BCC	S1 は 0 先頭桁送信(標準値)、1 送信せず
Ean_13_cd	Esc,0,4,81,m,S1,BCC	S1 は 0 チェックデジット送信(標準値)、送信せず
Ean_13_bookland	Esc,0,4,82,m,S1,BCC	S1 は 0 bookland EAN 有効、1 無効(標準値)
Ean_8_set	Esc,0,4,83,m,S1,BCC	S1 は、S1.b0 は Ean_8_enable、S1.b1 は Ean_8_ld、S1.b2 は Ean_8_cd
Ean_8_enable	Esc,0,4,84,m,S1,BCC	S1 は 0 無効、1 有効(標準値)
Ean_8_ld	Esc,0,4,85,m,S1,BCC	S1 は 0 先頭桁送信(標準値)、1 送信せず
Ean_8_cd	Esc,0,4,86,m,S1,BCC	S1 は 0 チェックデジット送信(標準値)、1 送信せず
Supplement_set	Esc,0,4,87,m,S1,BCC	S1 は、S1.b0 は Supplement_two、 s1.b1 は Supplement_five、S1.b2 は Supplement_mh、 S1.b3 は Supplement_ssi

Supplement_two	Esc,0,4,88,m,S1,BCC	S1 は 0 オフ(標準値)、1 オン
Supplement_five	Esc,0,4,89,m,S1,BCC	S1 は 0 オフ(標準値)、1
Supplement_mh	Esc,0,4,8a,m,S1,BCC	S1 は 0 あれば送信(標準値)、1 なければならない
Supplement_ssi	Esc,0,4,8b,m,S1,BCC	S1 は 0 スペースが挿入された、 1 スペースが挿入されていない(標準値)
Delta_code_set	Esc,0,4,8c,m,S1,BCC	S1 は、S1.b0 は Delta_c_enable、S1.b1 は Delta_code_cdc、S1.b2 は Delta_code_cds
Delta_c_enable	Esc,0,4,8d,m,S1,BCC	S1 は 0 無効(標準値)、1 有効
Delta_code_cdc	Esc,0,4,8e,m,S1,BCC	S1 は 0 チェックデジット計算(標準値)、1 計算しない
Delta_code_cds	Esc,0,4,8f,m,S1,BCC	S1 は =0 チェックデジット送信、1 送信しない(標準値)
Get_version	Esc,0,3,90,2,BCC	ファームウェアバージョンを得る
DumpSetting	Esc,Lh,Ll,91,m,S1...Si,BCC	Lh/Ll はコマンド長。Si は s1 から S255 の範囲。m=0 はダウンロード設定、 m=1 は設定エリアを FF にリセット。m=2 はアップロード設定。 実際には以下のフォーマットが必要: ダウンロード: Esc,1,02,91,0,s1,...,s255,BCC アップロード: Esc,0,3,91,2,BCC
EAN128BraceRemove	Esc,0,4,92,m,S1,BCC	S1 は =0 無効(標準値)、1 有効(brace 消去)
AimingTime	Esc,0,4,93,m,S1,BCC	S1 は =0 0.5s、1 1s(標準値)、2 1.5s、 3 2s
Exchange data	Esc,Lh,Ll,a3,S1,S2,...,Sn, BCC	• ACK を予期 (Esc,80H+ACK(06H)) • ホストと ICC 間のデータを交換 • Execute/Enquiry コマンド送出後に予期される戻り: Esc,Lh,Ll,0xa3,AH,data,BCC ここで: AH=0 成功 =1 タイムアウト =2 カードがない データ: 応答データとステータスワード

注: 1D デコーダ はこれらのコマンドをバッファに保存し、Execute コマンド (Esc,ENQ)を受信するまで実行しません。 1D デコーダ は " Esc,ENQ" を受信後にコマンドを実行し、そして応答を返します。データの最大長は 264 です。 m と応答は以下の様に定義されます。

1D デコーダからホストへのデータ					
データフォーマット: Code_number,Lh,Ll,string ここで: Lh/Ll はストリング長、 Lh は上位バイト、 Ll は下位バイト。ストリング長は Code_number と Lh/Ll に含まれません。ストリングは Code ID、pre_amble、scanned data、post_amble、そしてターミネータを含みます。 Code_number は以下の数に H’ 80 を加えたものです。					
0 Code 39 full ASCII	1 Code 39 standard or EDP Code			2 EAN 13	3 UPC A
4 EAN 8	5 UPC E	6 I25	7 Codabar	8 Code 128	9 Code 93
10 S25	11 MSI	12 EAN 128	13 Code 32	14 Delta	15 Label
16 Plessey	17 Code 11	18 Toshiba	19 reserved	20 Track 1	21 Track 2
22 Track 3	23 More than 1 track	24 reserved	25 RS232	26 reserved	27 reserved
28 reserved	29 reserved	30 reserved	31 reserved	32 reserved	33 reserved



注: ホストはACKまたは結果を受信しない場合1秒後にコマンドを再送します。

8. SysIOAPI.DLL

このDLLはユーザがスキャナ、LED、バックライトおよびPCカードスロットを制御するためのハードウェアに関するAPIを提供しています。API関数はアプリケーションを書くプログラマにDLLを通して提供されます。二つのファイルSysIOAPI.LIBとSysIOAPI.HがSDKにあります。
http://w3.tw.ute.com/pub/cs/software/Sample_Program/SysIOAPI/SysIOAPISample.zip

注：プログラミングについては、Unitechのビルトイン DLLを使用して動的に DLLを読み込む必要があります。（Unitechは、Windows Mobile OS用に *.H と *.LIB を提供しておりません。プログラミングについては、第9章をご覧下さい。

注: すべてのデバイスに対してこれらの関数がサポートされているわけではありません。

8.1 キーパッド関連関数

8.1.1 電源ボタンを有効または無効にする

関数の説明:

この関数は電源ボタンを有効または無効にします。

関数コール:

VOID DisablePowerButton(BOOL)

パラメータ(入力):

True = 電源ボタンを無効にします。
False = 電源ボタンを有効にします。

戻りコード:

なし

8.1.2 キーパッドユーティリティ入力モードをセット

関数の説明:

ターミナルにはGetVKと呼ぶフル英文字入力をエミュレートするユーティリティがあります。入力モードは「alpha」キーを押すか、以下の関数によって切り替わります。

関数コール:

Void SetGetVKWorkingMode(int)

パラメータ(入力):

0 = 選択ウインドウを隠す
1 = 小文字選択ウインドウを表示
2 = 大文字選択ウインドウを表示

戻りコード:

なし

8.1.3 キーパッドユーティリティ入力モードを得る

関数の説明:

この関数は Alpha キー入力モードをチェックするために使用します。

関数コール:

BYTE GetAlphaKeyWorkingMode(void);

戻りコード:

0 = 通常
1 = 小文字
2 = 大文字

8.1.4 Alpha キーが押されたかどうかをチェック

関数の説明:

この関数は Alpha キーが押されたかどうかをチェックするために使用されます。

関数コール:

BOOL GetKeypadAlphaKeyStatus(void);

戻りコード:

TRUE = Alpha キーが押された。
FALSE = Alpha キーが解放された。

8.1.5 Alpha Key を有効/無効にする

関数の説明:

この関数は、Alpha キーを有効/無効にするために使用されます。

関数コール:

void SetAlphaKeyDisable (BOOL bDisable)

パラメータ(入力):

TRUE = Alpha キー無効
FALSE = Alpha キー有効

8.1.6 Alpha Key ステータスをチェック

関数の説明:

この関数は、Alpha キーが有効かどうかをチェックするために使用されます。

関数コール:

BOOL GetAlphaKeyStatus (void)

戻りコード:

TRUE = Alpha キーが有効
FALSE = Alpha キーが無効

8.1.7 ファンクションキーのステータスをチェックする

関数の説明:

この関数は、ファンクションキーが有効かどうかをチェックするために使用されます。

関数コール:

BOOL GetFnKeyStatus(void);

戻りコード:

TRUE = ファンクションキーは有効 。
FALSE = ファンクションキーは無効。

8.1.8 ファンクションキーを有効/無効にする

関数の説明:

この関数は、ファンクションキーを有効/無効にするために使用されます。

関数コール:

void SetFnKeyDisable(BOOL bOff);

戻りコード:

TRUE = ファンクションキー無効 。
FALSE = ファンクションキー有効。

8.1.9 ファンクションモードをセット

関数の説明:

この関数は、キーパッドをファンクション(機能)モードにするか、しないかをセットするために使用されます。

関数コール:

void SetFnKeyWorkingMode (BOOL bEnable)

パラメータ(入力):

TRUE = キーパッドをファンクションモードにセット
FALSE = キーパッドを通常モードにセット

8.1.10　ファンクションモードを得る

関数の説明:

この関数は、キーパッドがファンクション(機能)モードかどうかを得るために使用されます。

関数コール:

BOOL GetFnKeyWorkingMode ()

戻りコード:

TRUE = キーパッドはファンクションモード
FALSE = キーパッドは通常モード

8.1.11　ファンクションキーステータスを検出する

関数の説明:

この関数は、ファンクションキーが押されたかどうかを得るために使用されます。

関数コール:

BOOL GetKeypadFnKeyStatus ()

戻りコード:

TRUE = ファンクションキーが押された
FALSE = ファンクションキーが離された

8.1.12　Start キーステータスをチェック

関数の説明:

この関数は、Start キーが有効かどうかをチェックするために使用されます。

関数コール:

BOOL GetStartKeyStatus (void)

戻りコード:

TRUE = Start キーが有効
FALSE = Start キーが無効

8.1.13　Start キーを有効/無効にする

関数の説明:

この関数は、Start キーを有効/無効にするために使用されます。

関数コール:

void SetStartKeyDisable (BOOL bDisable)

パラメータ(入力):

TRUE = Start キーを無効にする
FALSE = Start キーを有効にする

8.1.14　Talk キーステータスをチェック

関数の説明:

この関数は、Talk キーが有効かどうかをチェックするために使用されます。

関数コール:

BOOL GetPhoneTalkKeyStatus (void)

戻りコード:

TRUE = Talk キーは有効
FALSE = Talk キーは無効

8.1.15　Talk キーを有効/無効にする

関数の説明:

この関数は、Talk キーを有効/無効にするために使用されます。

関数コール:

void SetPhoneTalkKeyDisable(BOOL bDisable)

パラメータ(入力):

TRUE = Talk キーを無効にする
FALSE = Talk キーを有効にする

8.1.16　Phone End キーステータスをチェック

関数の説明:

この関数は、Phone End キーが有効かどうかをチェックするために使用されます。

関数コール:

BOOL GetPhoneEndKeyStatus (void)

戻りコード:

TRUE = Phone End キーは有効
FALSE = Phone End キーは無効

8.1.17　Phone End キーを有効/無効にする

関数の説明:

この関数は、Phone End キーを有効/無効にするために使用されます。

関数コール:

void SetPhoneEndKeyDisable (BOOL bDisable)

パラメータ(入力):

TRUE = Phone End キーを無効にする
FALSE = Phone End キーを有効にする

8.1.18　キーパッドステータスをチェック

関数の説明:

この関数は、キーパッドがロックされているかどうかをチェックするために使用されます。

関数コール:

BOOL GetKeypadLockStatus (void)

戻りコード:

TRUE = キーパッドはロックされている
FALSE = キーパッドはロックされていない

8.1.19　キーパッドをロック/アンロックする

関数の説明:

この関数は、キーパッドをロック/アンロックするために使用されます。

関数コール:

void SetKeypadLock (BOOL bLock)

パラメータ(入力):

TRUE = キーパッドをロックする
FALSE = キーパッドをアンロックする

8.2 スキャナ関連関数

電源を節約するために、デコーダ IC はスキャナが使用されないときは無効になっています。これは USI 関数で有効にすることができます。以下の関数はデコーダ関数が有効な場合にのみ意味があります。

8.2.1 スキャナトリガキーを有効にする/無効にする

関数の説明:

この関数はトリガキーを有効または無効にします。

関数コール:

void EnableScannerTrigger(BOOL fOn)

パラメータ (入力)

fON: BOOL: TRUE = トリガキーを有効にする 。
FALSE = トリガキーを無効にする。

戻りコード:

8.2.2 スキャンエンジンの電源をオンにする/オフにする

関数の説明:

この関数はスキャンエンジンの電源をオンまたはオフにするために使用されます。

関数コール:

void PowerOnScanner (BOOL fOn)

パラメータ(入力)

fON: BOOL: TRUE = スキャンエンジンの電源をオンにする。
FALSE= スキャンエンジンの電源をオフにする。

戻りコード:

なし

8.2.3 スキャンエンジンをオン/オフする

関数の説明:

この関数はスキャンエンジンをオンまたはオフにするためにトリガキーをエミュレートします。これはトリガキーが無効になっている場合に働きます。

関数コール:

void SetScannerOn(BOOL fON)

パラメータ(入力)

fON: BOOL: TRUE = スキャンエンジンをオンにする。
FALSE= スキャンエンジンをオフにする。

戻りコード:

なし

8.2.4 トリガキーステータスを得る

関数の説明:

この関数はトリガキーの有効/無効ステータスを戻します。

関数コール:

BOOL GetScannerTrigger(void)

戻りコード:

TRUE = トリガキーは有効
FALSE = トリガキーは無効

8.2.5 スキャナステータスを得る

関数の説明:

この関数はスキャナエンジンのステータスを戻します。

関数コール:

BOOL GetScannerStatus(void)

戻りコード:

TRUE = スキャンエンジンはオン
FALSE = スキャンエンジンはオフ

8.2.6 トリガキーが押されたかどうかをチェック

関数の説明:

この関数は右または左のトリガキーが押されたかどうかをチェックするために使用されます。

関数コール:

BOOL TriggerKeyStatus(int key);

パラメータ(入カ):

Key:	int:	LEFT_TRIGGER_KEY	;左トリガキー
		RIGHT_TRIGGER_KEY	;右トリガキー

戻りコード:

TRUE = トリガキーが押された。
FALSE = トリガキーは押されていない。

例:

```
#define kKeybdTriggerEventName          TEXT("KeybdTriggerChangeEvent")
#define kKeybdAlphaKeyEventName         TEXT("KBDAlphaKeyChangeEvent")
```

```
        #define LEFT_TRIGGER_KEY  1
        #define RIGHT_TRIGGER_KEY 2

        gKeyEvents[0] = CreateEvent(NULL, TRUE, FALSE, kKeybdTriggerEventName);
        gKeyEvents[1] = CreateEvent(NULL, TRUE, FALSE, kKeybdAlphaKeyEventName);
        while (1)
        {
            WaitForMultipleObjects(2, gKeyEvents, FALSE, INFINITE);
            TriggerKeyStatus(LEFT_TRIGGER_KEY);
        TriggerKeyStatus(RIGHT_TRIGGER_KEY);
}
```

8.3 LED 関連関数

8.3.1 読み取り完了 LED をオン/オフする

関数の説明:

この関数は、読み取り完了 LED をオン/オフするために使用されます。

関数コール:

void GoodReadLEDOn(BOOL fON)

パラメータ(入力)

fON: BOOL: TRUE = 読み取り完了 LED をオンにする
FALSE = 読み取り完了 LED をオフにする

8.3.2 カメラ LED をオン/オフする

関数の説明:

この関数は、カメラ LED をオン/オフするために使用されます。

関数コール:

void CameraLEDOn (BOOL bOn)

パラメータ(入力):

TRUE = カメラ LED をオンにする
FALSE = カメラ LED をオフにする

8.3.3 緑 LED をオン/オフする

関数の説明:

この関数は、緑 LED をオン/オフするために使用されます。.

