UTWS.dll ライブラリ仕様書

第 2.0.1.19 版

2022 年 2 月 14 日

ユニオンツール株式会社

更新履歴

date	version	comment
11/09/15	0.0.0.1	ベータ初版
11/09/29	0.0.0.2	· struct WHS1Config において，BYTE cpu_id[12]; を UINT cpu_id; に変更，BYTE set_serial_id[8]; を UINT set_serial_id; に変更した. · WHS-1 のメモリモードにおける書き込み方法で，メモリフルノときに「上書きする」と「止める」のモードがあるが，この状態を返すコマンドを新設したため struct WHS1Config において，BYTE mem_mode; を追加した.
11/10/05	0.0.0.3	· UTWSWHS1GetReceivedMemory 関数と UTWSWHS1SaveReceivedMemory 関数に引数 BOOL _time_adjust を追加，これを TRUE にすることによって，256 バイトの中間を埋めるタイムスタンプを生成する. · 構造体 WHS1Config のメンバ BYTE acc_mode の論理が反していたのを修正した. · 構造体 WHS1EcgData で次のメンバ変数を削除した：WORD start_year, start_month, start_day, start_hour, start_mi, start_sec; WORD end_year, end_month, end_day, end_hour, end_mi, end_sec; · 構造体 WHS1EcgData で時刻を表すメンバ変数 WORD hour, mi, sec; を WORD year, month, day, hour, mi, sec; に変更した.
11/10/19	0.0.0.7	· ファーム ver1.48 のバグ対策として，2011/0, 2011/1, 2011/2 をそれぞれ 2011/10, 2011/11, 2011/12 と解釈することとした.
11/10/23	0.0.1.9	· かつての RRI500Hz サンプリングを，RRI1 拍 1 送信モードに割り付けた.
11/11/29	0.0.3.12	· 受信時刻 msec に対応するため UTWSRRD1GetDataEx 関数と RRD1DataEx 構造体を追加した. · 64bit に対応した.
11/12/19	0.0.3.13	RRI1 拍 1 送信モードを削除.
12/1/20	0.0.4.18	· コード一部修正. · ドキュメントのバグ修正.
12/1/24	0.0.4.19	· 関数 UTWSWHS1GetReadedMemory と関数 UTWSWHS1SaveReadedMemory において，取得あるいは保存できる WHS-1 のメモリデータがないときは FALSE を返し，エラー値として UTWS_ERR_NO_DATA を返すようにした. · WHS-1 のメモリのヘッダにある日付情報が不正なとき，該当する 256 バイトのブロックを読み飛ばすこととした.
12/2/6	1.0.0.20	リリース初版

date	version	comment
12/2/27	1.0.1.21	· 関数 UTWSWHS1GetReadedMemory と関数 UTWSWHS1SaveReadedMemory をそれぞれ，UTWSWHS1GetReceivedMemory と UTWSWHS1SaveReceivedMemory に変更した．旧名称と新名称の関数で動作に違いはない．また，旧名称の関数も引き続き利用できる． · エラー識別子 UTWS_ERR_HAVE_OPEND_ALREDY を UTWS_ERR_ALREADY_OPEN に変更した．ただし，UTWS_ERR_HAVE_OPEND_ALREDY も引き続き利用できる．
12/3/1	1.0.2.22	· C 言語での開発を想定し，構造体の定義を typdef Type_ {} Type; のスタイルに変更した． · UTWSRRD1CallBakFunc の typedef を変更した．それに伴い関数 UTWSRRD1StartReceiving と UTWSRRD1SetCallBackDisconnect の引数の表示を変更した．(表示上の変更で，型そのものの変化はないため，上位側のソースコードを変更する必要はない．) · インクルードファイル utwsapi.h におけるインクルードファイルを<afxwin.h>から<windows.h>に変更した．
12/12/1	1.1.0.24	· 無線モードで 1 拍 1 送信かつ加速度の + ピーク値と − ピーク値を出力するモードと，メモリモードで加速度を 1 秒ごとに更新し + ピーク値と − ピーク値を保存するモードを追加したため，関数の追加と変更があった． · 構造体 RRD1Data と RRD1DataEx のメンバ変数 sampling_freq で心拍間隔 1 モードか心拍間隔 2 モードかを得られるようにした． · 構造体 WHS1Config のメンバ変数 ecg_mode で心拍間隔 1 モードと心拍間隔 2 モードを設定できるようにした． · 加速度 + ピーク値と − ピーク値に対応するため，構造体 WHS1EcgDataEx を追加した． · WHS-1 のフラッシュメモリからダウンロードすると同時にヘッダ情報を得るために UTWSWHS1ReadMemoryEx 関数を追加した． · 加速度 + ピーク値と − ピーク値に対応するため，UTWSWHS1GetReceivedMemoryEx 関数を追加した． · 加速度 + ピーク値と − ピーク値に対応するため，UTWSWHS1SaveReceivedMemory 関数の動作を変更した． · WHS-1 のファームウェアバージョンを得るため，UTWSWHS1Version 関数を追加した．
14/6/4	1.1.0.26	UTWSWHS1ReadConfig 関数でフラッシュ書き込み回数を読み取れなくなるバグを修正．

date	version	comment
15/1/14	1.1.0.28	· Windows8,8.1 において RRD-1 をオープンできないことがあるバグを修正した. · 本書のバグを修正した. · Windows xp と Windows vista のサポート終了と，windows8, 8.1 のサポート追加.
15/4/6	1.2.0.32	· メモリ + 心拍間隔 2 記録モードかつ加速度ピークホールドにおいて，WHS-1 フラッシュメモリ内のデータを途中までしか読み出しできない場合があるバグを修正した. · UTWSRRD1GetLocalAddress 関数においてアドレスを正しく取得できない場合があるバグを修正した.
19/5/13	1.2.0.40	· `UTWSWHS1SetDestinationAddress` 関数において，引数`_address` に英小文字が含まれると宛先アドレスが正しく設定できない場合がある問題を修正した. · `UTWSWHS1GetReceivedMemoryEx` 関数と `UTWSWHS1GetReceivedMemory` 関数の第 5 引数と `UTWSWHS1SaveReceivedMemory` 関数の第 3 引数で TRUE を設定したときになされる測定時刻の調整方法の仕様を変更した. · WHS-1 からフラッシュメモリの内容に，連続する-0.03125,-0.03125 の加速度値が著しく多くあるとき，フラッシュメモリからのダウンロードが途中で中断される問題を修正した. · 「無線 + 心拍間隔 2 出力モード」と「メモリ + 心拍間隔 2 記録モード」の記述を修正した.
19/7/8	1.2.0.43	· RRD-1 の受信アルゴリズムを変更し受信データの重複が発生しないようにした. · 開発環境を Visual Studio 17 以降へ変更した.
22/2/14	2.0.1.19	· WHS-1ver500 以降へ対応した. · WHS-1 の非同期読み出しに対応した. · Windows 7 SP1 と Windows 8.1 のサポート終了. · 32bit 環境のサポート終了.

目次

1 はじめに

本製品は，ユニオンツール製ウェアラブル心拍センサ WHS-1 および WHS-1ver500 以降 (S/N:0010020000 以降の WHS-1 が，WHS-1ver500 以降に該当します．本書では以降，特に区別する必要がない場合 WHS-1 と表記します．) とその受信ドングル RRD-1 を用いて，ユーザアプリケーションを開発する方を対象にしています．本製品を使用してアプリケーションプログラムを開発するには，Windows プログラミングの経験と C 言語に関する知識が必要です．本書はこれらの知識があることを前提として書かれています．

1.1 開発環境

開発環境は，64bit の Windows 10 と Visual Studio 2022 以降を推奨します．添付されたサンプルプログラムは，Windows 10 環境下の Visual Studio 2022 で作成されています．

1.2 ユーザアプリケーションの作成方法

プログラム作成に必要なファイルは次の通りです．

- `UTWS.dll`: WHS-1 と RRD-1 のための API がこのダイナミックリンクライブラリ (以下 DLL) で提供されます．ディレクトリ \x64 に配置されています．
- `UTWS.lib`: `UTWS.dll` を使用するためのインポートライブラリです．ディレクトリ \x64 に配置されています．
- `utwsapi.h`: `UTWS.dll` 内の関数が宣言されているヘッダーファイルです．ディレクトリ \include に配置されています．
- `utwsstruct.h`: 構造体が定義されています．`utwsapi.h` から読みこまれます．ディレクトリ \include に配置されています．
- `utwserrdef.h`: エラー定数が定義されています．`utwsapi.h` から読みこまれます．ディレクトリ \include に配置されています．

プログラムソースにヘッダーファイル `utwsapi.h` をインクルードして下さい．リンク時にインポートライブラリ `UTWS.lib` をリンクして下さい．DLL ファイル `UTWS.dll` をパスの通ったディレクトリ，または実行ファイルと同じディレクトリに配置して下さい．

1.3 UTWS.DLL 実行に必要な DLL

WHS-1 を操作するには Future Technology Devices Intl. Ltd.(以下 FTDI 社) の FTD2XX.DLL が必要です．この DLL は D2XX Direct Driver をインストールすれば，パスの通ったディレクトリに配置されます．2021 年 7 月現在，ドライバは以下のサイトからダウンロードできます．

`https://ftdichip.com/drivers/d2xx-drivers/`

D2XX Direct Driver のインストール方法については，以下のサイトを参照して下さい．

`https://ftdichip.com/document/installation-guides/`

UTWS ライブラリが提供する関数で FTD2XX.DLL が必要なものは，11 節以降の説明の中で記します．

1.4 マルチスレッド

本ライブラリが提供する関数の呼び出しには，接続したデバイス毎に割り当てられるハンドルが必要です．接続したデバイスに対する操作を独立に処理するため，各ハンドルに対し 1 つ以上のスレッドとバッファが割り当てられています．従って，異なるハンドルに対してはリエントラントですが，同一ハンドルによる関数呼び出しに対して，リエントラントではありません．

2 デバイスのオープンとクローズ

WHS-1 と RRD-1 を操作するには，それらを `UTWSOpenDevice` 関数 (11.2 節) を使ってオープンし、ハンドルを得る必要があります．また，必要な操作がすべて完了したら，クローズしなければなりません．

2.1 RRD-1 をオープンする

RRD-1 をオープンする例を次に示します．

```
HANDLE hRRD1 = UTWSOpenDevice(UTWS_RRD_1, 0);
```

返されるハンドルは，以降 RRD-1 を操作する関数で使用します．第 1 引数は RRD-1 をオープンするための識別子です．第 2 引数は無視されます．

2.2 WHS-1 をオープンする

WHS-1 は，同時に 10 台オープンできます．そこで，WHS-1 を `UTWSOpenDevice` 関数でオープンする前に，WHS-1 が何台接続されているか調べる必要があります．これには，`UTWSWHS1CountConnected` 関数 (11.15 節) を使います．

```
UINT count = UTWSWHS1CountConnected(1, 5000);
```

第 1 引数には接続予定台数を与えて下さい．第 2 引数は単位 ms のタイムアウトで，このタイムアウトの間に，20ms 間隔で WHS-1 が接続されているか調べます．もし，タイムアウト内で予定接続台数の WHS-1 が検出されたら，すぐに制御を返します．戻り値は接続されている WHS-1 の台数です．次に，`UTWSOpenDevice` 関数で WHS-1 をオープンします．

```
HANDLE hWHS[10];
hWHS[0] = UTWSOpenDevice(UTWS_WHS_1, 0);
```

第 1 引数は WHS-1 をオープンするための識別子です．第 2 引数で何番目の WHS-1 を開くか指定して下さい．上記の例では 0 番目の WHS-1 を開こうとしています．

2.3 クローズする

WHS-1 や RRD-1 の 1 台を指定してクローズするときは，UTWSCloseDevice 関数 (11.3 節) を使います．閉じようとするデバイスを特定するために，引数には UTWSOpenDevice 関数の戻り値として得られたハンドルを与えます．例を以下に示します．

```
HANDLE hRRD1 = UTWSOpenDevice(UTWS_RRD_1, 0);
UTWSCloseDevice(hRRD1);
```

オープンしたデバイスをすべて閉じたいときには，UTWSCloseAll 関数 (11.4 節) を利用できます．引数はありません．例を以下に示します．

```
UTWSCloseAll();
```

2.4 FTD2XX.DLL がロードされるタイミング

FTDI 社の FTD2XX.DLL は，UTWSWHS1CountConnected 関数または UTWSOpenDevice 関数の第 1 引数に UTWS_WHS_1 を伴って呼び出したときにロードされます．

3 DLL 内のエラー状態

UTWS.dll が提供している関数がエラーを返した場合，UTWSGetErrorMessage 関数 (11.1 節) でどのようなエラーが発生したか知ることができます．次に例を示します．

```
char buffer[256];
UINT err_id = UTWSGetErrorMessage(hWHS1, buffer, 256);
```

第 1 引数は，エラーの状態を知りたいデバイスへのハンドルです．第 2 引数はエラーに関する説明文を格納するバッファへのポインタを与えます．第 3 引数にはバッファのサイズです．戻り値はエラーの状況を表すエラー識別子で，その一覧は 13 節にあります．もしエラー識別子だけ得たいときは第 2 引数を 0 として下さい．

UTWSOpenDevice 関数，UTWSCloseDevice 関数または UTWSCloseAll 関数，あるいはハンドルを引数に取らない関数に関するエラー状態を知りたいとき，デバイスへのハンドルを 0 とします．

```
HANDLE hRRD1 = UTWSOpenDevice(UTWS_RRD_1, 0);
char buffer[256];
if (0 == hRRD1) {
  UINT err_id = UTWSGetErrorMessage(0, buffer, 256);
}
```

4　WHS-1 と RRD-1 の通信を成立させるために

WHS-1 と RRD-1 の間で通信が成立するには，WHS-1 の「宛先アドレス」と RRD-1 の「自アドレス」が一致していなければなりません．WHS-1 の宛先アドレスは ROM 領域に書き込まれるため，一度設定すれば再設定の必要はありません．RRD-1 の自アドレスは RAM 領域に書き込まれるため，オープンするたびに再設定が必要ですので注意して下さい．

WHS-1 の宛先アドレスと RRD-1 の自アドレスはともに 10 桁の 16 進数で，UTWS.dll の関数を使った設定や取得には，null ターミネートの 10 バイトの ANSI 文字列として扱います．

4.1　RRD-1 の宛先アドレス読み出し，および設定

RRD-1 のデフォルトの自アドレスは C2C2C2C2C2 です．

自アドレスを設定するには UTWSRRD1SetLocalAddress 関数 (11.13 節) を用います．次に例を示します．

```
char address[11] = "ACAC121245";
UTWSRRD1SetLocalAddress(hRRD1, address);
```

第 1 引数は RRD-1 のハンドルです．第 2 引数は自アドレスが格納されたバッファへのポインタです．

RRD-1 の自アドレスを取得するには，UTWSRRD1GetLocalAddress 関数 (11.12 節) を用います．次に例を示します．

```
char address[11];
UTWSRRD1GetLocalAddress(hRRD1, address);
```

第 1 引数は RRD-1 のハンドルです．第 2 引数の指すバッファに自アドレスが格納されます．バッファには 11 バイト以上を確保して下さい．

4.2　WHS-1 の宛先アドレス読み出し，および設定

WHS-1 の宛先アドレスを設定するには UTWSWHS1SetDestinationAddress 関数 (11.19 節) を用います．次に例を示します．

```
char address[11] = "ACAC121245";
UTWSWHS1SetDestinationAddress(hWHS1, address);
```

第 1 引数は WHS-1 のハンドルです．第 2 引数は宛先アドレスが格納されたバッファへのポインタです．

WHS-1 の宛先アドレスを取得するには，UTWSWHS1GetDestinationAddress 関数 (11.18 節) を用います．次に例を示します．

```
char address[11];
UTWSWHS1GetDestinationAddress(hWHS1, address);
```

第 1 引数は WHS-1 のハンドルです．第 2 引数の指すバッファに宛先アドレスが格納されます．バッファには 11 バイト以上を確保して下さい．

5　WHS-1 の仕様と設定

5.1　WHS-1 の仕様とモード

表 2 を使って WHS-1 の仕様を説明します．WHS-1 のデータ出力モードには，無線モードとメモリモードがあります．無線モードでは心拍波形，心拍間隔および心拍数のいずれかと加速度，温度を無線で出力し RRD-1 で受信することができます．メモリモードでは，心拍間隔と心拍数のいずれかと加速度，温度を WHS-1 内蔵フラッシュメモリに保存することができます．

表 2　WHS-1 の仕様.

