WB－1J

アプリケーションインターフェース
マニュアル

(Version 1.01)

改訂履歴

版	日付	変更内容	担当
1.00	2010/12/20	初版作成	
1.01	2011/4/22	2.2.1、2.2.2、2.2.6 の ReadFile、WriteFile の nNumberOfBytesTORead 及び nNumberOfBytesTOWrite には、固定値であることを追記 2.2.2　LED 消灯時の設定を追記 2.2.3　輝度、コントラストのレジストリ値の意味を追記 2.2.3　輝度、コントラストのリアルタイムでの変更には、画面を再描画させるためのイベントも必要であることを追加 2.2.5　のプログラミング例で、BatteryDrvrGetLevels()は不要のため削除 2.2.7　取得したバージョン文字列のフォーマットについて追記 その他誤記修正	

ー　目次　ー

1　はじめに 1

1.1　目的 1

2　アプリケーションインターフェース仕様 2

2.1　アプリケーションインターフェースの機能一覧 2

2.2　アプリケーションインターフェースの機能 3

2.2.1　デジタル入出力ポート制御 *3*

2.2.2　LED 制御 *6*

2.2.3　バックライト制御 *10*

2.2.4　スリープ移行 *11*

2.2.5　バッテリー状況取得 *12*

2.2.6　人感センサー設定 *13*

2.2.7　バージョン取得 *17*

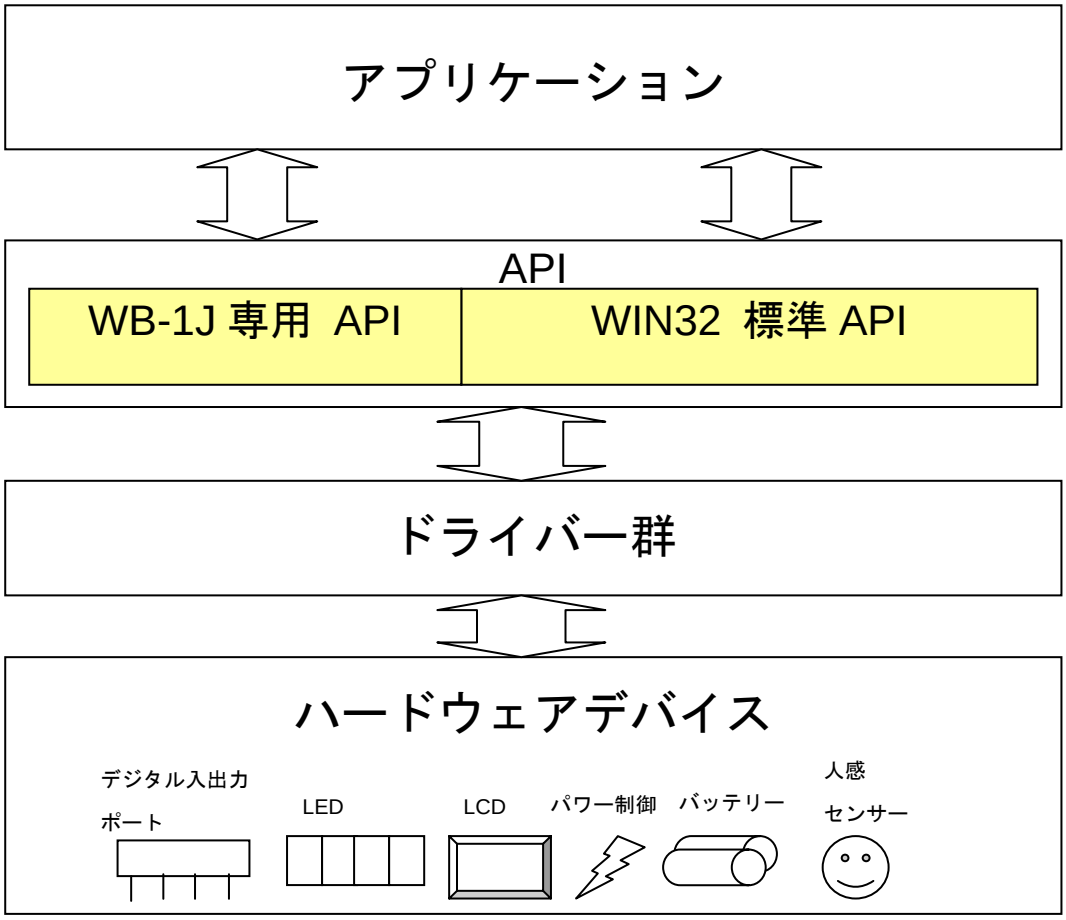
1　はじめに

1.1　目的

本書は、*WB－IJ*に搭載されるアプリケーションインターフェースの機能、及び、利用方法の解説を目的としたものである。

2　アプリケーションインターフェース仕様

WB-1J に内蔵された固有のハードウェアデバイスをアプリケーションから制御する場合、アプリケーションはアプリケーションインターフェースを介して、各ハードウェアデバイスを制御する。
アプリケーションインターフェースには WB-1J 専用 API と WIN32 標準 API（ハードウェアの違いをドライバーレベルで吸収しているもの）の２種類がある。



2.1　アプリケーションインターフェースの機能一覧

No.	機能	インターフェース	種別
1	デジタル入出力ポート制御	標準ストリーム入出力	WB-1J 専用 API
2	ＬＥＤ制御	標準ストリーム入出力	WB-1J 専用 API
3	バックライト制御	レジストリ＋イベント	WIN32 標準 API
4	スリープ移行	関数呼び出し	WIN32 標準 API
5	バッテリー状況取得	関数呼び出し	WIN32 標準 API
6	人感センサー設定	関数呼び出し	WB-1J 専用 API

2.2　アプリケーションインターフェースの機能

2.2.1　デジタル入出力ポート制御

目的：

外部機器などを制御する為にデジタル入出力ポートの制御を行う。
標準ストリーム入出力インターフェースとして API を提供する。
入力用 API、出力用 API を提供する。
入力と出力用共に 5 ポートの制御が可能で、スリープ状態においても出力状態は保持される。

機能詳細：

デジタル入出力ポートを標準ストリーム入出力インターフェースとして使用し、
リード（入力）、ライト（出力）を行う。