関数コール:

void GreenLEDOn (BOOL bOn)

パラメータ(入力):

TRUE = 緑 LED をオンにする
FALSE = 緑 LED をオフにする

8.3.4 赤 LED をオン/オフする

関数の説明:

この関数は、赤 LED をオン/オフするために使用されます。

関数コール:

void RedLEDOn (BOOL bOn)

パラメータ(入力):

TRUE = 赤 LED をオンにする
FALSE = 赤 LED をオフにする

8.4 バックライト関連関数

スクリーンバックライトとキーパッドバックライトの二つのバックライト制御があります。これらは個別に制御されます。スクリーンバックライトについては、バックライトの明るさを調節することもできます。

8.4.1 スクリーンバックライト制御

関数の説明:

この関数はスクリーンのバックライトをオンまたはオフします。

関数コール:

void BacklightOn(BOOL fON)

パラメータ(入力)

fON: BOOL: TRUE = スクリーンのバックライトをオン
FALSE= スクリーンのバックライトをオフ

戻りコード:

8.4.2 スクリーンのバックライトステータスを得る

関数の説明:

この関数はスクリーンのバックライトステータスを戻します。

関数コール:

BOOL GetBacklightStatus(void)

戻りコード:

TRUE = スクリーンのバックライトがオン
FALSE = スクリーンのバックライトがオフ

8.4.3 キーパッドのバックライト制御

関数の説明:

この関数はキーパッドのバックライトをオンまたはオフします。

関数コール:

void KeypadLightOn(BOOL fON)

パラメータ(入力)

fON: BOOL: TRUE = キーパッドのバックライトをオン
FALSE = キーパッドのバックライトをオフ

戻りコード:

8.4.4 キーパッドのバックライトステータスを得る

関数の説明:

この関数はキーパッドバックライトのステータスを戻します。

関数コール:

BOOL GetKeypadLightStatus(void)

戻りコード:

TRUE = キーパッドバックライトはオン
FALSE = キーパッドバックライトはオフ

8.4.5 スクリーンバックライトの明暗制御

関数の説明:

この関数はスクリーンバックライトの明暗を調整します。

関数コール:

void BrightnessUp(BOOL fup)

パラメータ(入力)

Fup: BOOL: TRUE = 一つ明るさを上げる
FALSE = 一つ明るさを落とす

戻りコード:

8.5 SD スロット関連関数

8.5.1 SD スロットのステータスを得る

関数の説明:

この関数は、SD スロットの有効/無効ステータスを得ます。

関数コール:

BOOL GetSDStatus()

戻りコード:

TRUE = スロット は有効
FALSE = スロットは無効

8.5.2 SD スロットを有効/無効にする

関数の説明:

この関数は、SD スロットを有効/無効にします。

関数コール:

void EnableSDSlot(BOOL bEnable);

パラメータ(入力)

bEnable:　　BOOL:　　TRUE = スロットを有効にする
FALSE = スロットを無効にする

8.5.3　SD スロットの動作モードを得る

関数の説明:

この関数は、SD スロットモードを得るために使用されます。

関数コール:

DWORD GetSDMode (void)

戻りコード:

0x01：SDMMC モード
0x02：SDIO モード
0x11：SDMMC レジューム中に外さない

8.5.4　SD スロットの動作モードをセットする

関数の説明:

この関数は、SD スロットの動作モードをセットアップするために使用されます。

関数コール:

void SetSDMode(DWORD nMode)

パラメータ:(入力)

nMode：UINT32:　　0x01：SDMMC モード
0x02：SDIO モード
0x11：SDMMC レジューム中に外さない

8.5.5　バイブレータを有効/無効

関数の説明:

この関数は、バイブレータを有効/無効にします。

関数コール:

void VibrationOn(BOOL bEnable);

パラメータ(入力)

bEnable:　　BOOL:　　TRUE = オン
FALSE = オフ

8.5.6 カメラのオートフォーカス

関数の説明:

この関数は、カメラのフォーカスをコールします。

関数コール:

BOOL TriggerAutoFocus(void);

戻りコード:

TRUE = オートフォーカスが成功
FALSE = オートフォーカスが失敗.

8.6 デバイス情報

8.6.1 スマートバッテリ ID を得る

関数の説明:

この関数は、スマートバッテリ ID を得るために使用されます。

関数コール:

BYTE GetSmartBatteryID()

戻りコード:

0x00/0x01：スマートバッテリ ID をサポートしていない
その他の値：スマートバッテリ ID.

8.6.2 クレードルステータスをチェックする

関数の説明:

この関数は、デバイスがクレードルにあるかどうかをチェックするために使用されます。

関数コール:

BOOL GetCradleStatus()

戻りコード:

TRUE = デバイスがクレードルにある
FALSE = デバイスがクレードルにない

8.7 マイク、受話器、およびスピーカの制御

8.7.1 受話器とスピーカを切り換える

関数の説明:

この関数は、受話器またはスピーカからの音声出力を切り換えるために使用されます。

関数コール:

void Sound_To_Speaker_Or_Receiver (BOOL fOn)

パラメータ(入力)

fOn: BOOL: TRUE = オン
FALSE = オフ

8.7.2 メインマイクを得る

関数の説明:

この関数は、メインマイクを得るために使用されます。

関数コール:

DWORD Get_Main_Mic()

戻りコード:

1：前方側マイク
2：後方側マイク

8.7.3 メインマイクをセットする

関数の説明:

この関数は、メインマイクを切り換えるために使用されます。

関数コール:

void Set_Main_Mic(DWORD dwIndex)

パラメータ(入力)

dwIndex: DWORD： 1：前方側マイク
2：後方側マイク

8.8 無線モジュール関連の関数

8.8.1 無線モジュールのステータスを得る

関数の説明:

この関数は、無線モジュールの有効/無効ステータスを戻します。

関数コール:

BOOL GetWLANStatus()

戻りコード:

TRUE = モジュールは有効
FALSE = モジュールは無効

8.8.2 無線モジュールを有効/無効にする

関数の説明:

この関数は、無線モジュールを有効/無効にします。

関数コール:

void WLANPowerEnable(BOOL bOn);

パラメータ(入力)

bOn: BOOL: TRUE = モジュールを有効にする
FALSE = モジュールを無効にする

8.9 Bluetooth モジュール関連関数

8.9.1 Bluetooth 電源ステータスを有効/無効

関数の説明:

Bluetooth モジュールの電源 ON/OFF を有効にする

関数コール:

void BT_PowerEnable (BOOL bEnable)

パラメータ (入力)

bON: BOOL: TRUE = 有効
FALSE = 無効

8.9.2 Bluetooth 電源ステータスを得る

関数の説明:

Bluetooth モジュールの電源ステータスを得る

関数コール:

BYTE BT_PowerStatus (void)

戻りコード:

BYTE: 1 = Bluetooth モジュールの電源が ON
0 = Bluetooth モジュールの電源が OFF

8.10 PCMCIA/CF スロット関連関数

HT660 は CF スロットのみサポートしており、PA96X/PA982 には I/O カード用のスロットが二つあります。PCMCIA は HT6x0 では動作しません。

8.10.1 物理スロット ID を得る

関数の説明:

A96X/PA982 は PCMCIA と CF 用にスロット 0 とスロット 1 の二つの PC スロットがあります。この関数は、どちらのスロットが PCMCIA か CF かを戻します。

関数コール:

UNIT GetPCMCIASlotID(UNIT)

パラメータ(入力)

0 = PCMCIA(PA962/PA966/PA982 のみ)
1 = CF

戻りコード:

物理スロット ID

8.10.2　PCMCIA または CF スロットを有効/無効にする

関数の説明:

この関数は I/O スロットを有効/無効にします。PA96x/PA982 は、物理スロット 0 を CF に、そしてスロット 1 を PCMCIA に割り当てます。これは以前の説明とは逆になっています。以下の関数は、互換性を保っています。これは同じ uSocket の値ですが、内部では逆になっています。

関数コール:

void EnablePCMCIASlot(UINT uSocket, BOOL bEnable)

パラメータ(入力)

uSocket:	UINT:	0 = PCMCIA スロット (PA962/PA966/PA982 のみ)
		1 = コンパクトフラッシュスロット
bEnable：	BOOL:	TRUE = 指定したスロットを有効にする
		FALSE = 指定したスロットを無効にする

8.10.3　I/O スロットを有効/無効にする

関数の説明:

この関数は I/O スロットを有効/無効にします。プラットフォームに独立であるために関数 GetPCMCIASlotID() と一緒に使われることをお勧めします。

関数コール:

void EnablePCMCIASlot1(UINT uSocket, BOOL bEnable)

パラメータ(入力):

uSocket:	UINT:	適用されるスロット
bEnable：	BOOL:	TRUE = 指定したスロットを有効にする
		FALSE = 指定したスロットを無効にする

例:

PCMCIA スロットを無効にし、CF スロットを有効にする。

```
#define PCMCIA_SOCKET   0   (PA962/PA966/PA982 のみ)
#define CF_SOCKET   1
EnablePCMCIASlot1(GetPCMCIASlotID(PCMCIA_SLOT),FALSE);
EnablePCMCIASlot1(GetPCMCIASlotID(CF_SLOT),TRUE);
```

8.10.4　PCMCIA/CF スロットのステータスを得る

関数の説明:

この関数はスロットの有効/無効ステータスを戻します。

関数コール:

BOOL GetPCMCIAStatus(UINT uSocket)

パラメータ(入力)

uSocket:	UINT:	0 = PCMCIA スロット(PA962/PA966/PA982 のみ)

1 = コンパクトフラッシュスロット

戻り値

bEnable：	BOOL:	TRUE = スロットは有効 FALSE = スロットは無効

8.10.5　IO スロットステータスを得る

関数の説明:

この関数は有効/無効のステータスを戻します。プラットフォームに独立であるために関数 GetPCMCIASlotID() と一緒に使われることをお勧めします。

関数コール:

BOOL GetPCMCIAStatus1(UINT uSocket)

パラメータ(入力):

uSocket:	UINT:	適用されるスロット

戻り:

bEnable：	BOOL:	TRUE = 指定したスロットは有効 FALSE = 指定したスロットは無効

例

PCMCIA スロットのステータスをチェック　(PA962/PA966/PA982 のみ)

```
#define PCMCIA_SOCKET   0
#define CF_SOCKET 1
if (GetPCMCIAStatus1(GetPCMCIASlotID(PCMCIA_SLOT))) {
}
```

8.10.6　レジューム時に PCMCIA/CF スロットを無効にする

関数の説明:

この関数は、スロットがサスペンド前に有効であってもレジューム後に指定したスロットを無効にします。

関数コール:

void DisablePCMCIAUponResume(UINT uSocket, BOOL bDisable);

パラメータ(入力):

uSocket:	UINT:	1 = 物理スロット 1 0 = 物理スロット 0
bDisable:	BOOL:	TRUE レジューム時に無効 FALSE レジューム時に有効

戻り

なし

8.10.7　LCD スクリーンを有効/無効

関数の説明:

LCD スクリーンをオン/オフします。

関数コール:

void PowerOnColorLCD(BOOL fON)

パラメータ(入力)

fON:　BOOL:　　TRUE　= LCD スクリーンの電源をオン
　　FALSE　= LCD スクリーンの電源をオフ

戻り

なし

8.10.8　RFID モジュールの電源をオン/オフ

関数の説明:

RFID モジュールの電源をオン/オフ (RH768 のみ)

関数コール:

void RFIDPowerOn(BOOL bON)

パラメータ(入力)

bON:　　BOOL:　　TRUE = RFID モジュールをオン
　　FALSE　= RFID モジュールをオフ

9. 動的な DLL 読み込み

ダイナミックロード DLL を使用している場合コンパイラは DLL を読み込みません。アプリケーションを実行中にこれがあると DLL の読み込みを助けます。以下に例を示します。

ノート: ユーザがヘッダと Lib ファイルを含む必要が無くても関数定義を知る必要があります。

```
////////////////////////////////////////////////////////////////////////////////
//C++

HINSTANCE g_hUSIDLL;
typedef BOOL (*lpfnUSI_GetScannerVersion)(LPTSTR model, LPTSTR firmware, LPTSTR sdk, int blen);
lpfnUSI_GetScannerVersion USI_GetScannerVersion;

g_hUSIDLL = LoadLibrary(L"¥¥Windows¥¥USI.dll");

if (g_hUSIDLL != NULL)
{
      USI_GetScannerVersion = (lpfnUSI_GetScannerVersion)GetProcAddress(g_hUSIDLL,
                                    TEXT("USI_GetScannerVersion"));
}
else
{
   MessageBox(_T("Load library USI.dll fail"), NULL, MB_OK);
   return;
}

TCHAR lstrmodel[50], lstrfirmware[50], lstrsdk[50];

if (USI_GetScannerVersion != NULL)
      rc = USI_GetScannerVersion(lstrmodel, lstrfirmware, lstrsdk, sizeof(lstrmodel) +
                        sizeof(lstrfirmware) + sizeof(lstrsdk));
else
      MessageBox(_T("USI_GetScanerVersion does not find"), NULL, MB_OK);

if (g_hUSIDLL != NULL)
      FreeLibrary(g_hUSIDLL);

////////////////////////////////////////////////////////////////////////////////

//C#
   [DllImport("SysIOAPI.dll", EntryPoint = "VibrationOn")]
public static extern void VibrationOn(bool bEnable);
```

10. HF RFID リーダ

HF RFID リーダをプログラミングするには、C++ ライブラリ “ RDINT.dll” と RDINT.h” が必要です。以下の URL から入手して下さい。

http://w3.tw.ute.com/pub/cs/SDK/RFID/RFID_SDK.zip

10.1 一般的な機能

10.1.1 ライブラリのバージョンを得る

関数の説明:

ライブラリバージョンを得ます。

関数コール:

INT32 RDINTsys_GetAPIVersionString (LPWSTR strVersion);

パラメータ:

strVersion: ライブラリバージョン

戻りコード:

セクション 10.8 をご覧下さい。

10.1.2 RFID リーダに接続する

関数の説明:

コントロールをする前にリーダと接続します。

関数コール:

INT32 RDINTsys_OpenReader (BYTE u8COMPort, UINT32 u32Baudrate, CONST LPTSTR strAccessCode, BYTE u1SecurityMode, UINT32 u32OpenDelayMs, PUINT32 pu32Baudrate)

パラメータ:

u8COMPort:	リーダの COM ポート番号 (1 – 255)
u32Baudrate:	リーダの転送速度、標準値は 19200。9600, 19200, 38400 と 115200 をサポート
strAccessCode:	リーダのアクセスコード、標準値は “00000000”
u1SecurityMode:	セキュリティモードを使用するかどうかをセット
TURN_ON :	オープン
TURN_OFF :	クローズ
u32OpenDelayMs:	リーダの初期化を待つ遅延時間、この値の推奨値は 700 です。
pu32Baudrate:	現在のリーダの転送速度を受信、値を受け取れない場合は NULL。

戻りコード:

セクション 10.8 をご覧下さい。

10.1.3　リーダを終了

関数の説明:

リーダのコントロールを終了します。

関数コール:

INT32 RDINTsys_CloseReader (BYTE u8COMPort);

パラメータ:

u8COMPort:　　リーダの COM ポート番号 (1 – 255)

戻りコード:

10.8 をご覧下さい。

10.1.4　カードタイプを選択

関数の説明:

この API は各カードタイプについてリーダの動作を変更します。カードを読む前に呼び出さなければなりません。

関数コール:

INT32 RDINT_WorkingType (BYTE u8COMPort, BYTE u8Type);

パラメータ:

u8COMPort:　　リーダの COM ポート番号 (1 – 255)
u8Type:　　WT_ISO14443_TypeA
WT_ISO14443_TypeB　　(PA692 はサポートしていません)
WT_ISO15693
WT_SR176_SRIX4K　　(PA692 はサポートしていません)

戻りコード:

セクション 10.8 をご覧下さい。

10.1.5　リーダ情報を得る

関数の説明:

リーダのシリアル番号とファームウェアバージョンを得ます。

関数コール:

INT32 RDINTv2_ReaderInfo (BYTE u8COMPort, LPBYTE pu8SerialNum, LPBYTE pu8FirmwareVer);

パラメータ:

u8COMPort:　　リーダの COM ポート番号 (1 – 255)
pu8SerialNum:　　リーダのシリアル番号 (Length: READER_SERIAL_LEN)
pu8FirmwareVer:　　リーダのファームウェアバージョン (Length: FIRMWARE_VER_LEN)

戻りコード:

セクション 10.8 をご覧下さい。

10.1.6 戻りデータ配列を読む

関数の説明:

最後のコマンドの戻りデータを得る。

関数コール:

INT32 RDINT_GetReturnDataArray (BYTE COMPort, BYTE Index, BYTE Offset, LPBYTE Data);

パラメータ:

COMPort:	リーダの COM ポート番号 (1 – 255)
Index:	データ配列の開始インデックス
Offset:	データ数
Data:	戻りデータとオフセット長.