WHS-1		無線モード				メモリモード		
		心拍波形	心拍間隔1	心拍間隔2	心拍数	心拍間隔1	心拍間隔2	心拍数
心拍	サンプリング周波数	128Hz	1000Hz	1000Hz	1000Hz	1000Hz	1000Hz	1000Hz
	更新周期	7.8ms	心拍ごと	心拍ごと	4s	心拍ごと	心拍ごと	4s
加速度	サンプリング周波数	128Hz	32Hz	32Hz	32Hz	32Hz	32Hz	32Hz
	更新周期	7.8ms	心拍ごと	心拍ごと	4s	4s	1s	4s
温度	サンプリング周波数	1Hz	1Hz	1Hz	1Hz	1Hz	1Hz	1Hz
	更新周期	39ms	心拍ごと	心拍ごと	4s	約80s	約40s	約160s
	送信間隔	39ms	心拍間隔×3	心拍間隔×1	12s	-	-	-

WHS-1 ver500以降		無線モード			メモリモード	
		心拍波形	心拍間隔1	心拍間隔2	心拍間隔1	心拍間隔2
心拍	サンプリング周波数	128Hz	1000Hz	1000Hz	1000Hz	1000Hz
	更新周期	7.8ms	心拍ごと	心拍ごと	心拍ごと	心拍ごと
加速度	サンプリング周波数	128Hz	32Hz	32Hz	32Hz	32Hz
	更新周期	7.8ms	心拍ごと	心拍ごと	4s	1s
温度	サンプリング周波数	1Hz	1Hz	1Hz	1Hz	1Hz
	更新周期	39ms	心拍ごと	心拍ごと	約80s	約40s
	送信間隔	39ms	心拍間隔×3	心拍間隔×1	-	-

5.1.1　無線 + 心拍波形出力モード

無線モードにおける心拍波形出力モードでは，WHS-1 に取り付けられた電極から得られる心拍信号を 10bit(WHS-1ver500 以降に於いては 12bit でサンプリング，出力 10bit)，128Hz でサンプリング，また加速度を 128Hz でサンプリングし，それらを 5 組まとめて温度とともに無線で送信します．したがって心拍信号と加速度の値の更新間隔は 8ms，温度の更新間隔は 39ms，無線の送信間隔は 39ms です．

5.1.2 無線 + 心拍間隔 1 出力モード

無線モードにおける心拍間隔 1 出力モードでは，WHS-1 に取り付けられた電極から得られる心拍信号を 10bit(WHS-1ver500 以降に於いては 12bit)，1000Hz でサンプリングし，心拍波形の R 波のピーク間隔を測定することで，心拍間隔を測定します．前述のように得られた心拍間隔を加速度と温度を組にし，3 組まとめて送信します．つまり，心拍数が 60bpm のときは加速度と温度の更新間は 1s で，無線の送信間隔は 3s です．

5.1.3 無線 + 心拍間隔 2 出力モード

無線モードにおける心拍間隔 2 出力モードでは，心拍間隔の測定方法は心拍間隔 1 モードと同様です．加速度をピークホールドにしたとき，加速度の測定値として 3 軸それぞれについて，最大値と最小値を出力します．加速度を平均値としたとき，3 軸それぞれの平均値を出力します．このモードの特徴は心拍間隔と加速度と温度を組にし，1 組づつ送信することです．つまり，心拍数が 60bpm のときは加速度と温度の更新間隔は 1s で，無線の送信間隔は 1s です．

5.1.4 無線 + 心拍数出力モード (WHS-1ver500 以降では実装されていません)

無線モードにおける心拍数モードでは，心拍間隔 1 モードと同様の方法で得られた心拍間隔の過去 8 個を平均し，平均心拍間隔を測定します．この平均心拍間隔と加速度と温度を 4s 間隔で更新し，3 組まとめて無線で送信します．従って無線の送信間隔は 12s です．

5.1.5 メモリ + 心拍間隔 1 記録モード

メモリモードにおける心拍間隔 1 モードにおける，心拍間隔測定方法は無線 + 心拍間隔 1 モードと同様です．心拍間隔は心拍ごとに，加速度は 4s ごとに，温度はデータ量 256 バイトに 1 回メモリに書き込みます．従って心拍数が 60bpm のとき，心拍間隔の更新間隔は 1s，加速度の更新間隔は 4s，温度の更新間隔は約 80s です．

5.1.6 メモリ + 心拍間隔 2 記録モード

メモリモードにおける心拍間隔 2 モードにおける，心拍間隔測定方法は無線 + 心拍間隔 1 モードと同様です．このモードでメモリ + 心拍間隔 1 モードと異なる点は，加速度が 1s ごとに更新されることにあります．また，加速度をピークホールドとしたとき，加速度の測定値として 3 軸それぞれについて，最大値と最小値が記録されます．加速度を平均値としたとき，3 軸それぞれの平均値が記録されます．心拍間隔は心拍ごとに，加速度は 1s ごとに，温度はデータ量 256 バイトに 1 回メモリに書き込むので，心拍数が 60bpm のとき，心拍間隔と加速度の更新間隔は 1s，温度の更新間隔は約 40s です．

5.1.7 メモリ + 心拍数記録モード (WHS-1ver500 以降では実装されていません)

メモリモードにおける心拍数モードでは，心拍間隔 1 モードと同様の方法で得られた心拍間隔の過去 8 個を平均し，平均心拍間隔を測定します．メモリモードの心拍数モードでは平均心拍数と加速度の組を 4s 間隔でメモリに書き込みます．温度はデータ量 256 バイトに 1 回書き込むので，更新間隔は約 160s です．

5.1.8 モニターモード (WHS-1ver500 以降のみ)

WHS-1ver500 以降において，メモリ + 心拍間隔 1 記録モード，またはメモリ + 心拍間隔 2 記録モードに設定し，同時にモニターモードを有効にすると，WHS-1 の電源を入れた直後の 60 秒間，WHS-1 は無線 + 心拍波形出力モードで動作します．

5.1.9 加速度測定

心拍間隔 1, 2 モードまたは心拍数モードのとき，加速度センサのサンプリング周波数は 32Hz です．一方，加速度の更新周期は，心拍間隔か 1s または 4s です．このギャップを埋めるために，加速度センサは移動平均モードまたはピークホールドモードのいずれかを選択できます．移動平均モードでは，更新周期間の平均値を加速度の値として出力します．ピークホールドモードでは，更新周期のピーク値を加速度の値として出力します．WHS-1ver500 以降では，ピークホールドモードに固定されていて，移動平均モードは選択できません．

5.1.10 温度測定

温度センサのサンプリング周期は 1s ですが，WHS-1 の一時メモリでは 4s 間隔で値が更新されています．温度は変化のタイムスケールが大きいので，加速度センサのような処置はとらず，値出力時直近の一時メモリ上の値を温度の値として出力します．従って，得られる値は少なくとも 4 秒間変化しません．

5.1.11 フラッシュメモリ

WHS-1 は 16Mbit のフラッシュメモリを内蔵し，メモリモードでは心拍間隔または平均心拍間隔，加速度，温度を記録できますが，それらがいずれの時間帯のデータか示すために，データ 256 バイトごとに時刻を記録します．従って，WHS-1 をメモリモードに設定するときには，現在時刻の設定も必要です．また，メモリがいっぱいになったとき，そこで記録を止めるか，最過去のデータから上書きするかを選択することができます．ただし，WHS-1ver500 以降ではメモリがいっぱいになったとき，記録を止めます．

上記のように，WHS-1 には種々モードがあります．この節ではこれらの設定方法について説明します．

5.2 WHS-1 の設定状態の取得

WHS-1 の設定状態を取得するには `UTWSWHS1ReadConfig` 関数 (11.16 節) を使います，例を次に示します．

```
WHS1Config rc;
UTWSWHS1ReadConfig(hWHS, &rc);
```

第 1 引数は WHS-1 のハンドルです．第 2 引数は読み出した設定情報が格納されている構造体 `WHS1Config` です．この構造体に関する詳細は 12.4 節を参照して下さい．

5.3 WHS-1 のモード設定

WHS-1 のモードを設定するには，UTWSWHS1WriteConfig 関数 (11.17 節) を使います．まず，無線モードに設定するときの例を示します．

```
WHS1Config rc;
rc.mode = 0;       //無線モード
rc.ecg_mode = 2;   //心拍間隔 1 モード
rc.acc_mode = 0;   //加速度ピークホールド
rc.temp_id = 100; //仮 ID
UTWSWHS1WriteConfig(hWHS, &rc, 0, 0, 0, 0, 0, 0, 0);
```

第 1 引数は WHS-1 のハンドルです．第 2 引数は設定情報を格納した構造体 WHS1Config を指すポインタです．他の引数は，無線モードでは必要ないので，0 としてかまいません．

次にメモリモードに設定する例を示します．

```
WHS1Config rc;
rc.mode = 1;           //メモリモード
rc.ecg_mode = 2;       //心拍間隔 1 モード
rc.acc_mode = 0;       //加速度ピークホールド
rc.temp_id = 100;      //仮 ID
rc.monitor_mode = 1; //モニターモード
SYSTEMTIME tm;
GetLocalTime(&tm);
UTWSWHS1WriteConfig(hWHS, &rc,
                    0, //最過去データを上書き
                    tm.wYear, tm.wMonth, tm.wDay,
                    tm.wHour, tm.wMinute, tm.wSecond);
```

第 1 引数は WHS-1 のハンドルです．第 2 引数は設定情報を格納した構造体 WHS1Config を指すポインタです．第 3 引数は，メモリがいっぱいになったときの動作を指定します．0 で過去データを上書き，1 で記録停止です．以下の引数には現在時刻の年，月，日，時，分，秒を与えて下さい．モニターモードは WHS-1ver500 以降で利用可能です．

6 RRD-1 での受信方法

WHS-1 が無線モードならば，WHS-1 の宛先アドレスに一致する自アドレスを持った RRD-1 で WHS-1 が送信するデータを受信することができます．

受信を開始するには，UTWSRRD1StartReceiving 関数 (11.5 節) を使います．RRD-1 で受信したデータはいったん UTWS.dll 内の受信バッファに格納されます．受信バッファのデータは UTWSRRD1GetData 関数 (11.8 節) または UTWSRRD1GetDataEx 関数 (11.9 節) によって取り出すことができます．両者の関数の使

用方法はほとんど同じなので，ここでは，UTWSRRD1GetData 関数を使って説明します．この関数が実行されると，取り出されたデータは受信バッファから削除されます．UTWSRRD1GetData 関数が実行されないと，受信バッファの末尾に受信したデータが追加されます．受信バッファには最大 1024 個の受信データを格納できます．あふれた場合は最過去のデータから消去されます．受信バッファに格納されているデータ数は UTWSRRD1DataCount(11.7 節) で得られます．受信を停止するには UTWSRRD1StopReceiving 関数 (11.6 節) を使用します．

6.1 ポーリングで受信データを取り出す

ポーリング手法で受信データを取り出す例を示します．

```
 1:  SetTimer(100, 1, 0);
 2:  UTWSRRD1StartReceiving(hRRD1, 0, 0, 0);

 3:void OnTimer(UINT_PTR nIDEvent)
 4:{
 5:  UINT count = UTWSRRD1DataCount(hRRD1);
 6:  if (-1 != count && count) {
 7:    RRD1Data data;
 8:    UTWSRRD1GetData(hRRD1, &data);
 9:    displayReceivedData(data);
10:  }
11:}
```

まず，第 1 行でタイマーを設定しポーリングの準備をします．第 2 行で受信を開始します．UTWSRRD1StartReceiving 関数の第 1 引数は RRD-1 のハンドルです．今はイベントもコールバック関数も使用しないので，第 2 引数以降は 0 とします．タイマーイベントが発生するたびに関数 OnTimer が呼び出されると仮定します．タイマーイベントが発生すると，第 5 行で UTWSRRD1DataCount 関数を使って UTWS.dll の受信バッファにデータがあるか調べます．この関数の第 1 引数も RRD-1 のハンドルで，戻り値は受信バッファにあるデータ数です．受信バッファにデータがあるとき，第 8 行の UTWSRRD1GetData 関数で受信バッファからデータを読み出します．この関数の第 1 引数は RRD-1 のハンドルで，第 2 引数は受信データを格納する構造体 RRD1Data(12.2 節) へのポインタです．UTWSRRD1GetDataEx 関数を使う場合は，第 2 引数は構造体 RRD1DataEx(12.3 節) へのポインタです．

6.2 イベントで受信データを取り出す

RRD-1 でのデータ受信を，イベントオブジェクトを使って監視することができます．

```
 1:  HANDLE hevent = CreateEvent(0, TRUE, FALSE, 0);
 2:  UTWSRRD1StartReceiving(hRRD1, hevent, 0, 0);
 3:  AfxBeginThread(thread, 0);
```

```
 4:UINT __cdecl thread(LPVOID)
 5:{
 6:  RRD1Data data;
 7:  while (controler) {
 8:    if (WAIT_OBJECT_0 == WaitForSingleObject(hevent, 1)) {
 9:      if (UTWSRRD1GetData(hRRD1, &data))
10:        displayReceivedData(data);
11:    }
12:  }
13:}
```

まず，第 1 行で受信時にシグナル状態になるイベントオブジェクトを生成します．ここでは必ず手動イベントを生成して下さい．第 2 行で受信を開始します．第 1 引数は RRD-1 のハンドルです．第 2 引数に第 1 行で生成したイベントオブジェクトのハンドルを与えます．このときイベントを非シグナル状態として下さい．第 3 引数以降は 0 とします．このように受信を開始すると，RRD-1 がデータを受信したとき hevent が指すイベントオブジェクトがシグナル状態となります．第 3 行でイベントの状態を監視するスレッドを生成します．第 8 行でイベントオブジェクトがシグナル状態になるのを待ちます．第 9 行で UTWS.dll 内の受信バッファから 1 つの受信データを取り出します．UTWSRRD1GetData 関数の実行によって UTWS.dll 内の受信バッファが空になるとイベントオブジェクトは非シグナル状態となります．ユーザがイベントを非シグナル状態にする必要はありません．

6.3 コールバック関数で受信データを取り出す

RRD-1 のデータ受信を，コールバック関数を使って知ることができます．

```
 1: RRD1Data rrd1_data;
 2: UTWSRRD1StartReceiving(hRRD1, 0, RRD1Receive, &rrd1_data);

 3: VOID CALLBACK RRD1Receive(VOID* p, VOID*)
 4: {
 5:   RRD1Data* rrd1_data = reinterpret_cast<RRD1Data*>(p);
 6:   UINT count;
 7:   while (-1 != (count = UTWSRRD1DataCount(hRRD1))&& count) {
 8:     UTWSRRD1GetData(hRRD1, rrd1_data);
 9:     SendMessage(hWnd, WM_PAINT, 0, 0);
10:   }
11: }
```