データ形式：

	BIT7	BIT6	BIT5	BIT4	BIT3	BIT2	BITI	BIT0
ＤＩ	X	X	X	PIC_GPI5	PIC_GPI4	PIC_GPI3	PIC_GPI2	PIC_GPI1
ＤＯ	X	X	X	PIC_GPO5	PIC_GPO4	PIC_GPO3	PIC_GPO2	PIC_GPO1

PIC_GPI1～5；DI1～5 に相当。　入力データの各 bit 共　1：OFF、0：ON
PIC_GPO1～5；DO1～5 に相当。　出力データの各 bit 共　1：ON、0：OFF

インターフェース：

１）オープン
デジタル入出力ポートのデバイスをオープンする。デジタル入出力ポートのデバイスは１つしかない為（DI、DO 計 10 ポートで１つのデバイス）、同時に複数のオープンを行う事は出来ない。

呼び出し形式：

```
HANDLE CreateFile(LPCTSTR lpFileName,
                  DWORD dwDesiredAccess,
                  DWORD dwShareMode,
                  LPSECURITY_ATTRIBUTES lpSecurityAttributes,
                  DWORD dwCreationDispostion,
                  DWORD dwFlagsAndAttributes,
                  HANDLE hTemplateFile);
```

引数：

lpFileName	“DIO1:”（デジタル入出力デバイスのデバイス名：固定値）
dwDesiredAccess	GENERIC_READ | GENERIC_WRITE（固定値）
dwShareMode	0（固定値）
lpSecurityAttributes	0（固定値）

dwCreationDispostion	OPEN_EXISTING（固定値）
dwFlagsAndAttributes	0（固定値）
hTemplateFileClose	NULL（固定値）

返り値：
成功：デバイスのハンドル
失敗：INVALID_HANDLE_VALUE
拡張エラー情報を取得する場合は GetLastError 関数を呼び出す。

２）クローズ
デバイスをクローズする。

呼び出し形式：
```
BOOL CloseHanlde( HANDLE hObject );
```

引数：
hObject　　オープン時に取得したハンドル
※クローズを行った後もデジタル入出力ポートの状態は保持したままとなる。

返り値：
成功：True
失敗：False
拡張エラー情報を取得する場合は GetLastError 関数を呼び出す。

３）デジタル入力
デジタル入力のデータを取得する。

呼び出し形式：
```
BOOL ReadFile (HANDLE hFile,
               LPVOID lpBuffer,
               DWORD nNumberOfBytesTORead,
               LPDWORD lpNumberOfBytesRead,
               LPOVERLAPPED lpOverlapped);
```

