JUNGO

WinDriver

USB リファレンス

エクセルソフト株式会社

Jungo Ltd.

POB 8493 Netanya Zip - 42504 Israel

Phone (USA) 1-877-514-0537 (WorldWide) +972-9-8859365

Fax (USA) 1-877-514-0538 (WorldWide) +972-9-8859366

ご注意

- このソフトウェアの著作権はイスラエル国 Jungo Ltd. 社にあります。
- このマニュアルに記載されている事項は、予告なしに変更されることがあります。
- このソフトウェアおよびマニュアルは、本製品のソフトウェア ライセンス契約に基づき、登録者の管理下でのみ使用することができます。
- このソフトウェアの仕様は予告なしに変更することがあります。
- このマニュアルの一部または全部を、エクセルソフト株式会社の文書による承諾なく、無断で複写、複製、転載、文書化することを禁じます。

エクセルソフト株式会社

〒108-0014 東京都港区芝 5-1-9 ブゼンヤビル 4F

TEL 03-5440-7875　FAX 03-5440-7876

E-MAIL: xlsoftkk@xlsoft.com

Home Page: http://www.xlsoft.com/

Rev. 9.0 - 6/2007

目次

付録 A
WinDriver USB PC Host API のリファレンス

注意

この関数リファレンスは、C 言語指向です。WinDriver .NET、Visual Basic および Delphi API を C コードに似た形で表現することで、すべてのユーザーに対しての理解度を向上します。各言語の実装および使用例は、WinDriver .NET、VB および Delphi ソース コードを参照してください。

A.1 WinDriver USB (WDU) ライブラリの概要

このセクションでは、以下の内容を含んだ WinDriver の USB ライブラリ (WDU) について説明します。

- `WDU_xxx` APIのコール順序の概要 - セクション A.1.1。
- 以前の WinDriver USB API (バージョン 5.22 以降) で開発されたコードのアップグレード方法 (向上した `WDU_xxx` API の使用) – セクション A.1.3。
  WinDriver の以前のバージョンで開発した USB ドライバ コードをアップグレードしない場合、このセクションをスキップしてください。

WDU ライブラリのインターフェイスは、WDU API を呼ぶソースファイルが含まれた **WinDriver/include/wdu_lib.h** および **WinDriver/include/windrvr.h** ヘッダー ファイルに保存されています。(**wdu_lib.h** は、既に **windrvr.h** に含まれています。)

A.1.1 WD_DriverName()

目的

- 呼び出し元アプリケーションにより使用される WinDriver カーネル モジュールの名前を指定します。

注意:

- デフォルトのドライバ名は **`windrvr6`** です。この関数が呼び出されない場合に使用されます。
- この関数は、サンプルおよび DriverWizard で生成される WinDriver アプリケーションのように、他の WinDriver 関数 (`WD_Open()` / `WDU_Init()` を含む) を呼び出す前に、アプリケーションの始めで 1 度だけ呼び出してください。サンプルおよび DriverWizard で生成される WinDriver アプリケーションでは、デフォルトのドライバ名 (**`windrvr6`**) でこの関数を呼び出しています。
- Windows および Linux では、WinDriver ユーザーズ ガイドのセクション 15.2 で説明するように、WinDriver カーネル モジュールの名前 (**`windrvr6.sys/.o/.ko`**) を変更する場合、アプリケーションによる `WD_DriverName()` の呼び出しに新しい名前が使用されていることを確認してください。
- `WD_DriverName()` 関数を使用するためには、`WD_DRIVER_NAME_CHANGE` プリプロセッサー フラグ (例: Visual Studio および gcc の場合は `-DWD_DRIVER_NAME_CHANGE`) を使用して、ユーザーモードのドライバ プロジェクトをビルドする必要があります。サンプルおよび DriverWizard で生成される Windows および Linux の WinDriver プロジェクトまたは makefile では、既にこのプリプロセッサ フラグが設定されています。

プロトタイプ

```
const char*  DLLCALLCONV WD_DriverName(const char* sName);
```

パラメータ

名前	型	入出力
➢ sName	const char*	入力

説明

名前	説明
sName	アプリケーションにより使用される WinDriver カーネル モジュールの名前 注意: ドライバ名には、ファイル拡張子を含めないでください。たとえば、**windrvr6.sys** や **windrvr6.o** ではなく、**windrvr6** とします。

戻り値

正常終了した場合、指定したドライバ名を返します。失敗した場合 (例: 同じアプリケーションから 2 度呼び出された場合)、NULL を返します。

注釈

- WinDriver ユーザーズ ガイドのセクション 15.2 で説明するように、WinDriver カーネル モジュールの名前変更は、Windows および Linux でサポートしています。
- Windows CE では、WD_DriverName() 関数は、常にデフォルトの WinDriver カーネル モジュール名 (**windrvr6**) で呼び出す必要があります。そうでない場合、この関数を呼び出さないでください。

A.1.2 WinDriver USB のコール順序

WinDriver の WDU_xxx USB API は、ユーザー モード USB アプリケーションと USB デバイス間のイベントドリブン転送をサポートするよう設計されています。これは以前のバージョンにはありませんでした。以前のバージョンでは、USB デバイスは関数呼出しの特定の順序を使用して初期化およびコントロールされていました。

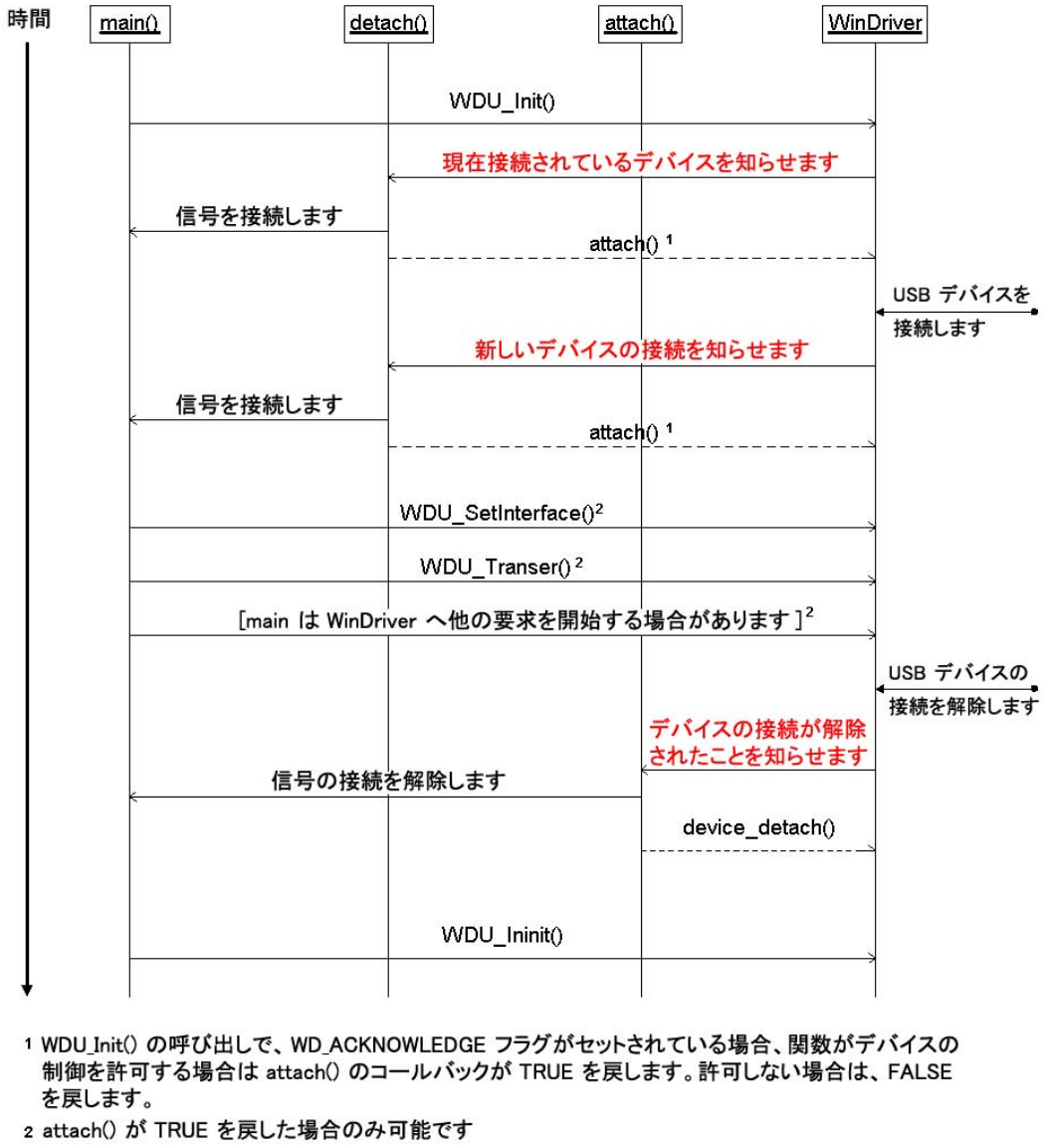
次のセクションでは、3 つのユーザー コールバック関数 (WDU_ATTACH_CALLBACK [A.2.1]、WDU_DETACH_CALLBACK [A.2.2]、および WDU_POWER_CHANGE_CALLBACK [A.2.3]) を実装できます。これらの関数は、USB デバイスの装着または検出などの、システム イベントの発生時に、アプリケーションに通知するのに使用されます。最善のパフォーマンスは、最小の実行が 3 つの関数で行われます。