データ受信時に呼び出されるコールバック関数を第 3 行以降で定義しています．第 2 行でデータ受信を開始します．第 1 引数は RRD-1 のハンドルです．イベントを用いないので第 2 引数を 0 とします．第 3 引数にコールバック関数へのポインタを，第 4 引数にコールバック関数の第 1 引数を与えて下さい．上記の例では第

1 行で定義した変数へのポインタがコールバック関数の第 1 引数です．RDD-1 がデータを受信するたびに，第 3 行のコールバック関数が呼び出されます．第 7 行で UTWS.dll 内受信バッファのデータ数を調べ，第 8 行で受信データを取り出します．与えられたコールバック関数は UTWS.dll から呼び出されますが，それは UTWSRRD1StartReceiving 関数を呼び出したスレッドとは異なることに注意して下さい．コールバック関数での処理が大きく制御が戻るのに時間がかかれば，RRD-1 の受信データを取りこぼす危険があります．

6.4 受信の停止

RRD-1 の受信を停止する例を以下に示します．

```
UTWSRRD1StopReceiving(hRRD1);
```

引数は RRD-1 のハンドルです．

7 RRD-1 の状態の検出

7.1 RRD-1 が開いているか

RRD-1 がオープンしているかどうかを調べるには，UTWSRRD1IsOpen 関数 (11.10 節) を使います．例を以下に示します．

```
if (UTWSRRD1IsOpen(hRRD1)) {
  //オープンしている
}
```

引数は RRD-1 のハンドルです．戻り値が FALSE ならば RRD-1 が閉じているか，あるいは関数が失敗したかです．いずれか調べるには UTWSGetErrorMessage 関数を用いて下さい．

7.2 RRD-1 が PC から取り外されたか

RRD-1 が PC から取り外されたことを，コールバック関数で知ることができます．RRD-1 が取り外されたときに呼び出されるコールバック関数を登録するには UTWSRRD1SetCallBackDisconnect 関数 (11.11 節) を用います．例を以下に示します．

```
UTWSRRD1SetCallBackDisconnect(hRRD1, callbackDisconnected, 0);

VOID CALLBACK callbackDisconnected(VOID* p, VOID* h)
{
  //第 2 引数 h は抜けた RRD-1 のハンドル．
  MessageBox(0, "ドングルが抜けました．", 0, MB_OK);
}
```

第 1 引数は RRD-1 のハンドルです．第 2 引数は RRD-1 が PC から取り外されたときに呼び出されるコールバック関数へのポインタです．第 3 引数はコールバック関数の第 1 引数にそのまま渡されます．呼び出された

コールバック関数の第 2 引数は，取り外された RRD-1 のハンドルです．

8 WHS-1 のメモリ読み出し

WHS-1 を PC に接続，オープンした後，UTWSWHS1ReadMemoryEx 関数 (11.21 節) を使って，WHS-1 のフラッシュメモリの内容を PC へダウンロードすることができます．WHS-1 にデータが記録されているかどうかは，UTWSWHS1GetRecordedSize(11.26) 関数で調べることができます．ダウンロードしたデータはいったん UTWS.dll 内のバッファに保存されます．これをメイン側のバッファにコピーするには UTWSWHS1GetReceivedMemory 関数 (11.22 節) か UTWSWHS1GetReceivedMemoryEx 関数 (11.23 節) を使います．また，ファイルに保存するには UTWSWHS1SaveReceivedMemory 関数 (11.24 節) を用います．

8.1 WHS-1 のデータをメイン側にコピーする

まず，UTWS.dll にストアされたフラッシュメモリのデータをメイン側にコピーする方法を示します．

```
 1:  UINT recorded;
 2:  UTWSWHS1GetRecordedSize(hWHS, &recorded);
 3:  if (!recordded)
 4:    return;
 5:  WHS1MemDataHeader header;
 6:  UTWSWHS1ReadMemoryEx(hWHS, &header);
 7:  UINT n = header.data_count;
 8:  if (3 == header.ecg_mode && 0 == header.acc_mode) {
 9:    //メモリ + 心拍間隔 2 記録モードのとき
10:    WHS1EcgDataEx* data = new WHS1EcgDataEx[n];
11:    UTWSWHS1GetReceivedMemoryEx(hWHS, &header, data, n, TRUE);
12:  } else {
13:    WHS1EcgData* data = new WHS1EcgData[n]
14:    UTWSWHS1GetReceivedMemory(hWHS, &header, data, n, TRUE);
15:  }
```

第 2 行の UTWSWHS1GetRecordedSize 関数で，WHS-1 のフラッシュメモリにデータが記録されているかどうか調べます．この関数の第 1 引数は WHS-1 のハンドルで，第 2 引数に記録されているデータブロック数が返ります．第 6 行で UTWSWHS1ReadMemoryEx 関数を使い WHS-1 のフラッシュメモリのデータを UTWS.dll のバッファへダウンロードします．この関数の第 1 引数は WHS-1 のハンドル，第 2 引数が WHS1MemDataHeader 型 (12.7 節) へのポインタで，ここにダウンロードしたデータ数や動作モードなどのヘッダ情報が返されます．UTWS.dll のバッファにダウンロードされたデータをメイン側にコピーするとき，WHS-1 の動作モードによって使用する関数が異なります．

心拍間隔 2 記録モードの時 (5.1.6 節) は次の通りです．第 10 行で WHS1EcgDataEx 型 (12.6 節) のメモリをダウンロードしたデータ数だけ用意します．第 11 行で，UTWSWHS1GetReceivedMemoryEx 関数を実行しメイン側に UTWS.dll のバッファをコピーします．この関数の第 1 引数は WHS-1 のハンドル，第 2 引数はヘッ

ダ情報のコピー先である WHS1MemDataHeader 型変数へのポインタ，第 3 引数は UTWS.dll にストアされた心拍情報，温度，加速度を保存できるだけのメモリを確保した WHS1EcgDataEx 型のバッファへのポインタ，第 4 引数は準備した WHS1EcgDataEx 型の数です．

心拍間隔 2 記録モード以外の時は，第 13，14 行のようにそれぞれ WHS1EcgData 型 (12.5 節) をメモリ確保し，UTWSWHS1GetReceivedMemory 関数でメイン側にコピーします．

UTWSWHS1GetReceivedMemoryEx 関数と UTWSWHS1GetReceivedMemory 関数の第 5 引数は 8.3 節で説明します．

8.2 WHS-1 のデータをファイルに保存する

次に，UTWS.dll にストアされたフラッシュメモリのデータをファイルに保存する方法を示します．

```
1:  UTWSWHS1ReadMemory(hWHS);
2:  UTWSWHS1SaveReceivedMemory(m_hWHS, "c:/data.csv", TRUE);
```

第 1 行で UTWSWHS1ReadMemory 関数を実行し WHS-1 のフラッシュメモリのデータを UTWS.dll のバッファにダウンロードします．この関数の第 1 引数は WHS-1 のハンドルです．第 2 行で，UTWSWHS1SaveReceivedMemory 関数を使い UTWS.dll のバッファの内容をファイルに書き込みます．この関数の第 1 引数は WHS-1 のハンドルで，第 2 引数は保存先のパス名です．UTWSWHS1SaveReceivedMemory 関数の第 3 引数は 8.3 節で説明します．

8.3 時刻情報の調整

節 5.1 で説明したように，時刻情報は WHS-1 のフラッシュメモリ上の 256 バイトごとに記録されています．UTWSWHS1GetReceivedMemoryEx 関数と UTWSWHS1GetReceivedMemory 関数の第 5 引数，UTWSWHS1SaveReceivedMemory 関数と UTWSWHS1BinaryFileToData の第 3 引数，および UTWSWHS1BinaryToData 関数の第 4 引数は，UTWS.dll のバッファに保存されている心拍情報，温度，加速度の組に対し時刻をどのように付与するか指示します．

この引数に FALSE を与えると，フラッシュメモリ上の 1 つの 256 バイトブロック内に記録されている心拍情報，加速度の組に同じ時刻情報が付与されます．

引数が TRUE のときは，心拍間隔モードと心拍数モードで動作が異なります．心拍間隔モードのとき：256 バイトブロック A に記録された時刻を t_A とします．次の 256 バイトブロックに記録された時刻を t_B とします．このときブロック A にデータを記録するのに要した時間は $t_{AB} = t_B - t_A$ です．一方，ブロック A に記録された心拍間隔の積分を $s_A = \sum_{i=2}^{n} r_i + r(t_B)$ とします．ここで r_i はブロック A に n 個ある内の i 番目の心拍間隔，$r(t_B)$ はブロック B に最初に記録された心拍間隔値です．時間 t_{AB} と積分 s_A には差がないはずですが，256 バイトごとに記録する時刻は秒単位なので，時間と心拍間隔積分が精密に一致することはありません．そこで，時間 t_{AB} と積分 s_A の差を $\Delta t = t_{AB} - s_A$ として，ブロック A の i 番目の心拍間隔と加速度と温度の組の時刻 t_{Ai} を，

$$t_{A1} = t_A, \quad (i = 1) \tag{1}$$

$$t_{Ai} = t_A + \sum_{k=2}^{i} r_k + \frac{\Delta t}{n}(i-1), \quad (i > 1) \tag{2}$$

とします．ただし Δt が 10sec を超える場合，WHS-1 の電源が一時切られたという状況が考えられ，またフラッシュメモリに記録された最後のブロックでは Δt が計算できないことから，

$$t_{Ai} = t_A + \sum_{k=2}^{i} r_k, \quad (i > 1) \tag{3}$$

とします．

心拍数モードの時，心拍数と加速度の組は 4 秒ごとに書き込まれます．従って，ブロック A の i 番目の心拍間隔と加速度と温度の組の時刻 t_{Ai} を，

$$t_{Ai} = t_A + 4(i-1), \tag{4}$$

とします．

9 WHS-1 メモリの非同期読み出し，バイナリデータ保存

9.1 WHS-1 のデータを非同期で読み出し，完了をイベントで検出する

WHS-1 のメモリを非同期で読み出しし，イベントオブジェクトで読み出し完了を待つ方法を示します．

```
 1: DWORD WINAPI check_read_size(LPVOID pParam)
 2: {
 3:   HANDLE stop_event = reinterpret_cast<HANDLE>(pParam);
 4:   while (WAIT_OBJECT_0 != WaitForSingleObject(stop_event, 1)) {
 5:     UINT read_size, size_for_read;
 6:     UTWSWHS1ReadSize(hWHS, &read_size, &size_for_read);
 7:   }
 8:   return 0;
 9: }

10: void read()
11: {
12:   UINT block_count;
13:   UTWSWHS1GetRecordedSize(hWHS, &block_count);
14:   if (!block_count)
15:     return;
16:
17:   HANDLE hevent = CreateEvent(0, TRUE, FALSE, 0);
18:   UTWSWHS1ReadMemoryBinary(hWHS, hevent, 0, 0);
19:   HANDLE stop_thread = CreateEvent(0, TRUE, FALSE, 0);
20:   HANDLE handle_thread = CreateThread(0,0,check_read_size, stop_thread, 0, 0);
21:
22:   WaitForSingleObject(hevent, INFINITE);
```

```
23:   SetEvent(stop_thread);
24:   WaitForSingleObject(handle_thread, INFINITE);
25:   if (UTWS_NO_ERR == UTWSWHS1ReadMemoryBinaryResult(hWHS)) {
26:
27:     UINT read_byte_size;
28:     UTWSWHS1GetMemoryBinarySize(hWHS, &read_byte_size);
29:
30:     BYTE* buf = new BYTE[read_byte_size];
31:     UTWSWHS1GetMemoryBinary(hWHS, buf, read_byte_size);
32:
33:     UTWSWHS1SaveMemoryBinary(hWHS, "file.bin");
34:
35:     UTWSCloseDevice(hWHS);
36:     HANDLE hBin = UTWSOpenDevice(UTWS_WHS_1_BINARYDATA, 0);
37:     UINT data_count = UTWSWHS1BinaryToData(hBin, buf, read_byte_size, TRUE, FALSE);
38:
39:     WHS1MemDataHeader memHeader;
40:     WHS1EcgDataEx* mem_data = new WHS1EcgDataEx[data_count];
41:     UTWSWHS1GetDataFromBinary(hBin, &memHeader, mem_data, data_count);
42:
43:     UTWSWHS1SaveDataFromBinary(hBin, "file.csv");
44:
45:     delete[] mem_data;
46:     delete[] data;
47:     delete[] buf;
48:   }
49:   CloseHandle(hevent);
50:   CloseHandle(stop_thread);
51: }
```

非同期読み出しの開始には，行 18 の UTWSWHS1ReadMemoryBinary 関数 (11.27) を使います．第 1 引数は，WHS-1 のハンドルで，第 2 引数はイベントオブジェクトのハンドルです．イベントで読み出しの完了を検出するとき，第 3 及び 4 引数は 0 としてください．読み出しが完了すると，イベントがシグナル状態になるので，これがシグナル状態になるのを待っているのは行 22 です．非同期読み出し中の進捗を行 6 の UTWSWHS1ReadSize 関数 (11.32) で調べられます．第 1 引数は，WHS-1 のハンドルで，第 2 引数には読み出したデータブロック数，第 3 引数には読み出されるべき全データブロック数が返されます．読み出し結果を，行 25 のを UTWSWHS1ReadMemoryBinaryResult 関数 (11.31) で調べ，戻り値が UTWS_NO_ERR ならば読み出し成功です．引数は WHS-1 のハンドルです．読み出したバイナリデータのバイト数は，行 28 の UTWSWHS1GetMemoryBinarySize 関数 (11.28) で得られます．第 1 引数は WHS-1 のハンドル，第 2 引数にバイナリデータのバイト数が返されます．バイナリデータを直接得るには，行 31 の

UTWSWHS1GetMemoryBinary 関数 (11.29) を使います．第 1 引数は WHS-1 のハンドル，第 2 引数はバイナリデータが返されるバッファへのポインタ，第 3 引数はバッファサイズです．バイナリデータをファイルに保存するには行 33 の UTWSWHS1SaveMemoryBinary 関数 (11.30) を用います．第 1 引数は WHS-1 のハンドル，第 2 引数は保存パスです．この時点で，WHS-1 のフラッシュメモリのバイナリデータは呼び出し側にあるので，行 35 で WHS-1 をクローズしても，UTWS.dll を使ってデータを操作することができます．心拍，加速度などのデータを得るには，まず行 36 の UTWSOpenDevice 関数で引数に UTWS_WHS_1_BINARYDATA を指定して，バイナリデータへのハンドルを得ます．次に，バイナリデータから UTWS.dll 内で心拍，加速度などの生体情報へ変換するため，行 37 の UTWSWHS1BinaryToData 関数 (11.34) を用います．第 1 引数はバイナリデータのハンドル，第 2 引数は書き戻すバイナリデータバッファへのポインタ，第 3 引数は書き戻すバイナリデータのバイト数，第 4 引数は 8.3 節で説明した時刻調整を有効にする場合は TRUE を，第 5 引数は常に FALSE を指定します．呼び出し側で心拍，加速度などの生体情報を得るには，行 41 の UTWSWHS1GetDataFromBinary 関数 (11.36) を使います．第 1 引数はバイナリデータのハンドル，第 2 引数はヘッダ情報のコピー先である WHS1MemDataHeader 型変数へのポインタ，第 3 引数は UTWS.dll にストアされた心拍情報，温度，加速度を保存できるだけのメモリを確保した WHS1EcgDataEx 型のバッファへのポインタ，第 4 引数は準備した WHS1EcgDataEx 型の数です．バイナリデータのハンドルから，心拍，加速度などを CSV ファイルとして保存するには，行 43 の UTWSWHS1SaveDataFromBinary 関数 (11.37) を用います，第 1 引数はバイナリデータのハンドル，第 2 引数は保存パスです．