引数：

hFile	オープン時に取得したハンドル
lpBuffer	取得したデジタル入力のデータを格納するバッファへポインタ
nNumberOfBytesTORead	1（取得するデータのバイト数：固定値）
lpNumberOfBytesRead	実際に取得した値のバイト数
lpOverlapped	0（固定値）

※nNumberOfBytesTORead に２以上を指定しても、１バイトのみリードする。

返り値：
成功：True
失敗：False
拡張エラー情報を取得する場合は GetLastError 関数を呼び出す。

４）デジタル出力

データのデジタル出力を行う。

呼び出し形式：

```
BOOL WriteFile (HANDLE hFile,
        LPCVOID lpBuffer,
        DWORD nNumberOfBytesTOWrite,
        LPDWORD lpNumberOfBytesWritten,
        LPOVERLAPPED lpOverlapped) ;
```

引数：

hFile	オープン時に取得したハンドル
lpBuffer	出力するデータを格納するバッファへのポインタ
nNumberOfBytesTOWrite	1（出力するデータのバイト数：固定値）
lpNumberOfBytesWritten	実際に出力したデータのバイト数
lpOverlapped	0（固定値）

※nNumberOfBytesTOWrite に２以上を指定しても、１バイトのみのライトとなる。

返り値：

成功：True

失敗：False

拡張エラー情報を取得する場合は GetLastError 関数を呼び出す。

５）プログラミング例

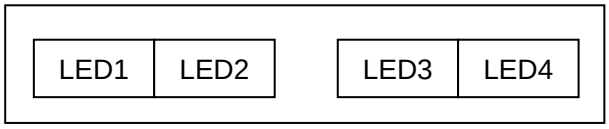
```
HANDLE hDIO;
BOOL rt;
DWORD NumberOfBytesWritten
DWORD NumberOfBytesRead;
BYTE DI=0,DO=0;

hDIO = CreateFile(L “DIO1:”,
                GENERIC_READ | GENERIC_WRITE,
                0,
                0,
                OPEN_EXISTING,
                0,
                NULL);

rt=ReadFile(hDIO,&DI,1,&NumberOfBytesRead,0);
rt=WriteFile(hDIO,&DO,1,&NumberOfBytesWritten,0);
rt=CloseHandle(hDIO);
```

2.2.2 LED 制御

目的：

各種状態表示のためにアプリケーションで LED を点灯する機能を提供する。
標準ストリーム入出力インターフェースとして API を提供する。
状態取得用 API、出力用 API を提供する。
LED は 4 個、色は RGB を組み合わせる事により、7 色の出力を可能とする。

WB-1J の LED 配置については、以下のとおり。



機能詳細：

ＬＥＤデバイスを標準ストリーム入出力インターフェースとして使用する

データの形式：

１バイト目: 赤色ＬＥＤの指定
BIT0: LED1: 1=ON 0=OFF
BIT2: LED2: 1=ON 0=OFF
BIT4: LED3: 1=ON 0=OFF
BIT6: LED4: 1=ON 0=OFF

２バイト目: 緑色ＬＥＤの指定
BIT0: LED1: 1=ON 0=OFF
BIT2: LED2: 1=ON 0=OFF
BIT4: LED3: 1=ON 0=OFF
BIT6: LED4: 1=ON 0=OFF

３バイト目:青色ＬＥＤの指定
BIT0: LED1: 1=ON 0=OFF
BIT2: LED2: 1=ON 0=OFF
BIT4: LED3: 1=ON 0=OFF
BIT6: LED4: 1=ON 0=OFF

RGB の組み合わせにより、以下の色を出力出来る。

No.	Color	R	G	B
1	Red	ON	OFF	OFF
2	Green	OFF	ON	OFF
3	Blue	OFF	OFF	ON
4	Yellow	ON	ON	OFF
5	Magenta	ON	OFF	ON
6	Cyan	OFF	ON	ON
7	White	ON	ON	ON