アプリケーションが WDU_Init() [A.3.1] を呼び出し、デバイスが関連しているかしていないかをシステムが識別するための基準を設けます。WDU_Init()がユーザー コールバック関数へポインタを渡す必要があります。

アプリケーションはただイベントの通知を待機するだけです。通知を受信するまで、処理が続きます。アプリケーションは、下記の高水準または低水準 API で定義したどの関数でも使用します。高水準関数は低水準関数 (WinDriver カーネル モジュールとユーザー モード アプリケーション間の通信を可能にする IOCTL を使用する) を使用します。

既存の場合、アプリケーションは指定の基準に一致するデバイスへのリスンを停止し、それらのデバイスへのコールバックの通知を解除するのに `WDU_Uninit()` [A.3.6] 関数を呼び出します。

次の図は、上記で説明したコール順序を示しています。縦線が、関数またはプロセスを表しています。横線の矢印が、信号または要求を表し、開始位置から受信までを描いています。上から下が時間軸です。



1 WDU_Init() の呼び出しで、WD_ACKNOWLEDGE フラグがセットされている場合、関数がデバイスの制御を許可する場合は attach() のコールバックが TRUE を戻します。許可しない場合は、FALSE を戻します。

2 attach() が TRUE を戻した場合のみ可能です

図 A.1: WinDriver USB の呼び出し順序

次のコードをユーザー モード アプリケーションのコードのフレームワークとして使用できます。

```
attach()
{
    ...
    if this is my device
        /*
        Set the desired alternate setting ;
        Signal main() about the attachment of this device
        */

        return TRUE;
    else
        return FALSE;
}

detach()
{
    ...
    signal main() about the detachment of this device
    ...
}

main()
{
    WDU_Init(...);

    ...
    while (...)
    {
        /* wait for new devices */

        ...

        /* issue transfers */

        ...
    }
    ...
    WDU_Uninit();
}
```

A.1.3 WD_xxx USB API から WDU_xxx API へのアップグレード

バージョン 6.00 から提供されている WinDriver の WDU_xxx USB API は、ユーザー モード USB アプリケーションと USB デバイス間のイベントドリブン転送をサポートするよう設計されています。これは以前のバージョンにはありませんでした。以前のバージョンでは、USB デバイスは関数呼出しの特定の順序を使用して初期化およびコントロールされていました。

この変更の結果、Microsoft Windows だけではなく対応するすべてのプラット フォーム上で WinDriver 6.X で動作するように、WinDriver の以前のバージョンのインターフェイス用に設計された USB アプリケーションを修正する必要があります。セクション A.1.2 で説明した meta-code の一部のフレームワークと一致するようにアプリケーションのコードを作り直す必要があります。

更に、USB API を定義している関数が変更されています。次のセクションで説明しますが、新しい関数は、改良したユーザー モード USB アプリケーションと WinDriver カーネル モード間のインターフェイスを提供します。新しい関数は引数を直接、受信します。古い関数は、構造体を使用して引数を受信していました。

次の表に、左側の項目に古い関数を、右側の項目に各古い関数を置き換えた関数を表示しています。この表を使用して、新しいコードではどの新しい関数を使用するかを素早く判断します。

問題	解決方法
高水準 API	
この関数が...	次のように置き換えられました...
WD_Open() WD_Version() WD_UsbScanDevice()	WDU_Init() [A.3.1]
WD_UsbDeviceRegister()	WDU_SetInterface() [A.3.2]
WD_UsbGetConfiguration()	WDU_GetDeviceInfo() [A.3.4]
WD_UsbDeviceUnregister()	WDU_Uninit() [A.3.6]
低水準 API	
この関数が...	次のように置き換えられました...
WD_UsbTransfer()	WDU_Transfer() [A.3.7.1] WDU_TransferDefaultPipe() [A.3.7.3] WDU_TransferBulk() [A.3.7.4] WDU_TransferIsoch() [A.3.7.5] WDU_TransferInterrupt() [A.3.7.6]
USB_TRANSFER_HALT option	WDU_HaltTransfer() [A.3.7.2]
WD_UsbResetPipe()	WDU_ResetPipe() [A.3.9]
WD_UsbResetDevice() WD_UsbResetDeviceEx()	WDU_ResetDevice() [A.3.10]

A.2 USB – ユーザー コールバック関数

A.2.1 WDU_ATTACH_CALLBACK()

目的

- WinDriver は、まだ他のドライバに制御されていない、指定した基準に一致した新しいデバイスが装着されたときに、この関数を呼び出します。
  この callback を各一致するインターフェイスに対して一度呼びます。

プロトタイプ

```
typedef BOOL (DLLCALLCONV *WDU_ATTACH_CALLBACK)(
    WDU_DEVICE_HANDLE hDevice,
    WDU_DEVICE *pDeviceInfo,
    PVOID pUserData);
```

パラメータ

名前	型	入出力
➢ hDevice	WDU_DEVICE_HANDLE	入力
➢ pDeviceInfo	WDU_DEVICE*	入力
➢ pUserData	PVOID	入力

説明

名前	説明
hDevice	デバイス / インターフェイスのユニークな識別子。
pDeviceInfo	USB デバイス情報の構造体 [A.4.3] へのポインタ。関数の終了まで有効。
pUserData	コールバックするユーザーモード データ。イベント テーブル パラメータ (`pEventTable->pUserData`) で `WDU_Init()` [A.3.1] へ渡されます。

戻り値

`WDU_Init()` [A.3.1] (dwOptions 引数内で) を呼び出すように `WD_ACKNOWLEDGE` フラグを設定した場合、コールバック関数がデバイスを制御するかチェックし、制御する場合は、`TRUE` を返し、そうでない場合、`FALSE` を返します。
`WDU_Init()`への呼び出しで、`WD_ACKNOWLEDGE` フラグを設定しない場合、コールバック関数の戻り値は、意味がありません。

A.2.2 WDU_DETACH_CALLBACK()

目的

- WinDriver は、デバイスがシステムから取り外されたときに、この関数を呼び出します。

プロトタイプ

```
typedef void (DLLCALLCONV *WDU_DETACH_CALLBACK)(
    WDU_DEVICE_HANDLE hDevice,
    PVOID pUserData);
```

パラメータ

名前	型	入出力
➢ hDevice	WDU_DEVICE_HANDLE	入力
➢ pUserData	PVOID	入力

説明

名前	説明
hDevice	デバイス / インターフェイスのユニークな識別子。
pUserData	コールバックするユーザーモード データ。イベント テーブル パラメータ (`pEventTable->pUserData`) で `WDU_Init()` [A.3.1] へ渡されます。

戻り値

なし。

A.2.3 WDU_POWER_CHANGE_CALLBACK()

目的

- WinDriver は、デバイスの電源の設定を変更したときに、この関数を呼び出します。

プロトタイプ

```
typedef BOOL (DLLCALLCONV *WDU_POWER_CHANGE_CALLBACK)(
    WDU_DEVICE_HANDLE hDevice,
    DWORD dwPowerState,
    PVOID pUserData);
```

パラメータ

名前	型	入出力
➢ dwPowerState	DWORD	入力
➢ pUserData	PVOID	入力

説明

名前	説明
hDevice	デバイス / インターフェイスのユニークな識別子。
dwPowerState	選択した電源状態の番号。
pUserData	コールバックするユーザーモード データ。イベント テーブル パラメータ (`pEventTable->pUserData`) で `WDU_Init()` [A.3.1] へ渡されます。

戻り値

TRUE / FALSE。現在、戻り値に重要な意味はありません。

注釈

- Windows 2000 以降の Windows オペレーティング システムでのみ、この callback をサポートします。

A.3 USB – 関数

A.3.1 WDU_Init()

目的

- 入力基準に一致するデバイスへリスンを開始し、デバイスの通知コールバックを登録します。

プロトタイプ

```
DWORD WDU_Init(
    WDU_DRIVER_HANDLE *phDriver,
    WDU_MATCH_TABLE *pMatchTables,
    DWORD dwNumMatchTables,
    WDU_EVENT_TABLE *pEventTable,
    const char *sLicense,
    DWORD dwOptions);
```

パラメータ

名前	型	入出力
➢ phDriver	WDU_DRIVER_HANDLE *	出力
➢ pMatchTables	WDU_MATCH_TABLE*	入力
➢ dwNumMatchTables	DWORD	入力
➢ pEventTable	WDU_EVENT_TABLE*	入力
➢ sLicense	const char*	入力
➢ dwOptions	DWORD	入力

説明

名前	説明
phDriver	イベントおよび基準の登録へのハンドル。
pMatchTables	デバイスの基準を定義しているマッチ テーブル [A.4.1] の配列。
dwNumMatchTables	pMatchTables のエレメントの番号。
pEventTable	ユーザーモードのデバイス ステータス変更通知コールバック関数 [A.2] およびコールバック関数に渡されるデータのアドレスを保持するイベント テーブル構造体 [A.4.2] へのポインタ。
sLicense	WinDriver のライセンス文字列。

dwOptions	0 または : • **WD_ACKNOWLEDGE** - WDU_ATTACH_CALLBACK [A.2.1] に値が戻ってくるときに、ユーザーはデバイスを制御できます。

戻り値

正常終了した場合、WD_STATUS_SUCCESS (0) を返します。その他の場合、エラーコードを返します [A.7]。

A.3.2 WDU_SetInterface()