9.2 WHS-1 のデータを非同期で読み出し，完了をコールバック関数で検出する

WHS-1 のメモリを非同期で読み出しし，コールバック関数を使って読み出し完了を検出する方法を示します．

```
 1: void callback_function(void* p)
 2: {
 3:   HANDLE h = reinterpret_cast<HANDLE>(p);
 4:   if (UTWS_NO_ERR == UTWSWHS1ReadMemoryBinaryResult(h)) {
 5:     //読み込み完了後の処理
 6:   }
 7: }

 8: void CUTWSRefMunSampleDlg::OnBnClickedButton3()
 9: {
10:   UINT block_count;
11:   UTWSWHS1GetRecordedSize(hWHS, &block_count);
12:   if (!block_count)
13:     return;
14:   UTWSWHS1ReadMemoryBinary(hWHS, 0, callback_function, hWHS);
15: }
```

非同期読み出しの開始には，行 14 の UTWSWHS1ReadMemoryBinary 関数を使います．第 1 引数は，WHS-1

のハンドルで，第 2 引数はイベントを用いないので 0，第 3 引数はコースバック関数のポインタ，第 4 引数はコールバック関数の引数です．この例では，コールバック関数は行 1〜7 で定義されています．UTWS.dll は読み出しが完了するとコールバック関数を呼び出すので，その後の処理は，コールバック関数の中に記述してください．このコールバック関数は UTWS.dll 側のスレッドで動作していることに注意してください．

9.3 WHS-1 のバイナリ形式で保存されたファイルから読み込む

UTWSWHS1SaveMemoryBinary 関数で保存した，WHS-1 のバイナリデータファイルの読み出し方法を以下に示します．

```
1: HANDLE hBin = UTWSOpenDevice(UTWS_WHS_1_BINARYDATA, 0);
2: UINT count = UTWSWHS1BinaryFileToData(hBin, "file.bin", TRUE, FALSE);
3: //読み込み完了後の操作
4: UTWSCloseDevice(hBin);
```

バイナリデータをストレージから読み込み前に，行 1 の UTWSOpenDevice 関数を使って，バイナリデータのハンドルを取得してください．行 2 の UTWSWHS1BinaryFileToData 関数 (11.35) でバイナリデータを読み込みます．第 1 引数はバイナリデータのハンドル，第 2 引数は保存パス，第 3 引数は 8.3 節の時刻補間をする場合に TURE，第 4 引数は常に FALSE を指定してください．戻り値は，データ数です．

9.4 バイナリデータのフォーマット

先頭の 1 バイトは WHS-1 の仮 ID です．これに続いて WHS-1 から読み出されたバイナリデータが続きます．WHS-1 から送出されるデータは 1 ブロック 256 バイトのヘッダとフッタにそれぞれ 0x01 と 0x04 が付与されていますが，これらは削除されています．またデータの終端を示す 0xFFFFFFFF も同様に削除されています．

10 WHS-1 のファームウェアバージョンを取得する

WHS-1 のファームウェアバージョンを取得したいときは UTWSWHS1Version(11.25 節) を使います．例を次に示します．

```
1:  UINT ver;
2:  UTWSWHS1Version(hWHS, &ver);
```

第 1 引数は WHS-1 へのハンドルです．第 2 引数は UINT 型へのポインタで，ファームウェアバージョンが返されます．このバージョンが 238 以上のとき心拍間隔 2 モードが使えます．また，バージョン 500 以上では，心拍数モード，加速度平均値モード，フラッシュメモリフル時の上書きモードが削除されて，モニタモードが追加されました．

11 API リファレンス

11.1 UTWSGetErrorMessage

DLL 内のエラー状態を返します.

構文

```
UINT WINAPI UTWSGetErrorMessage(
  HANDLE _handle,
  LPSTR _err_message,
  UINT _max_length
);
```

引数

`HANDLE _handle` (*in*)

エラー情報を得ようとするデバイスのハンドを指定します．UTWSOpenDevice 関数 (11.2)，UTWSCloseDevice 関数 (11.3) および UTWSCloseAll 関数 (11.4) ではデバイス毎のエラー情報を取得できないので，0 としてください.

`LPSTR _err_message` (*out opt*)

エラーに関する説明が格納される文字列バッファへのポインタ．この文字列が必要ないときは 0 としてかまいません.

`UINT _max_length` (*in opt*)

`_err_message` の指す文字列バッファのバイト数．ここには，終端の NULL 文字も含みます.

戻り値

直前に発生したエラーのエラー識別子が返されます．エラー識別子はヘッダファイル `utwserrdef.h`(13 節) で定義されています.

解説

`_err_message` の指す文字列バッファは, `_max_length` バイト以上確保して下さい．エラーに関する説明文が`_max_length` よりも長いときは，説明文は`_max_length` の長さまでに切り詰められます．`_err_message` が 0 のとき，または`_max_length` が 1 以下のときは，エラー識別子のみを返します.

11.2 UTWSOpenDevice

WHS-1 や RRD-1 などのデバイスを開きます.

構文

```
HANDLE WINAPI UTWSOpenDevice(
  UINT _device_id,
  UINT _device_no
);
```

引数

UINT _device_id (*in*)

オープンしたいデバイスの識別子を指定してください．デバイスと識別子の対応は以下の通りです．

デバイス	識別子
WHS-1	UTWS_WHS_1
WHS-1 から読み込んだバイナリデータ	UTWS_WHS_1_BINARYDATA
RRD-1	UTWS_RRD_1

UINT _device_no (*in*)

WHS-1 をオープンするときは，何番目のデバイスをオープンするか指定して下さい．たとえば，5 台接続されていて 3 番目のデバイスを開きたいときは 2 を与えて下さい．1 台しか接続していないときは 0 として下さい．引数_device_id に UTWS_WHS_1 以外を指定したとき，この引数は無視されます．

戻り値

オープンされたデバイスのハンドルが返されます．ハンドルは他の関数で利用するので保存して下さい．エラーのときは 0 が返されます．エラーの状態は UTWSGetErrorMessage 関数 (11.1 節) で調べることができます．

解説

第 1 引数に UTWS_WHS_1 を伴って呼び出すと，FTD2XX.DLL がロードされます．

エラー値

関数実行後 UTWSGetErrorMessage 関数で返されるエラーの状態は次の通りです．

UTWS_NO_ERR

正常終了．

UTWS_ERR_UNKNOWN_DEVICE_ID

第 1 引数で与えられた識別子が間違っています．

UTWS_ERR_CANNOT_OPEN

デバイスを開くことができませんでした．

UTWS_ERR_CANNOT_CHANGE_MODE

RRD-1 で受信モードを変更することができませんでした.

UTWS_ERR_CLOSED
RRD-1 で受信モードに変更しようとしたとき，すでにデバイスが閉じられていた.

UTWS_ERR_CANNOT_GET_SERIAL
WHS-1 で，デバイスのシリアル番号を取得できなかった．あるいは，WHS-1 が接続されていない.

UTWS_ERR_CANNOT_LOAD_FTD2XX
パスの通ったディレクトリに FTD2XX.DLL が存在しないか，何らかの理由で読み込めませんでした.

UTWS_ERR_UNKNOWN
原因不明の例外が発生した.

11.3 UTWSCloseDevice

WHS-1 や RRD-1 などのデバイスをクローズします.

構文

```
BOOL WINAPI UTWSCloseDevice(
  HANDLE _handle
);
```

引数

HANDLE _handle (in)
クローズしようとするデバイスのハンドル.

戻り値

成功すると TRUE，エラーのときは FALSE が返ります．エラーの状態は UTWSGetErrorMessage 関数 (11.1 節) で調べることができます.

解説

この関数を実行すると，引数のハンドルは無効な値となります.

UTWSOpenDevice 関数の第 1 引数に UTWS_WHS_1 を伴って呼び出すことで得られたハンドルを引数として UTWSCloseDevice を実行するには FTD2XX.DLL が必要です.

エラー値

関数実行後 UTWSGetErrorMessage 関数で返されるエラーの状態は次の通りです.

UTWS_NO_ERR
正常終了.

UTWS_ERR_INVALID_HANDLE
第 1 引数で与えられたハンドルが，有効ではありません.

UTWS_ERR_UNKNOWN
原因不明の例外が発生した.

11.4 UTWSCloseAll

WHS-1 や RRD-1 などのすべてのデバイスをクローズします.

構文

```
VOID WINAPI UTWSCloseAll(VOID);
```

引数

ありません.

戻り値

ありません.

解説

この関数を実行すると，関数 UTWSOpenDevice で得られたすべてのハンドルは無効になります.
オープンされたデバイスに WHS-1 が含まれているとき，FTD2XX.DLL が必要です.

エラー値

関数実行後 UTWSGetErrorMessage 関数で返されるエラーの状態は次の通りです.

UTWS_NO_ERR
正常終了.

UTWS_ERR_UNKNOWN
原因不明の例外が発生した.

11.5 UTWSRRD1StartReceiving

RRD-1 でデータ受信を開始します.

構文

```
BOOL WINAPI UTWSRRD1StartReceiving(
  HANDLE _handle,
  HANDLE _event,
  UTWSRRD1CallBakFunc _callBack,
```

```
  VOID* _callBackArg
);
```

引数

`HANDLE _handle` (*in*)
受信を開始する RRD-1 のハンドル.

`HANDLE _event` (*in opt*)
Win32 API の `CreateEvent` 関数で得られるイベントオブジェクトのハンドル．WHS-1 からデータを受信するとシグナル状態になります．イベントオブジェクトを使用しないときは 0 として下さい.

`UTWSRRD1CallBakFunc _callBack` (*in opt*)
WHS-1 からデータを受信すると呼び出されるコールバック関数へのポインタ．コールバック関数を使用しないときは 0 として下さい．関数の型 `UTWSRRD1CallBakFunc` については 11.14 節を参照して下さい.

`VOID* _callBackArg` (*in opt*)
コールバック関数 `(*_callBack)()` が呼び出されるときの第 1 引数として`_callBackArg` が使われます．コールバック関数を使用しないときは無視されます.

戻り値

成功すれば TRUE を，失敗で FALSE を返します．エラーの状態は `UTWSGetErrorMessage` 関数 (11.1 節) で調べることができます.

解説

データ受信が開始されると，RRD-1 で受信したデータはいったん UTWS.dll 内の受信バッファに格納されます．受信バッファのデータは `UTWSRRD1GetData` 関数 (11.8 節) または `UTWSRRD1GetDataEx` 関数 (11.9 節) によって取り出すことができます．この関数が実行されると，取り出されたデータは受信バッファから削除されます．`UTWSRRD1GetData` 関数または `UTWSRRD1GetDataEx` 関数が呼び出されない間，受信バッファの末尾に受信したデータが追加されます．受信バッファには最大 1024 個の受信データを格納できます．あふれた場合は最過去のデータから消去されます.

イベントオブジェクトを使用する場合，第 2 引数に与えるオブジェクトは `CreateEvent` 関数から手動イベントとして生成されたもので，かつ非シグナル状態でなければなりません．WHS-1 からデータを受信するとイベントはシグナル状態になり，`UTWSRRD1GetData` 関数または `UTWSRRD1GetDataEx` 関数によって UTWS.dll 内の受信バッファが空になると非シグナル状態となります．ユーザがイベントを非シグナル状態にする必要はありません.

エラー値

関数実行後 `UTWSGetErrorMessage` 関数で返されるエラーの状態は次の通りです.

`UTWS_NO_ERR`

正常終了.

UTWS_ERR_INVALID_HANDLE
第 1 引数で与えられたハンドルが，有効ではありません.

UTWS_ERR_CLOSED
第 1 引数で与えられたハンドルのデバイスは，オープンされていません.

UTWS_ERR_TIMEOUT
RRD-1 が応答しませんでした.

UTWS_ERR_UNKNOWN
原因不明の例外が発生しました.

11.6 UTWSRRD1StopReceiving

WHS-1 からのデータ受信を停止します.

構文

```
BOOL CALLBACK UTWSRRD1StopReceiving(
  HANDLE _handle
);
```

引数

HANDLE _handle (in)
受信を停止する RRD-1 のハンドル.

戻り値

成功すれば TRUE を，失敗で FALSE を返します．エラーの状態は UTWSGetErrorMessage 関数 (11.1 節) で調べることができます.

解説

この関数が実行されると，UTWS.dll 内の受信バッファは空になります.

エラー値

関数実行後 UTWSGetErrorMessage 関数で返されるエラーの状態は次の通りです.

UTWS_NO_ERR
正常終了.

UTWS_ERR_INVALID_HANDLE

第 1 引数で与えられたハンドルが，有効ではありません．

UTWS_ERR_CLOSED
第 1 引数で与えられたハンドルのデバイスは，オープンされていません．

UTWS_ERR_UNKNOWN
原因不明の例外が発生しました．

11.7 UTWSRRD1DataCount

RRD-1 で受信したデータが UTWS.dll 内の受信バッファに何個入っているか返します．

構文

```
UINT WINAPI UTWSRRD1DataCount(
  HANDLE _handle
 );
```

引数

HANDLE _handle (in)
受信バッファのデータ数を調べようとする RRD-1 のハンドル．

戻り値

UTWS.dll 内の受信バッファにある受信データ数を返します．失敗のとき-1 を返します．エラーの状態は UTWSGetErrorMessage 関数 (11.1 節) で調べることができます．

エラー値

関数実行後 UTWSGetErrorMessage 関数で返されるエラーの状態は次の通りです．

UTWS_NO_ERR
正常終了．

UTWS_ERR_INVALID_HANDLE
第 1 引数で与えられたハンドルが，有効ではありません．

UTWS_ERR_UNKNOWN
原因不明の例外が発生しました．

11.8 UTWSRRD1GetData

UTWS.dll の受信バッファから受信データを 1 つ取り出し，取り出した受信データを受信バッファから削除します．

構文

```
BOOL WINAPI UTWSRRD1GetData(
  HANDLE _handle,
  RRD1Data* _data
 );
```

引数

`HANDLE _handle` (*in*)
　受信データを取り出そうとする RRD-1 のハンドル.

`RRD1Data* _data` (*out*)
　受信データが格納される構造体へのポインタ．構造体 `RRD1Data` については 12.2 節を参照して下さい.

戻り値

　成功するか，`_data` が 0 ならば何もせずに TRUE を返します．失敗するか UTWS.dll 内の受信バッファに取り出すべきデータがないときは FALSE を返します．エラーの状態は `UTWSGetErrorMessage` 関数 (11.1 節) で調べることができます.

エラー値

　関数実行後 `UTWSGetErrorMessage` 関数で返されるエラーの状態は次の通りです.

`UTWS_NO_ERR`
　正常終了.

`UTWS_ERR_INVALID_HANDLE`
　第 1 引数で与えられたハンドルが，有効ではありません.

`UTWS_ERR_EMPTY_DATA`
　UTWS.dll 内の受信バッファが空でした.

`UTWS_ERR_UNKNOWN`
　原因不明の例外が発生しました.