戻りコード:

セクション 10.8 をご覧下さい

10.2 ISO-15693

10.2.1 インベントリ

関数の説明:

カードを StayQuiet モードにセットし、カード ID を戻します。

関数コール:

INT32 RDINT_ISO15693Inventory(BYTE u8COMPort, BYTE u8Flag, BYTE u8Afi, BYTE u8MaskLen, LPBYTE pu8Mask, LPBYTE pu8Dsfid, LPBYTE pu8Uid);

パラメータ:

u8COMPort:	リーダの COM ポート番号 (1 – 255)
u8Flag:	VICC によって実行される動作と対応するフィールドがあるかどうかを指定します。
u8Afi:	Application Family Identifier パラメータ，ISO15693 仕様をご覧下さい。
u8MaskLen:	このマスク長はマスク値の有効ビット数を示します。これは 16 スロットが使用される場合 0 と 60 の間の任意の値、そして 1 スロットが使用される場合は 0 と 64 の間の任意の値を持つことができます。LSB は最初送信されます。
pu8Mask:	マスク値は整数バイトを含みます。LSB は最初に送信されます。
pu8Dsfid:	Data Storage Format Identifier パラメータ、ISO15693 仕様をご覧下さい。
pu8Uid:	タグ ID を受信するバッファのポインタ

戻りコード:

セクション 10.8 をご覧下さい。

10.2.2 StayQuiet モードのセット

関数の説明:

カードを StayQuiet モードにセットします。

関数コール:

INT32 RDINT_ISO15693StayQuiet(BYTE u8COMPort, BYTE u8Flag, LPBYTE pu8Uid);

パラメータ:

u8COMPort:	リーダの COM ポート番号 (1 – 255)
u8Flag:	VICC によって実行される動作と、対応するフィールドがあるかどうかを指定します。
pu8Uid:	タグ ID を含むバッファのポインタ

戻りコード:

セクション 10.8 をご覧下さい。

10.2.3 Select モードのセット

関数の説明:

カードを Select モードにセットします。

関数コール:

INT32 RDINT_ISO15693Select(BYTE u8COMPort, BYTE u8Flag, LPBYTE pu8Uid);

パラメータ:

u8COMPort:	リーダの COM ポート番号 (1 – 255)。
u8Flag:	VICC によって実行される動作と、対応するフィールドがあるかどうかを指定します。
pu8Uid:	タグ ID を含むバッファのポインタ

戻りコード:

セクション 10.8 をご覧下さい。

10.2.4 Ready モードのセット

関数の説明:

カードを StayQuiet または Select モードについて Ready モードをセットします。

関数コール:

INT32 RDINT_ISO15693Reset2Ready(BYTE u8COMPort, BYTE u8Flag, LPBYTE pu8Uid);

パラメータ:

u8COMPort:	リーダの COM ポート番号 (1 – 255)。
u8Flag:	VICC によって実行される動作と、対応するフィールドがあるかどうかを

指定します。

pu8Uid: タグ ID を含むバッファのポインタ。

戻りコード:

セクション 10.8 をご覧下さい。

10.2.5 ISO15693 タグからブロックデータを読む

関数の説明:

ISO15693 タグからブロックデータを読みます。

関数コール:

INT32 RDINT_ISO15693Read(BYTE u8COMPort, BYTE u8Flag, LPBYTE pu8Uid, BYTE u8BlockStart, BYTE u8BlockCount, LPBYTE pu8Data);

パラメータ:

u8COMPort: リーダの COM ポート番号 (1 – 255)

u8Flag: VICC によって実行される動作と、対応するフィールドがあるかどうかを指定します。

pu8Uid: タグ ID を含むバッファのポインタ

u8BlockStart: 読みたい最初のブロック(例：0, 1, 2…)

u8BlockCount: 読みたいブロック数

pu8Data: ブロックデータを受信するバッファのポインタ

戻りコード:

セクション 10.8 をご覧下さい。

10.2.6 ブロックデータを ISO15693 タグに書く

関数の説明:

ブロックデータを ISO15693 タグに書きます。

関数コール:

INT32 RDINT_ISO15693Write(BYTE u8COMPort, BYTE u8Flag, LPBYTE pu8Uid, BYTE u8Block, LPBYTE pu8Data);

パラメータ:

u8COMPort: リーダの COM ポート番号 (1 – 255)

u8Flag: VICC によって実行される動作と、対応するフィールドがあるかどうかを指定します。

pu8Uid: タグ ID を含むバッファのポインタ

u8Block: 書き込みたいブロック(例：0, 1, 2…)

pu8Data: データを含むバッファのポインタ

戻りコード:

セクション 10.8 をご覧下さい。

10.2.7　マルチデータブロックを ISO15693 タグに書く

関数の説明:

マルチデータブロックを指定した ISO15693 タグに書きます。

関数コール:

INT32 RDINT_ISO15693WriteMultiBlock(BYTE u8COMPort, BYTE u8Flag, LPBYTE pu8Uid, BYTE u8BlockStart, BYTE u8BlockCount, LPBYTE pu8Data);

パラメータ:

u8COMPort:	リーダの COM ポート番号 (1 – 255)
u8Flag:	VICC により実行されるアクションと、対応するフィールドの有無を指定します。
pu8Uid:	タグ Id を含むバッファのポインタ
u8BlockStart:	書き込みたい最初のブロック(例：0, 1, 2…)
u8BlockCount:	書き込みたいブロック数
pu8Data:	データを含むバッファのポインタ

戻りコード:

セクション 10.8 をご覧下さい。

10.2.8　ISO15693 ブロックをロック

関数の説明:

ISO15693 タグのブロックをロックします。

関数コール:

INT32 RDINT_ISO15693LockBlock(BYTE u8COMPort, BYTE u8Flag, LPBYTE pu8Uid, BYTE u8Block);

パラメータ:

u8COMPort:	リーダの COM ポート番号 (1 – 255)
u8Flag:	VICC によって実行される動作と、対応するフィールドがあるかどうかを指定します。
pu8Uid:	タグ ID を含むバッファのポインタ
u8Block:	書き込みたいブロック(例：0, 1, 2…).

戻りコード:

セクション 10.8 をご覧下さい。

10.2.9　ISO15693 タグに AFI を書く

関数の説明:

ISO15693 タグに AFI を書き込みます。

関数コール:

INT32 RDINT_ISO15693WriteAfi(BYTE u8COMPort, BYTE u8Flag, LPBYTE pu8Uid, BYTE u8AfiValue);

パラメータ:

u8COMPort:	リーダの COM ポート番号 (1 – 255)
u8Flag:	VICC によって実行される動作と、対応するフィールドがあるかどうかを指定します。
pu8Uid:	タグ ID を含むバッファのポインタ
u8AfiValue:	AFI の値とこの値については、 ISO15693 仕様をご覧下さい。

戻りコード:

セクション 10.8 をご覧下さい。

10.2.10 ISO15693 の AFI をロック

関数の説明:

ISO15693 タグの AFI をロックします。

関数コール:

INT32 RDINT_ISO15693LockAfi(BYTE u8COMPort, BYTE u8Flag, LPBYTE pu8Uid);

パラメータ:

u8COMPort:	リーダの COM ポート番号 (1 – 255)
u8Flag:	VICC によって実行される動作と、対応するフィールドがあるかどうかを指定します。
pu8Uid:	タグ ID を含むバッファのポインタ

戻りコード:

セクション 10.8 をご覧下さい。

10.2.11 ISO15693 タグに DSFID を書く

関数の説明:

ISO15693 タグに DSFID を書きます。

関数コール:

INT32 RDINT_ISO15693WriteDsfid(BYTE u8COMPort, BYTE u8Flag, LPBYTE pu8Uid, BYTE u8DsfidValue);

パラメータ:

u8COMPort:	リーダの COM ポート番号 (1 – 255)
u8Flag:	VICC によって実行される動作と、対応するフィールドがあるかどうかを指定します。
pu8Uid:	タグ ID を含むバッファのポインタ
u8DsfidValue:	DSFID の値と値については、ISO15693 の仕様をご覧下さい。

戻りコード:

セクション 10.8 をご覧下さい。

10.2.12 ISO15693 の DSFID をロック

関数の説明:

ISO15693 タグの DSFID をロックします。

関数コール:

INT32 RDINT_ISO15693LockDsfid(BYTE u8COMPort, BYTE u8Flag, LPBYTE pu8Uid);

パラメータ:

u8COMPort:	リーダの COM ポート番号 (1 – 255)
u8Flag:	VICC によって実行される動作と、対応するフィールドがあるかどうかを指定します。
pu8Uid:	タグ ID を含むバッファのポインタ

戻りコード:

セクション 10.8 をご覧下さい。

10.3 TI の ISO-15693

10.3.1 インベントリ

関数の説明:

TI カードを選択し、カードをレディモードにして、カード Id を得ます。

関数コール:

INT32 TI_ISO15693Inventory (BYTE u8COMPort, BYTE u8Afi, BYTE u8MaskLen, CONST LPBYTE pcu8Mask, LPBYTE pu8Dsfid, LPBYTE pu8Uid);

パラメータ:

u8COMPort:	リーダの COM ポート番号 (1 – 255)
u8Afi:	Application Family Identifier パラメータ: ISO15693 仕様をご覧下さい。
u8MaskLen:	使用するマスク長、マスクリストから使用するマスク長を選択します。
pcu8Mask:	マスク値は整数バイト数を含みます。
pu8Dsfid:	Data Storage Format Identifier パラメータ、ISO15693 仕様をご覧下さい。
pu8Uid:	タグ ID を受信するバッファのポインタ

戻りコード:

セクション 10.8 をご覧下さい。

10.3.2 Stay Quiet

関数の説明:

カードを Stay Quiet モードにセットします。レディモードに入った後で、この関数を実行することができます。カードがこのモードにセットされたとき、これはレディモードまたは選択モードに、そして通常モードにリセットしなければなりません。

関数コール:

INT32 TI_ISO15693StayQuiet(BYTE u8COMPort);

パラメータ:

u8COMPort:	リーダの COM ポート番号 (1 – 255)

戻りコード:

セクション 10.8 をご覧下さい。

10.3.3　読み込み

関数の説明:

指定したブロック番号に従って TI カードのブロックデータを読みます。

関数コール:

INT32 TI_ISO15693Read(BYTE u8COMPort, BYTE u8BlockStart, BYTE u8BlockCount, LPBYTE pu8Data);

パラメータ:

u8COMPort:	リーダの COM ポート番号 (1 – 255)
u8BlockStart:	読み取る最初のブロック
u8BlockCount:	読み取るブロック数
pu8Data:	読み取りデータを戻す

戻りコード:

セクション 10.8 をご覧下さい。

10.3.4　書き込み

関数の説明:

指定したブロック番号に従ってデータを TI カードブロックに書き込みます。

関数コール:

INT32 TI_ISO15693Write(BYTE u8COMPort, BYTE u8Block, CONST LPBYTE pu8Data);

パラメータ:

u8COMPort:	リーダの COM ポート番号 (1 – 255)
u8Block:	書き込むブロック
pu8Data:	書き込むデータ

戻りコード:

セクション 10.8 をご覧下さい。

10.3.5　ブロックをロック

関数の説明:

指定したブロック番号に従って TI カードをブロックします。
指定した番号に従って、ロックされた TI カードブロックは読み込みのみ許され、無効な書き込みを避けるためにデータを書き込むことはできません。

関数コール:

INT32 TI_ISO15693LockBlock(BYTE u8COMPort, BYTE u8Block);

パラメータ:

u8COMPort:	リーダの COM ポート番号 (1 – 255)
u8Block:	ロックするブロック番号

戻りコード:

セクション 10.8 をご覧下さい。

10.3.6 キル

関数の説明:

カードのすべての機能を削除します。

関数コール:

INT32 TI_ISO15693Kill(BYTE u8COMPort,CONST LPBYTE pu8CardPassword);

パラメータ:

u8COMPort: リーダの COM ポート番号 (1 – 255)
pu8CardPassword: カードのパスワード

戻りコード:

セクション 10.8 をご覧下さい。

10.3.7 パスワードの書き込み

関数の説明:

指定したブロックに新しいパスワードをセットします。

関数コール:

INT32 TI_ISO15693WriteSingleBlockPwd(BYTE u8COMPort, CONST LPBYTE pu8CardPassword, BYTE pu8Block, CONST LPBYTE pu8NewPassword);

パラメータ:

u8COMPort: リーダの COM ポート番号 (1 – 255)
pu8CardPassword: カードのパスワード
pu8Block: パスワードを変更するブロック番号
pu8NewPassword: 新しいパスワード

戻りコード:

セクション 10.8 をご覧下さい。

10.4 ISO-14443A

10.4.1 標準値キーの書き込み

関数の説明:

標準値キーをリーダに書き込みます。

関数コール:

INT32 RDINT_WriteDefaultKey(BYTE u8COMPort, BYTE u8DefaultKeyIndx, LPBYTE pu8DefaultKey);

パラメータ:

u8COMPort:	リーダの COM ポート番号 (1 – 255)
u8DefaultKeyIndx:	リーダの標準値キーインデックス
pu8DefaultKey:	キーを含むバッファのポインタ

戻りコード:

セクション 10.8 をご覧下さい。

10.4.2 ISO-14443A カードをオープン

関数の説明:

ISO-14443A タグをロックし、タグ ID を得ます。カードタイプを選択した後で、ユーザは ISO-14443A タグをコントロールする前にこの API を呼ばなければなりません。

関数コール:

INT32 RDINT_OpenCard(BYTE u8COMPort, BYTE u1AutoFind, LPBYTE pu8Uid, LPBYTE pu8Atqa, LPBYTE pu8Sak);

パラメータ:

u8COMPort:	リーダの COM ポート番号 (1 – 255)
u1AutoFind:	自動カード検索
	TURN_ON
	TURN_OFF
pu8Uid:	タグ ID 受信
pu8Atqa:	ATQA カードタイプを戻す
	ATQA_MIFAER_S50
	ATQA_MIFAER_S70
	ATQA_ULTRA_LIGHT
pu8Sak:	SAK カードタイプを戻す
	SAK_ISO14443_3
	SAK_ISO14443_4

戻りコード:

セクション 10.8 をご覧下さい。

10.4.3 ISO-14443A カードをクローズ

関数の説明:

ISO-14443A タグのロックを解除します。ISO-14443A タグのコントロール後は、ユーザはタグのロックを解除するためにこの API を呼ばなければなりません。

関数コール:

INT32 RDINT_CloseCard(BYTE u8COMPort);

パラメータ:

u8COMPort: リーダの COM ポート番号 (1 – 255)

戻りコード:

セクション 10.8 をご覧下さい。

10.4.4 ISO-14443A リセット

関数の説明:

ISO14443A カードをリセットします。

関数コール:

INT32 RDINT_ISO14443AReset(BYTE u8COMPort, LPBYTE pu8GRLCardInf);

パラメータ:

u8COMPort: リーダの COM ポート番号 (1 – 255)
pu8GRLCardInf: 戻りデータ長
データを得るには RDINT_GetReturnDataArray を使用します。

戻りコード:

セクション 10.8 をご覧下さい。

10.4.5 ISO-14443A ブロックデータ読み込み

関数の説明:

特定のブロックデータを読みます。

関数コール:

INT32 RDINT_ReadMifareOneBlock(BYTE u8COMPort, BYTE u1KeyType, BYTE u1DefaultKey, BYTE u8DefaultKeyIndx, BYTE u8Block, LPBYTE pu8Key, LPBYTE pu8Data);

パラメータ:

u8COMPort: リーダの COM ポート番号 (1 – 255)
u1KeyType: プライベートキータイプ
CARD_KEY_A
CARD_KEY_B
u1DefaultKey: リーダの標準パスワードを使用

TURN_ON
TURN_OFF

u8DefaultKeyIndx: リーダの標準キーのインデックス

u8Block: 読みたいブロック (例：0, 1, 2…).

pu8Key: ユーザ定義のキーの値

pu8Data: 受信データ

戻りコード:

セクション 10.8 をご覧下さい。

10.4.6 ISO-14443A セクタデータ読み込み

関数の説明:

特定のセクタデータを読みます。

関数コール:

INT32 RDINT_ReadMifareOneSector(BYTE u8COMPort, BYTE u1KeyType, BYTE u1DefaultKey, BYTE u8DefaultKeyIndx, BYTE u8Sector, LPBYTE pu8Key, LPBYTE pu8Data);

パラメータ:

u8COMPort: リーダの COM ポート番号 (1 – 255)

u1KeyType: プライベートキータイプ

CARD_KEY_A

CARD_KEY_B

u1DefaultKey: リーダの標準パスワードを使用

TURN_ON

TURN_OFF

u8DefaultKeyIndx: リーダの標準キーのインデックス

u8Sector: 読みたいセクター (例: 0, 1, 2…).

pu8Key: ユーザ定義のキーの値

pu8Data: 受信データ

戻りコード:

セクション 10.8 をご覧下さい。

10.4.7　ISO-14443A ブロックデータ書き込み

関数の説明:

特定のブロックにデータを書きます。

関数コール:

INT32 RDINT_WriteMifareOneBlock(BYTE u8COMPort, BYTE u1KeyType, BYTE u1DefaultKey, BYTE u8DefaultKeyIndx, BYTE u8Block, LPBYTE pu8Key, LPBYTE pu8Data);