目的

- 指定したインターフェイスの代替の設定を設定します。

プロトタイプ

```
DWORD WDU_SetInterface(
    WDU_DEVICE_HANDLE hDevice,
    DWORD dwInterfaceNum,
    DWORD dwAlternateSetting);
```

パラメータ

名前	型	入出力
➢ hDevice	WDU_DEVICE_HANDLE	入力
➢ dwInterfaceNum	DWORD	入力
➢ dwAlternateSetting	DWORD	入力

説明

名前	説明
hDevice	デバイス / インターフェイスのユニークな識別子。
dwInterfaceNum	インターフェイスの番号。
dwAlternateSetting	要求した代替の設定値。

戻り値

正常終了した場合、WD_STATUS_SUCCESS (0) を返します。その他の場合、エラーコードを返します [A.7]。

A.3.3 WDU_GetDeviceAddr()

目的

- 指定されたデバイスの USB アドレスを取得します。

プロトタイプ

```
DWORD WDU_GetDeviceAddr(
    WDU_DEVICE_HANDLE hDevice,
    ULONG *pAddress);
```

パラメータ

名前	型	入出力
➢ hDevice	WDU_DEVICE_HANDLE	入力
➢ pAddress	ULONG	出力

説明

名前	説明
hDevice	デバイス / インターフェイスのユニークな識別子。
pAddress	関数により返されるアドレスへのポインタ

戻り値

正常終了した場合、WD_STATUS_SUCCESS (0) を返します。その他の場合、エラーコードを返します [A.7]。

注釈

- Windows 2000 およびそれ以降でのみ、この関数をサポートします。

A.3.4 WDU_GetDeviceInfo()

目的

- すべてのデバイス記述子を含む、デバイスからの設定情報を取得します。

注意: 呼び出し元は、関数によって返される`*ppDeviceInfo` ポインタを解放するために、`WDU_PutDeviceInfo()` [A.3.5] を呼び出す必要があります。

プロトタイプ

```
DWORD WDU_GetDeviceInfo(
    WDU_DEVICE_HANDLE hDevice,
    WDU_DEVICE **ppDeviceInfo);
```

パラメータ

名前	型	入出力
➢ hDevice	WDU_DEVICE_HANDLE	入力
➢ ppDeviceInfo	WDU_DEVICE**	出力

説明

名前	説明
hDevice	デバイス / インターフェイスのユニークな識別子。
ppDeviceInfo	USB デバイス情報の構造体 [A.4.3] へのポインタをポイントします。

戻り値

正常終了した場合、WD_STATUS_SUCCESS (0) を返します。その他の場合、エラーコードを返します [A.7]。

A.3.5 WDU_PutDeviceInfo()

目的

- 以前の `WDU_GetDeviceInfo()` [A.3.4] の呼び出しで割り当てられた、デバイス情報のポインタを受信します。

プロトタイプ

```
void WDU_PutDeviceInfo(WDU_DEVICE *pDeviceInfo);
```

パラメータ

名前	型	入出力
➢ pDeviceInfo	WDU_DEVICE*	入力

説明

名前	説明
pDeviceInfo	`WDU_GetDeviceInfo()` [A.3.4] への以前の呼び出しで返された USB デバイス情報の構造体へのポインタ [A.4.3]

戻り値

なし。

A.3.6 WDU_Uninit()

目的

- 入力基準に一致するデバイスへリスンを開始し、デバイスの通知コールバックを解除します。

プロトタイプ

```
void WDU_Uninit(WDU_DRIVER_HANDLE hDriver);
```

パラメータ

名前	型	入出力
➢ hDriver	WDU_DRIVER_HANDLE	入力

説明

名前	説明
hDriver	`WDU_Init()` [A.3.1] から受信した登録をハンドルします。

戻り値

なし。

A.3.7 シングル ブロッキング転送関数

このセクションでは、WinDriver のシングル ブロッキング データ転送関数について説明します。

詳細は、WinDriver ユーザーズ ガイドのセクション 9.5 を参照してください。

A.3.7.1 WDU_Transfer()

目的

- デバイスへまたはデバイスからデータを転送します。

プロトタイプ

```
DWORD WDU_Transfer(
    WDU_DEVICE_HANDLE hDevice,
    DWORD dwPipeNum,
    DWORD fRead,
    DWORD dwOptions,
    PVOID pBuffer,
    DWORD dwBufferSize,
    PDWORD pdwBytesTransferred,
    PBYTE pSetupPacket,
    DWORD dwTimeout);
```

パラメータ

名前	型	入出力
➢ hDevice	WDU_DEVICE_HANDLE	入力
➢ dwPipeNum	DWORD	入力
➢ fRead	DWORD	入力
➢ dwOptions	DWORD	入力

➢ pBuffer	PVOID	入力
➢ dwBufferSize	DWORD	入力
➢ pdwBytesTransferred	PDWORD	出力
➢ pSetupPacket	PBYTE	入力
➢ dwTimeout	DWORD	入力

説明

名前	説明
hDevice	`WDU_Init()` [A.3.1] から取得したデバイス / インターフェイスのユニークな識別子
dwPipeNum	データ転送に使用するパイプの数
fRead	read (読み取り) の場合は TRUE、write (書き込み) の場合 FALSE
dwOptions	以下のいずれかのフラグのビット マスク。 • **`USB_ISOCH_NOASAP`** -等時性 (アイソクロナス) 転送。このオプションを設定することで、データ転送の実行中に下位 USB スタック ドライバ (**`usbd.sys`**) は (次に利用可能なフレームの代わりに) 現在のフレーム番号を使用するよう命令します。低スピードまたは高スピードのデバイス (USB 1.1 のみ) で転送中に未使用のフレームがある場合、このフラグを使用します (Windows のみ。ただし、Windows CE は除く)。 • **`USB_ISOCH_RESET`** -データ転送を行なう前に等時性パイプをリセットします。また、マイナーなエラーが発生した際にパイプをリセットします (データ転送は続行されます)。 • **`USB_ISOCH_FULL_PACKETS_ONLY`** -パケット サイズ以下のデータを等時性パイプに転送しません。 • **`USB_BULK_INT_URB_SIZE_OVERRIDE_128K`** - URB (USB Request Block) のサイズを 128KB に制限します。
pBuffer	データ バッファのアドレス
dwBufferSize	転送するバイト数。バッファ サイズは、デバイスの最大パケット サイズに制限されません。このため、バッファ サイズを最大パケット サイズの倍数に設定して、より大きなバッファを使用できます。大きなバッファを使用してコンテキスト スイッチの数を減らし、パフォーマンスを向上します。
pdwBytesTransferred	実際に転送されたバイト数
pSetupPacket	パイプを制御するために転送する 8 バイトのパケット
dwTimeout	転送にかかるミリ秒 (ms) 単位での最大所要時間。0 の場合、タイムアウトせずに、無限に待ちます。

戻り値

正常終了した場合、WD_STATUS_SUCCESS (0) を返します。その他の場合、エラーコードを返します [A.7]。

注釈

- タイムアウト (dwTimeout パラメータ) の最小単位は、オペレーティング システムのスケジューラのタイムスロットに依存します。たとえば、Windows の場合、タイムアウトの最小単位は 10 ミリ秒 (ms) です。

A.3.7.2 WDU_HaltTransfer()

目的

- 指定されたパイプの転送を停止します (WinDriver では、1 つのパイプにつき、同時に 1 つの転送のみ許可されています)。

プロトタイプ

```
DWORD WDU_HaltTransfer(
    WDU_DEVICE_HANDLE hDevice,
    DWORD dwPipeNum);
```

パラメータ

名前	型	入出力
➢ hDevice	WDU_DEVICE_HANDLE	入力
➢ dwPipeNum	DWORD	入力

説明

名前	説明
hDevice	デバイス / インターフェイスのユニークな識別子
dwPipeNum	パイプの数

戻り値

正常終了した場合、WD_STATUS_SUCCESS (0) を返します。その他の場合、エラーコードを返します [A.7]。

A.3.7.3 WDU_TransferDefaultPipe()

目的

- デフォルトのパイプを使用して、デバイスへ またはデバイスからデータを転送します。

プロトタイプ

```
DWORD WDU_TransferDefaultPipe(
    WDU_DEVICE_HANDLE hDevice,
    DWORD fRead,
    DWORD dwOptions,
    PVOID pBuffer,
    DWORD dwBufferSize,
    PDWORD pdwBytesTransferred,
    PBYTE pSetupPacket,
    DWORD dwTimeout);
```

パラメータ

`WDU_Transfer()` [A.3.7.1] のパラメータを参照してください。
`dwPipeNum` はこの関数のパラメータではありません。

説明

`WDU_Transfer()` [A.3.7.1] の説明を参照してください

戻り値

正常終了した場合、WD_STATUS_SUCCESS (0) を返します。その他の場合、エラーコードを返します [A.7]。

A.3.7.4 WDU_TransferBulk()

目的

- デバイスへまたはデバイスからバルク データ転送を実行します。

プロトタイプ

```
DWORD WDU_TransferBulk(
    WDU_DEVICE_HANDLE hDevice,
    DWORD dwPipeNum,
    DWORD fRead,
    DWORD dwOptions,
    PVOID pBuffer,
    DWORD dwBufferSize,
    PDWORD pdwBytesTransferred,
    DWORD dwTimeout);
```

パラメータ

`WDU_Transfer()` [A.3.7.1] のパラメータを参照してください。
`pSetupPacket` はこの関数のパラメータではありません。

説明

`WDU_Transfer()` [A.3.7.1] の説明を参照してください。

戻り値

正常終了した場合、WD_STATUS_SUCCESS (0) を返します。その他の場合、エラーコードを返します [A.7]。

A.3.7.5 WDU_TransferIsoch()