11.9　UTWSRRD1GetDataEx

　UTWS.dll の受信バッファから受信データを 1 つ取り出し，取り出した受信データを受信バッファから削除します．`RRD1DataEx` 構造体を通じてミリ秒単位の受信時刻を得られます.

構文

```
BOOL WINAPI UTWSRRD1GetData(
```

```
  HANDLE _handle,
  RRD1DataEx* _data
 );
```

引数

`HANDLE _handle` (*in*)

受信データを取り出そうとする RRD-1 のハンドル.

`RRD1DataEx* _data` (*out*)

受信データが格納される構造体へのポインタ．構造体 `RRD1DataEx` については 12.3 節を参照して下さい.

戻り値

成功するか，`_data` が 0 ならば何もせずに TRUE を返します．失敗するか UTWS.dll 内の受信バッファに取り出すべきデータがないときは FALSE を返します．エラーの状態は `UTWSGetErrorMessage` 関数 (11.1 節) で調べることができます.

エラー値

関数実行後 `UTWSGetErrorMessage` 関数で返されるエラーの状態は次の通りです.

`UTWS_NO_ERR`

正常終了.

`UTWS_ERR_INVALID_HANDLE`

第 1 引数で与えられたハンドルが，有効ではありません.

`UTWS_ERR_EMPTY_DATA`

UTWS.dll 内の受信バッファが空でした.

`UTWS_ERR_UNKNOWN`

原因不明の例外が発生しました.

11.10 UTWSRRD1IsOpen

RRD-1 がオープンしているか調べる.

構文

```
BOOL WINAPI UTWSRRD1IsOpen(
  HANDLE _handle
);
```

引数

HANDLE _handle (*in*)
　調べようとする RRD-1 のハンドル.

戻り値

　デバイスがオープンしていれば TURE，デーバイスがクローズしているか関数が失敗すれば FALSE を返します．エラーの状態は UTWSGetErrorMessage 関数 (11.1 節) で調べることができます.

エラー値

　関数実行後 UTWSGetErrorMessage 関数で返されるエラーの状態は次の通りです.

UTWS_NO_ERR
　正常終了.

UTWS_ERR_INVALID_HANDLE
　第 1 引数で与えられたハンドルが，有効ではありません.

UTWS_ERR_UNKNOWN
　原因不明の例外が発生しました.

11.11　UTWSRRD1SetCallBackDisconnect

　RRD-1 が PC から取り外されたときに呼び出されるコールバック関数を登録します.

構文

```
BOOL WINAPI UTWSRRD1SetCallBackDisconnect(
  HANDLE _handle,
  UTWSRRD1CallBakFunc _callBack,
  VOID* _callBackArg
);
```

引数

HANDLE _handle (*in*)
　RRD-1 のハンドル.

UTWSRRD1CallBakFunc _callBack (*in*)
　RRD-1 が PC から取り外されたときに呼び出されるコールバック関数へのポインタ．コールバック関数を使用しないときは 0 として下さい．関数の型 UTWSRRD1CallBakFunc については 11.14 節を参照して下さい.

VOID* _callBackArg (*in*)

コールバック関数 (*_callBack)() が呼び出されるときの第 1 引数として_callBackArg が使われます．コールバック関数を使用しないときは無視されます．

戻り値

成功すれば TRUE を，失敗で FALSE を返します．エラーの状態は UTWSGetErrorMessage 関数 (11.1 節) で調べることができます．

解説

RRD-1 が PC から取り外されると，当該 RRD-1 のハンドルはクローズされます．そして，コールバック関数が登録されていれば，それが呼び出されます．コールバック関数の第 1 引数はここで説明する UTWSRRD1SetCallBackDisconnect 関数の第 3 引数で，第 2 引数は PC から取り外された RRD-1 のハンドルです．コールバック関数は UTWSRRD1SetCallBackDisconnect 関数を呼び出したスレッドと異なるスレッドで呼び出されることに注意して下さい．

エラー値

関数実行後 UTWSGetErrorMessage 関数で返されるエラーの状態は次の通りです．

UTWS_NO_ERR
正常終了．

UTWS_ERR_INVALID_HANDLE
第 1 引数で与えられたハンドルが，有効ではありません．

UTWS_ERR_UNKNOWN
原因不明の例外が発生しました．

11.12 UTWSRRD1GetLocalAddress

RRD-1 の自アドレスを取得します．

構文

```
BOOL WINAPI UTWSRRD1GetLocalAddress(
  HANDLE _handle,
  CHAR* _address
);
```

引数

HANDLE _handle (*in*)
自アドレスを取得しようとする RRD-1 のハンドル．

CHAR* _address (*out*)

自アドレスが返されるバッファへのポインタ．null ターミネート文字列で自アドレスを返します．11 バイト以上確保して下さい．

戻り値

成功すれば TRUE を，失敗で FALSE を返します．エラーの状態は UTWSGetErrorMessage 関数 (11.1 節) で調べることができます．

エラー値

関数実行後 UTWSGetErrorMessage 関数で返されるエラーの状態は次の通りです．

UTWS_NO_ERR
正常終了．

UTWS_ERR_INVALID_HANDLE
第 1 引数で与えられたハンドルが，有効ではありません．

UTWS_ERR_CLOSED
第 1 引数で与えられたハンドルのデバイスは，オープンされていません．

UTWS_ERR_CANNOT_CHANGE_MODE
RRD-1 が設定モードに移行できませんでした．

UTWS_ERR_FAILED
RRD-1 の設定情報を得ることができませんでした．

UTWS_ERR_TIMEOUT
RRD-1 が応答しませんでした．

UTWS_ERR_UNKNOWN
原因不明の例外が発生しました．

11.13 UTWSRRD1SetLocalAddress

RRD-1 の自アドレスを設定します．

構文

```
BOOL WINAPI UTWSRRD1SetLocalAddress(
  HANDLE _handle,
  const CHAR* _address
);
```

引数

`HANDLE _handle` (in)

自アドレスを設定しようとする RRD-1 のハンドル.

`const CHAR* _address` (in)

設定しようとする自アドレスが格納されたバッファへのポインタ．null ターミネート文字列としてください.

戻り値

成功すれば TRUE を，失敗で FALSE を返します．エラーの状態は `UTWSGetErrorMessage` 関数 (11.1 節) で調べることができます.

解説

自アドレスは 10 文字で，16 進数を表す文字列です．したがって ‘0’∼‘9’，‘ A’∼‘F’，‘a’∼‘f’ 以外の文字列が与えられると関数は失敗します.

エラー値

関数実行後 `UTWSGetErrorMessage` 関数で返されるエラーの状態は次の通りです.

`UTWS_NO_ERR`

正常終了.

`UTWS_ERR_INVALID_HANDLE`

第 1 引数で与えられたハンドルが，有効ではありません.

`UTWS_ERR_CLOSED`

第 1 引数で与えられたハンドルのデバイスは，オープンされていません.

`UTWS_ERR_CANNOT_CHANGE_MODE`

RRD-1 が設定モードに移行できませんでした.

`UTWS_ERR_BAD_DESTINATION_ADDRESS`

第 2 引数で与えられた自アドレスが間違っています.

`UTWS_ERR_FAILED`

RRD-1 の設定情報を得ることができませんでした.

`UTWS_ERR_TIMEOUT`

RRD-1 が応答しませんでした.

`UTWS_ERR_UNKNOWN`

原因不明の例外が発生しました.

11.14 UTWSRRD1CallBakFunc

UTWSRRD1StartReceiving 関数などで用いられるコールバック関数の typedef.

構文

```
typedef VOID (CALLBACK *UTWSRRD1CallBakFunc)(VOID*, VOID*);
```

11.15 UTWSWHS1CountConnected

接続されている WHS-1 を検索し，その台数を調べます.

構文

```
UINT WINAPI UTWSWHS1CountConnected(
  UINT _numConnected,
  UINT _timeout
);
```

引数

UINT _numConnected (in)

PC に接続している WHS-1 の台数を与えて下さい．もし，台数が不明な場合は 0 を与えて下さい.

UINO _timeout (in)

WHS-1 を検索する時間，タイムアウトを ms 単位で与えて下さい.

戻り値

検索の結果，検出された WHS-1 の台数を返します．エラーのときは-1 が返されます．エラーの状態は UTWSGetErrorMessage 関数 (11.1 節) で調べることができます.

解説

この関数は_timeout 間，20ms 間隔で WHS-1 を検索します．_numConnected が 0 でない場合，検出した接続数が_numConnected 以上のとき制御が戻ります．このときの戻り値は_numConnected よりも大きいことがあります．_numConnected が 0 のとき，_timeout 間制御が戻りません.

この関数を実行すると FTD2XX.DLL がロードされます.

エラー値

関数実行後 UTWSGetErrorMessage 関数で返されるエラーの状態は次の通りです.

UTWS_NO_ERR

正常終了.

UTWS_ERR_FAILED
検索に失敗しました.

UTWS_ERR_CANNOT_LOAD_FTD2XX
パスの通ったディレクトリに FTD2XX.DLL が存在しない，または何らかの理由で読み込めませんでした.

UTWS_ERR_UNKNOWN
原因不明の例外が発生しました.

11.16 UTWSWHS1ReadConfig

WHS-1 の設定状態を読み取る.

構文

```
BOOL WINAPI UTWSWHS1ReadConfig(
  HANDLE _handle,
  WHS1Config* _readConfig
);
```

引数

HANDLE _handle (in)
設定状態を取得しようとする WHS-1 のハンドル.

WHS1Config* _readConfig (out)
WHS-1 の設定状態が返される WHS1Config 構造体へのポインタ．返される設定状態の詳細は 12.4 節を参照して下さい.

戻り値

成功すれば TRUE を，失敗で FALSE を返します．エラーの状態は UTWSGetErrorMessage 関数 (11.1 節) で調べることができます.

解説

この関数の実行には FTD2XX.DLL が必要です.

エラー値

関数実行後 UTWSGetErrorMessage 関数で返されるエラーの状態は次の通りです.

UTWS_NO_ERR
正常終了.

UTWS_ERR_INVALID_HANDLE

第 1 引数で与えられたハンドルが，有効ではありません．

UTWS_ERR_FAILED
WHS-1 の設定が失敗しました．

UTWS_ERR_TIMEOUT
WHS-1 が応答しませんでした．

UTWS_ERR_UNKNOWN
原因不明の例外が発生しました．

11.17 UTWSWHS1WriteConfig

WHS-1 のモードを設定する．

構文

```
BOOL WINAPI UTWSWHS1WriteConfig(
  HANDLE _handle,
  const WHS1Config* _readConfig,
  BYTE _mem_mode,
  WORD _year,
  WORD _month,
  WORD _day,
  WORD _hour,
  WORD _min,
  WORD _sec
);
```

引数

HANDLE _handle (*in*)
モード設定しようとする WHS-1 のハンドル．

const WHS1Config* _readConfig (*in*)
WHS-1 の設定状態が格納された WHS1Config 構造体へのポインタ．節 12.4 のメンバ説明で，「設定可能」とされたメンバすべてに有効な値を設定する必要があります．

BYTE _mem_mode (*in opt*)
メモリがいっぱいになったときの書き込み動作を指定します．メモリモードに設定するとき，以下のいずれかの値を必ず設定して下さい．無線モードに設定するときは無視されます．また WHS-1ver500 以降は，メモリがいっぱいになったとき常に記録停止なので，この引数は無視されます．

モード	値
最過去データから上書き	0
記録停止	1

WORD _year (*in opt*)
WORD _month (*in opt*)
WORD _day (*in opt*)
WORD _hour (*in opt*)
WORD _min (*in opt*)
WORD _sec (*in opt*)

メモリモードに設定するときは，同時に現在時刻を設定する必要があります．上記の引数に現在時刻を与えて下さい．引数_year は 2000 から 2255 までとし，その他の引数は日付として有効な値を与えて下さい．無線モードに設定するときはこれらの引数は無視されます．

戻り値

成功すれば TRUE を，失敗で FALSE を返します．エラーの状態は UTWSGetErrorMessage 関数 (11.1 節) で調べることができます．

解説

この関数の実行には FTD2XX.DLL が必要です．

エラー値

関数実行後 UTWSGetErrorMessage 関数で返されるエラーの状態は次の通りです．

UTWS_NO_ERR
正常終了．

UTWS_ERR_INVALID_HANDLE
第 1 引数で与えられたハンドルが，有効ではありません．

UTWS_ERR_INVALID_PARAMETER
引数の設定が間違っています．日付けの他，操作の対象が WHS-1ver500 以降かどうかに注意してください．

UTWS_ERR_FAILED
WHS-1 の設定が失敗しました．

UTWS_ERR_TIMEOUT
WHS-1 が応答しませんでした．

UTWS_ERR_UNKNOWN
原因不明の例外が発生しました．

11.18 UTWSWHS1GetDestinationAddress

WHS-1 の宛先アドレスを取得します.

構文

```
BOOL WINAPI UTWSWHS1GetDestinationAddress(
  HANDLE _handle,
  CHAR* _address
);
```

引数

`HANDLE _handle` (*in*)

宛先アドレスを取得しようとする WHS-1 のハンドル.

`CHAR* _address` (*out*)

宛先アドレスが返されるバッファへのポインタ．null ターミネート文字列で宛先アドレスを返します．11 バイト以上確保して下さい.

戻り値

成功すれば TRUE を，失敗で FALSE を返します．エラーの状態は `UTWSGetErrorMessage` 関数 (11.1 節) で調べることができます.

解説

この関数の実行には FTD2XX.DLL が必要です.

エラー値

関数実行後 `UTWSGetErrorMessage` 関数で返されるエラーの状態は次の通りです.

`UTWS_NO_ERR`

正常終了.

`UTWS_ERR_INVALID_HANDLE`

第 1 引数で与えられたハンドルが，有効ではありません.

`UTWS_ERR_FAILED`

WHS-1 の設定が失敗しました.

`UTWS_ERR_TIMEOUT`

WHS-1 が応答しませんでした.

`UTWS_ERR_UNKNOWN`

原因不明の例外が発生しました.

11.19 UTWSWHS1SetDestinationAddress

WHS-1 の宛先アドレスを設定します.

構文

```
BOOL WINAPI UTWSWHS1SetDestinationAddress(
  HANDLE _handle,
  const CHAR* _address
);
```

引数

`HANDLE _handle` (in)

宛先アドレスを設定しようとする WHS-1 のハンドル.

`const CHAR* _address` (in)

設定しようとする宛先アドレスが格納されたバッファへのポインタ．null ターミネート文字列としてください.

戻り値

成功すれば TRUE を，失敗で FALSE を返します．エラーの状態は `UTWSGetErrorMessage` 関数 (11.1 節) で調べることができます.

解説

宛先アドレスは 10 文字で，16 進数を表す文字列です．したがって ‘0’〜‘9’，‘ A’〜‘F’，‘a’〜‘f’ 以外の文字列が与えられると関数は失敗します.

この関数の実行には FTD2XX.DLL が必要です.

エラー値

関数実行後 `UTWSGetErrorMessage` 関数で返されるエラーの状態は次の通りです.

`UTWS_NO_ERR`

正常終了.

`UTWS_ERR_INVALID_HANDLE`

第 1 引数で与えられたハンドルが，有効ではありません.

`UTWS_ERR_BAD_DESTINATION_ADDRESS`

第 2 引数で与えられた自アドレスが間違っています.

`UTWS_ERR_FAILED`

WHS-1 の設定が失敗しました.

UTWS_ERR_TIMEOUT
WHS-1 が応答しませんでした.

UTWS_ERR_UNKNOWN
原因不明の例外が発生しました.