パラメータ:

u8COMPort:	リーダの COM ポート番号 (1 – 255)
u1KeyType:	プライベートキータイプ
	CARD_KEY_A
	CARD_KEY_B
u1DefaultKey:	リーダの標準パスワードを使用
	TURN_ON
	TURN_OFF
u8DefaultKeyIndx:	リーダの標準キーのインデックス
u8Block:	書きたいブロック (例: 0, 1, 2…).
pu8Key:	ユーザ定義のキーの値
pu8Data:	書きたいデータ

戻りコード:

セクション 10.8 をご覧下さい。

10.4.8　ISO-14443A Ultra Light ブロックデータを読む

関数の説明:

Ultra Light タグのブロックデータを読みます。

関数コール:

INT32 RDINT_ReadUltraLight(BYTE u8COMPort, BYTE u8Block, LPBYTE pu8Data);

パラメータ:

u8COMPort:	リーダの COM ポート番号 (1 – 255)
u8Block:	読みたいブロック (例: 0, 1, 2…).
pu8Data:	読みたいデータ

戻りコード:

セクション 10.8 をご覧下さい。

10.4.9 ISO-14443A Ultra Light ブロックデータの書き込み

関数の説明:

Ultra Light タグの指定したブロックにデータを書き込みます。

関数コール:

INT32 RDINT_WriteUltraLight(BYTE u8COMPort, BYTE u8Block, LPBYTE pu8CmdData);

パラメータ:

u8COMPort: リーダの COM ポート番号 (1 – 255)
u8Block: 書き込みたいブロック (例: 0, 1, 2…)
pu8CmdData: 書き込みたいデータ

戻りコード:

セクション 10.8 をご覧下さい。

10.5 ISO-14443B(PA692 はサポートしていません)

10.5.1 ST カードを選択

関数の説明:

ST カードを選択します。

関数コール:

INT32 RDINT_STCardSelect(BYTE u8COMPort, LPBYTE pu8IDNum);

パラメータ:

u8COMPort: リーダの COM ポート番号 (1 – 255)
pu8IDNum: カードデータ

戻りコード:

セクション 10.8 をご覧下さい。

10.5.2 ST カードをリリース

関数の説明:

ST カードをリリースします。

関数コール:

INT32 RDINT_STCardIntoDeactive(BYTE u8COMPort);

パラメータ:

u8COMPort: リーダの COM ポート番号 (1 – 255)

戻りコード:

セクション 10.8 をご覧下さい。

10.5.3 SR176 カードのブロックデータを読む

関数の説明:

特定の SR176 カードからブロックを読みます。

関数コール:

INT32 RDINT_SR176ReadBlock(BYTE u8COMPort, BYTE u8BlkNo, LPBYTE pu8Data);

パラメータ:

u8COMPort:	リーダの COM ポート番号 (1 – 255)
u8BlkNo:	読み込みたいブロック数 (例: 0, 1, 2…).
pu8Data:	受信データ

戻りコード:

セクション 10.8 をご覧下さい。

10.5.4 SR176 カードのブロックデータを書き込む

関数の説明:

特定の SR176 カードのブロックにデータを書きます。

関数コール:

INT32 RDINT_SR176WriteBlock(BYTE u8COMPort, BYTE u8BlkNo, LPBYTE pu8Data);

パラメータ:

u8COMPort:	リーダの COM ポート番号 (1 – 255)
u8BlkNo:	書きたいブロック番号 (例: 0, 1, 2…).
pu8Data:	ブロックに書きたいデータ

戻りコード:

セクション 10.8 をご覧下さい。

10.5.5 SR176 ブロックをロック

関数の説明:

SR176 カードの特定ブロックをロックします。

関数コール:

INT32 RDINT_SR176LockBlock(BYTE u8COMPort, BYTE u8BlkNo);

パラメータ:

u8COMPort:	リーダの COM ポート番号 (1 – 255)
u8BlkNo:	ロックしたいブロック番号 (例: 0, 1, 2…).

戻りコード:

セクション 10.8 をご覧下さい。

10.5.6　SRIX4K カードのブロックデータを読み込む

関数の説明:

特定の SRIX4K カードからブロックを読み込みます。

関数コール:

INT32 RDINT_SRIX4KReadBlock (BYTE u8COMPort, BYTE u8BlkNo, LPBYTE pu8Data);

パラメータ:

u8COMPort:	リーダの COM ポート番号 (1 – 255)
u8BlkNo:	読み込みたいブロック番号 (例: 0, 1, 2…).
pu8Data:	受信データ

戻りコード:

セクション 10.8 をご覧下さい。

10.5.7　SRIX4K カードのブロックデータに書き込む

関数の説明:

特定の SRIX4K カードのブロックにデータを書き込みます。

関数コール:

INT32 RDINT_SR176WriteBlock(BYTE u8COMPort, BYTE u8BlkNo, LPBYTE pu8Data);

パラメータ:

u8COMPort:	リーダの COM ポート番号 (1 – 255)
u8BlkNo:	書き込みたいブロック番号 (例: 0, 1, 2…).
pu8Data:	ブロックに書き込みたいデータ

戻りコード:

セクション 10.8 をご覧下さい。

10.5.8　SRIX4K カードの認証

関数の説明:

特定の SRIX4K カードを認証します。

関数コール:

INT32 RDINT_SRIX4KAuth(BYTE u8COMPort, LPBYTE pu8Auth);

パラメータ:

u8COMPort:	リーダの COM ポート番号 (1 – 255)
pu8Auth:	認証データ

戻りコード:

セクション 10.8 をご覧下さい。

10.5.9　SRIX4K カード ID を読む

関数の説明:

SRIX4K カード ID を読みます。

関数コール:

INT32 RDINT_SRIX4KReadUID(BYTE u8COMPort, LPBYTE pu8Uid);

パラメータ:

u8COMPort:	リーダの COM ポート番号 (1 – 255)
pu8Uid:	読んだカード ID

戻りコード:

セクション 10.8 をご覧下さい。

10.5.10 ISO14443B リセット

関数の説明:

ISO14443B カードをリセットします。

関数コール:

INT32 RDINT_ISO14443BReset(BYTE COMPort, BYTE FindType, BYTE SlotNum, BYTE AFI, LPBYTE GRLCardInf);

パラメータ:

COMPort:	リーダの COM ポート番号 (1 – 255)
FindType:	カードを見つける方法(FT_REQB, FT_WUPB)
SlotNum:	スロット番号
AFI:	AFI
GRLCardInf:	戻りデータ長 データを得るには RDINT_GetReturnDataArray を使用します。

戻りコード:

セクション 10.8 をご覧下さい。

10.6 NFC(PA692 のみサポート)

10.6.1 自動検出開始

関数の説明:

自動検出機能をオープンします。

関数コール:

INT32 RDINT_NFC_StartAutoDetect(BYTE COMPort, BYTE WorkMode);

パラメータ:

COMPort: リーダの COM ポート番号 (1 – 255)

WorkMode: NFC 機能をオープン

- Bit 0 リーダモード ISO14443 Type A
- Bit 1 リーダモード ISO14443 Type B
- Bit 2 リーダモード Felica
- Bit 3 リーダモード ISO15693
- Bit 4 NFCIP モード Target
- Bit 5 NFCIP モード Initiator
- Bit 6 ISO14443A カードモード
- Bit 7 ISO14443B カードモード

戻りコード:

セクション 10.8 をご覧下さい。

10.6.2 自動検出終了

関数の説明:

自動検出機能を閉じます。

関数コール:

INT32 RDINT_NFC_StopAutoDetect(BYTE COMPort);

パラメータ:

COMPort: リーダの COM ポート番号 (1 – 255)

戻りコード:

セクション 10.8 をご覧下さい。

10.6.3　自動検出リセット

関数の説明:

自動検出機能を再度オープンします。

関数コール:

INT32 RDINT_NFC_ResetAutoDetect(BYTE COMPort);

パラメータ:

COMPort:　　リーダの COM ポート番号 (1 – 255)

戻りコード:

セクション 10.8 をご覧下さい。

10.6.4　イベント発生

関数の説明:

タグがあったらデータを受信します。

関数コール:

INT32 RDINT_NFC_EventRaise(BYTE COMPort, LPBYTE EventInformation, PUINT16 Length);

パラメータ:

COMPort:　　リーダの COM ポート番号 (1 – 255)

EventInformation:　　リーダから戻るデータ。

0x00 + ATQA + SAK + UID　ISO14443 Type A が検出された。

0x01 + AFI + PUPI　　ISO14443 Type B カードが検出された。

0x02 + System Code + IDM + PMM　Felica カードが検出された。

0x03 + DSFID + UID　　ISO15693 カードが検出された。

0x04 + ATR_REQ　　NFCIP-1 Initiator が検出された。

0x05 + ATR_RES　　NFCIP-1 Target が検出された。

0x06 + Initiator Data　　NFCIP-1 Initiator のデータ。

0x07 + Target Data　　NFCIP-1 Target のデータ。

0x20 + ISO14443 Type A リーダデータ

ISO14443A リーダからのデータ

0x21 + ISO14443 Type B リーダデータ

ISO14443B リーダからのデータ

Length:　　EventInformation のデータ長

戻りコード:

セクション 10.8 をご覧下さい。

10.6.5 Mifare コマンド送出

関数の説明:

Mifare コマンドを送出します。

関数コール:

INT32 RDINT_NFC_SendMifareCommand(BYTE COMPort, LPBYTE Data, PUINT32 DataLen);

パラメータ:

COMPort: リーダの COM ポート番号 (1 – 255)

Data: コマンド/データを送信/受信

Byte0 : 0x60 認証 A
0x61 認証 B
0x30 16 バイト読み込み
0xA0 16 バイト書き込み
0xA2 4 バイト書き込み
0xC1 インクリメント
0xC0 デクリメント
0xB0 転送
0xC2 リストア
0x38 セクタ読み込み
0xA8 セクタ書き込み

Byte1: ブロック番号

Byte2~5: 4 バイト Mifare カード UID

Byte6~n: データ

DataLen: コマンドまたは戻りデータの長さ

戻りコード:

セクション 10.8 をご覧下さい。

10.6.6 ISO14443 Type A コマンド送出

関数の説明:

ISO14443-4 Type A コマンドを送出します。

関数コール:

INT32 RDINT_NFC_SendISO14443TypeACommand(BYTE COMPort, LPBYTE Data, PUINT32 DataLen);

パラメータ:

COMPort: リーダの COM ポート番号 (1 – 255)

Data: コマンド/データの送信/受信

DataLen: コマンドまたは戻りデータの長さ

戻りコード:

セクション 10.8 をご覧下さい。

10.6.7 ISO14443 Type B コマンド送出

関数の説明:

ISO14443-4 Type B コマンドを送出します。

関数コール:

INT32 RDINT_NFC_SendISO14443TypeBCommand(BYTE COMPort, LPBYTE Data, PUINT32 DataLen);

パラメータ:

COMPort:	リーダの COM ポート番号 (1 – 255)
Data:	コマンド/データの送信/受信
DataLen:	コマンドまたは戻りデータの長さ

戻りコード:

セクション 10.8 をご覧下さい。

10.6.8 Felica コマンド送出

関数の説明:

Felica コマンドを送出します。

関数コール:

INT32 RDINT_NFC_SendFelicaCommand(BYTE COMPort, LPBYTE Data, PUINT32 DataLen);

パラメータ:

COMPort:	リーダの COM ポート番号 (1 – 255)
Data:	コマンド/データを送信/受信
Byte0:	カードの応答を待つ時間 (0x00~0xFF).
Byte1:	ステータス、通常は 0x00 をセット。
Byte2:	コマンドの長さ
Byte3:	コマンドタイプ 0x00 ポーリング 0x02 リクエストサービス 0x04 リクエスト応答 0x06 暗号無し読み取り 0x08 暗号無し書き込み 0x0A サービスコード検索 0x0C システムコード要求
DataLen:	コマンドまたは戻りデータの長さ

戻りコード:

セクション 10.8 をご覧下さい。

10.6.9　ISO15693 コマンド送出

関数の説明:

ISO15693 コマンドを送出します。

関数コール:

INT32 RDINT_NFC_SendISO15693Command(BYTE COMPort, LPBYTE Data, PUINT32 DataLen);

パラメータ:

COMPort:　リーダの COM ポート番号 (1 – 255)
Data:　コマンド/データを送信/受信
　Byte0:　ISO15693 フラグ
　Byte1:　ISO15693 コマンドコード
　　0x01 インベントリ
　　0x02 クワイエット
　　0x20 1 ブロック読み取り
　　0x21 1 ブロック書き込み
　　0x22 ブロックをロック
　　0x25 選択
　Byte2:　パラメータ
　Byte3~10:　8 バイト ISO15693 カード UID
　Byte11~n:　データ
DataLen:　コマンドまたは戻りデータの長さ

戻りコード:

セクション 10.8 をご覧下さい。

10.6.10 NFC IP データ送出

関数の説明:

NFC IP データを送出します。

関数コール:

INT32 RDINT_NFC_SendNFCIPData(BYTE COMPort, LPBYTE Data, PUINT32 DataLen);

パラメータ:

COMPort:　リーダの COM ポート番号 (1 – 255)
Data:　コマンド/データの送信/受信
DataLen:　コマンドまたは戻りデータの長さ

戻りコード:

セクション 10.8 をご覧下さい。

10.6.11 ISO14443 Type A カードデータ送出

関数の説明:

ISO14443 Type A カードモードデータを送出します。

関数コール:

INT32 RDINT_NFC_SendISO14443TypeACardData(BYTE COMPort, LPBYTE OutputData, PUINT32 OutputDataLen);

パラメータ:

COMPort:	リーダの COM ポート番号 (1 – 255)
OutputData:	送信されるデータ
OutputDataLen:	OutputData の長さ

戻りコード:

セクション 10.8 をご覧下さい。

10.6.12 ISO14443 Type B カードデータ送出

関数の説明:

ISO14443 Type B カードモードデータを送出します。

関数コール:

INT32 RDINT_NFC_SendISO14443TypeBCardData(BYTE COMPort, LPBYTE OutputData, PUINT32 OutputDataLen);

パラメータ:

COMPort:	リーダの COM ポート番号 (1 – 255)
OutputData:	送信されるデータ
OutputDataLen:	OutputData の長さ

戻りコード:

セクション 10.8 をご覧下さい。

10.6.13 ISO14443 Type B ポーリング

関数の説明:

ISO14443 Type B カード ID を得ます。

関数コール:

INT32 RDINT_NFC_SendISO14443BPolling(BYTE COMPort, LPBYTE Pupi);

パラメータ:

COMPort:	リーダの COM ポート番号 (1 – 255)
Pupi:	4 バイトカード ID

戻りコード:

セクション 10.8 をご覧下さい。

10.6.14 Felica カードを得る

関数の説明:

Felica カード ID を得ます。

関数コール:

INT32 RDINT_NFC_FelicaGetCard(BYTE COMPort, LPBYTE IDm, LPBYTE PMm);

パラメータ:

COMPort:	リーダの COM ポート番号 (1 – 255)
IDm:	8 バイト IDm
PMm:	8 バイト PMm

戻りコード:

セクション 10.8 をご覧下さい。

10.6.15 Felica ポーリング

関数の説明:

Felica カードの IDm データを得ます。

関数コール:

INT32 RDINT_NFC_FelicaPolling(BYTE COMPort, LPBYTE IDm, LPBYTE PMm, LPBYTE RequestData);

パラメータ:

COMPort:	リーダの COM ポート番号 (1 – 255)
IDm:	8 バイト IDm
PMm:	8 バイト PMm
RequestData:	2 バイトカードデータ

戻りコード:

セクション 10.8 をご覧下さい。

10.6.16 Felica リクエストサービス

関数の説明:

Felica カードの全てのエリアとサービスのキーバージョンを得ます。

関数コール:

INT32 RDINT_NFC_FelicaRequestService(BYTE COMPort, LPBYTE IDm, LPBYTE Count, PUINT16 NodeCode, PUINT16 NodeKeyVerList);

パラメータ:

COMPort:	リーダの COM ポート番号 (1 – 255)
IDm:	8 バイト IDm
Count:	入力：ノード番号 出力：キーバージョン番号

NodeCode: ノード
NodeKeyVerList: キーバージョン.