目的

- デバイスへまたはデバイスから等時性 (isochronous) データ転送を実行します。

プロトタイプ

```
DWORD WDU_TransferIsoch(
    WDU_DEVICE_HANDLE hDevice,
    DWORD dwPipeNum,
    DWORD fRead,
    DWORD dwOptions,
    PVOID pBuffer,
    DWORD dwBufferSize,
    PDWORD pdwBytesTransferred,
    DWORD dwTimeout);
```

パラメータ

`WDU_Transfer()` [A.3.7.1] のパラメータを参照してください。
`pSetupPacket` はこの関数のパラメータではありません。

説明

`WDU_Transfer()` [A.3.7.1] の説明を参照してください。

戻り値

正常終了した場合、WD_STATUS_SUCCESS (0) を返します。その他の場合、エラーコードを返します [A.7]。

A.3.7.6 WDU_TransferInterrupt()

目的

- デバイスへまたはデバイスから割り込みデータ転送を実行します。

プロトタイプ

```
DWORD WDU_TransferInterrupt(
    WDU_DEVICE_HANDLE hDevice,
    DWORD dwPipeNum,
    DWORD fRead,
    DWORD dwOptions,
    PVOID pBuffer,
    DWORD dwBufferSize,
    PDWORD pdwBytesTransferred,
    DWORD dwTimeout);
```

パラメータ

`WDU_Transfer()` [A.3.7.1] のパラメータを参照してください。
`pSetupPacket` はこの関数のパラメータではありません。

説明

`WDU_Transfer()` [A.3.7.1] の説明を参照してください。

戻り値

正常終了した場合、WD_STATUS_SUCCESS (0) を返します。その他の場合、エラーコードを返します [A.7]。

A.3.8 ストリーミング データ転送関数

このセクションでは、WinDriver のストリーミング データ転送関数について説明します。

ストリーム転送および Windriver を使用した実装に関する詳細は、WinDriver ユーザーズ ガイドのセクション 9.5 を参照してください。このセクションで説明する API は、現在 Windows 2000 以降でのみサポートしています。

A.3.8.1 WDU_StreamOpen()

目的

- 指定されたパイプの新しいストリームを開きます。
- コントロール パイプ (Pipe 0) を除く、すべてのパイプにストリームを関連付けることができます。
- ストリームのデータ転送方向 (読み取り/書き込み) は、パイプの方向により決定されます。

プロトタイプ

```
DWORD DLLCALLCONV WDU_StreamOpen(
    WDU_DEVICE_HANDLE hDevice,
    DWORD dwPipeNum,
    DWORD dwBufferSize,
    DWORD dwRxSize,
    BOOL  fBlocking,
    DWORD dwOptions,
    DWORD dwRxTxTimeout,
    WDU_STREAM_HANDLE *phStream);
```

パラメータ

名前	型	入出力
➢ hDevice	WDU_DEVICE_HANDLE	入力
➢ dwPipeNum	DWORD	入力
➢ dwBufferSize	DWORD	入力
➢ dwRxSize	DWORD	入力
➢ fBlocking	BOOL	入力
➢ dwOptions	DWORD	入力
➢ dwRxTxTimeout	DWORD	入力
➢ phStream	WDU_STREAM_HANDLE*	出力

説明

名前	説明
hDevice	デバイス / インターフェイスのユニークな識別子。
dwPipeNum	ストリームを開くパイプの番号
dwBufferSize	ストリーム データ バッファのバイト単位でのサイズ
dwRxSize	ストリームによりデバイスから読み取るデータ ブロックのバイト単位でのサイズ。このパラメータは、読み取りストリームに対してのみ使用され、`dwBufferSize` パラメータ の値以下でなければなりません。
fBlocking	ブロッキング I/O を使用するブロッキング ストリームの場合は `TRUE`。ノンブロッキング I/O を使用するノンブロッキング ストリームの場合は `FALSE`。詳細は、WinDriver ユーザーズ ガイドのセクション 9.5.3.1 を参照してください。
dwOptions	以下のいずれかのフラグのビット マスク。 **`USB_ISOCH_NOASAP`** - 等時性 (isochronous) データ転送。このオプションを設定することで、データ転送の実行中に下位 USB スタック ドライバ (**`usbd.sys`**) は (次に利用可能なフレームの代わりに) 現在のフレーム番号を使用するよう命令します。低スピードまたは高スピードのデバイス (USB 1.1 のみ) で転送中に未使用のフレー

	ムがある場合、このフラグを使用します (Windows のみ)。 **USB_ISOCH_RESET** - データ転送を行なう前に等時性パイプをリセットします。また、マイナーなエラーが発生した際にパイプをリセットします (データ転送は続行されます)。 **USB_ISOCH_FULL_PACKETS_ONLY** - パケット サイズ以下のデータを等時性パイプに転送しません。 **USB_BULK_INT_URB_SIZE_OVERRIDE_128K** - URB (USB Request Block) のサイズを 128KB に制限します。
dwRxTxTimeout	ストリームとデバイス間のデータ転送のミリ秒 (ms) 単位での最大所要時間。0 の場合、タイムアウトせずに、無限に待ちます。
phStream	ストリームのユニークな識別子へのポインタ。この関数により返され、その他の `WDU_StreamXXX()` 関数に渡されます。

戻り値

正常終了した場合、WD_STATUS_SUCCESS (0) を返します。その他の場合、エラーコードを返します [A.7]。

A.3.8.2 WDU_StreamStart()

目的

- ストリーム (例: ストリームとデバイス間の転送) を開始します。
- データは、ストリームの方向 (読み取り/書き込み) に転送されます。

プロトタイプ

```
DWORD DLLCALLCONV WDU_StreamStart(
    WDU_STREAM_HANDLE hStream);
```

パラメータ

名前	型	入出力
➢ hStream	WDU_STREAM_HANDLE	入力

説明

名前	説明
hStream	`WDU_StreamOpen()` によって返されるストリームのユニークな識別子

戻り値

正常終了した場合、WD_STATUS_SUCCESS (0) を返します。その他の場合、エラーコードを返します [A.7]。

A.3.8.3 WDU_StreamRead()

目的

- 読み取りストリームからアプリケーションへデータを読み取ります。
- ブロッキング ストリームの場合 (`fBlocking=TRUE - WDU_StreamOpen()` を参照)、指定されたデータ量 (バイト) を読み取るか、ストリームによるデバイスからの読み取りがタイムアウト (例: `WDU_StreamOpen()` の `dwRxTxTimeout` パラメータ [A.3.8.1] で設定されたストリームとデバイス間の転送のタイムアウトに到達した場合) になるまでこの関数の呼び出しはブロックされます。
- ノンブロッキング ストリームの場合 (`fBlocking=FALSE`)、この関数は要求されたデータをできるだけ多く (ストリーム データ バッファにある利用可能なデータ量により異なる) アプリケーションに転送して、直ちにリターンします。
- ブロッキング転送およびノンブロッキング転送の両方の場合で、この関数はストリームから実際に読み取ったバイト数を、`pdwBytesRead` パラメータに格納して返します。

プロトタイプ

```
DWORD DLLCALLCONV WDU_StreamRead(
    HANDLE hStream,
    PVOID pBuffer,
    DWORD bytes,
    DWORD *pdwBytesRead);
```

パラメータ

名前	型	入出力
➢ hStream	WDU_STREAM_HANDLE	入力
➢ pBuffer	PVOID	出力
➢ bytes	DWORD	入力
➢ pdwBytesRead	DWORD*	出力

説明

名前	説明
hStream	`WDU_StreamOpen()` によって返されるストリームのユニークな識別子
pBuffer	ストリームから読み取るデータを保持するためのデータ バッファへのポインタ
bytes	ストリームから読み取るバイト数
pdwBytesRead	ストリームから実際に読み取ったバイト数へのポインタ

戻り値

正常終了した場合、WD_STATUS_SUCCESS (0) を返します。その他の場合、エラーコードを返します [A.7]。

A.3.8.4 WDU_StreamWrite()

目的

- アプリケーションから書き込みストリームへデータを書き込みます。
- ブロッキング ストリームの場合 (`fBlocking=TRUE - WDU_StreamOpen()` を参照)、すべてのデータを書き込むか、ストリームによるデバイスへの書き込みがタイムアウト (例: `WDU_StreamOpen()` の `dwRxTxTimeout` パラメータ [A.3.8.1] で設定されたストリームとデバイス間の転送のタイムアウトに到達した場合) になるまでこの関数の呼び出しはブロックされます。
- ノンブロッキング ストリームの場合 (`fBlocking=FALSE`)、この関数はできるだけ多くのデータをストリーム データ バッファに書き込み、直ちにリターンします。
- ブロッキング転送およびノンブロッキング転送の両方の場合で、この関数はストリームへ実際に書き込んだバイト数を、`pdwBytesWritten` パラメータに格納して返します。

プロトタイプ

```
DWORD DLLCALLCONV WDU_StreamWrite(
    HANDLE hStream,
    const PVOID pBuffer,
    DWORD bytes,
    DWORD *pdwBytesWritten);
```

パラメータ

名前	型	入出力
➢ hStream	WDU_STREAM_HANDLE	入力
➢ pBuffer	PVOID	入力
➢ bytes	DWORD	入力
➢ pdwBytesWritten	DWORD*	出力