11.20 UTWSWHS1ReadMemory

この関数は互換性のために残されており，使用は推奨されません.
WHS-1 からフラッシュメモリの内容をダウンロードし，UTWS.dll 内のバッファに保存します.

構文

```
UINT WINAPI UTWSWHS1ReadMemory(
  HANDLE _handle
);
```

引数

HANDLE _handle (in)
フラッシュメモリの内容をダウンロードしようとする WHS-1 のハンドル.

戻り値

ダウンロードしたデータ数を返します．関数が失敗したときは -1 を返します．エラーの状態は UTWSGetErrorMessage 関数 (11.1 節) で調べることができます.

解説

ダウンロードしたデータをメイン側で得るには UTWSWHS1GetReceivedMemory 関数 (11.22 節) を用います．ファイルに保存する場合は UTWSWHS1SaveReceivedMemory 関数 (11.24 節) を利用できます.
この関数の実行には FTD2XX.DLL が必要です.

エラー値

関数実行後 UTWSGetErrorMessage 関数で返されるエラーの状態は次の通りです.

UTWS_NO_ERR
正常終了.

UTWS_ERR_INVALID_HANDLE
第 1 引数で与えられたハンドルが，有効ではありません.

UTWS_ERR_FAILED

WHS-1 からフラッシュメモリの内容のダウンロードに失敗しました．あるいは，フラッシュメモリにデータがありませんでした．

`UTWS_ERR_TIMEOUT`
WHS-1 が応答しませんでした．

`UTWS_ERR_UNKNOWN`
原因不明の例外が発生しました．

11.21 UTWSWHS1ReadMemoryEx

WHS-1 からフラッシュメモリの内容をダウンロードし，UTWS.dll 内のバッファに保存します．

構文

```
BOOL WINAPI UTWSWHS1ReadMemoryEx(
  HANDLE _handle,
  WHS1MemDataHeader* _header
);
```

引数

`HANDLE _handle` (in)
フラッシュメモリの内容をダウンロードしようとする WHS-1 のハンドル．

`WHS1MemDataHeader* _header` (out)
WHS-1 のフラッシュメモリに記録されたデータのヘッダ情報を格納する構造体 `WHS1MemDataHeader` へのポインタ．構造体の内容については 12.7 節を参照してください．

戻り値

成功すれば TRUE を，失敗で FALSE を返します．エラーの状態は `UTWSGetErrorMessage` 関数 (11.1 節) で調べることができます．

解説

ダウンロードしたデータをメイン側で得るには `UTWSWHS1GetReceivedMemory` 関数 (11.22 節) を，または心拍間隔 2 モードかつ加速度ピークホールド

`(3 == _header->ecg_mode && 0 == _header->acc_mode)`

のとき `UTWSWHS1GetReceivedMemoryEx` 関数 (11.23 節) 用います．ファイルに保存する場合は `UTWSWHS1SaveReceivedMemory` 関数 (11.24 節) を利用できます．

この関数の実行には FTD2XX.DLL が必要です．

エラー値

関数実行後 `UTWSGetErrorMessage` 関数で返されるエラーの状態は次の通りです．

UTWS_NO_ERR
正常終了.

UTWS_ERR_INVALID_HANDLE
第 1 引数で与えられたハンドルが，有効ではありません.

UTWS_ERR_FAILED
WHS-1 からフラッシュメモリの内容のダウンロードに失敗しました．あるいは，フラッシュメモリにデータがありませんでした.

UTWS_ERR_TIMEOUT
WHS-1 が応答しませんでした.

UTWS_ERR_UNKNOWN
原因不明の例外が発生しました.

11.22 UTWSWHS1GetReceivedMemory

関数 UTWSWHS1ReadMemoryEx によって UTWS.dll のバッファに保存された WHS-1 のフラッシュメモリの内容をメイン側のバッファにコピーします．関数 UTWSWHS1ReadMemoryEx で返されたヘッダ情報が，心拍間隔 2 モードかつ加速度ピークホールドのときは，UTWSWHS1GetReceivedMemoryEx 関数 (11.23 節) を使用して下さい.

構文

```
BOOL WINAPI UTWSWHS1GetReceivedMemory(
  HANDLE _handle,
  WHS1MemDataHeader* _header,
  WHS1EcgData* _data,
  UINT _size,
  BOOL _time_adjust
);
```

引数

HANDLE _handle (*in*)
フラッシュメモリの内容をコピーしようとする WHS-1 のハンドル.

WHS1MemDataHeader* _header (*out*)
WHS-1 のフラッシュメモリに記録されたデータのヘッダ情報を格納する構造体 WHS1MemDataHeader へのポインタ．構造体の内容については 12.7 節を参照してください.

WHS1EcgData* _data (*out opt*)

WHS-1 のフラッシュメモリに記録された時刻，心拍情報，温度，加速度の組を表す構造体 WHS1EcgData へのポインタ．この引数が 0 のとき，ヘッダ情報のみが得られます．構造体の内容については 12.5 節を参照して下さい．

UINT _size (*in opt*)
メイン側でバッファとして用意した WHS1EcgData の数．UTWS.dll がこれより多くのデータ数をストアしているとき，_data の指すバッファへは_size 個のデータだけがコピーされます．_data が 0 のとき，この引数は無視されます．

BOOL _time_adjust(*in opt*)
第 3 引数で返される構造体 WHS1EcgData の時刻要素の調整方法を与えます．詳しくは 8.3 節を参照してください．第 3 引数が 0 のとき無視されます．

戻り値
成功すれば TRUE を，失敗で FALSE を返します．エラーの状態は UTWSGetErrorMessage 関数 (11.1 節) で調べることができます．

解説
関数 UTWSWHS1ReadMemoryEx から得られるヘッダ情報から，ダウンロードされたデータ数を調べ，少なくともそのデータ数だけ構造体 WHS1EcgData の領域を確保し，そのポインタを第 3 引数に与えて下さい．UTWS.dll 内のストアのデータ数が 0 だった場合 FALSE が帰ります．
この関数の実行によって，UTWS.dll 内にストアしたフラッシュメモリの内容が消去されることはありません．

エラー値
関数実行後 UTWSGetErrorMessage 関数で返されるエラーの状態は次の通りです．

UTWS_NO_ERR
正常終了．

UTWS_ERR_INVALID_HANDLE
第 1 引数で与えられたハンドルが，有効ではありません．

UTWS_ERR_NO_DATA
UTWS.dll 内のストアに取得可能なデータがありませんでした．

UTWS_ERR_UNKNOWN
原因不明の例外が発生しました．

11.23 UTWSWHS1GetReceivedMemoryEx

関数 UTWSWHS1ReadMemoryEx によって UTWS.dll のバッファに保存された WHS-1 のフラッシュメモリの内容をメイン側のバッファにコピーします．関数 UTWSWHS1ReadMemoryEx で返されたヘッダ情報が，心拍

間隔 2 モードかつ加速度ピークホールドのときにだけ使用して下さい.

構文

```
BOOL WINAPI UTWSWHS1GetReceivedMemoryEx(
  HANDLE _handle,
  WHS1MemDataHeader* _header,
  WHS1EcgDataEx* _data,
  UINT _size,
  BOOL _time_adjust
);
```

引数

`HANDLE _handle` (*in*)
フラッシュメモリの内容をコピーしようとする WHS-1 のハンドル.

`WHS1MemDataHeader* _header` (*out*)
WHS-1 のフラッシュメモリに記録されたデータのヘッダ情報を格納する構造体 `WHS1MemDataHeader` へのポインタ．構造体の内容については 12.7 節を参照してください.

`WHS1EcgDataEx* _data` (*out opt*)
WHS-1 のフラッシュメモリに記録された時刻，心拍情報，温度，加速度の組を表す構造体 `WHS1EcgDataEx` へのポインタ．この引数が 0 のとき，ヘッダ情報のみが得られます．構造体の内容については 12.6 節を参照して下さい.

`UINT _size` (*in opt*)
メイン側でバッファとして用意した `WHS1EcgData` の数．UTWS.dll がこれより多くのデータ数をストアしているとき，`_data` の指すバッファへは`_size` 個のデータだけがコピーされます．`_data` が 0 のとき，この引数は無視されます.

`BOOL _time_adjust`(*in opt*)
第 3 引数で返される構造体 `WHS1EcgData` の時刻要素の調整方法を与えます．詳しくは 8.3 節を参照してください．第 3 引数が 0 のとき無視されます.

戻り値

成功すれば TRUE を，失敗で FALSE を返します．エラーの状態は `UTWSGetErrorMessage` 関数 (11.1 節) で調べることができます.

解説

関数 `UTWSWHS1ReadMemoryEx` から得られるヘッダ情報から，ダウンロードされたデータ数を調べ，少なくともそのデータ数だけ構造体 `WHS1EcgDataEx` の領域を確保し，そのポインタを第 3 引数に与えて下さい．UTWS.dll 内のストアのデータ数が 0 だった場合 FALSE が帰ります.

この関数の実行によって，UTWS.dll 内にストアしたフラッシュメモリの内容が消去されることはありません．

エラー値

関数実行後 UTWSGetErrorMessage 関数で返されるエラーの状態は次の通りです．

UTWS_NO_ERR
正常終了．

UTWS_ERR_INVALID_HANDLE
第 1 引数で与えられたハンドルが，有効ではありません．

UTWS_ERR_NO_DATA
UTWS.dll 内のストアに取得可能なデータがありませんでした．

UTWS_ERR_UNKNOWN
原因不明の例外が発生しました．

11.24 UTWSWHS1SaveReceivedMemory

関数 UTWSWHS1ReadMemory によって UTWS.dll のバッファに保存された WHS-1 のフラッシュメモリの内容をファイルに保存します．

構文

```
BOOL WINAPI UTWSWHS1SaveReceivedMemory(
  HANDLE _handle,
  LPCSTR _path,
  BOOL _time_adjust
);
```

引数

HANDLE _handle (in)
フラッシュメモリの内容をファイルに保存しようとする WHS-1 のハンドル．

LPCSTR _path (in)
保存先のパス名．null ターミネート文字列として下さい．

BOOL _time_adjust$(in\ opt)$
心拍情報，加速度，温度の組に付与される時刻情報の調整方法を与えます．詳しくは 8.3 節を参照してください．

戻り値

成功すれば TRUE を，失敗で FALSE を返します．エラーの状態は `UTWSGetErrorMessage` 関数 (11.1 節) で調べることができます．

解説

この関数を実行したときに UTWS.dll 内のストアのデータ数が 0 だった場合 FALSE が帰ります．必ず関数 `UTWSWHS1ReadMemoryEx` を先行して実行し，WHS-1 のフラッシュメモリの内容を UTWS.dll 内のストアに保存して下さい．

保存されるファイルのフォーマットはすべての行でカンマ ‘,’ で区切られた CSV 形式です．各行の終端は復帰文字が付加されます．

第 1 行目は WHS-1 の仮 ID が保存されます．第 1 列は固定値 “TempID” です，第 2 列は 0〜255 までの仮 ID で，これが x ならば第 1 行は，

```
TempID,x
```

となります．

第 2 行目は心拍モードで 1 列のみです．心拍間隔モードの時 “RRI”，心拍数モードの時 “HR” です．つまり，

```
RRI|HR
```

です．

第 3 行目は加速度モードを表します．第 1 列は固定値で “acc_mode” です．第 2 列は加速度モードがピークホールドの時 “peak”，移動平均の時 “avg” です．つまり

```
acc_mode,peak|avg
```

です．

第 4 行はデータの列名です．データの列順は時刻，心拍情報，温度，x 軸加速度，y 軸加速度，z 軸加速度を表すので，

```
time,HR|RRI,temperature, Acc X,  Acc Y, Acc Z[, Acc nX, Acc nY, Acc nZ]
```

です．2 列目は心拍間隔モードの時 “RRI”，心拍数モードの時 “HR” です．心拍間隔 2 モードかつ加速度ピークホールドの時，`[, Acc nX, Acc nY, Acc nZ]` が加わります．ここで `Acc X` の列は絶対値が最大である加速度値で，`Acc nX` は，`Acc X` との差の絶対値が最大である加速度値です．Y，Z 軸加速度についても同様です．

5 行目以降がデータ部です．第 1 列の時刻は次のフォーマットに従います．

```
yyyy/mm/dd hh:mm:ss.000
```

年，月，日，時，分，秒，ミリ秒は上記に表示された通りの桁数の固定長で，桁に満たない数値のときは ‘0’ で詰められます．第 2 列は心拍間隔モードの時．心拍数間隔．心拍数モードの時，過去 8 拍の平均心拍間隔で

す．共に単位は ms です．第 3 列は温度で，単位は摂氏です．第 4 から 6 列，心拍間隔 2 モードかつ加速度ピークホールドでは第 4 から 9 列は加速度で，単位は重力加速度 g です．

エラー値

関数実行後 UTWSGetErrorMessage 関数で返されるエラーの状態は次の通りです．

UTWS_NO_ERR
正常終了．

UTWS_ERR_INVALID_HANDLE
第 1 引数で与えられたハンドルが，有効ではありません．

(UTWS_ERR_CANNOT_OPEN_FILE
第 2 引数で与えられたファイルを開くことができませんでした．

UTWS_ERR_NO_DATA
UTWS.dll 内のストアに保存可能なデータがありませんでした．

UTWS_ERR_UNKNOWN
原因不明の例外が発生しました．

11.25 UTWSWHS1Version

WHS-1 のファームウェアのバージョンを返します．

構文

```
UTWS_API BOOL WINAPI UTWSWHS1Version(
  HANDLE _handle,
  UINT* _ver
);
```

引数

HANDLE _handle (*in*)
ファームウェアのバージョンを得たい WHS-1 のハンドル．

UINT _ver (*out*)
バージョン番号の保存先．

戻り値

成功すれば TRUE を，失敗で FALSE を返します．エラーの状態は UTWSGetErrorMessage 関数 (11.1 節) で調べることができます．

エラー値

関数実行後 UTWSGetErrorMessage 関数で返されるエラーの状態は次の通りです.

UTWS_NO_ERR

正常終了.

UTWS_ERR_INVALID_HANDLE

第 1 引数で与えられたハンドルが，有効ではありません.

UTWS_ERR_UNKNOWN

原因不明の例外が発生しました.

11.26 UTWSWHS1GetRecordedSize

WHS-1 に記録されたデータのブロック数を取得します.

構文

```
UTWS_API BOOL WINAPI UTWSWHS1GetRecordedSize(
  HANDLE _handle,
  UINT* _size_to_read
);
```

引数

HANDLE _handle (in)

データブロック数を得たい WHS-1 のハンドル.

UINT* _size_to_read

得られたデータブロック数を格納するためのバッファへのポインタ.

戻り値

成功すれば TRUE を，失敗で FALSE を返します．エラーの状態は UTWSGetErrorMessage 関数 (11.1 節) で調べることができます.

エラー値

関数実行後 UTWSGetErrorMessage 関数で返されるエラーの状態は次の通りです.

UTWS_NO_ERR

正常終了.