消灯する際は、全ての BIT を OFF に設定する。

インターフェース：

１）オープン
ＬＥＤデバイスをオープンする。ＬＥＤデバイスは１つしかない為（ＬＥＤ１～４で１つのデバイス）、同時に複数のオープンを行う事は出来ない。

呼び出し形式：

```
HANDLE CreateFile(LPCTSTR lpFileName,
                  DWORD dwDesiredAccess,
                  DWORD dwShareMode,
                  LPSECURITY_ATTRIBUTES lpSecurityAttributes,
                  DWORD dwCreationDispostion,
                  DWORD dwFlagsAndAttributes,
                  HANDLE hTemplateFile)
```

引数：

lpFileName	“LED1:”（ＬＥＤデバイスのデバイス名：固定値）
dwDesiredAccess	GENERIC_READ | GENERIC_WRITE（固定値）
dwShareMode	0（固定値）
lpSecurityAttributes	0（固定値）
dwCreationDispostion	OPEN_EXISTING（固定値）
dwFlagsAndAttributes	0（固定値）
hTemplateFileClose	NULL（固定値）

返り値：
成功：ＬＥＤデバイスのハンドル
失敗：INVALID_HANDLE_VALUE
拡張エラー情報を取得する場合は GetLastError 関数を呼び出す。

２）クローズ
デバイスをクローズする。

呼び出し形式：

```
BOOL CloseHanlde( HANDLE hObject );
```

引数：
hObject　　オープン時に取得したハンドル

※クローズを行った後も状態は保持したままとなる。

返り値：
成功：True
失敗：False
拡張エラー情報を取得する場合は GetLastError 関数を呼び出す。

３）ＬＥＤ点灯状態の取得

ＬＥＤ出力時に保持しているＬＥＤデータを取得する。

呼び出し形式：

```
BOOL ReadFile (HANDLE hFile,
          LPVOID lpBuffer,
          DWORD nNumberOfBytesTORead,
          LPDWORD lpNumberOfBytesRead,
          LPOVERLAPPED lpOverlapped);
```

引数：

hFile	オープン時に取得したハンドル
lpBuffer	取得したＬＥＤデータを格納するバッファへのポインタ
nNumberOfBytesTORead	3（取得するＬＥＤデータのバイト数：固定値）
lpNumberOfBytesRead	実際に取得したＬＥＤデータのバイト数
lpOverlapped	0（固定値）

※nNumberOfBytesTORead に４以上を指定しても、３バイトのみのリードとなる。

返り値：

成功：True

失敗：False

拡張エラー情報を取得する場合は GetLastError 関数を呼び出す。

４）ＬＥＤ出力

ＬＥＤの点灯、消灯を設定する。ＬＥＤデータはドライバー内で保持される。

呼び出し形式：

```
BOOL WriteFile (HANDLE hFile,
        LPCVOID lpBuffer,
        DWORD nNumberOfBytesTOWrite,
        LPDWORD lpNumberOfBytesWritten,
        LPOVERLAPPED lpOverlapped) ;
```

引数：

hFile	オープン時に取得したハンドル
lpBuffer	ＬＥＤデータを格納するバッファへのポインタ
nNumberOfBytesTOWrite	3（ＬＥＤデータのバイト数：固定値）
lpNumberOfBytesWritten	実際に出力したＬＥＤデータのバイト数
lpOverlapped	0（固定値）

※nNumberOfBytesTOWrite に４以上を指定しても、３バイトのみのライトとなる。

返り値：

成功：True

失敗：False

拡張エラー情報を取得する場合は GetLastError 関数を呼び出す。

５）プログラミング例

```
HANDLE hLED;
BOOL rt;
BYTE LEDDATA[3];
DWORD NumberOfBytesWritten

LEDDATA[0]＝RData;
LEDDATA[1]＝GData;
LEDDATA[2]＝BData;

hLED＝CreateFile(L”LED1:”,GENERIC_READ | GENERIC_WRITE,0,0,OPEN_EXISTING,0,NULL);
rt=WriteFile(hLED,&LEDDATA,3,&NumberOfBytesWritten,0);
rt=CloseHandle(hLED);
```

2.2.3　バックライト制御

目的：

LCD の輝度とコントラストのデフォルト設定を変更したいユーザに対して、制御機能を提供する。
設定はレジストリの値を変更する事で制御可能である。
輝度 0～31 まで、コントラストは 0～31 の範囲で指定出来る。
また、輝度とコントラストの設定を変更した場合は「BackLightLevelChangeEvent」というイベントを発生させる事により、設定した値で画面が表示される。