説明

名前	説明
hStream	`WDU_StreamOpen()` によって返されるストリームのユニークな識別子
pBuffer	ストリームへ書き込むデータを保持するデータ バッファへのポインタ
bytes	ストリームへ書き込むバイト数
pdwBytesWritten	ストリームへ実際に書き込んだバイト数へのポインタ

戻り値

正常終了した場合、WD_STATUS_SUCCESS (0) を返します。その他の場合、エラーコードを返します [A.7]。

A.3.8.5 WDU_StreamFlush()

目的

- 書き込みストリームをフラッシュします (例: ストリーム データ バッファのすべてのコンテンツをデバイスに書き込みます)。
- ストリームのすべての待機中 I/O が処理されるまでブロックします。
- この関数は、ブロッキング ストリームとノンブロッキング ストリームの両方で呼び出すことができます。

プロトタイプ

```
DWORD DLLCALLCONV WDU_StreamFlush(
    WDU_STREAM_HANDLE hStream);
```

パラメータ

名前	型	入出力
➢ hStream	WDU_STREAM_HANDLE	入力

説明

名前	説明
hStream	`WDU_StreamOpen()` によって返されるストリームのユニークな識別子

戻り値

正常終了した場合、WD_STATUS_SUCCESS (0) を返します。その他の場合、エラーコードを返します [A.7]。

A.3.8.6 WDU_StreamStop()

目的

- アクティブなストリーム (例: ストリームとデバイス間の転送) を停止します。
- 書き込みストリームの場合、この関数はストリームを停止する前にフラッシュ (ストリームのコンテンツをデバイスに書き込むなど) します。

プロトタイプ

```
DWORD DLLCALLCONV WDU_StreamStop(
    WDU_STREAM_HANDLE hStream);
```

パラメータ

名前	型	入出力
➢ hStream	WDU_STREAM_HANDLE	入力

説明

名前	説明
hStream	`WDU_StreamOpen()` によって返されるストリームのユニークな識別子

戻り値

正常終了した場合、WD_STATUS_SUCCESS (0) を返します。その他の場合、エラーコードを返します [A.7]。

A.3.8.7 WDU_StreamClose()

目的

- 開かれているストリームを閉じます。
- この関数はストリームを閉じる前に、ストリームを停止し、データをデバイスにフラッシュします (書き込みストリームの場合)。

プロトタイプ

```
DWORD DLLCALLCONV WDU_StreamClose(
    WDU_STREAM_HANDLE hStream);
```

パラメータ

名前	型	入出力
➢ hStream	WDU_STREAM_HANDLE	入力

説明

名前	説明
hStream	`WDU_StreamOpen()` によって返されるストリームのユニークな識別子

戻り値

正常終了した場合、WD_STATUS_SUCCESS (0) を返します。その他の場合、エラーコードを返します [A.7]。

A.3.9 WDU_ResetPipe()

目的

- パイプのホスト側の停止状態とエンドポイントのストール状態の両方をクリアして、パイプをリセットします。
- pipe00 を除くすべてのパイプで有効です。

プロトタイプ

```
DWORD WDU_ResetPipe(
    WDU_DEVICE_HANDLE hDevice,
    DWORD dwPipeNum);
```

パラメータ

名前	型	入出力
➢ hDevice	WDU_DEVICE_HANDLE	入力
➢ dwPipeNum	DWORD	入力

説明

名前	説明
hDevice	デバイス / インターフェイスのユニークな識別子
dwPipeNum	パイプの番号

戻り値

正常終了した場合、WD_STATUS_SUCCESS (0) を返します。その他の場合、エラーコードを返します [A.7]。

注釈

- この関数は、パイプの停止状態をクリアするために使用します。

A.3.10 WDU_ResetDevice()

目的

- デバイスをリセットします。

プロトタイプ

```
DWORD WDU_ResetDevice(
    WDU_DEVICE_HANDLE hDevice,
    DWORD dwOptions);
```

パラメータ

名前	型	入出力
➢ hDevice	WDU_DEVICE_HANDLE	入力
➢ dwOptions	DWORD	入力

説明

名前	説明
hDevice	デバイス / インターフェイスのユニークな識別子
dwOptions	0 または: • **WD_USB_HARD_RESET** -デバイスが無効に設定されていない場合でもリセットします。このオプションを使用した後に、WDU_SetInterface() [A.3.2] を使用して、インターフェイス デバイスを設定することを推奨します。 • **WD_USB_CYCLE_PORT** - デバイスをリセットせずに再度列挙するようにオペレーティング システムに求め、デバイスの取り外しおよび再装着のシミュレーションを行います。このオプションは、Windows XP 以降でのみサポートしています。

戻り値

正常終了した場合、WD_STATUS_SUCCESS (0) を返します。その他の場合、エラーコードを返します [A.7]。

注釈

- WDU_ResetDevice() は Windows および Windows CE 5.0 以降でサポートしています。WD_USB_CYCLE_PORT オプションは、Windows XP 以降でサポートしています。
- この関数は、Windows USB ドライバから、ハブ ポートのリセット要求を発行します (Windows USB ドライバでこの機能がサポートされていることを前提としています)。

A.3.11 WDU_SelectiveSuspend()

目的

- 指定されたデバイスのサスペンド要求を送信 (選択的サスペンド)、または送信済みサスペンド要求をキャンセルします。

プロトタイプ

```
DWORD DLLCALLCONV WDU_SelectiveSuspend(
    WDU_DEVICE_HANDLE hDevice,
    DWORD dwOptions);
```

パラメータ

名前	型	入出力
➢ hDevice	WDU_DEVICE_HANDLE	入力
➢ dwOptions	DWORD	入力

説明

名前	説明
hDevice	デバイス / インターフェイスのユニークな識別子
dwPipeNum	以下のいずれかの WDU_SELECTIVE_SUSPEND_OPTIONS の値を指定できます。 **WDU_SELECTIVE_SUSPEND_SUBMIT** - デバイスのサスペンド要求を送信します。 **WDU_SELECTIVE_SUSPEND_CANCEL** - 送信済みのデバイスのサスペンド要求をキャンセルします。

戻り値

正常終了した場合、WD_STATUS_SUCCESS (0) を返します。その他の場合、エラーコードを返します [A.7]。

サスペンド要求を送信した際にデバイスがビジーな場合 (dwOptions = WDU_SELECTIVE_SUSPEND_SUBMIT)、WD_OPERATION_FAILED を返します。

注釈

- WDU_SelectiveSuspend() は、Windows XP 以降でサポートしています。

A.3.12 WDU_Wakeup()

目的

- ウェイクアップ機能を有効 / 無効にします。

プロトタイプ

```
DWORD WDU_Wakeup(
    WDU_DEVICE_HANDLE hDevice,
    DWORD dwOptions);
```

パラメータ

名前	型	入出力
➢ hDevice	WDU_DEVICE_HANDLE	入力
➢ dwOptions	DWORD	入力

説明

名前	説明
hDevice	デバイス / インターフェイスのユニークな識別子。
dwOptions	以下のいずれか: • **WDU_WAKEUP_ENABLE** - ウェイクアップ機能を有効にします。

	または • **WDU_WAKEUP_DISABLE** - ウェイクアップ機能を無効にします。

戻り値

正常終了した場合、WD_STATUS_SUCCESS (0) を返します。その他の場合、エラーコードを返します [A.7]。

A.3.13 WDU_GetLangIDs()

目的

- デバイスからサポートされている言語 ID のリストと数を読み取ります。

プロトタイプ

```
DWORD DLLCALLCONV WDU_GetLangIDs(
    WDU_DEVICE_HANDLE hDevice,
    PBYTE pbNumSupportedLangIDs,
    WDU_LANGID *pLangIDs,
    BYTE bNumLangIDs);
```

パラメータ

名前	型	入出力
➢ hDevice	WDU_DEVICE_HANDLE	入力
➢ pbNumSupportedLangIDs	PBYTE	出力
➢ pLangIDs	WDU_LANGID*	出力
➢ bNumLangIDs	BYTE	入力

説明

名前	説明
hDevice	デバイス/インターフェイス用のユニークな識別子
pbNumSupportedLangIDs	サポートされている言語 ID の数を受け取るパラメータ
pLangIDs	言語 ID の配列。**bNumLangIDs** が 0 でない場合、デバイスでサポートされている言語 ID が格納されます。
bNumLangIDs	pLangIDs 配列に格納されている ID の数

戻り値

正常終了した場合、WD_STATUS_SUCCESS (0) を返します。その他の場合、エラーコードを返します [A.7]。

注釈

- **dwNumLangIDs** が 0 の場合、この関数はサポートされている言語 ID の数 (**pbNumSupportedLangIDs**)のみを返し、実際の ID で **pLangIDs** 配列を更新しません。このため、**pLangIDs** は (参照されたないため) NULL にすることができまが、**pbNumSupportedLangIDs** は NULL にしないでください。ただし、ユーザーがサポートされている言語 ID の数ではなく、そのリストだけを必要とする場合、**pbNumSupportedLangIDs** を NULL にすることができます。この場合、**bNumLangIDs** を 0 にしたり、**pLangIDs** を NULL にすることはできません。
- デバイスがどの言語 ID もサポートしていない場合、この関数は正常終了します。このため、この関数がリターンした後に、呼び出し元で ***pbNumSupportedLangIDs** の値を確認する必要があります。
- **pLangIDs** 配列のサイズ (**bNumLangIDs**) がデバイスでサポートされている ID の数 (***pbNumSupportedLangIDs**) よりも小さい場合、この関数はサポートされている言語 ID の中から最初の **bNumLangIDs** のみ読み取って返します。