UTWS_ERR_INVALID_HANDLE

第 1 引数で与えられたハンドルが，有効ではありません．

`UTWS_ERR_FAILED`
原因不明のエラーが生じました．

`UTWS_ERR_TIMEOUT`
WHS-1 との通信タイムアウトが発生しました．

`UTWS_ERR_UNKNOWN`
原因不明の例外が発生しました．

11.27 UTWSWHS1ReadMemoryBinary

WHS-1 のフラッシュメモリの内容を非同期に読み出します．

構文

```
UTWS_API BOOL WINAPI UTWSWHS1ReadMemoryBinary(
  HANDLE _handle,
  HANDLE _user_event,
  void (*_user_call_back)(void*),
  void* _user_call_back_arg
);
```

引数

`HANDLE _handle` (*in*)
バイナリデータを読み込みたい WHS-1 のハンドル．

`HANDLE _user_event` (*in opt*)
読み込み完了時にシグナル状態になるイベントへのハンドル．CreateEvent 関数でハンドルを取得し，あらかじめ非シグナル状態としてください．使用しないときは 0 としてください．

`void (*_user_call_back)(void*)` (*in opt*)
読み込み完了時に呼び出されるコールバック関数．使用しないときは 0 としてください．

`void* _user_call_back_arg` (*in, opt*)
コールバック関数`_user_call_bak` の引数として呼び出される．

戻り値

読み出し開始に成功すれば TRUE を, 失敗で FALSE を返します．エラーの状態は `UTWSGetErrorMessage` 関数 (11.1 節) で調べることができます．

解説

WHS-1 のフラッシュメモリをバイナリデータのまま非同期で読み出します.

WHS-1 のメモリ読み出しが開始されると，呼び出し側へすぐに制御が返ります．読み出し開始に失敗すると FALSE を返し，このときイベント_user_event は非シグナル状態のままで，コールバック関数\user_call_back も呼び出されません．読み出しが完了したら，イベント_user_event がシグナル状態になり，コールバック関数_user_call_back が呼び出されます.

読み出し完了を検出したら，UTWSWHS1ReadMemoryBinaryResult 関数 (11.31) で，読み出しが成功したかどうか調べてください．読み出しが成功したら読み出した WHS-1 のバイナリデータのバイト数を UTWSWHS1GetMemoryBinarySize 関数 (11.28) で調べ，これと同じサイズのメモリを確保した後，UTWSWHS1GetMemoryBinary 関数 (11.29) を使って呼び出し側のメモリにバイナリデータをコピーできます．また，UTWSWHS1SaveMemoryBinary 関数 (11.30) を使えばバイナリデータをそのままファイルへ保存できます.

バイナリデータを生体情報へ変換したり，CSV ファイルとして保存するには

```
HANDLE h = UTWSOpenDevice(UTWS_WHS_1_BINARYDATA, 0)
```

でバイナリデータを扱うためのハンドルを取得する必要があります．このハンドルを使って，バイナリデータを操作してください.

さらに詳しくは，9.1 節を参照してください.

エラー値

関数実行後 UTWSGetErrorMessage 関数で返されるエラーの状態は次の通りです.

UTWS_NO_ERR

正常終了.

UTWS_ERR_INVALID_HANDLE

第 1 引数で与えられたハンドルが，有効ではありません.

UTWS_ERR_FAILED

何らかの原因で，WHS-1 へのコマンド送信が失敗しました.

UTWS_ERR_TIMEOUT

何らかの原因で，タイムアウトが発生しました.

UTWS_ERR_NO_DATA

メモリにデータがありません.

UTWS_ERR_UNKNOWN

原因不明の例外が発生しました.

11.28 UTWSWHS1GetMemoryBinarySize

WHS-1 から読み出されたバイナリデータのバイト数を取得します.

構文

```
UTWS_API BOOL WINAPI UTWSWHS1GetMemoryBinarySize(
  HANDLE _handle,
  UINT* _size
);
```

引数

`HANDLE _handle` (in)

読み出されたバイナリデータのバイト数を取得したい WHS-1 のハンドル.

`UINT* _size` (out)

読み出されたバイナリデータのバイト数が返されます. この値が 0 のとき何もしません.

戻り値

バイト数取得に成功すれば TRUE を, 失敗で FALSE を返します. エラーの状態は `UTWSGetErrorMessage` 関数 (11.1 節) で調べることができます.

エラー値

関数実行後 `UTWSGetErrorMessage` 関数で返されるエラーの状態は次の通りです.

`UTWS_NO_ERR`

正常終了.

`UTWS_ERR_INVALID_HANDLE`

第 1 引数で与えられたハンドルが, 有効ではありません.

`UTWS_ERR_UNKNOWN`

原因不明の例外が発生しました.

11.29 UTWSWHS1GetMemoryBinary

WHS-1 のフラッシュメモリから読み出されたバイナリデータを関数呼び出し側のバッファへコピーします.

構文

```
UTWS_API BOOL WINAPI UTWSWHS1GetMemoryBinary(
  HANDLE _handle,
```

```
  BYTE* _buf,
  UINT _bufsize
);
```

引数

HANDLE _handle (*in*)

バイナリデータのコピーを得たい WHS-1 のハンドル.

BYTE* _buf (*out*)

このバッファへバイナリデータがコピーされます．バッファは UTWSWHS1GetMemoryBinarySize 関数 (11.28) で返されるバイト数以上の領域を確保してください．0 のとき関数は FALSE を返します.

UINT _bufsize (*in*)

ポインタ_buf に確保されている領域のバイト数を与えてください．読み出した WHS-1 のバイナリデータのサイズがこれよりも大きいとき，関数は FALSE を返します.

戻り値

バイナリデータのコピーに成功すれば TRUE を，失敗で FALSE を返します．エラーの状態は UTWSGetErrorMessage 関数 (11.1 節) で調べることができます.

解説

先頭の 1 バイトは WHS-1 の仮 ID です．これに続いて WHS-1 から読み出されたバイナリデータが続きます．WHS-1 から送出されるデータは 1 ブロック 256 バイトのヘッダとフッタにそれぞれ 0x01 と 0x04 が付与されていますが，これらは削除されています．またデータの終端を示す 0xFFFFFFFF も同様に削除されています.

エラー値

関数実行後 UTWSGetErrorMessage 関数で返されるエラーの状態は次の通りです.

UTWS_NO_ERR

正常終了.

UTWS_ERR_INVALID_HANDLE

第 1 引数で与えられたハンドルが，有効ではありません.

UTWS_ERR_FAILED

WHS-1 から読み出されたバイナリデータのバイト数が，引数_bufsize よりも大きい.

UTWS_ERR_INVALID_PARAMETER

バッファへのポインター_buf が 0 である.

UTWS_ERR_UNKNOWN

原因不明の例外が発生しました.

11.30 UTWSWHS1SaveMemoryBinary

WHS-1 から読み出されたバイナリデータをそのままファイルへ保存します.

構文

```
UTWS_API BOOL WINAPI UTWSWHS1SaveMemoryBinary(
  HANDLE _handle,
  LPCSTR _filename
);
```

引数

HANDLE _handle (in)

読み出したバイナリデータを保存する WHS-1 のハンドル.

LPCSTR _filename (in)

ファイル名.

戻り値

保存に成功すれば TRUE を，失敗で FALSE を返します．エラーの状態は UTWSGetErrorMessage 関数 (11.1 節) で調べることができます.

エラー値

関数実行後 UTWSGetErrorMessage 関数で返されるエラーの状態は次の通りです.

UTWS_NO_ERR

正常終了.

UTWS_ERR_INVALID_HANDLE

第 1 引数で与えられたハンドルが，有効ではありません.

UTWS_ERR_CANNOT_OPEN_FILE

ファイルを開けませんでした.

UTWS_ERR_UNKNOWN

原因不明の例外が発生しました.

11.31 UTWSWHS1ReadMemoryBinaryResult

関数 UTWSWHS1ReadMemoryBinary を使った非同期読み出しの結果を取得します.

構文

```
UTWS_API UINT WINAPI UTWSWHS1ReadMemoryBinaryResult(
  HANDLE _handle
);
```

引数

HANDLE _handle (*in*)

非同期読み出しの結果を調べたい WHS-1 のハンドル.

戻り値

戻り値が UTWS_NO_ERR のとき，バイナリデータは正常に読み出されています．UTWS_ERR_TIMEOUT のとき，WHS-1 が USB から抜かれるなどして途中で読み出しが途絶えています．UTWS_ERR_FAILED のとき，その他の問題が生じています.

エラー値

関数実行後 UTWSGetErrorMessage 関数で返されるエラーの状態は次の通りです.

UTWS_NO_ERR

正常終了.

UTWS_ERR_INVALID_HANDLE

第 1 引数で与えられたハンドルが，有効ではありません.

11.32 UTWSWHS1ReadSize

関数 UTWSWHS1ReadMemoryBinary を使った非同期読み出しの中の進捗を調べることができます.

構文

```
UTWS_API BOOL WINAPI UTWSWHS1ReadSize(
  HANDLE _handle,
  UINT* _size_read,
  UINT* _size_for_read
);
```

引数

HANDLE _handle (*in*)

非同期呼び出しの進捗を調べたい WHS-1 のハンドル.

UINT* _size_read (*out*)

現在までに読み出し完了したブロック数を格納される 0 以外のポインタ.

`UINT* _size_for_read` (*out*)
読み出し予定の全ブロック数が格納される 0 以外のポインタ.

戻り値

値の取得に成功すると TRUE を，失敗で FALSE を返します．エラーの状態は `UTWSGetErrorMessage` 関数 (11.1 節) で調べることができます.

解説

WHS-1 の 1 ブロックは 256 バイトです.

エラー値

関数実行後 `UTWSGetErrorMessage` 関数で返されるエラーの状態は次の通りです.

`UTWS_NO_ERR`
正常終了.

`UTWS_ERR_INVALID_HANDLE`
第 1 引数で与えられたハンドルが，有効ではありません.

`UTWS_ERR_INVALID_PARAMETER`
引数のポインター値が 0 です.

`UTWS_ERR_UNKNOWN`
原因不明の例外が発生しました.

11.33 UTWSWHS1SetCallBackDisconnect

WHS-1 が USB から抜き取られたことを検出したときに呼び出されるコールバック関数を登録します.

構文

```
UTWS_API BOOL WINAPI UTWSWHS1SetCallBackDisconnect(
  HANDLE _handle,
  void (*_callBack)(void*),
  void* _callBackArg
);
```

引数

`HANDLE _handle` (*in*)
コールバック関数を登録したい WHS-1 のハンドル.

`void (*_callBack)(void*)` (in)
コールバック関数へのポインタ．コールバック関数の登録を解除したいとき 0 とします．

`void* _callBackArg` (in)
コールバック関数の引数．

戻り値

コールバック関数の登録に成功すると TRUE を，失敗で FALSE を返します．エラーの状態は `UTWSGetErrorMessage` 関数 (11.1 節) で調べることができます．

解説

WHS-1 が USB から抜き取られると，USB シリアル通信はクローズしますが，WHS-1 のために確保したメモリは解放されません．呼び出されたコールバック関数において，`UTWSCloseDevice`(11.3 節) を使ってハンドルをクローズしてください．

エラー値

関数実行後 `UTWSGetErrorMessage` 関数で返されるエラーの状態は次の通りです．

`UTWS_NO_ERR`
正常終了．

`UTWS_ERR_INVALID_HANDLE`
第 1 引数で与えられたハンドルが，有効ではありません．

`UTWS_ERR_UNKNOWN`
原因不明の例外が発生しました．

11.34 UTWSWHS1BinaryToData

WHS-1 バイナリデータから生体情報を復元します．

構文

```
UTWS_API UINT WINAPI UTWSWHS1BinaryToData(
  HANDLE _handle,
  BYTE* _binary,
  UINT  _size,
  BOOL _time_adjust,
  BOOL _dtr_rri
);
```

引数

`HANDLE _handle` (in)

WHS-1 バイナリデータオブジェクトへのハンドル.

`BYTE* _binary` (*in*)
WHS-1 バイナリデータへのポインタ.

`UINT  _size` (*in*)
WHS-1 バイナリデータのバイト数.

`BOOL _time_adjust` (*in*)
心拍情報，加速度，温度の組に付与される時刻情報の調整方法を与えます．詳しくは 8.3 節を参照してください.

`BOOL _dtr_rri` (*in*)
常時 FALSE としてください.

戻り値
成功すると生成された生体情報のレコード数が返ります．失敗すると-1 が返ります.

解説
入力された WHS-1 バイナリデータに基づいて，UTWS.dll で生体情報へ変換されます．この生体情報を取り出すには `UTWSWHS1GetDataFromBinary` 関数 (11.36) を使います.

エラー値
関数実行後 `UTWSGetErrorMessage` 関数で返されるエラーの状態は次の通りです.

`UTWS_NO_ERR`
正常終了.

`UTWS_ERR_INVALID_HANDLE`
第 1 引数で与えられたハンドルが，有効ではありません.

`UTWS_ERR_UNKNOWN`
原因不明の例外が発生しました.

11.35 UTWSWHS1BinaryFileToData

保存されている WHS-1 バイナリデータから生体情報を復元します.

構文
```
UTWS_API UINT WINAPI UTWSWHS1BinaryFileToData(
  HANDLE _handle,
  LPCTSTR _binaryfile_name,
```

```
  BOOL _time_adjust,
  BOOL _dtr_rri
);
```

引数

`HANDLE _handle` (in)

WHS-1 バイナリデータオブジェクトへのハンドル.

`LPCTSTR _binaryfile_name` (in)

WHS-1 バイナリデータが保存されたファイルのファイル名.

`BOOL _time_adjust` (in)

心拍情報，加速度，温度の組に付与される時刻情報の調整方法を与えます．詳しくは 8.3 節を参照してください.

`BOOL _dtr_rri` (in)

常時 FALSE としてください.

戻り値

成功すると生成された生体情報のレコード数が返ります．失敗すると-1 が返ります.

解説

入力された WHS-1 バイナリデータに基づいて，UTWS.dll で生体情報へ変換されます．この生体情報を取り出すには `UTWSWHS1GetDataFromBinary` 関数 (11.36) を使います.

エラー値

関数実行後 `UTWSGetErrorMessage` 関数で返されるエラーの状態は次の通りです.

`UTWS_NO_ERR`

正常終了.

`UTWS_ERR_INVALID_HANDLE`

第 1 引数で与えられたハンドルが，有効ではありません.

`UTWS_ERR_CANNOT_OPEN_FILE`

ファイルを開けませんでした.

`UTWS_ERR_EMPTY_DATA`

読み込むバイナリの形式が間違っているなどして，変換できませんでした.

`UTWS_ERR_UNKNOWN`

原因不明の例外が発生しました.

11.36 UTWSWHS1GetDataFromBinary

関数 UTWSWHS1BinaryToData または UTWSWHS1BinaryFileToData によって WHS-1 バイナリデータから生体情報に復元されたデータを呼び出し側にコピーします.

構文

```
UTWS_API BOOL WINAPI UTWSWHS1GetDataFromBinary(
  HANDLE _handle,
  WHS1MemDataHeader* _header,
  WHS1EcgDataEx* _data,
  UINT _size
);
```

引数

HANDLE _handle (in)

生体情報が復元されている WHS-1 バイナリデータオブジェクトへのハンドル.

WHS1MemDataHeader* _header (out)

データヘッダ情報を格納する構造体 WHS1MemDataHeader へのポインタ．構造体の詳細は 12.7 節を参照してください.

WHS1EcgDataEx* _data (out)

1 組の生体情報を表す構造体 WHS1EcgDataEx へのポインタ．あらかじめ_size だけのメモリが確保してください.

UINT _size (in)

呼び出し側でバッファとして用意し WHSEcgDataEx の数．WHS-1 バイナリオブジェクト内で復元されている生体情報の数よりも，_size が小さいと関数は失敗します.

戻り値

成功のとき TURE，失敗のとき FALSE を返します.

解説

この関数を実行する前に，UTWSWHS1BinaryToData 関数または UTWSWHS1BinaryFileToData 関数を実行し，生成された生体情報のレコード数を得てください．生体情報のレコード数だけ構造体 WHS1EcgDataEx のメモリを確保してください.