戻りコード:

セクション 10.8 をご覧下さい。

10.6.17 Felica システムコードを得る

関数の説明:

Felica カードからすべてのシステムコードを得ます。

関数コール:

INT32 RDINT_NFC_FelicaEnumSystemCode(BYTE COMPort, LPBYTE IDm, PUINT16 Systemcode, LPBYTE Count);

パラメータ:

COMPort: リーダの COM ポート番号 (1 – 255)
IDm: 8 バイト IDm
Systemcode: システムコードのデータ
Count: システムコードの数

戻りコード:

セクション 10.8 をご覧下さい。

10.6.18 Felica サービスを得る

関数の説明:

Felica カードからすべてのシステムサービスデータを得ます。

関数コール:

INT32 RDINT_NFC_FelicaEnumService(BYTE COMPort, LPBYTE IDm, PUINT16 SystemService, LPBYTE Count);

パラメータ:

COMPort: リーダの COM ポート番号 (1 – 255)
IDm: 8 バイト IDm
SystemService: システムサービス
Count: システムサービスの数

戻りコード:

セクション 10.8 をご覧下さい。

10.6.19 Felica 暗号無しで読む

関数の説明:

暗号なしでカードを読みます。

関数コール:

INT32 RDINT_NFC_FelicaReadWithoutEncryption(BYTE COMPort, LPBYTE IDm, BYTE ServiceCodeListCount, PUINT16 ServiceCodeList, BYTE BlockListCount, LPBYTE BlockList, LPBYTE StatusFlag1, LPBYTE StatusFlag2, LPBYTE BlockCount, LPBYTE BlockData);

パラメータ:

COMPort:	リーダの COM ポート番号 (1 – 255)
IDm:	8 バイト IDm
ServiceCodeListCount:	サービスコードの数
ServiceCodeList:	システムコードのデータ
BlockListCount:	ブロック数
BlockList:	ブロックのデータ
StatusFlag1:	応答ステータス
StatusFlag2:	応答ステータス
BlockCount:	読み取りブロック数
BlockData:	読み取りブロックデータと長さ BlockCount * 16.

戻りコード:

セクション 10.8 をご覧下さい。

10.6.20 Felica 暗号無しで書く

関数の説明:

暗号無しにカードに書き込みます。

関数コール:

INT32 RDINT_NFC_FelicaWriteWithoutEncryption(BYTE COMPort, LPBYTE IDm, BYTE ServiceCodeListCount, PUINT16 ServiceCodeList, BYTE BlockCount, LPBYTE BlockList, LPBYTE BlockData, LPBYTE StatusFlag1, LPBYTE StatusFlag2);

パラメータ:

COMPort:	リーダの COM ポート番号 (1 – 255)
IDm:	8 バイト IDm
ServiceCodeListCount:	サービスコードの数
ServiceCodeList:	システムコードのデータ
BlockCount:	ブロック数
BlockList:	ブロックのデータ
BlockData:	書き込むデータと長さ BlockCount * 16.
StatusFlag1:	応答ステータス
StatusFlag2:	応答ステータス

戻りコード:

セクション 10.8 をご覧下さい。

10.7 SAM

10.7.1 SAM リセット

関数の説明:

SAM モジュールをリセットします。

関数コール:

INT32 RDINT_SAM_Reset(BYTE COMPort, BYTE u8SAMSlot);

パラメータ:

COMPort:	リーダの COM ポート番号 (1 – 255)
U8SAMSlot:	SAM の接続されたスロット

戻りコード:

セクション 10.8 をご覧下さい。

10.7.2 SAM APDU

関数の説明:

APDU コマンドを送出、この API を使用する前に RDINT_SAM_Reset を呼ばなければなりません。

関数コール:

INT32 RDINT_SAM_APDU(BYTE COMPort, BYTE CLA, BYTE INS, BYTE P1, BYTE P2, LPWSTR adpuData, LPBYTE Resp, PUINT16 pu16RespLen, BYTE u8SAMSlot);

パラメータ:

COMPort:	リーダの COM ポート番号 (1 – 255)
CLA:	クラス分けコード
INS:	インストラクションコード
P1:	パラメータ 1
P2:	パラメータ 2
ApduData:	送信されるデータ
Resp:	SAM から戻るデータ
Pu16RespLen:	Resp の長さ
U8SAMSlot:	SAM の接続されたスロット

戻りコード:

セクション 10.8 をご覧下さい。

10.7.3 SAM ボーレート

関数の説明:

SAM のボーレートをセットします。

関数コール:

INT32 RDINT_SAM_Baudrate(BYTE COMPort, BYTE Baud, BYTE u8SAMSlot);

パラメータ:

COMPort:	リーダの COM ポート番号 (1 – 255)
Baud:	0x00 9600
	0x01 38400(標準値)
U8SAMSlot:	SAM の接続されたスロット

戻りコード:

セクション 10.8 をご覧下さい。

10.8 エラーコード

名前	値	説明
LRSUCCESS	0x00	要求が正常に完了
LRSYSTEM	0x01	不明のエラー
LRLASTCARD	0x02	最後のカードがまだある
LRNOCARD	0x03	カードがない
LRCTYPE	0x04	カードタイプエラー
LRPARAM	0x05	パラメータ要求エラー
LRACCESS	0x06	カードアクセスエラー
LRREAD	0x07	カード読み取りエラー
LRWRITE	0x08	カード書き込みエラー
LRINCR	0x09	Purse インクリメントエラー
LRDECR	0x0a	Purse デクリメントエラー
LRTRANSFER	0x0b	Purse 値転送エラー
LRRESTORE	0x0c	Purse リストアエラー
LRPURSE	0x0d	Purse 値が壊れている
LRMADERR	0x0e	カードディレクトリエラー
LRFIXERR	0x0f	Purse フィックスエラー
LRFIXED	0x10	Purse が壊れたが直った
LRNOTOPEN	0x11	カードはオープンしていない
LRNOFILE	0x12	ファイルがない
LRBADSIZE	0x13	ファイルサイズがおかしい
LRABORTED	0x14	要求が中止
LRMANYCARD	0x15	カードの要求が多すぎる
LRFORMAT	0x16	カードフォーマットエラー
LRCREATE	0x17	カードファイル作成エラー
LRDELETE	0x18	カードファイル削除エラー
LRALREADOPEN	0x19	カードはすでにオープンされている
LRALREADCLOSED	0x1a	カードはすでにクローズされている
LRMSTRKEYLOAD	0x1b	マスターキーをロードできない
LRAPPKEYLOAD	0x1c	アプリケーションをロードできない
LRKEYCARD	0x1d	キーカードエラー
LRUNFORMAT	0x1e	カードにファイルがある
LRNOKBDCHAR	0x20	キーボード文字がない
LRAPIWRITE	0x21	API データ書き込みエラー
LRAPIREAD	0x22	API データ読み込みエラー
LRBLOCKERROR	0x23	ブロック番号エラー
LRNOTIMPL	0x7f	機能が実装されていない
LRUNKNOWN	0x80	不明のエラー

LRCCRBUSY	0xbb	リーダがビジー
LRCHANGEWROKTYPE	0xee	リーダ動作タイプ変更エラー
LRNOINIT	0xef	リーダはオープンされていない
LRCRDNOTOPEN	0xfa	カードはオープンされていない
LRINUSE	0xfb	カードは他のアプリケーションで使用されている
LRAPPLICERR	0xfc	API システムエラー
LRLINKLOST	0xfd	リーダとのリンクが失われた
LRBADCOMPORT	0xfe	COM ポートにアクセスできない
LRNOINIT	0xff	リーダがオープンされていない
LRNOCRYPTBOX	0xf9	キーボックスがない
LRBADAPPACCESS	0xf8	無効なアプリケーションアクセスコード
LRBADRDACCESS	0xf7	無効なリーダアクセスコード
LRNOMAIDFILE	0xf6	MAID 定義をオープンできない
LRBOXREAD	0xf5	キーボックスから読めない
LRBOXWRITE	0xf4	キーボックスに書けない
LRBOXNOKEYS	0xf3	ボックスのキー数はゼロ、または
LRSECURE	0xf2	Comms MAC チェック異常
LRERRSELREADER	0xf1	選択したリーダを変更できない
LRRDNORESP	0xf0	リーダの応答がない
LRSAMUNKNOWN	0xED	SAMカードの不明なエラー

11. RH767 Plus / RH768 UHF リーダ

このライブラリ “ RFID18K6CReader.dll” は、RH767 Plus および RH768 UHF RFID リーダをコントロールするために使用されます。ライブラリは以下の URL からダウンロードして下さい。

http://w3.tw.ute.com/pub/cs/SDK/RFID/RFID_SDK.zip

11.1 RH767 Plus / RH768 RFID リーダ の API について

プロジェクトをビルドするには、すべてのヘッダファイルと RFID186KCReader.lib をプロジェクトフォルダにコピーし、 rfidstruct.h と RFID18K6CReaderAPI.h をプロジェクトにインクルードしてください。

RFID リーダインターフェースを使用するためのステップは次の通りです。

1. RFID リーダインターフェースの初期化
2. RFID リーダをオープンする
3. RFID リーダを設定する
   動作モード、応答データモード、リーダの出力、状態等
4. アンテナの設定をして、有効にする
5. タグのアクセスを実行する
   インベントリ、リード、ライト等
6. RFID リーダをクローズする
7. RFID リーダインターフェースをシャットダウンする

アドバイス：RFID リーダインターフェースを使用する前に、確実に初期化しなければなりません。そして、内部リソースを解放するためにアプリケーションの終了前に適切にシャットダウンしなければなりません。
もしアプリケーションが RFID リーダインターフェースのシャットダウンに失敗した場合、RFID リーダは他のアプリケーションで使用できなくなります。これは、RFID モジュールをリセットするか、ターミナルをリブートする必要があるかもしれません。

11.2 インターフェースマネージメント

11.2.1 RFID リーダインターフェースの初期化

関数の説明:

RFID リーダインターフェースの内部データ構造を適切に初期化することを可能にし、そしてレディ状態にします。 この関数は、どの RFID リーダインターフェース関数よりも先にコールしなければなりません。

関数コール:

RFID_STATUS RFIDCreate(char *pszVer);

パラメータ:

pszVer – 文字列のポインタ、RFID リーダインターフェースのバージョンを含む。

戻り値:

RFID_STATUS

11.2.2 RFID リーダインターフェースのシャットダウン

関数の説明:

RFID リーダインターフェースが内部に保持しているリソースを適切にリリースします。リソースのリークを避けるために、アプリケーションは、アプリケーションを終了する前に適切に RFID リーダをシャットダウンしなければなりません。

関数コール:

RFID_STATUS RFIDDestroy();

パラメータ:

戻り値:

RFID_STATUS

11.3 RFID リーダの設定

RFID リーダが初期化された後で、RFID リーダをオープンし、動作環境を設定するために RFID リーダの構成パラメータを与えます。これらは、動作モード、データ応答フォーマット、RFID リーダの電源状態等があります。

11.3.1 RFID リーダをオープンする

関数の説明:

RFID リーダをオープンします。アプリケーションは、RFID リーダをコントロールする前にこの関数をコールしなければなりません。

関数コール:

RFID_STATUS RFIDOpen(int nRadio = 0);

パラメータ:

nRadio – 将来のために予約されています。

戻り値:

RFID_STATUS

11.3.2 RFID リーダをクローズする

関数の説明:

RFID コントロールをリリースします。

関数コール:

RFID_STATUS RFIDClose(int nRadio = 0);

パラメータ:

nRadio -将来のために予約されています。

戻り値:

RFID_STATUS

11.3.3 RFID リーダの動作モードを設定する

関数の説明:

RFID リーダモジュールは、連続モードまたは非連続モードのいずれかで動作します。 **連続モード**では、タグプロトコル動作サイクル(例、すべての有効なアンテナポートを一巡)が完了したとき、RFID リーダモジュールは最初に有効になったアンテナポートで新しいタグプロトコル動作サイクルを開始し、アプリケーションによって動作が明確にキャンセルされるまで動作を続けます。**非連続モード**は、RFID リーダモジュールで 1 回だけタグプロトコル動作サイクルが実行されます。

関数コール:

RFID_STATUS RFIDSetOperationMode(RFID_RADIO_OPERATION_MODE mode);

パラメータ:

mode – RFID リーダの動作モード。

戻り値:

RFID_STATUS

11.3.4 RFID リーダの動作モードを得る

関数の説明:

RFID リーダの動作モードを得ます。

関数コール:

RFID_RADIO_OPERATION_MODE RFIDGetOperationMode();

パラメータ:

なし

戻り値:

RFID_STATUS

11.3.5 RFID リーダの応答データのモードをセット

関数の説明:

アプリケーションが、タグアクセス動作に関するデータレポートのモードをコントロールすることができます。標準では、レポートモードは "Normal" にセットされています。
RFID18K6CReaderAPI.dll は、 “Compact” と “Normal” フォーマットのみをサポートしています。**Compact モードは**、タグプロトコル動作の結果をアプリケーションに戻すために必要な最低限のデータを含んでいます。**Normal モード**は、アプリケーションが検出できた動作の結果のステータス/前後の情報を加えることによりコンパクトモードのデータより多くなっています。例えば、インベントリの開始、新しいアンテナが使用された時間等です。

関数コール:

RFID_STATUS RFIDSetResponseMode(RFID_RESPONSE_MODE mode);

パラメータ:

mode – 要求するデータレポートモード。“Normal” または “Compact”。

戻り値:

RFID_STATUS

11.3.6 RFID リーダの応答データのモードを得る

関数の説明:

タグアクセス動作についてのデータレポートのモードを得ます。

関数コール:

RFID_STATUS RFIDGetResponseMode(RFID_RESPONSE_MODE *pMode);

パラメータ:

pMode – データレポートモードを含む RFID_RESPONSE_MODE のポインタ。

戻り値:

RFID_STATUS_OK

11.3.7 RFID リーダの電源状態をセット

関数の説明:

RFID リーダモジュールの電源状態をセットします(アンテナの出力ではありません。)。

関数コール:

RFID_STATUS RFIDSetPowerState(RFID_RADIO_POWER_STATE state)

パラメータ:

state - RFID リーダモジュールの電源状態

戻り値:

RFID_STATUS_OK

11.3.8 RFID リーダの電源状態を得る

関数の説明:

RFID リーダの電源の状態を得ます(アンテナの出力ではありません)。

関数コール:

RFID_STATUS RFIDGetPowerState(RFID_RADIO_POWER_STATE *pState)

パラメータ:

pState – RFID リーダモジュールの電源状態を含む RFID_RADIO_POWER_STATE のポインタ

戻り値:

RFID_STATUS

11.3.9 RFID リーダの低レベルパラメータをセット

関数の説明:

RFID リーダモジュールの低レベル構成パラメータをセットします。例、MAC レジストリ値。

関数コール:

RFID_STATUS RFIDSetConfiguration パラメータ:(INT16U パラメータ:, INT32U value)

パラメータ:

パラメータ: – セットする構成パラメータ
value – 構成パラメータをセットするための値

戻り値:

RFID_STATUS

11.3.10 RFID リーダの低レベルパラメータを得る

関数の説明:

低レベル RFID リーダモジュール構成パラメータを得ます。

関数コール:

RFID_STATUS RFIDGetConfiguration パラメータ:(INT16U パラメータ:, INT32U *pValue)

パラメータ:

パラメータ: – 得られるパラメータ

pValue – 構成パラメータの値を含む変数のポインタ

戻り値:

RFID_STATUS

11.4 アンテナポートの構成

RFID リーダモジュールは、一つもしくは複数の論理的なアンテナポートの使用をサポートしており、それぞれは物理的な送信ポートと物理的な受信ポートにマップされています。アプリケーションは、ステータスといくつかの構成パラメータを得ることができます。これらは、RFID モジュールを有効/無効にする、電源レベル、滞留(Dwell) 時間、インベントリサイクル数、アンテナポートの論理-物理マッピング等です。以下の説明をお読み下さい。

11.4.1 アンテナを無効/有効にする

関数の説明:

RFID リーダモジュールのアンテナポートの状態をセットします。

関数コール:

RFID_STATUS RFIDSetAntennaPortState(INT32 antennaPort,
RFID_ANTENNA_PORT_STATE state)

パラメータ:

antennaPort – 有効または無効にする論理アンテナポート
State – 論理アンテナポートの新しい状態

戻り値:

RFID_STATUS

11.4.2 RFID リーダのアンテナポートステータスを得る

関数の説明:

RFID リーダモジュールのアンテナポートのステータスを得ます。

関数コール:

RFID_STATUS RFIDGetAntennaPortStatus(INT32U antennaPort,
RFID_ANTENNA_PORT_STATUS *pStatus);

パラメータ:

antennaPort – ステータスを得る論理アンテナポート。
Pstatus – アンテナポートのステータスが含まれる構造体のポインタ。 NULL ではいけません。

戻り値:

RFID_STATUS

11.4.3 アンテナポートパラメータを構成

論理アンテナポートについての構成を設定または取得するときに、アプリケーションは設定/取得するいくつかのパラメータを持っています。
RFID_ANTENNA_PORT_CONFIG 構造体をご覧下さい。

関数の説明:

アプリケーションは、一つの論理アンテナポートに対していくつかのパラメータ、例えば、 滞留(dwell) 時間、出力レベルを構成することができます。アプリケーションは、まずアンテナポートの現在の設定を取得して、その後変更した構造体の値をアップデートします。

関数コール:

RFID_STATUS RFIDSetAntennaPortConfiguration(INT32U　antennaPort, const RFID_ANTENNA_PORT_CONFIG *pConfig)

パラメータ:

antennaPort – 構成する論理アンテナポート
Pconfig – アンテナポートの構成パラメータが含まれる構造体のポインタ
NULL ではいけません。

戻り値:

RFID_STATUS

11.4.4　アンテナポートの構成を得る

関数の説明:

論理アンテナポートが無効になっていない場合、一つの論理アンテナポートの構成パラメータを得ます。例えば、 滞留(dwell) 時間、出力レベル、インベントリサイクル数です。

関数コール:

RFID_STATUS RFIDGetAntennaPortConfiguration(INT32U antennaPort, RFID_ANTENNA_PORT_CONFIG *pConfig);

パラメータ:

antennaPort – 構成された論理アンテナポート
Pconfig – アンテナポートの構成パラメータを含む構造体のポインタ

戻り値:

RFID_STATUS

11.5 ISO 1800-6C タグアクセス

インターフェースは、以下のタグアクセス動作をサポートします。

・インベントリ
・読み取り
・書き込み
・キル
・消去
・ロック

タグアクセスは三つの動作を組み合わせます。

タグ選択の基準を指定: アプリケーションは、アクセスコマンドを出す前に、グループ分けされ、論理的に区分されたタグの総数が必要になります。タグが区分された後、指定された動作がグループの一つに適用されます。アプリケーションはタグが識別(シンギュレーション)される前にタグの区分(パーティション)を実行するためにタグ選択基準を指定します。

識別後(ポストシンギュレーション)マッチマスクを適用: RFID リーダモジュールがタグを識別(シンギュレーション)した後で、アプリケーションが提供する識別後マッチマスク(アプリケーションの提供するポストシンギュレーション一致マスク)を識別したタグの EPC にさらにフィルタするためにオプションで適用することができます。

ISO 18000-6C アクセスコマンドを適用: オプションで提供された選択基準とポストシンギュレーションマッチマスクにマッチしたタグだけが、アクセスコマンドの適用を受けます。

注意: タグの読み取り、書き込み、キル、ロックの動作を行う場合、RFID リーダモジュールは最初に有効になった論理アンテナのみを使用します。

タグ動作関数

アプリケーションがインターフェースにタグ動作(例、インベントリ、読み取り、等)を出した場合、アプリケーション定義コールバック関数のポインタも提供します。インターフェースは順に RFID リーダに要求を出し、そして、アプリケーション定義コールバックより動作結果を戻します。 タグ動作関数は、ブロックまたは非ブロックモードのいずれかを実行します。**ブロックモード**では、関数はタグの動作の完了を待ち、そして LPACCESS_STATUS フィールドが動作結果を含みます。**非ブロックモード**では、この関数はすぐに戻り、そして動作結果はアプリケーション定義コールバック関数によって戻ります。現在は非ブロックモードのみサポートされています。

11.5.1 コールバック関数

LRESULT (CALLBACK* RFIDPROC)(HWND, UINT, WPARAM, LPARAM);

関数の説明:

アプリケーション定義のコールバック関数。

パラメータ:

HWND – Window がタグ動作を発する

UNIT – 未定義、将来使うために予約
WPARAM –未定義、将来使うために予約
LPARAM – 動作結果を含む ACCESS_STATUS 構造体のポインタ

戻り値:

なし

11.5.2 アンテナの応答ステータスを得る

関数の説明:

タグアクセス動作を実行した後で、アンテナのスタータスを得るためにこの関数を呼びます。ブロックモードでは、タグアクセス関数に続いてこの関数を呼びます。非ブロックモードでは、アプリケーション定義のコールバック関数でこの関数をコールします。

関数コール:

BOOL RFIDGetAntennaStatus(int nAntenna, LPANTENNA_STATUS lpAntennaStatus);

パラメータ:

nAntnna – どのアンテナのステータスを得るかを示します。
lpAntennaStatus –アンテナのステータスを含む ANTENNA_STATUS 構造体のポインタ。

戻り値:

TRUE/FALSE

11.5.3 タグアクセス応答データを得る

関数の説明:

タグアクセス動作を実行した後で、アクセス応答データを得るためにこの関数を呼びます。**ブロックモード**では、タグアクセス関数に続いてこの関数を呼びます。**非ブロックモード**では、アプリケーション定義のコールバック関数でこの関数をコールします。

関数コール:

BOOL RFIDGetAccessData(int nAntenna, int nIndex, LPACCESS_DATA lpAccessData);

パラメータ:

nAntenna – どのアンテナのアクセスデータを得るかを示します。
nIndex – アクセスデータのインデックス
lpAccessData – アクセスデータを含む ACCESS_DATA 構造体のポインタ。

戻り値:

TRUE/FALSE

11.5.4 タグ動作ストップカウントをセット

関数の説明:

タグ動作が適用されるタグの最大数をセットします。この数がゼロの場合、動作はすべての選択されたタグに適用されます。この数がゼロでない場合、アンテナポートの滞留(dwell) 時間とイ

ンベントリラウンドサイクルが適用されます。(バージョン 1.0 については、このフィールドは最大値が 1 です。)

関数コール:

void RFIDSetStopCount(int nStopCount);

パラメータ:

nStopCount – タグ動作が適用されるタグの最大数。標準値はゼロです。

戻り値:

なし

11.5.5 タグの動作ストップカウントを得る

関数の説明:

タグ動作が適用されるダグの最大数を得ます。詳細については、 “11.5.4 タグ動作ストップカウントをセット” をご覧下さい。

関数コール:

int RFIDGetStopCount();

パラメータ:

なし

戻り値:

int – タグの最大数

11.5.6 タグインベントリ操作

関数の説明:

対象となる全てのタグについてタグインベントリを実行します。選択基準とポストシンギュレーションが指定された場合、タグは最初に区分されます。

関数コール:

void RFIDInventory(RFID_INVENTORY stInventory,
LPACCESS_STATUS lpAccessStatus, BOOL bBlock = FALSE, int nTimeout = 3000);

パラメータ:

stInventory – インベントリ動作パラメータ
lpAccessStatus - 動作結果を含む。ブロックモードで使用
bBlock – ブロックモード
nTimeout –将来のために予約。非ブロックモードに対して使用

戻り値:

なし

11.5.7　タグ読み取り動作

関数の説明:

タグのメモリバンクのいずれかから一つまたは複数の 16 ビットワードを読みます。一方、読み込みはタグ EPC データのセットを得るために使用されます。もし、EPC だけが欲しいデータであれば、インベントリ動作を行うとより効率的です。 (読み取りは 16 ビットバウンダリーでのみ、そして 16 ビットワードの複数で行われます。) オフセット/カウントの組み合わせによって指定された一つまたは複数のメモリワードがないか、読み取りロックされていた場合、タグからの読み取りは失敗します。

関数コール:

void RFIDTagRead(RFID_READ stRead, LPACCESS_STATUS lpAccessStatus, BOOL bBlock = FALSE, int nTimeout = 3000);
void RFIDTagReadEx(RFID_READ_EX stReadEx, LPACCESS_STATUS lpAccessStatus, BOOL bBlock = FALSE, int nTimeout = 3000);

パラメータ:

stRead/stReadEx – 読み込み動作パラメータ
lpAccessStatus –　操作結果を含む。ブロックモードで使用
bBlock - ブロックモード
nTimeout -　アクセス操作のタイムアウト。ブロックモードで使用

戻り値:

なし

11.5.8　タグ書き込み動作

関数の説明:

指定したメモリバンクに一つまたは複数の 16 ビットワードを書き込みます。(書き込みは、指定した 16 ビットオフセットでのみ始まり、書き込みの 16 ビットワードの最大数は 8 です。)
タグに一度に 8 つの 16 ビットワード以上書き込みたい場合、RFIDTagWriteEx を使用して下さい。

関数コール:

void RFIDTagWrite(RFID_WRITE stWrite, LPACCESS_STATUS lpAccessStatus, BOOL bBlock = FALSE, int nTimeout = 3000);
void RFIDTagWriteEx(RFID_WRITE_EX stWriteEx, LPACCESS_STATUS lpAccessStatus, int nTimeout = 3000);

パラメータ:

stWrite/stWriteEx　– 読み込み動作パラメータ
lpAccessStatus – 操作結果を含む。ブロックモードで使用
bBlock - ブロックモード
nTimeout -　アクセス操作のタイムアウト。非ブロックモードに対して使用。

戻り値:

なし

11.5.9 EPC 操作の変更

関数の説明:

ターゲットタグの EPC を変更する。

関数コール:

void RFIDTagWriteEPC(RFID_WRITE_EPC stEPC, LPACCESS_STATUS lpAccessStatus, int nTimeout = 3000);

パラメータ:

stEPC – EPC 操作パラメータの変更を含む
lpAccessStatus – 操作結果を含む
nTimeout – アクセス操作のタイムアウト

戻り値:

なし

11.5.10 タグのキル操作

関数の説明:

対象のタグをキルします。

関数コール:

void RFIDTagKill(RFID_KILL stKill, LPACCESS_STATUS lpAccessStatus, BOOL bBlock = FALSE, int nTimeout = 3000);
void RFIDTagKillEx(RFID_KILL_EX stKillEx, LPACCESS_STATUS lpAccessStatus, BOOL bBlock = FALSE, int nTimeout = 3000);

パラメータ:

stKill/stKillEx – キル操作 パラメータ
lpAccessStatus –操作結果を含む。ブロックモードで使用
bBlock – ブロックモード
nTimeout – アセス操作のタイムアウト、ブロックモードで使用

戻り値:

RFID_STATUS

11.5.11 タグロック操作

関数の説明:

タグロックを実行します。(タグのアクセス許可を設定)

関数コール:

void RFIDTagLock(RFID_LOCK stLock, LPACCESS_STATUS lpAccessStatus, BOOL bBlock = FALSE, int nTimeout = 3000);
void RFIDTagLockEx(RFID_LOCK_EX stLockEx, LPACCESS_STATUS lpAccessStatus, BOOL bBlock, int nTimeout);

パラメータ:

stLock/stLockEx – ロック操作 パラメータ。
lpAccessStatus – 操作結果を含みます。ブロックモードで使用。
bBlock – ブロックモード 。
nTimeout – アクセス操作のタイムアウト、ブロックモードで使用。

戻り値:

RFID_STATUS

11.5.12 タグのプレシンギュレーション操作

関数の説明:

ISO 18000-6C 選択コマンドについてのタグ選択基準を構成します。このコマンドは、タグアクセス操作の前に出さなければなりません。タグ選択基準は、次のタグ選択基準のコールがあるまで有効なままです。

関数コール:

RFID_STATUS RFIDTagSelectCriteria(RFID_SELECT_CRITERIA *pCriteria, int *pnCount, BOOL bSet = TRUE);

パラメータ:

pCriteria – タグ選択基準 パラメータを含む
pnCount – 基準の数。値は 0 と 8 の間に入らなければなりません。 0 をセットした場合、タグ選択基準は削除されます。
bSet – TRUE は、タグ選択基準パラメータをセットします。
FALSE は、タグ選択基準パラメータを戻します。

戻り値:

RFID_STATUS

11.5.13 タグのポストシンギュレーション操作

関数の説明:

RFID 無線モジュールで使用されるポストシンギュレーションの一致基準を構成します。アプリケーションは、タグの EPC の全てもしくは一部を元にしてタグをフィルタするためにポストシンギュレーションを使用することが出来ます。ポストシンギュレーションの一致基準は、次のポストシンギュレーションコールまで有効なままです。(すべてのタグアクセス操作は、シンギュレートされたもののみに適用されます。)

関数コール:

RFID_STATUS RFIDTagPostSingulation(RFID_POST_SINGULATION *pCriteria, int *pnCount, BOOL bSet = TRUE);

パラメータ:

pCriteria – ポストシンギュレーション一致基準パラメータを含む。
pnCount – 基準の数。ポストシンギュレーション一致基準をセットする場合、この値は 1 でなければなりません。ポストシンギュレーション一致基準を削除する場合、この値は 0 でなければなりません。
bSet – TRUE ポストシンギュレーションパラメータをセットします。
FALSE ポストシンギュレーションパラメータを戻します。

戻り値:

RFID_STATUS

11.5.14 タグクエリー操作

関数の説明:

どのタググループが引き続いてアクセス操作を適用するかを指定します。

関数コール:

RFID_STATUS RFIDTagQueryGroup(RFID_18K6C_TAG_GROUP *pGroup, BOOL bSet = TRUE);

パラメータ:

pGroup – タググループ指定のパラメータを含みます。
bSet – TRUE タググループを指定
FALSE タググループを戻す

戻り値:

RFID_STATUS

11.5.15 現在のシンギュレーションアルゴリズムをセット

関数の説明:

現在のシンギュレーションアルゴリズムを選択します。使用するシナリオによっては、別のシンギュレーションアルゴリズム(例. Q-adustment) が求められます。

関数コール:

RFID_STATUS RFIDSingulationAlgorithm(RFID_18K6C_SINGULATION_ALGORITHM * pAlgorithm, BOOL bSet = TRUE);

パラメータ:

pAlgorithm – 有効なシンギュレーションアルゴリズム
bSet – TRUE 現在のシンギュレーションアルゴリズムをセット
FALSE 現在のシンギュレーションアルゴリズムを戻す

戻り値:

RFID_STATUS

11.5.16 シンギュレーションアルゴリズムパラメータを指定する

関数の説明:

特定のシンギュレーションアルゴリズムに対して設定を構成することをアプリケーションに許します。

関数コール:

RFID_STATUS RFIDSingulationAlgorithmParameters(
RFID_18K6C_SINGULATION_ALGORITHM algorithm, void *pParms, BOOL bSet = TRUE);

パラメータ:

algorithm – 構成するシンギュレーションアルゴリズム
pParms – シンギュレーションアルゴリズムパラメータを含む構造体のポインタ
bSet – 指定したシンギュレーションアルゴリズムパラメータをセット、または戻す

戻り値:

RFID_STATUS

11.5.17 タグ操作のキャンセル

関数の説明:

RFID リーダで現在行われているタグの操作を停止します。

関数コール:

RFID_STATUS RFIDCancelOperation();

パラメータ:

なし

戻り値:

RFID_STATUS

11.5.18 タグ操作の中止

関数の説明:

タグの操作を即時に中止します。応答パケットは放棄されます。

関数コール:

RFID_STATUS RFIDAbortOperation();

パラメータ:

なし

戻り値:

RFID_STATUS

11.5.19 RFID リーダモジュールのエラー状態をクリア

関数の説明:

RFID リーダモジュールの MAC ファームウェアのエラー状態をクリアします。

関数コール:

RFID_STATUS RFIDClearError();

パラメータ:

なし

戻り値:

RFID_STATUS

11.5.20 重複タグの保持または放棄

関数の説明:

インベントリを行ったときに重複タグを保持または放棄します。インベントリの前にこの関数をコールします。

関数コール:

RFID_STATUS RFIDEnableDuplicate(BOOL bEnable = TRUE);

パラメータ:

bEnable – TRUE 重複タグを保持
FALSE 重複タグを放棄

戻り値:

RFID_STATUS

11.6 その他の API

11.6.1 RFID リーダのファームウェアバージョンを得る

関数の説明:

RFID リーダのライブラリとファームウェアバージョンを得ます。

関数コール:

RFID_STATUS RFID MacGetVersion(char *pszVer);

パラメータ:

pszVer - RFID リーダのライブラリとファームウェアバージョンを含みます。

戻り値:

RFID_STATUS

11.7 ライブラリの構造体

11.7.1 RFID_ANTENNA_PORT_CONFIG

説明:

論理アンテナポートに対する構成パラメータ。

定義:

```
typedef struct {
    INT32U  length;
    INT32U  powerLevel;
    INT32U  dwellTime;
    INT32U  numberInventoryCycles;
    INT32U  physicalRxPort;
    INT32U  physicalTxPort;
    INT32U  antennaSenseThreshold;
    } RFID_ANTENNA_PORT_CONFIG;
```