A.3.14 WDU_GetStringDesc()

目的

- 文字列インデックスによりデバイスから文字列記述子を読み取ります。

プロトタイプ

```
DWORD DLLCALLCONV WDU_GetStringDesc(
    WDU_DEVICE_HANDLE hDevice,
    BYTE bStrIndex,
    PCHAR pcDescStr,
    DWORD dwSize,
    WDU_LANGID langID);
```

パラメータ

名前	型	入出力
➢ hDevice	WDU_DEVICE_HANDLE	入力
➢ bStrIndex	BYTE	入力
➢ pbBuf	PBYTE	出力
➢ dwBufSize	DWORD	入力
➢ langID	WDU_LANGID	入力
➢ pdwDescSize	PDWORD	出力

説明

名前	説明
hDevice	デバイス/インターフェイスのユニークな識別子
bStrIndex	文字列 インデックス

pbBuf	読み取り文字列記述子 (記述子はバイト配列として返されます)
dwBufSize	`pbBuf` のサイズ
langID	デバイスに送信する文字列記述子の取得要求で使用される言語 ID。`langID` パラメータが 0 の場合、この関数はデバイスから返されるサポートされている言語 ID の中から最初のもの (存在する場合) を使用します。
pdwDescSize	`NULL` でない場合、デバイスから返される記述子のサイズで更新されます。

戻り値

正常終了した場合、WD_STATUS_SUCCESS (0) を返します。その他の場合、エラーコードを返します [A.7]。

注釈

- **`pbBuf`** のサイズが文字列記述子を保持するのに十分でない場合 (**`dwBufSize`** < **`*pdwDescSize`**) 、デバイスから返された記述子は `dwBufSize` のサイズに切り捨てられます。

A.4 USB – 構造

次の図は、WinDriver の USB API が使用する構造階層を表しています。階層の各レベルに位置する配列は、図で表されている以上に要素を持っています。矢印はポインタを表しています。分かり易くするために、階層の各レベルの 1 つの構造のみを、詳細に説明しています (リストされたすべての要素とポインタ)。

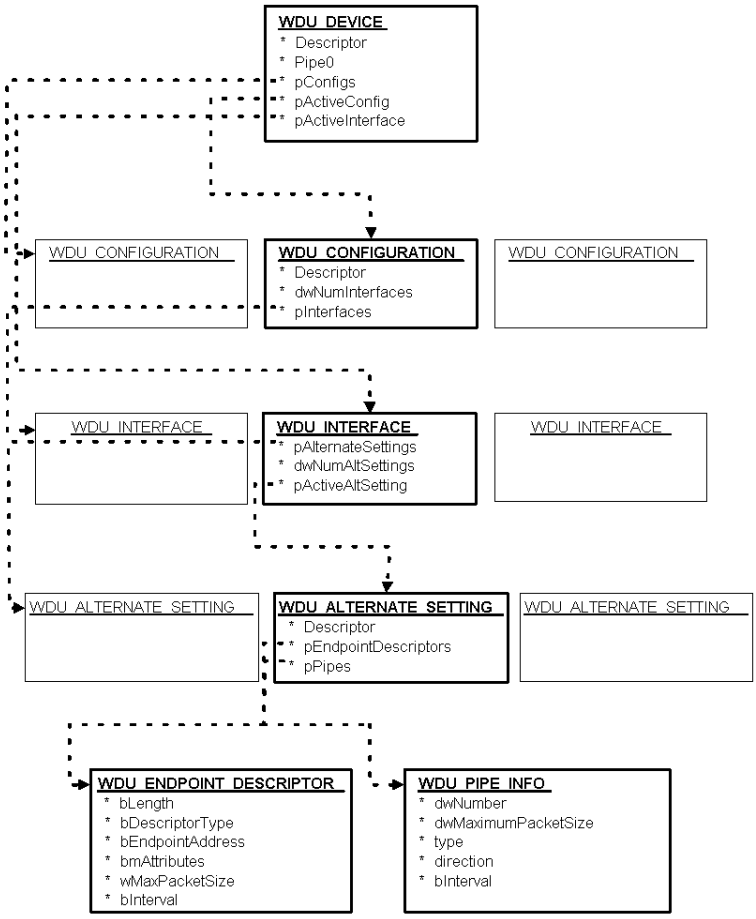


図 A.2: WinDriver USB の構造

A.4.1 WDU_MATCH_TABLE 構造体

USB 一致テーブルは、以下の要素で構成されています。

注意

すべての項目の (*) は、値を 0 に設定すると、すべて一致します。

名前	型	説明
wVendorId	WORD	USB-IF に割り当てられる、検出に必要な USB ベンダー ID。(*)
wProductId	WORD	製造元に割り当てられる、検出に必要な USB プロダクト ID。(*)
bDeviceClass	BYTE	USB-IF に割り当てられる、デバイスのクラス コード。(*)
bDeviceSubClass	BYTE	USB-IF に割り当てられる、デバイスのサブ クラス コード。(*)
bInterfaceClass	BYTE	USB-IF に割り当てられる、インターフェイスのクラス コード。(*)
bInterfaceSubClass	BYTE	USB-IF に割り当てられる、インターフェイスのサブ クラス コード。(*)
bInterfaceProtocol	BYTE	USB-IF に割り当てられる、インターフェイスのプロトコル コード。(*)

A.4.2 WDU_EVENT_TABLE 構造体

USB イベント テーブルは、以下の要素で構成されています。

名前	型	説明
pfDeviceAttach	WDU_ATTACH_CALLBACK	デバイスを装着したときに WinDriver に呼ばれます。
pfDeviceDetach	WDU_DETACH_CALLBACK	デバイスを取り外したときに WinDriver に呼ばれます。
pfPowerChange	WDU_POWER_CHANGE_CALLBACK	デバイスの電源状態が変化すると WinDriver から呼ばれます。
pUserData	PVOID	コールバックへ渡されるユーザーモード データへのポインタ。

A.4.3 WDU_DEVICE 構造体

USB デバイス情報の構造体は、以下の要素で構成されています。

名前	型	説明
Descriptor	WDU_DEVICE_DESCRIPTOR	デバイス記述子情報の構造体 [A.4.7]。
Pipe0	WDU_PIPE_INFO	デバイスのデフォルトのパイプ (Pipe 0) に関する情報の構造体 [A.4.11] 。
pConfigs	WDU_CONFIGURATION*	デバイスの設定情報の構造体 [A.4.4] へのポインタ。
pActiveConfig	WDU_CONFIGURATION*	デバイスのアクティブな設定に関する情報の構造体 [A.4.4] へのポインタ。
pActiveInterface	WDU_INTERFACE*	デバイスのアクティブなインターフェイスに関する情報の構造体 [A.4.5] へのポインタの配列。

A.4.4 WDU_CONFIGURATION 構造体

設定情報の構造体は、以下の要素で構成されています。

名前	型	説明
Descriptor	WDU_CONFIGURATION_DESCRIPTOR	設定記述子情報の構造体 [A.4.8] 。
dwNumInterfaces	DWORD	この設定でサポートされるインターフェイスの数。
pInterfaces	WDU_INTERFACE*	設定のインターフェイスに関する情報の構造体 [A.4.5] 配列の初めへのポインタ。

A.4.5 WDU_INTERFACE 構造体

インターフェイス情報の構造体は、以下の要素で構成されています。

名前	型	説明
pAlternateSettings	WDU_ALTERNATE_SETTING*	インターフェイスの代替設定に関する情報の構造体 [A.4.6] 配列の初めへのポインタ。
dwNumAltSettings	DWORD	このインターフェイスでサポートされている代替設定の数。
pActiveAltSetting	WDU_ALTERNATE_SETTING*	インターフェイスのアクティブな代替設定に関する情報の構造体 [A.4.6] へのポインタ。

A.4.6 WDU_ALTERNATE_SETTING 構造体

代替設定情報の構造体は、以下の要素で構成されています。

名前	型	説明
Descriptor	WDU_INTERFACE_DESCRIPTOR	インターフェイス記述子情報の構造体 [A.4.9]。
pEndpointDescriptors	WDU_ENDPOINT_DESCRIPTOR*	代替設定のエンドポイント記述子情報の構造体 [A.4.10] 配列の初めへのポインタ
pPipes	WDU_PIPE_INFO*	代替設定のパイプ情報の構造体 [A.4.11] 配列の初めへのポインタ

A.4.7 WDU_DEVICE_DESCRIPTOR 構造体

USB デバイス記述子情報の構造体は、以下の要素で構成されています。

名前	型	説明
bLength	UCHAR	記述子のバイト サイズ (18 バイト)。
bDescriptorType	UCHAR	デバイス記述子 (0x01)。

bcdUSB	USHORT	デバイスが対応する USB 仕様の数。
bDeviceClass	UCHAR	デバイスのクラス。
bDeviceSubClass	UCHAR	デバイスのサブ クラス。
bDeviceProtocol	UCHAR	デバイスのプロトコル。
bMaxPacketSize0	UCHAR	転送されるパケットの最大サイズ。
idVendor	USHORT	USB-IF に割り当てられるベンダー ID。
idProduct	USHORT	製造元に割り当てられるプロダクト ID。
bcdDevice	USHORT	デバイス リリース番号。
iManufacturer	UCHAR	製造元文字列記述子のインデックス。
iProduct	UCHAR	製品文字列記述子のインデックス。
iSerialNumber	UCHAR	シリアル番号文字列記述子のインデックス。
bNumConfigurations	UCHAR	設定可能な数。