エラー値

関数実行後 UTWSGetErrorMessage 関数で返されるエラーの状態は次の通りです.

UTWS_NO_ERR
正常終了.

UTWS_ERR_INVALID_HANDLE
第 1 引数で与えられたハンドルが，有効ではありません.

UTWS_ERR_INVALID_PARAMETER
第 2 引数または第 3 引数が 0，または生成済みの生体情報のレコード数が第 4 引数よりも小さい.

UTWS_ERR_UNKNOWN
原因不明の例外が発生しました.

11.37 UTWSWHS1SaveDataFromBinary

関数 UTWSWHS1BinaryToData または UTWSWHS1BinaryFileToData によって WHS-1 バイナリデータから生体情報に復元されたデータを CSV ファイルに保存します.

構文

```
UTWS_API BOOL WINAPI UTWSWHS1SaveDataFromBinary(
  HANDLE _handle,
  LPCSTR _file_name
);
```

引数

HANDLE _handle (in)
生体情報が復元されている WHS-1 バイナリデータオブジェクトへのハンドル.

LPCSTR _file_name (in)
保存先ファイル名.

戻り値

成功のとき TRUE，失敗のとき FALSE が返ります.

エラー値

関数実行後 UTWSGetErrorMessage 関数で返されるエラーの状態は次の通りです.

UTWS_NO_ERR
正常終了.

UTWS_ERR_INVALID_HANDLE
第 1 引数で与えられたハンドルが，有効ではありません.

UTWS_ERR_CANNOT_OPEN_FILE
ファイルを開けませんでした.

UTWS_ERR_UNKNOWN
原因不明の例外が発生しました.

12 構造体リファレンス

12.1 struct RRD1EcgData

構造体 RRD1Data で使われ，心拍信号，加速度および温度データの組を表す構造体.

構文

```
typedef struct RRD1EcgData_
{
  WORD ecg;
  double temp;
  double acc_x, acc_y, acc_z;
} RRD1EcgData;
```

メンバ

WORD ecg
心拍波形モードのとき，心拍信号の 10bit サンプリング値.
心拍間隔モードのとき心拍間隔で単位は ms.
心拍数モードのとき平均心拍間隔で単位は ms．この値の逆数に 1000ms/s×60s/min をかけ算すれば心拍数 bpm が得られます.

double temp
温度．単位は摂氏です.

double acc_x, acc_y, acc_z
それぞれ，x，y，z 軸の加速度．単位は重力加速度 g です．最大で $\pm 4g$ です.

12.2 struct RRD1Data

RRD-1 の 1 つの受信データを表す構造体.

構文

```
typedef struct RRD1Data_
{
  WORD year, month, day, hour, min, sec;
```

```
  BYTE mode;
  BYTE tempID;
  BYTE sendedID;

  BYTE ecg_mode;
  BYTE acc_mode;
  BYTE lowbattery;
  BYTE sampling_freq;

  BYTE data_count;
  RRD1EcgData data[10];
} RRD1Data;
```

メンバ

`WORD year, month, day, hour, min, sec`
受信した年，月，日，時，分，秒.

`BYTE mode`
未使用.

`BYTE tempID;`
WHS-1 の仮 ID．RRD-1 は複数台の WHS-1 から受信できるが，それらを区別するために仮 ID を使用します.

`BYTE sendedID`
WHS-1 がデータ送信のたびにインクリメントする値．0xFF を超えると 0x00 に戻ります.

`BYTE ecg_mode`
心拍モード.

モード	値
心拍波形モード	0
心拍間隔モード	1
心拍数モード	2

`BYTE acc_mode`
加速度センサのモード．心拍波形モードのときは無意味な値です.

モード	値
移動平均モード	0
ピークホールドモード	1

`BYTE lowbattery`

WHS-1 がローバッテリーのとき 1，それ以外は 0.

BYTE sampling_freq

心拍間隔モードのとき有効です．モードに関する詳細は 5.1 節を参照してして下さい．

モード	値
心拍間隔 2 モード	0
心拍間隔 1 モード	1

BYTE data_count

以下の data メンバ変数の有効なデータ数．たとえば data_count が 3 のとき，data[0]，data[1] と data[2] が有効です．

RRD1EcgData data[10]

心拍信号，加速度および温度データの組の配列．詳しくは 12.1 節を参照して下さい．

心拍間隔 2 モード (5.1.3 節参照) かつ加速度ピークホールドの時，有効なデータ数は 2 ですが，このとき下記の data[0] と data[1] の心拍間隔と温度は等しい値です．data[0] の加速度の値は，絶対値をとったとき最大値となる加速度値で，data[1] の加速度は，data[0] の加速度値との差の絶対値が最大である加速度値です．

12.3 struct RRD1DataEx

RRD-1 の 1 つの受信データを表す構造体．ミリ秒単位の受信時刻をサポートします．

構文

```
typedef struct RRD1DataEx_
{
  WORD year, month, day, hour, min, sec, msec;
  BYTE mode;
  BYTE tempID;
  BYTE sendedID;

  BYTE ecg_mode;
  BYTE acc_mode;
  BYTE lowbattery;
  BYTE sampling_freq;

  BYTE data_count;
  RRD1EcgData data[10];
  BYTE saturation;
  BYTE reserve[9];
} RRD1DataEx;
```

メンバ

`WORD year, month, day, hour, min, sec, msec`
受信した年，月，日，時，分，秒，ミリ秒.

`BYTE mode`
未使用.

`BYTE tempID;`
WHS-1 の仮 ID．RRD-1 は複数台の WHS-1 から受信できるが，それらを区別するために仮 ID を使用します.

`BYTE sendedID`
WHS-1 がデータ送信のたびにインクリメントする値．0xFF を超えると 0x00 に戻る.

`BYTE ecg_mode`
心拍モード.

モード	値
心拍波形モード	0
心拍間隔モード	1
心拍数モード	2

`BYTE acc_mode`
加速度センサのモード．心拍波形モードのときは無意味な値です.

モード	値
移動平均モード	0
ピークホールドモード	1

`BYTE lowbattery`
WHS-1 がローバッテリーのとき 1，それ以外は 0.

`BYTE sampling_freq`
心拍間隔モードのとき有効です．モードに関する詳細は 5.1 節を参照してして下さい.

モード	値
心拍間隔 2 モード	0
心拍間隔 1 モード	1

`BYTE data_count`
以下の `data` メンバ変数の有効なデータ数．たとえば `data_count` が 3 のとき，`data[0]`，`data[1]` と `data[2]` が有効です.

`RRD1EcgData data[10]`
心拍信号，加速度および温度データの組の配列．詳しくは 12.1 節を参照して下さい.

心拍間隔 2 モード (5.1.3 節参照) かつ加速度ピークホールドの時，有効なデータ数は 2 ですが，このとき下記の data[0] と data[1] の心拍間隔と温度は等しい値です．data[0] の加速度の値は，絶対値をとったとき最大値となる加速度値で，data[1] の加速度は，data[0] の加速度値との差の絶対値が最大である加速度値です．

`BYTE saturation`

WSH-1 の入力信号が飽和しているとき 1，それ以外のときは 0．WHS-1ver500 以降のとき有効です．

`BYTE reserve[9]`

未使用．

12.4 struct WHS1Config

WHS-1 の設定情報をまとめた構造体．

構文

```
typedef struct WHS1Config_
{
  BYTE ecg_mode;
  BYTE mode;
  UINT flush_write_count;
  BYTE acc_mode;
  CHAR tmp_offset;
  UINT cpu_id;
  UINT set_serial_id;
  BYTE temp_id;
  BYTE mem_mode;
  BYTE monitor_mode;
  BYTE reserve;
} WHS1Config;
```

メンバ

`BYTE ecg_mode`(設定可能)

心拍信号のモードを表す．設定可能です．各モードの無線，メモリモードおよび WHS-1，WHS-1ver500 以降で設定可能なモードには○を付します．

モード	値	WHS-1 無線	WHS-1 メモリ	WHS-1ver500 以降 無線	WHS-1ver500 以降 メモリ
心拍波形モード	1	○	×	○	×
心拍間隔 1 モード	2	○	○	○	○
心拍間隔 2 モード	3	○	○	○	○
心拍数モード	4	○	○	×	×

BYTE mode(設定可能)

無線モードかメモリモードを表す．設定可能．心拍波形モードのときメモリモードに設定しないで下さい．また，下記以外の値を与えないで下さい．

モード	値
無線モード	0
メモリモード	1

UINT flush_write_count

内蔵フラッシュメモリの書き込み回数です．読み取りのみ．

BYTE acc_mode(設定可能)

加速度センサのモードを表す．設定可能．心拍波形モードのときこの値は無視されますが，0 または 1 のいずれかに設定して下さい．WHS-1ver500 以降では，移動平均モードを使用できません．

モード	値
移動平均モード	0
ピークホールドモード	1

CHAR tmp_offset

温度センサの校正値を表す．読み取りのみ．WHS-1ver500 以降では常に 0 です．

UINT cpu_id

マイコンの CPU ID を表す．読み取りのみ．

UINT set_serial_id

デバイスのシリアル番号を表す．読み取りのみ．

BYTE temp_id(設定可能)

WHS-1 の仮 ID を表す．0x00 から 0xFF の値を設定可能です．RRD-1 は同時に複数台の WHS-1 から受信できますが，仮 ID によってどの WHS-1 からの送信か区別できます．

BYTE mem_mode

メモリがいっぱいになったときの書き込み動作を表す．読み取りのみ．WHS-1ver500 以降では，何れに設定しても記録停止で動作します．

モード	値
最過去データを上書き	0
記録停止	1

BYTE monitor_mode

モニターモードを ON にすると，メモリモードかつ，心拍間隔 1 または心拍間隔 2 モードに於いて，WHS-1 の電源を ON にすると，60 秒間，無線 +PQRST モードで動作します．WHS-1 が正しく装着できているか調べるときに便利な機能です．WHS-1ver500 以降で動作します．

モニターモード	値
有効	1
無効	0

`BYTE reserve`
将来の拡張のためのリザーブ.

12.5 struct WHS1EcgData

WHS-1 のフラッシュメモリデータ要素を表す構造体.

構文

```
typedef struct WHS1EcgData_
{
  WORD year, month, day, hour, mi, sec, msec;
  WORD ecg;
  double temp;
  double acc_x, acc_y, acc_z;
  BYTE additional;
} WHS1EcgData;
```

メンバ

`WORD year, month, day, hour, mi, sec, msec`
データの時刻情報．節 8.3 で説明する時刻調整方法によって，値が異なることに注意してください.

`WORD ecg`
心拍波形モードのとき，心拍信号の 10bit サンプリング値.
心拍間隔モードのとき心拍間隔で単位は ms.
心拍数モードのとき平均心拍間隔で単位は ms．この値の逆数に 1000ms/s×60s/min をかけ算すれば心拍数 bpm が得られます.

`double temp`
温度．単位は摂氏です．この値はフラッシュメモリ 256 バイト単位に付されるので，同じ値が連続します.

`double acc_x, acc_y, acc_z`
それぞれ，x，y，z 軸の加速度．単位は重力加速度 g です．最大で $\pm 4g$ です.

`BYTE additional`
当該データに対する付加情報です．将来の拡張のため.

12.6 struct WHS1EcgDataEx

WHS-1 のフラッシュメモリデータ要素を表す構造体.

構文

```
typedef struct WHS1EcgDataEx_
{
  WORD year, month, day, hour, mi, sec, msec;
  WORD ecg;
  double temp;
  double acc_px, acc_py, acc_pz;
  double acc_nx, acc_ny, acc_nz;
  BYTE additional;
} WHS1EcgDataEx;
```

メンバ

`WORD year, month, day, hour, mi, sec, msec`
データの時刻情報．節 8.3 で説明する時刻調整方法によって，値が異なることに注意してください．

`WORD ecg`
心拍波形モードのとき，心拍信号の 10bit サンプリング値．
心拍間隔モードのとき心拍間隔で単位は ms．
心拍数モードのとき平均心拍間隔で単位は ms．この値の逆数に 1000ms/s×60s/min をかけ算すれば心拍数 bpm が得られます．

`double temp`
温度．単位は摂氏です．この値はフラッシュメモリ 256 バイト単位に付されるので，同じ値が連続します．

`double acc_px, acc_py, acc_pz`
それぞれ，x，y，z 軸の最大加速度．単位は重力加速度 g です．最大で $\pm 4g$ です．

`double acc_nx, acc_ny, acc_nz`
それぞれ，x，y，z 軸の最小加速度．単位は重力加速度 g です．最大で $\pm 4g$ です．

`BYTE additional`
当該データに対する付加情報です．将来の拡張のため．

12.7 struct WHS1MemDataHeader

WHS-1 のフラッシュメモリに記録されたデータのヘッダ情報です．

構文

`//WHS-1` **のメモリデータ**

```
typedef struct WHS1MemDataHeader_
{
  BYTE ecg_mode;
  BYTE acc_mode;
  BYTE temp_id;
  UINT data_count;
  WORD mode_flag;
} WHS1MemDataHeader;
```

メンバ

`ecg_mode`
心拍信号のモードを表す.

モード	値
心拍間隔 1 モード	2
心拍間隔 2 モード	3
心拍数モード	4

`acc_mode`
加速度センサのモードを表す.

モード	値
ピークホールドモード	0
移動平均モード	1

`BYTE temp_id`
WHS-1 の仮 ID を表す.

`UINT data_count`
WHS-1 のフラッシュメモリからダウンロードし，UTWS.dll にストアされたデータ数．つまり，構造体 `WHS1EcgData`(12.5 節) の数を表します.

`BYTE mode_flag`
付加情報を表します.

値	情報
0x01	イベント記録あり

13 エラー一覧

//成功（エラーなし）

```
#define UTWS_NO_ERR                    0x00000000
```

//原因不明のエラー

```
#define UTWS_ERR_UNKNOWN            0xffffffff

//指定されたデバイス識別子は存在しない.
#define UTWS_ERR_UNKNOWN_DEVICE_ID  0x00000001

//指定されたデバイスはすでにオープンされています.
#define UTWS_ERR_ALREADY_OPEN       0x00000002

//デバイスを開くことができなかった.
#define UTWS_ERR_CANNOT_OPEN        0x00000003

//デバイスが閉じられています.
#define UTWS_ERR_CLOSED             0x00000004

//デバイスのモードを変更できなかった
#define UTWS_ERR_CANNOT_CHANGE_MODE 0x00000005

//無効なハンドルが指定された
#define UTWS_ERR_INVALID_HANDLE     0x00000006

//取り出し可能なデータがありません
#define UTWS_ERR_EMPTY_DATA         0x00000007

//失敗しました
#define UTWS_ERR_FAILED             0x00000008

//タイムアウト
#define UTWS_ERR_TIMEOUT            0x00000009

//デバイスのシリアル番号を取得できませんでした
#define UTWS_ERR_CANNOT_GET_SERIAL  0x0000000A

//ファイルを開けませんでした
#define UTWS_ERR_CANNOT_OPEN_FILE   0x0000000B

//宛先アドレスが間違っています
#define UTWS_ERR_BAD_DESTINATION_ADDRESS 0x0000000C

//FTD2XX.DLL が存在しません
#define UTWS_ERR_CANNOT_LOAD_FTD2XX 0x0000000D
```

```
//取得あるいは保存可能なデータが存在しません
#define UTWS_ERR_NO_DATA            0x0000000E

//無効なパラメータ
#define UTWS_ERR_INVALID_PARAMETER  0x0000000F

//不正な個別識別 ID です
#define UTWS_ERR_BAD_RFID           0x00000010
```