フィールド:

length – 構造体のバイト長。アプリケーションで、sizeof(RFID_ANTENNA_PORT_CONFIG) をセットしなければなりません。
powerLevel – 論理アンテナポートの物理送信アンテナの電力レベル。この値は 0.1 (1/10) dBm 単位で指定します。 値は、0 と 300 の間で、280 が最も適当な値です。
dwellTime – このアンテナポートの使用サイクル時間をミリ秒で指定します。ゼロはアンテナの使用が、 numberInventoryCycles フィールドでコントロールされることを示します。
numberInventoryCycles – このアンテナポートで実行するためのインベントリ連続数。ゼロはアンテナの使用が、dwellTime フィールドでコントロールされることを示します。
physicalRxPort – 論理アンテナポートに関連づけられた物理的な受信アンテナ。 0 と 3 の間。
physicalTxPort – 論理アンテナポートに関連づけられた物理的な送信アンテナ。 0 と 3 の間。
antennaSenseThreshold – 測定した抵抗値、オームで指定。

注意: *バージョン 1.0 では、physicalRxPort と physicalTxPort の値は、同じでなければなりません。 *dwellTime と numberInventoryCycles は両方ともゼロではいけません。 *length field は config 関数を呼ぶ前に構造体の長さを埋めなければなりません。

11.7.2 ACCESS_STATUS

説明:

タグ動作結果の情報を含みます。

定義:

```
typedef struct ACCESS_STATUS_TAG {
FILETIME ftStartTime;
FILETIME ftEndTime;
 INT32U dwResponseMode;
 INT32U dwOperationMode;
 INT16U unCommand;
 DWORD dwErrorCode;
 DWORD dwStatus;
 INT16U unAntennas;
 }ACCESS_STATUS, *LPACCESS_STATUS;
```

フィールド:

tStartTime – タグの動作開始時間。
ftEndTime – タグの動作終了時間。
dwResponseMode – 動作応答データレポートモードが Compat または Normal。
dwOperationMode – RFID リーダモジュールの動作モードが連続または、非連続。
dwErrorCode – タグ動作に失敗した場合、このフィールドはエラーコードを含みます。ゼロはエラーが無いことを示します。
dwStatus – タグ動作関数が示すライブラリステータスとエラーコードによって戻る。
unAntennas – アンテナ数。

11.7.3 ANTENNA_STATUS

説明:

タグ動作の実行に影響するアンテナの情報を含みます。

定義:

```
typedef struct ANTENNA_STATUS_TAG{
FILETIME ftStartTime;
FILETIME ftEndTime;
INT16U  unAntenna;
DWORD  dwErrorCode;
DWORD  dwStatus;
INT16U   unCount
```

} ANTENNA_STATUS, *LPANTENNA_STATUS;

フィールド:

ftStartTime – このアンテナがタグ動作を開始する時間。
ftEndTime – このアンテナがタグ動作を終了する時間。
unAntenna – アンテナ番号。
unCount – このアンテナの戻すタグの数。

11.7.4 ACCESS_DATA

説明:

アンテナによって戻るタグのアクセスデータを含みます。

定義:

```
typedef struct ACCESS_ DATA _TAG{
INT16U unEPCLength;
INT16U unRSSI;  //受信信号強度
BYTE pnEPC[68];       //PC + EPC + CRC
INT16U  unDataLength;
BYTE pnData[256];              //アクセスデータまたはアクセスステータス
} ACCESS_DATA, *LPACCESS_DATA;
```

フィールド:

unEPCLength – EPC の長さ(PC と CRC を含む)。
unRSSI – 受信信号強度
pnEPC[68] – EPC のデータ(2 バイト PC + EPC + 2 バイト CRC)
unDataLength – アクセスデータ長
pnData[256] – タグ書き込みのタグから得られるデータ

11.7.5 RFID_INVENTORY

説明:

インベントリ動作パラメータ。

定義:

```
typedef struct RFID_INVENTORY_TAG{
        HWND hWnd; //親ウインドウ
        RFIDPROC lpfnStartProc;
        RFIDPROC lpfnStopProc;
        }RFID_INVENTORY;
```

フィールド:

hWnd – タグ動作を発する親ウインドウ。
lpfnStartProc – アプリケーション定義コールバック関数。RFID リーダインターフェースはタグ動作を実行する前にこの関数を呼びます。
lpfnStopProc – アプリケーション定義コールバック関数。RFID リーダインターフェースはタグ動作の終了後にこの関数を呼びます。

11.7.6 RFID_READ

説明:

タグ読み取り動作パラメータ。

定義:

```
typedef struct RFID_READ_TAG{
        HWND hWnd; //親ウインドウ
        RFIDPROC lpfnStartProc;
        RFIDPROC lpfnStopProc;
        RFID_18K6C_MEMORY_BANK bank;
         INT16U offset;
         INT16U count;
                INT32U accessPassword;
        }RFID_READ;
```

フィールド:

hWnd - タグ動作を発する親ウインドウ。
lpfnStartProc - アプリケーション定義コールバック関数。RFID リーダインターフェースはタグ動作を実行する前にこの関数を呼びます。
lpfnStopProc - アプリケーション定義コールバック関数。RFID リーダインターフェースはタグ動作の終了後にこの関数を呼びます。
bank – 読み出しを開始するメモリバンク。
offset –指定したメモリバンクから読み出すための最初の 16 ビットワードのオフセット。
count – 読み出す 16 ビットワード数。(この値がゼロで、バンクが EPC の場合、読み取りはオフセットによって指定された 16 ビットワードで、EPC を開始し EPC の最後までの内容を戻します。

この値は、1 と 255 の間でなければなりません。)
accessPassword – タグのアクセスパスワード。ゼロの値はアクセスパスワードが無いことを示します。

11.7.7 RFID_READ_EX

説明:

タグ読み込み拡張関数のパラメータ。

定義:

```
typedef struct RFID_READ_EX_TAG {
        HWND hWnd;  //Parent window
        RFIDPROC lpfnStartProc;
        RFIDPROC lpfnStopProc;
        RFID_18K6C_MEMORY_BANK bank;
        INT16U offset;
        INT16U count;
        BYTE accessPassword[8];
}RFID_READ_EX;
```

フィールド:

hWnd - RFID_READ と同じ
lpfnStartProc – RFID_READ と同じ
lpfnStopProc – RFID_READ と同じ
bank – RFID_READ と同じ
offset – RFID_READ と同じ
count – RFID_READ と同じ
accessPassword[8] – タグのアクセスパスワード。8 バイト 16 進文字(0~f)

11.7.8 RFID_WRITE

説明:

タグの書き込み操作パラメータ。

定義:

```
  typedef struct RFID_WRITE_TAG{
        HWND hWnd; //親ウインドウ
        RFIDPROC lpfnStartProc;
        RFIDPROC lpfnStopProc;
        BOOL32 verify;
        INT32U verifyRetryCount;
        RFID_18K6C_MEMORY_BANK bank;
        INT32U accessPassword;
        INT16U offset;
        INT16U count;
```

```
        INT16U pnData[8];
}RFID_WRITE;
```

フィールド:

hWnd – タグ動作を発する親ウインドウ。
lpfnStartProc – アプリケーション定義コールバック関数。RFID リーダインターフェースは、タグ動作を実行する前にこの関数を呼びます。
lpfnStopProc – アプリケーション定義コールバック関数。RFID リーダインターフェースはタグ動作の終了後この関数を呼びます。
verify – タグに書き込まれたデータは正しく書き込まれたことを検証するためにタグから読み出されなければならないことを示すフラグ。ゼロでない値は、タグのメモリが検証のために読み出さなければならないことを示しています。
verifyRetryCount – 書き込み-検証に失敗した場合、書き込みをリトライする回数の最大値。この値は 0 と 7 の間でなければなりません。
accessPassword – タグのアクセスパスワード。ゼロの値はアクセスパスワードが無いことを示します。
bank – 書き込み開始のメモリバンク。
offset – 指定したメモリバンクから書き込むための最初の 16 ビットワードのオフセット。count –読み出す 16 ビットワード数。(この値がゼロで、バンクが EPC の場合、書き込みはオフセットによって指定された 16 ビットワードで、EPC を開始し EPC の最後までの内容を戻します。この値は、1 と 8 の間でなければなりません。)
pnData[8] – タグの指定したメモリバンクに書き込むデータを含みます。pnData[n] の上位バイトは、16 ビットオフセット(offset + n)でタグのメモリバンクに書き込まれます。 下位バイトは次のバイトに書き込みます。

11.7.9 RFID_WRITE_EX

説明:

タグ書き込み関数のパラメータ

定義:

```
typedef struct RFID_WRITE_EX_TAG {
        HWND hWnd; //親ウインドウ
        RFIDPROC lpfnStartProc;
        RFIDPROC lpfnStopProc;
        BOOL32 verify;
        INT32U verifyRetryCount;
        BYTE accessPassword[8];      //パスワードは 8 バイト 16 進文字(0~f)
        BYTE pnEPC[64];
        RFID_18K6C_MEMORY_BANK bank;
        INT16U offset;
        INT16U count;
        Int16U pnData[256];  //書き込むデータ(16 進数)
}RFID_WRITE_EX;
```

フィールド:

hWnd – RFID_WRITE と同じ
lpfnStartProc - RFID_WRITE と同じ
lpfnStopProc - RFID_WRITE と同じ
verify – RFID_WRITE と同じ
verifyRetryCount – RFID_WRITE と同じ
accessPassword[8] – タグのアクセスパスワード。8 バイトの 16 進文字(0~f)。
pnEPC[64] – 指定したタグに書き込む EPC。この パラメータがヌルの場合、データはフィールドにあるタグに書き込まれます。そうでなければ、データは EPC によって指定されたタグに書き込まれます。
bank – RFID_WRITE と同じ
offset – RFID_WRITE と同じ
count – RFID_WRITE と同じ
pnData[256] – タグの指定したメモリバンクに書き込まれたデータを含みます。(HEX 文字列)

11.7.10 RFID_WRITE_EPC

説明:

タグの EPC 変更の構造体。

定義:

```
typedef struct RFID_WRITE_EPC_TAG {
        HWND hWnd;          //親ウインドウ
        RFIDPROC lpfnStartProc;
        RFIDPROC lpfnStopProc;
        BOOL32 verify;
        INT32U verifyRetryCount;
        BYTE accessPassword[8];
        BYTE pnOldEPC[64];
        INT16U tagType;
        BYTE pnNewEPC[64];
    }RFID_WRITE_EPC;
```

フィールド:

hWnd - RFID_WRITE と同じ
lpfnStartProc – RFID_WRITE と同じ
lpfnStopProc – RFID_WRITE と同じ
verify – RFID_WRITE と同じ
verifyRetryCount – RFID_WRITE と同じ
accessPassword[8] – RFID_WRITE_EX と同じ
pnOldEPC[64] – 変更する古いタグの EPC
tagType – 将来のために予約
PnNewEPC[64] – ターゲットタグの新しい EPC

11.7.11 RFID_KILL

説明:

キルタグの構造体.

定義:

```
typedef struct RFID_KILL_TAG {
                HWND hWnd;  //親ウインドウ
                RFIDPROC lpfnStartProc;
                RFIDPROC lpfnStopProc;
                INT32U accessPassword;
                INT32U killPassword;
    }RFID_KILL;
```

フィールド:

hWnd –タグ 操作を出す親ウインドウ。

lpfnStartProc – アプリケーション定義のコールバック関数。RFID リーダは、タグ操作の前にこの関数をコールします。

lpfnStopProc – アプリケーション定義のコールバック関数。RFID リーダインターフェースはタグ操作完了の後にこの関数をコールします。

accessPassword – タグのアクセスパスワード。ゼロの値はアクセスパスワードがないことを表します。

killPassword – ダグのキルパスワード。ゼロでなければなりません。

11.7.12 RFID_KILL_EX

説明:

キルタグの拡張関数の パラメータ。

定義:

```
typedef struct RFID_KILL_EX_TAG {
        HWND hWnd; //親ウインドウ
        RFIDPROC lpfnStartProc;
        RFIDPROC lpfnStopProc;
        BYTE accessPassword[8]; //パスワードは 8 バイトの 16 進数文字(0~f)
        BYTE killPassword[8];   //パスワードは 8 バイトの 16 進数文字(0~f)
    }RFID_KILL_EX;
```

フィールド:

hWnd - RFID_ KILL と同じ

lpfnStartProc – RFID_ KILL と同じ

lpfnStopProc – RFID_ KILL と同じ

accessPassword[8] – タグのアクセスパスワード。8 バイト 16 進文字(0~f)

killPassword[8] – タグのキルパスワード。 8 バイトの 16 進文字(0~f)

11.7.13 RFID_LOCK

説明:

タグロック関数の構造体。

定義:

```
typedef struct RFID_LOCK_TAG {
        HWND hWnd; //親ウインドウ
        RFIDPROC lpfnStartProc;
        RFIDPROC lpfnStopProc;
        INT32U accessPassword;
        INT32U killPasswordPermissions;
        INT32U accessPasswordPermissions;
        INT32U epcBankPermissions;
        INT32U tidBankPermissions;
        INT32U userBankPermissions;
    }RFID_LOCK;
```

フィールド:

hWnd –タグ操作を発する親ウインドウ

lpfnStartProc – アプリケーション定義のコールバック関数。 RFID リーダインターフェースはタグ操作の実行前にこの関数をコールします。

lpfnStopProc – アプリケーション定義のコールバック関数。RFID リーダインターフェースは、タグ操作の終わった後、この関数をコールします。

accessPassword – タグのアクセスパスワード(ゼロはアクセスパスワードのないことを示します)。

killPasswordPermissions – タグのキルパスワードに対するアクセス許可

accessPasswordPermissions – タグのアクセスパスワードに対するアクセス許可

epcBankPermissions - タグの EPC メモリバンクに対するアクセス許可

tidBankPermissions - タグの TID メモリバンクに対するアクセス許可

userBankPermissions - タグの USER メモリバンクに対するアクセス許可

アクセス許可は以下のようにセットすることが出来ます。

名称	値	説明
許可	0	パスワード許可(Password permission): 読み込みと書き込みが可能。 バンク許可(Bank permission): メモリバンクは書き込み可能。
常に許可	1	パスワード許可(Password permission): 読み込みと書き込みが可能。 バンク許可(Bank permission): メモリバンクは書き込み可能。 この許可は常にセットされます。
パスワード保護	2	パスワード許可(Password permission): 読み込みと書き込みにパスワードが必要。 バンク許可: メモリバンクの書き込みにパスワードが必要。
常に拒否	3	パスワード許可(Password permission): 読み書き不可。 キルパスワード許可がこの値にセットされた場合、タグはキルすることができません。 バンク許可(Bank permission): メモリバンクは書き込みできません。 この許可は常にセットされます。
変化なし	4	許可は変更されません。

アクセス許可値の表

11.7.14 RFID_LOCK_EX

説明:

ロック拡張関数の パラメータ。

定義:

```
typedef struct RFID_LOCK_EX_TAG {
        HWND hWnd; //親ウインドウ
        RFIDPROC lpfnStartProc;
        RFIDPROC lpfnStopProc;
        BYTE accessPassword[8];
        INT32U killPasswordPermissions;
        INT32U accessPasswordPermissions;
        INT32U epcBankPermissions;
        INT32U tidBankPermissions;
        INT32U userBankPermissions;
    }RFID_LOCK_EX;
```

フィールド:

hWnd –RFID_LOCK と同じ
lpfnStartProc – RFID_LOCK と同じ
lpfnStopProc – RFID_LOCK と同じ
accessPassword[8] – タグのアクセスパスワード。 8 バイト 16 進文字(0~f)。
killPasswordPermissions - RFID_LOCK と同じ
accessPasswordPermissions - RFID_LOCK と同じ
epcBankPermissions - RFID_LOCK と同じ
tidBankPermissions - RFID_LOCK と同じ
userBankPermissions - RFID_LOCK と同じ

11.7.15 RFID_SELECT_CRITERIA

説明:

プレシンギュレーション一致基準の構造体。

定義:

```
typedef struct RFID_SELECT_CRITERIA_TAG {
        RFID_18K6C_MEMORY_BANK bank;
        INT32U offset;
        INT32U count;
        INT8U mask[RFID_18K6C_SELECT_MASKBYTE_LEN];
        RFID_18K6C_TARGET target;
        RFID_18K6C_ACTION action;
        BOOL32 enableTruncate;
 }RFID_SELECT_CRITERIA
```