A.4.8 WDU_CONFIGURATION_DESCRIPTOR 構造体

USB 設定記述子情報の構造体は、以下の要素で構成されています。

名前	型	説明
bLength	UCHAR	記述子のバイト サイズ。
bDescriptorType	UCHAR	デバイス記述子 (0x02)。
wTotalLength	USHORT	必要なデータの合計バイト長。
bNumInterfaces	UCHAR	インターフェイスの数。
bConfigurationValue	UCHAR	設定番号。
iConfiguration	UCHAR	この設定を記述する文字列記述子のインデックス。
bmAttributes	UCHAR	この設定の電源設定: • *D6* - 電源内臓の場合 • *D5* - リモート ウェイクアップの場合 (デバイスはホストをウェイク アップできます)。
MaxPower	UCHAR	2*mA* ユニットで、この設定の最大電力消費量。

A.4.9 WDU_INTERFACE_DESCRIPTOR 構造体

USB インターフェイス記述子情報の構造体は、以下の要素で構成されています。

名前	型	説明
bLength	UCHAR	記述子のバイト サイズ (9 バイト)。
bDescriptorType	UCHAR	デバイス記述子 (0x04)。
bInterfaceNumber	UCHAR	インターフェイス番号。

bAlternateSetting	UCHAR	代替設定番号。
bNumEndpoints	UCHAR	このインターフェイスで使用するエンドポイントの数。
bInterfaceClass	UCHAR	USB-IF に割り当てられるインターフェイスのクラス コード。
bInterfaceSubClass	UCHAR	USB-IF に割り当てられるインターフェイスのサブ クラス コード。
bInterfaceProtocol	UCHAR	USB-IF に割り当てられるインターフェイスのプロトコル コード。
iInterface	UCHAR	このインターフェイスを記述する文字列記述子のインデックス。

A.4.10 WDU_ENDPOINT_DESCRIPTOR 構造体

USB エンドポイント記述子情報の構造体は、以下の要素で構成されています。

名前	型	説明
bLength	UCHAR	記述子のバイト サイズ (7 バイト)。
bDescriptorType	UCHAR	エンドポイント記述子 (0x05)。
bEndpointAddress	UCHAR	エンドポイント アドレス: エンドポイント番号のビット 0-3 を使用します。ビット 4-6 を 0 に設定します。アウトバウンド データの 0 およびインバウンド データの 1 にビット 7 を設定します (制御エンドポイントのために無視されます)。
bmAttributes	UCHAR	このエンドポイントの転送タイプを指定します (制御、割り込み、等時性またはバルク)。詳細は、USB の仕様を参照してください。
wMaxPacketSize	USHORT	このエンドポイントが送受信可能なパケットの最大サイズ。
bInterval	UCHAR	フレーム カウントのエンドポイント データ転送のポーリング間隔。 バルクおよび制御コントロールのため無視されます。 等時性エンドポイントの 1 に等しいです。 割り込みエンドポイントの 1 から 255 の範囲です。

A.4.11 WDU_PIPE_INFO 構造体

USB パイプ情報の構造体は、以下の要素で構成されています。

名前	型	説明
dwNumber	DWORD	パイプ番号; デフォルトのパイプ番号は 0 です。
dwMaximumPacketSize	DWORD	このパイプを使用して転送できるパケットの最大サイズ。
type	DWORD	このパイプの転送タイプ。
direction	DWORD	転送の方法: • 等時性、バルクまたは割り込みパイプの場合、**USB_DIR_IN** または **USB_DIR_OUT**。 • 制御パイプの場合、**USB_DIR_IN_OUT**。
dwInterval	DWORD	ミリ秒 (*ms*) 間隔。 割り込みパイプにのみ適応されます。

A.5 一般的な WD_xxx 関数

A.5.1 WinDriver のコール順序 ー 一般用法

次に WinDriver API の一般的なコール順序を示します。

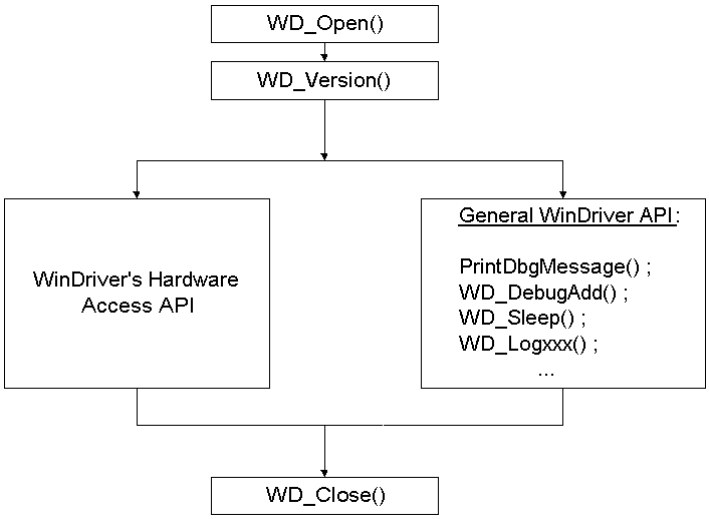


図 A.3: WinDriver API の呼び出し順序

注意

- `WD_Version()` [A.5.3] は、`WD_Open()` [A.5.2] をコールした後で他の WinDriver 関数をコールする前にコールすることを推奨します。その目的は、WinDriver カーネル モジュール (windrvr6.sys/.dll/.o/.ko) バージョン番号を返すことでアプリケーションおよび WinDriver カーネル モジュールのバージョンが互換であることを認識させます。
- `WD_Open()` のあと、`WD_DebugAdd()` [A.5.6] および `WD_Sleep()` [A.5.8] をどこからでもコールすることができます。

A.5.2 WD_Open()

目的

- WinDriver カーネル モジュールにアクセスするためにハンドルをオープンします。ハンドルはすべての WinDriver API によって使用されるため、他の WinDriver API がコールされる前にハンドルをコールしなければなりません。

プロトタイプ

```
HANDLE WD_Open(void);
```

戻り値

WinDriver カーネル モジュールへのハンドル。
デバイスをオープンできない場合は INVALID_HANDLE_VALUE を返します。

注釈

- 登録版を使用する場合、WinDriver のライセンスの登録方法に関しては、WD_License() [A.5.9] 関数の説明を参照してください。

例

```
HANDLE hWD;

hWD = WD_Open();
if (hWD == INVALID_HANDLE_VALUE)
{
    printf("Cannot open WinDriver device\n");
}
```

A.5.3 WD_Version()

目的

- 起動中の WinDriver カーネル モジュールのバージョン番号を返します。

プロトタイプ

```
DWORD WD_Version(
    HANDLE hWD,
    WD_VERSION *pVer);
```

パラメータ

名前	型	入出力
➢ hWD	HANDLE	入力
➢ pVer	WD_VERSION*	
☐ dwVer	DWORD	出力
☐ cVer	CHAR[128]	出力

説明

名前	説明
hWD	WD_Open() [A.5.2] から返される WinDriver のカーネル モード ドライバへのハンドル。
pVer	WinDriver バージョン情報の構造体へのポインタ。
dwVer	バージョン番号。
cVer	バージョン情報文字列。

	バージョン文字列のサイズは、128 文字までに制限されています (NULL 終端文字を含む)。

戻り値

正常終了した場合、WD_STATUS_SUCCESS (0) を返します。その他の場合、エラーコードを返します [A.7]。

例

```
WD_VERSION ver;

BZERO(ver);
WD_Version(hWD, &ver);
printf("%s\n", ver.cVer);
if (ver.dwVer < WD_VER)
{
    printf("Error - incorrect WinDriver version\n");
}
```

A.5.4 WD_Close()

目的

- WinDriver カーネル モジュールへのアクセスを終了します。

プロトタイプ

```
void WD_Close(HANDLE hWD);
```

パラメータ

名前	型	入出力
➢ hWD	HANDLE	入力

説明

名前	説明
hWD	WD_Open() [A.5.2] から返される WinDriver のカーネル モード ドライバへのハンドル。

戻り値

なし。

注釈

- WinDriver カーネル モジュールの使用を終了したときは、この関数を必ずコールします。

例

```
WD_Close(hWD);
```

A.5.5 WD_Debug()

目的

- デバッグメッセージを収集するレベルにデバッグレベルを設定 します。

プロトタイプ

```
DWORD WD_Debug(
    HANDLE hWD,
    WD_DEBUG *pDebug);
```

パラメータ

名前	型	入出力
➢ hWD	HANDLE	入力
➢ pDebug	WD_DEBUG*	入力
☐ dwCmd	DWORD	入力
☐ dwLevel	DWORD	入力
☐ dwSection	DWORD	入力
☐ dwLevelMessageBox	DWORD	入力
☐ dwBufferSize	DWORD	入力

説明

名前	説明
hWD	WD_Open() [A.5.2] から返される WinDriver のカーネル モード ドライバへのハンドル。
pDebug	デバッグ情報の構造体へのポインタ。
dwCmd	デバッグ コマンド: フィルタの設定、バッファのクリア等。 詳細は **windrvr.h** の DEBUG_COMMAND を参照してください。
dwLevel	dwCmd=DEBUG_SET_FILTER に使用されます。デバッグ レベルを Error、Warning、Info、Trace に設定します。 詳細は **windrvr.h** の DEBUG_LEVEL を参照してください。
dwSection	dwCmd=DEBUG_SET_FILTER に使用されます。セクションを I/O、Memory、Interrupt などに設定し、収集します (すべての場合は S_ALL を使用します)。 詳細は **windrvr.h** の DEBUG_SECTION を参照してください。
dwLevelMessageBox	dwCmd=DEBUG_SET_FILTER に使用されます。デバッグレベルをメッセージボックスに出力するように設定します。