機能詳細：

以下のレジストリを設定することにより、輝度、コントラストの指定が出来る。

１）レジストリ

輝度の設定

[HKEY_CURRENT_USER¥ControlPanel¥Backlight]

バッテリー駆動時の輝度設定　　“BattBacklightLevel”DWORD、設定値：0x00～0x1F

AC アダプタ、PoE 駆動時の輝度設定　　“ACBacklightLevel”　DWORD、設定値：0x00～0x1F

コントラストの設定

[HKEY_CURRENT_USER¥ControlPanel¥Contrast]

バッテリー駆動時のコントラスト設定　　“BattContrastLevel”　DWORD、設定値：0x00～0x1F

AC アダプタ、PoE 駆動時のコントラスト設定　“ACContrastLevel”　　DWORD、設定値：0x00～0x1F

※設定値を直ちに反映させる場合は、アプリケーションは BackLightLevelChangeEvent を発生させた上で、画面の再描画が起こるイベントを発生させる必要がある。

※31(0x1F)を超える値を設定した場合、31 を設定した時と同じ設定で動作する。

２）ＡＰＩコード

```
HANDLE hEvent＝CreateEvent(NULL, FALSE, FALSE,L"BackLightLevelChangeEvent")
SetEvent(hEvent); //Inform backlight driver to change the backlight level
```

３）プログラミング例

```
ULONG dwExtraInfo = 0;
HANDLE hEvent＝CreateEvent(NULL, FALSE, FALSE,L"BackLightLevelChangeEvent");
If(hEvent != INVALID_HANDLE_VALUE)
{
    SetEvent(hEvent);
    CloseHanle(hEvent);
    keybd_event(VK_0, 0, KEYEVENTF_KEYDOWN | KEYEVENTF_SILENT, & dwExtraInfo );
    keybd_event(VK_0, 0, KEYEVENTF_KEYUP | KEYEVENTF_SILENT, & dwExtraInfo );
}
```

2.2.4　スリープ移行

目的：

端末を省電力モードに移行させる機能を提供する。インターフェースとしてはWindows標準API。
スリープ状態（システムアイドル状態）：実際にはバックライト消灯、LEDの消灯、CPUのクロックダウン状態となり、電源ボタン短押し、タッチパネル押下、人感センサー反応により、通常状態に戻る。

機能詳細：

１）インクルードファイル

```
mpicomm.h
Pm.h
```

２）ＡＰＩコード
以下のAPIを“POWER_STATE_IDLE”のパラメータを指定しコールする。

```
SetSystemPowerState(ptr, POWER_STATE_IDLE, 0);
```

３）プログラミング例

```
UINT result = SetSystemPowerState(NULL, POWER_STATE_IDLE, 0);
```

〈注意事項〉
自由LEDについては、スリープ移行時に、一旦消灯し、スリープからの復帰時に、スリープ前の点灯状態に戻る。
ただし、スリープ中であってもプログラムは動作可能であり、スリープ中にプログラムからLED制御APIをコールした場合は、スリープ中であっても自由LEDは点灯する。

2.2.5　バッテリー状況取得

目的：

バッテリー残量や外部電源の状態により、アプリケーションの制御を変更したいユーザに対して、バッテリーの充電状態、残量、外部電源供給有無の各状態を取得する機能を提供する。
API としては、WIN32 標準 API の「GetSystemPowerStatusEX2」を使用する。
ただし、デバイスから取得できる情報に制限がある為、バッテリーの充電状態、残量、外部電源供給有無、電池電圧、電池温度のみデータを取得可能である。