フィールド:

bank – 一致するメモリバンク
offset – 一致する最初のビットのオフセット
count – マスク中のビット数
mask[RFID_18K6C_SELECT_MASK_BYTE_LEN] – 一致するビットパターン
target - 動作(S0~S4,SL)によって影響されるもの
action – 選択中にタグの数で実行される動作 (これは、一致もしくは非一致)
enableTruncate – タグがシンギュレーションされたときに EPC はトランケートされなければならないか? ゼロでない値は、EPC がトランケートされたことを示します。 enableTruncate が true の場合: バンクは EPC でなければなりません; ターゲットは RFID_18K6C_TARGET_SELECTED_FLAG でなければなりません。

11.7.16 RFID_POST_SINGULATION

説明:

ポストシンギュレーション一致基準の構造体。

定義:

```
typedef struct RFID_POST_SINGULATION_TAG {
        INT32U offset;
        INT32U count;
        INT8U mask[RFID_18K6C_SINGULATION_MASK_BYTE_LEN];
        BOOL32 match;
    }RFID_POST_SINGULATION;
```

フィールド:

offset – EPC の始めからのオフセットビット
count –マスクのビット数。ゼロの長さは、すべての EPC に一致します。もし (offset + count)がマスクの最後を越えると、タグは一致しないと考えられます。(有効な値は 0 から 396)。
mask[RFID_18K6C_SINGULATION_MASK_BYTE_LEN] – 一致するビットパターン(16 進文字)
match – 関係するタグ操作がマスクに一致するタグに適用されるかどうかを判断します。

11.8 エラーコード

二種類のエラーコードがあります。一つは、RFID ライブラリによって返され、RFID ライブラリのステータスを示し、他方は RFID モジュールファームウェアによって返され、アクセス操作の結果を含んでいます。

RFID ライブラリステータスとエラーコード:

名前	値	
RFID_STATUS_OK	0	成功
RFID_ERROR_ALREADY_OPEN	-9999	すでにオープンしている無線をオープンしようとした
RFID_ERROR_BUFFER_TOO_SMALL	-9998	与えられたバッファが小さすぎる
RFID_ERROR_FAILURE	-9997	一般的な異常
RFID_ERROR_DRIVER_LOAD	-9996	無線バスドライバの読み込み異常
RFID_ERROR_DRIVER_MISMATCH	-9995	ライブラリがシステムにある無線バスドライバのバージョンを使用することが出来ない
RFID_ERROR_EMULATION_MODE	-9994	ライブラリがエミュレーションモードにある間操作を実行することが出来ない
RFID_ERROR_INVALID_ANTENNA	-9993	アンテナ番号が無効
RFID_ERROR_INVALID_HANDLE	-9992	提供された無線ハンドルが無効
RFID_ERROR_INVALID_パラメータ	-9991	関数のパラメータの一つが無効
RFID_ERROR_NO_SUCH_RADIO	-9990	存在しない無線をオープンしようとした
RFID_ERROR_NOT_INITIALIZED	-9989	ライブラリは正しく初期化されなかった
RFID_ERROR_NOT_SUPPORTED	-9988	サポートされていない関数
RFID_ERROR_OPERATION_CANCELLED	-9987	操作はキャンセル操作をコールすることによってキャンセルされ、無線をクローズまたはライブラリをシャットダウンする
RFID_ERROR_OUT_OF_MEMORY	-9986	ライブラリはメモリ割り付けのエラーが発生した
RFID_ERROR_RADIO_BUSY	-9985	無線がビジー中であるので、操作を実行することが出来ない
RFID_ERROR_RADIO_FAILURE	-9984	対象の無線モジュールでエラーが発生
RFID_ERROR_RADIO_NOT_PRESENT	-9983	無線がシステムから切り離された
RFID_ERROR_CURRENTLY_NOT_ALLOWED	-9982	RFID 関数は現在使用できません。
RFID_ERROR_RADIO_NOT_RESPONDING	-9981	無線モジュールの MAC ファームウェアは、要求に応答しない

RFID_ERROR_NONVOLATILE_INIT_FAILED	-9980	MAC ファームウェアは非揮発メモリ更新を初期化中にエラーを発生。 MAC ファームウェアは無線モジュールをリセットせずに通常のアイドル状態に戻る
RFID_ERROR_NONVOLATILE_OUT_OF_BOUNDS	-9979	無線モジュールの非揮発メモリアドレスの有効な範囲外のアドレスにデータを書き込もうとした
RFID_ERROR_NONVOLATILE_WRITE_FAILED	-9978	MAC ファームウェアは、無線モジュールの非揮発メモリの領域に書き込み中にエラーを発生した。
RFID_ERROR_RECEIVE_OVERFLOW	-9977	トランスポートレイヤーは、入ってくるデータの 1 または複数バイトが落ちる事によるオーバーフローエラーを検出しました。動作は中止され、パイプラインのすべてのデータは捨てられます。

アクセス操作の結果:

値	説明
0x00	成功
0x01	書き込み後の読み出し検証で異常
0x02	タグコマンド送信中の問題
0x03	書き込みのタグ応答で CRC エラー
0x04	書き込みの確認時にパケット読み込みで CRC エラー
0x05	書き込み実行時の最大リトライ数
0x06	タグからデータを読み取り待機異常、タイムアウトの可能性
0x07	新しいタグハンドル要求中に異常
0x0A	タグ応答待ち中のエラー、タイムアウトの可能性
0x0B	キルするためのタグ応答時に CRC エラー
0x0C	タグキルの後半送信中に問題
0x0D	最初のキルコマンドでタグが無効なハンドルに応答
0xFA	タグはメモリ書き込みを実行するの十分なパワーがありません
0xFB	指定したメモリの場所は、ロックされているか、永久ロックされている
0xFC	指定したメモリの場所はありません
0xFD	タグはタイムアウト内に応答しません
0xFE	CRC は無効
0xFF	一般のエラー

12. .Net Compact Framework サポート

R1000ReaderCF.dll は、RFID18k6cReader.dll のネイティブ C/C++ DLL の C#ラップです。これは、RFID リーダをコントロールするための.NET コンパクトフレームワークの API を提供します。

12.1 “ R1000Reader” クラス

これはアプリケーションによって例示されるメインクラスです。これは RFID リーダにアクセスするためのプロパティとメソッドを提供します。R1000ReaderCF のメソッドは、ネイティブなものと同じ名前です。

12.2 プログラミングモデル

最初に、プロジェクトに R1000ReaderCF.dll の参照を追加します。

C# 例:

```
using Unitech.R1000.Reader;
using Unitech.R1000.Reader.Constants;
using Unitech.R1000.Reader.Structures;

//RFID ライブラリの作成と初期化
String strVersion = String.Empty;
R1000Reader.RFIDCreate(ref strVersion);
//RFID リーダをオープン
R1000Reader.RFIDOpen(0);
//インベントリとタグの EPC を取得する
RFID_INVENTORY stInventory = new RFID_INVENTORY();
ACCESS_STATUS stAccessStatus = new ACCESS_STATUS();
//operation in blocking mode
R1000Reader.RFIDInventory(stInventory, ref stAccessStatus, true, 3000);
if (stAccessStatus .dwStatus == 0 && stAccessStatus.dwErrorCode == 0)
{
   for (int i = 0; i < stAccessStatus.unAntennas; i++)
   {
      ANTENNA_STATUS stAntennaStatus = new ANTENNA_STATUS();
      R1000Reader.RFIDGetAntennaStatus(i, ref stAntennaStatus);
      for (int j = 0; j < stAntennaStatus.unCount; j++)
      {
         ACCESS_DATA accessData = new ACCESS_DATA();
         UInt32 nRet = R1000Reader.RFIDGetAccessData(i, j, ref accessData);
```

```
        if (nRet == 1 && accessData.unEPCLength > 0)
        {
                //Get EPC from accessData.pnEPC;
        }
      }
   }
}
```

//RFID リーダをクローズ

```
R1000Reader.RFIDClose(0);
```

//RFID ライブラリを消す

```
R1000Reader.RFIDDestroy();
```

13. SysIOAPI.DLL に含まれない便利な関数コール

以下の API は HT6xx/PA600/PA96X/PA982/RH767 を使用するのに便利でしょう。

13.1 ウォームブート, コールドブートと電源オフ

```
#include     <pkfuncs.h>
#include     "oemioctl.h"

 //ウォームブート
KernelIoControl(IOCTL_HAL_REBOOT, NULL, 0, NULL, 0, NULL);

// コールドブート
KernelIoControl(IOCTL_COLD_BOOT, NULL, 0, NULL, 0, NULL);

// 電源オフ
{
DWORD dwExtraInfo=0;
BYTE bScan=0;
keybd_event( VK_OFF, bScan, KEYEVENTF_SILENT, dwExtraInfo );
keybd_event( VK_OFF, bScan, KEYEVENTF_KEYUP, dwExtraInfo );
}
```

13.2 デバイス ID を得る

HT6xx/PA600/PA96X/PA982/RH767 にはユニークな ID が書き込まれており、ユーザは“ Func”+”9”キーを押してチェックすることができます。

デバイス ID 読み込みのためのサンプルコードは以下の通りです。

```
//////////////////////////////////////////////////////////////
TCHAR outBuf[512], szBuff[200];
DWORD bytesReturned;
TCHAR deviceID[64];
BYTE cProductID[255];

PDEVICE_ID pDeviceID = NULL;
pDeviceID = (PDEVICE_ID)outBuf;
pDeviceID->dwSize = sizeof(outBuf);

if (KernelIoControl(IOCTL_HAL_GET_DEVICEID, NULL, 0, outBuf, sizeof(outBuf),
&bytesReturned))
{
        memset(szBuff, 0, sizeof(szBuff));

        // Device ID for WinCE version
        memcpy((PBYTE)szBuff, (PBYTE)pDeviceID + pDeviceID->dwPresetIDOffset,
   pDeviceID->dwPresetIDBytes);
        swprintf(deviceID, _T("%s"), szBuff);


        // Device ID for Mobile version
        memset(cProductID, 0, sizeof(cProductID));
        memcpy((PBYTE) cProductID, (PBYTE)pDeviceID + pDeviceID->dwPresetIDOffset,
  pDeviceID->dwPresetIDBytes);
        MultiByteToWideChar(CP_ACP, 0, (char*)cProductID, strlen((char*)cProductID),
      szBuff, sizeof(szBuff));
        wcsncpy(deviceID, szBuff, 16);

    }
//////////////////////////////////////////////////////////
```

このコードは、 platformID に プラットフォーム IDを、 deviceID にデバイス IDを持ちます。

13.3 OEM 情報を得る

HT6xx/PA96x では、OEM ID はターミナルに書き込まれています。ユーザは “ Func” +” 9” を押してチェックすることができます。
OEM ID を読むサンプルコードは以下の通りです。

```
////////////////////////////////////////////////////////////////
    {
     TCHAR szBuff[500];

     ZeroMemory(szBuff, sizeof(szBuff));

     SystemParametesInfo(SPI_GETOEMINFO, 500, (LPVOID)szBuff, 0);

     MessageBox(szBuff);
    }
////////////////////////////////////////////////////////////
```

13.4 ファームウェアとブートローダのバージョン情報を得る

```
////////////////////////////////////////////////////////////////
{
        DEVICE_CONFIG cfg;
        if (KernelIoControl(IOCTL_GET_DEVICE_CONFIG, NULL, 0, &cfg, sizeof(cfg), NULL))
        {
                //Get OS Version
                mbstowcs(stringBuffer, cfg.swVersion, strlen(cfg.swVersion));
                swprintf(szProductID, _T("OS version : %s"), stringBuffer);
                Show(szProductID);

                //Get Bootloader Version
                memset (stringBuffer, 0, sizeof(stringBuffer));
                mbstowcs(stringBuffer, cfg.bootloadversion, strlen(cfg.bootloadversion));
                wsprintf(szProductID, _T("Bootloader version : %s"), stringBuffer);
                Show(szProductID);
        }
}
////////////////////////////////////////////////////////////////
```

14. カメラ関連の関数

以下の URL から SDK をダウンロードして下さい。
http://w3.tw.ute.com/pub/cs/SDK/Camera/CameraSDK.zip

ノート: この SDK は、PA550 と PA690 用です。

15. 指紋関連の関数

以下の URL からサンプルプログラムとマニュアルをダウンロードして下さい。
http://w3.tw.ute.com/pub/cs/software/Sample_Program/PA968/Fingerprint.zip

ノート: PA968 用です。

16. GPS 関連の関数

以下の URL からサンプルプログラムとマニュアルをダウンロードして下さい。(**Windows CE バージョンのみ**)

http://w3.tw.ute.com/pub/cs/software/Sample_Program/GPS/GPSSDK.zip

17. USI.NET Compact Framework コンポーネント

以下の URL からサンプルプログラムとマニュアルをダウンロードして下さい。
http://w3.tw.ute.com/pub/cs/SDK/USI/USICF.zip

18. USI ActiveX コントロール

以下よりバイナリファイルと html サンプルを入手して下さい。
http://w3.tw.ute.com/pub/cs/software/Sample_Program/USIActiveX/USIActiveX.zip

18.1 レジストリコントロール

a. Microsoft “ REGSVRCE.exe” をデバイスにコピー。
b. “ REGSVRCE.exe ScannerActiveX.dll” を実行してレジストリをコントロールします。
c. システムの変更を適用するためにウォームブートします。

18.2 html に組み込み

```
<OBJECT ID="Scanner"
    CLASSID="CLSID:E81DD955-9B99-4493-8035-355DFB5028D9"
    WIDTH=0 HEIGHT=0>
</OBJECT>
```

18.3 スクリプト言語による動作コントロール

a. スキャナを有効にする:

```
<SCRIPT LANGUAGE="Javascript">
    function OnRegister()   {   Scanner.Register=1   }
    function OnUnregister() {   Scanner.Register=0   }
    function OnEnable()     {   Scanner.Scan=1       }
    function OnDisable()    {   Scanner.Scan=0       }
</SCRIPT>

<INPUT NAME="REGISTER1" TYPE="BUTTON" VALUE="Register" onClick="OnRegister()" >
<INPUT NAME="ENABLE" TYPE="BUTTON" VALUE="Enable" onClick="OnEnable()" >
<INPUT NAME="DISABLE1" TYPE="BUTTON" VALUE="Disable" onClick="OnDisable()" >
<INPUT NAME="UNREGISTER" TYPE="BUTTON" VALUE="Unregister"
onClick="OnUnregister()" >
```

b. 1D デコーダ 設定の変更:

```
<SCRIPT LANGUAGE="Javascript">
    function OnUPCEnable()
    {
        Scanner.SetHamster(0x79,1);
    }

    function OnUPCDisable()
    {
        Scanner.SetHamster(0x79,0);
```

```
    }
</SCRIPT>

<INPUT NAME="UPC_Enable" TYPE="BUTTON" VALUE="UPC E Enable"
onClick="OnUPCEnable()" >
<INPUT NAME="UPC_Disable" TYPE="BUTTON" VALUE="UPC E Disable"
onClick="OnUPCDisable()" >
```

19. 32WAN GPRS ライブラリ

以下の URL からサンプルプログラムとマニュアルをダウンロードして下さい。(**Windows CE バージョンのみ**)

http://w3.tw.ute.com/pub/cs/software/Sample_Program/32WAN/32WAN_SDK.zip

20. アップデート情報

V1.0 最初のバージョン
V1.1 C# の URL リンク訂正
V1.2 PA982 サポート
V1.3 RH767 HF/UHF プログラミング追加
V1.4 RH767 HF プログラミング変更
V1.6 ロゴ変更
V1.7 SDK URL 変更
V1.8 RH767 UHF SkyeTek プログラミング追加
V1.9 HF API 追加
V1.10 PA968 サポート、カメラ、GPS と GPRS プログラミング
V1.11 HF API の更新
V1.12 HF マルチタグ API を除く
V1.13 Matrix 2 of 5 サポート追加、Toshiba code と共有
V1.14 ファームウェアとブートローダバージョンを得るための説明を追加
V1.15 RFID リーダ(Kitty)を追加
V1.16 HT680 について修正
V1.17 記載ミス修正 – PA968 SD/バイブレーション
RFID SDK リンクの変更
V1.18 HF リーダのエラーコード表修正
V1.19 Windows Mobile システムのデバイスについて修正
V1.20 PA690 COM ポート対応
V1.21 GPS リンクの変更
RFID SDK リンクの変更
Scanner3, ScanKey3, BTAPI 記述の削除
カメラの説明を変更
V1.22 PA500II の追加
15 章サンプルコードの変更
V1.23 第 10 章に TI カードの API を追加
PA690 COM ポートテーブルを修正
V1.23a RH768 加筆
V1.27 PA692 COM ポートリスト追加、RH768 API 追加
V1.29 PA692 HF RFID 追加