機能詳細：

１）GetSystemPowerStatusEX2 をコールすることにより、以下の電源状態を取得できる

```
DWORD GetSystemPowerStatusEX2 (
                PSYSTEM_POWER_STATUS_EX2 pSystemPowerStatusEX2,
                DWORD dwLen,
                BOOL fupdate
)

BYTE ACLineStatus：
                AC_LINE_OFFLINE          外部給電無し（AC,POE）
                AC_LINE_ONLINE           外部給電有り（AC,POE）
                AC_LINE_BACKUP_POWER     電池給電
                AC_LINE_UNKNOWN          給電状態不明
BYTE BatteryFlag：
                BATTERY_FLAG_HIGH        電池残量：高（25～100%）
                BATTERY_FLAG_LOW         電池残量：低（10～25%）
                BATTERY_FLAG_CRITICAL    充電が必要な状態（10%未満）
                BATTERY_FLAG_CHARGING    充電中
                BATTERY_FLAG_NO_BATTERY  電池未装着
BYTE BatteryLifePercent：
                電池容量（%）0～100%  255 の場合は容量取得不可の場合
DWORD BatteryVoltage：
                電池電圧（mV）0～65535mV
DWORD BatteryTemperature：
                電池温度（℃） 0.1℃単位 -3,276.8 ～ 3,276.7
```

２）プログラミング例

```
SYSTEM_POVVER_STATUS_EX2 pwr;

GetSystemPowerStatusEX2(&pwr,sizeof(pwr),TRUE);
```

2.2.6　人感センサー設定

目的：

設置場所、環境などにより人感センサーの検知レベルを変更したいユーザに対して、検知レベルを変更する API を提供する。
センサー検知レベル 4 段階（検知距離最小：1～検知距離最大：4）の設定と状態を取得する。

機能詳細：

1）レジストリ
以下のレジストリを設定することにより、人感センサーの初期値（起動時の値）の指定が出来る。
リアルタイムに変更する場合は、下記 API を使用することで変更する。ただし、API を使用した変更内容はメモリ上でのみ保存され、レジストリは更新されないため、初期値は変更されない。

```
[HKEY_LOCAL_MACHINE¥Drivers¥BuiltIn¥SENSOR]
"InitLevel"=dword:3
```

2）オープン
人感センサーデバイスをオープンする。同時に複数オープンする事は出来ない。

呼び出し形式：

```
HANDLE CreateFile(LPCTSTR lpFileName,
                  DWORD dwDesiredAccess,
                  DWORD dwShareMode,
                  LPSECURITY_ATTRIBUTES lpSecurityAttributes,
                  DWORD dwCreationDispostion,
                  DWORD dwFlagsAndAttributes,
                  HANDLE hTemplateFile)
```

引数：

lpFileName	“SNS1:”（人感センサーデバイスのデバイス名：固定値）
dwDesiredAccess	GENERIC_READ | GENERIC_WRITE（固定値）
dwShareMode	0（固定値）
lpSecurityAttributes	0（固定値）
dwCreationDispostion	OPEN_EXISTING（固定値）
dwFlagsAndAttributes	0（固定値）
hTemplateFileClose	NULL（固定値）

返り値：
成功：人感センサーデバイスのハンドル
失敗：INVALID_HANDLE_VALUE
拡張エラー情報を取得する場合は GetLastError 関数を呼び出す。

3）クローズ

デバイスをクローズする。

呼び出し形式：
BOOL CloseHanlde(HANDLE hObject);

引数：
hObject　　オープン時に取得したハンドル

※クローズを行った後も状態は保持したままとなる。

返り値：
成功：True
失敗：False
拡張エラー情報を取得する場合は GetLastError 関数を呼び出す。

４）人感センサー設定値の取得
BOOL ReadFile (HANDLE hFile,
LPVOID lpBuffer,
DWORD nNumberOfBytesTORead,
LPDWORD lpNumberOfBytesRead,
LPOVERLAPPED lpOverlapped);

引数：

hFile	オープン時に取得したハンドル
lpBuffer	取得した人感センサー設定値を格納するバッファへのポインタ
nNumberOfBytesTORead	1（取得する人感センサート設定値のバイト数：固定値）
lpNumberOfBytesRead	実際に取得したバイト数
lpOverlapped	0（固定値）

※nNumberOfBytesTORead に２以上を指定しても、１バイトのみリードする。

返り値：
成功：True
失敗：False
拡張エラー情報を取得する場合は GetLastError 関数を呼び出す。

５）人感センサー設定値の書き込み
BOOL WriteFile (HANDLE hFile,
LPCVOID lpBuffer,
DWORD nNumberOfBytesTOWrite,
LPDWORD lpNumberOfBytesWritten,
LPOVERLAPPED lpOverlapped) ;

引数：

hFile	オープン時に取得したハンドル
lpBuffer	人感センサー設定値を格納するバッファへのポインタ
nNumberOfBytesTOWrite	1（書き込む人感センサー設定値のバイト数：固定値）
lpNumberOfBytesWritten	実際に書き込んだバイト数

lpOverlapped　　0（固定値）
※nNumberOfBytesTOWrite に２以上を指定しても、１バイトのみのライトとなる。

返り値：
成功：True
失敗：False
拡張エラー情報を取得する場合は GetLastError 関数を呼び出す。

〈注意事項〉
人感センサーが反応し、省電力モードから復帰するのは人感センサーが人を最初に検出した時の１回のみである。（常にセンサーが反応している場合は省電力モード移行動作が優先される）

６）プログラミング例
＜人感センサー設定取得＞

```
DWORD lpNumberOfBytesRead;
BYTE bLevel;

// センサーレベル取得
HANDLE hSns = CreateFile(
                L"SNS1:",
                GENERIC_READ | GENERIC_WRITE,
                0,
                0,
                OPEN_EXISTING,
                FILE_ATTRIBUTE_NORMAL,
                0);
if (INVALID_HANDLE_VALUE != hSns)
{
    if (TRUE == ReadFile(hSns, &bLevel, 1, &lpNumberOfBytesRead, NULL))
    {
        *pSensorLevel = bLevel;
        bReturn = TRUE;
    }
    CloseHandle(hSns);
}
```

＜人感センサー設定書き込み＞

```
HKEY hKey;
DWORD dwType = 0;
DWORD dwSensorLevel = (DWORD)SensorLevel;
BYTE bSensorLevel = (BYTE)SensorLevel;

// センサーレベル設定
HANDLE hSns = CreateFile(
```

```
                L"SNS1:",
                GENERIC_READ | GENERIC_WRITE,
                0,
                0,
                OPEN_EXISTING,
                FILE_ATTRIBUTE_NORMAL,
                0);
if (INVALID_HANDLE_VALUE != hSns)
{
        DWORD NumberOfBytesWritten;
        if (TRUE == WriteFile(hSns, &bSensorLevel, 1, &NumberOfBytesWritten, 0))
        {
                //*********//
        }
        CloseHandle(hSns);
}
```

2.2.7　バージョン取得

目的：

稼働中の機種名及びバージョンを特定する為、バージョン取得 API を提供する。

機能詳細：

1）バージョン情報

装置情報、バージョン情報を取得する。

呼び出し形式：

```
BOOL GetWB1J_Version(LPBYTE DevType,
                     LPDWORD DevTypeSz,
                     LPBYTE VerStr,
                     LPDWORD VerStrSz)
```

引数：

DevType	装置名称を格納するバッファへのポインタを指定する。 バッファは 100Byte 以上を確保する。
DevTypeSz	DevType に格納した装置名称のバッファサイズを設定する。 DevType と同様 100 以上を指定する。
VerStr	WB-1J の装置バージョンを格納するバッファへのポインタを指定する。 バッファは 100Byte 以上を確保する。
VerStrSz	VerStrSz に格納した装置バージョンのバッファサイズを設定する。 VerStr と同様 100 以上を指定する。

返り値：

成功：True

失敗：False

拡張エラー情報を取得する場合は GetLastError 関数を呼び出す。

備考：

2011/4/22　現在

DevType には、装置名称として 5 桁の文字列“WB-1J” がセットされる。

VerStr には、装置バージョンとして 5 桁の文字列“XX.XX” がセットされる。

上記の値は、将来拡張される可能性があるため、100Byte 以上を確保すること。

2）プログラミング例

```
typedef BOOL (__stdcall * GetWB1J_Version)(     LPBYTE DevType,
                                                LPDWORD DevTypeSz,
                                                LPBYTE VerStr,
                                                LPDWORD VerStrSz);
```

```
HMODULE hDll = LoadLibrary(L"wb1jver.dll");
if (hDll != NULL) {
        GetWB1J_Version func = (GetWB1J_Version) GetProcAddress(hDll, L"GetWB1J_Version");
        if (func != NULL) {
                LPBYTE DevType = new BYTE[256];
                LPBYTE VerStr = new BYTE[256];
                DWORD DevTypeSz = 256;
                DWORD VerStrSz = 256;

                func(DevType, &DevTypeSz, VerStr, &VerStrSz);
        }
}
```