	詳細は **windrvr.h** の DEBUG_LEVEL を参照してください。
dwBufferSize	dwCmd=DEBUG_SET_BUFFER に使用されます。カーネルにあるバッファのサイズです。

戻り値

正常終了した場合、WD_STATUS_SUCCESS (0) を返します。その他の場合、エラーコードを返します [A.7]。

例

```
WD_DEBUG dbg;

BZERO(dbg);
dbg.dwCmd = DEBUG_SET_FILTER;
dbg.dwLevel = D_ERROR;
dbg.dwSection = S_ALL;
dbg.dwLevelMessageBox = D_ERROR;

WD_Debug(hWD, &dbg);
```

A.5.6 WD_DebugAdd()

目的

- デバッグ ログへデバッグ メッセージを送ります。ドライバ コードで使用します。

プロトタイプ

```
DWORD WD_DebugAdd(
    HANDLE hWD,
    WD_DEBUG_ADD *pData);
```

パラメータ

名前	型	入出力
➢ hWD	HANDLE	入力
➢ pData	WD_DEBUG_ADD*	
☐ dwLevel	DWORD	入力
☐ dwSection	DWORD	入力
☐ pcBuffer	CHAR [256]	入力

説明

名前	説明
hWD	WD_Open() [A.5.2] から返される WinDriver のカーネル モード ドライバへのハンドル。
pData	追加するデバッグ情報の構造体へのポインタ。

dwLevel	宣言されるデータに、Debug Monitor でレベルを割り当てます。 dwLevel が 0 の場合、D_ERROR が宣言されます。 詳細は **windrvr.h** の DEBUG_LEVEL を参照してください。
dwSection	宣言されるデータに、Debug Monitor でセクションを割り当てます。 dwSection が 0 の場合、S_MISC が宣言されます。 詳細は **windrvr.h** の DEBUG_SECTION を参照してください。
pcBuffer	メッセージ ログにコピーする文字列。

戻り値

正常終了した場合、WD_STATUS_SUCCESS (0) を返します。その他の場合、エラーコードを返します [A.7]。

例

```
WD_DEBUG_ADD add;

BZERO(add);
add.dwLevel = D_WARN;
add.dwSection = S_MISC;
sprintf(add.pcBuffer, "This message will be displayed in "
    "the debug monitor\n");
WD_DebugAdd(hWD, &add);
```

A.5.7 WD_DebugDump()

目的

- デバッグ メッセージ バッファを取り出します。

プロトタイプ

```
DWORD WD_DebugDump(
    HANDLE hWD,
    WD_DEBUG_DUMP *pDebugDump);
```

パラメータ

名前	型	入出力
➢ hWD	HANDLE	入力
➢ pDebug	WD_DEBUG_DUMP*	入力
☐ pcBuffer	PCHAR	入力/出力
☐ dwSize	DWORD	入力

説明

名前	説明
hWD	WD_Open() [A.5.2] から返される WinDriver のカーネル モード ドライバへのハンドル。
pDebugDump	デバッグ ダンプ情報の構造体へのポインタ。
pcBuffer	デバッグ メッセージを受け取るバッファ。
dwSize	バッファ サイズ (Byte)。

戻り値

正常終了した場合、WD_STATUS_SUCCESS (0) を返します。その他の場合、エラーコードを返します [A.7]。

例

```
char buffer[1024];
WD_DEBUG_DUMP dump;
dump.pcBuffer=buffer;
dump.dwSize = sizeof(buffer);
WD_DebugDump(hWD, &dump);
```

A.5.8 WD_Sleep()

目的

- 指定した時間分、実行を遅らせます。

プロトタイプ

```
DWORD WD_Sleep(
    HANDLE hWD,
    WD_SLEEP *pSleep);
```

パラメータ

名前	型	入出力
➢ hWD	HANDLE	入力
➢ pSleep	WD_SLEEP*	
☐ dwMicroSeconds	DWORD	入力
☐ dwOptions	DWORD	入力

説明

名前	説明
hWD	WD_Open() [A.5.2] から返される WinDriver のカーネル モード ドライバへのハンドル。
pSleep	スリープ情報の構造体へのポインタ。
dwMicroSeconds	実行を遅らせる時間 (マイクロ秒単位)。- 1/1,000,000 秒
dwOptions	以下のいずれかのフラグのビット マスク。 • **Zero (0)** - Busy sleep (デフォルト) または、 • **SLEEP_NON_BUSY** - CPU リソースを消費せずに実行を遅らせます。(17,000 マイクロ秒以下では無関係です。busy sleep より不正確です。)

戻り値

正常終了した場合、WD_STATUS_SUCCESS (0) を返します。その他の場合、エラーコードを返します [A.7]。

注釈

- 使用例: 応答の遅いハードウェアへのアクセス。

例

```
WD_Sleep slp;

BZERO(slp);
slp.dwMicroSeconds = 200;
WD_Sleep(hWD, &slp);
```

A.5.9 WD_License()

目的

- ライセンス文字列を WinDriver カーネル モジュールに転送して指定したライセンス文字列のライセンスの種類に関する情報を返します。

注意: WDU USB APIs [A.1] を使用する場合、WinDriver ライセンスの登録は WDU_Init() [A.3.1] への呼び出しで実行されるため、コードから直接 WD_License() を呼び出す必要はありません。

プロトタイプ

```
DWORD WD_License(
    HANDLE hWD,
    WD_LICENSE *pLicense);
```

パラメータ

名前	型	入出力
➢ hWD	HANDLE	入力
➢ pLicense	WD_LICENSE*	
☐ cLicense	CHAR[]	入力
☐ dwLicense	DWORD	出力
☐ dwLicense2	DWORD	出力

説明

名前	説明
hWD	`WD_Open()` [A.5.2] から渡された WinDriver のカーネル モード ドライバへのハンドル。
pLicense	WinDriver ライセンス情報の構造体へのポインタ。
cLicense	WinDriver カーネル モジュールへ転送されるライセンス文字列を含むためのバッファ。空の文字列が転送された場合、WinDriver カーネル モジュールはパラメータ `dwLicense` にライセンスの種類を返します。
dwLicense	ライセンス文字列 (`cLicnese`) が許可したライセンスの種類を返します。戻り値は、**windrvr.h** で enum として定義したライセンスのフラグのビット マスクです。0 は無効なライセンスを示します。必要な場合、ライセンスの種類を決定する追加のフラグは、**`dwLicense2`** に返されます。
dwLicense2	`dwLicense` が対応するすべての情報を持てない場合 (その他の場合は 0)、ライセンスの種類の決定に追加のフラグを返します。

戻り値

正常終了した場合、WD_STATUS_SUCCESS (0) を返します。その他の場合、エラーコードを返します [A.7]。

注釈

- 登録版を使用する際に、コードからライセンスを登録するには、`WD_Open()` [A.5.2] 以外の WinDriver API関数を呼ぶ前に、この関数を呼ぶ必要があります。

例

使用例: アプリケーションに登録用のルーチンを追加する。

```
DWORD RegisterWinDriver()
{
    HANDLE hWD;
    WD_LICENSE lic;
    DWORD dwStatus = WD_INVALID_HANDLE;

    hWD = WD_Open();
    if (hWD!=INVALID_HANDLE_VALUE)
    {
        BZERO(lic);
        /* Replace the following string with your license string: */
        strcpy(lic.cLicense, "12345abcde12345.CompanyName");
        dwStatus = WD_License(hWD, &lic);
        WD_Close(hWD);
    }

    return dwStatus;
}
```

A.6 ユーザーモード ユーティリティ関数

このセクションでは、さまざまな作業を実装するのに役に立つユーザーモード ユーティリティ関数を説明します。これらのユーティリティ関数をマルチ プラットフォーム (WinDriver がサポートしてるすべてのオペレーティング システム) で実装します。

A.6.1 Stat2Str()

目的

- ステータス コードに対応するステータス文字列を取得します。

プロトタイプ

```
const char *Stat2Str(DWORD dwStatus);
```

パラメータ

名前	型	入出力
➢ dwStatus	DWORD	入力

説明

名前	説明
dwStatus	ステータス コードの番号。

戻り値

指定したステータス コードの番号に対応するステータスの説明 (文字列) を返します。

注釈

ステータス コードと文字列の一覧は、セクション A.7 を参照してください。

A.6.2 get_os_type()

目的

- オペレーティング システムの種類を取得します。

プロトタイプ

```
OS_TYPE get_os_type(void);
```

戻り値

オペレーティング システムの種類を返します。
オペレーティング システムの種類を検出できなかった場合、OS_CAN_NOT_DETECT を返します。

A.6.3 ThreadStart()

目的

- スレッドを作成します。

プロトタイプ

```
DWORD ThreadStart(
    HANDLE *phThread,
    HANDLER_FUNC pFunc,
    void *pData);
```

パラメータ

名前	型	入出力
➢ phThread	HANDLE*	出力
➢ pFunc	HANDLER_FUNC	入力
➢ pData	VOID*	入力

説明

名前	説明
phThread	生成したスレッドへのハンドルを返します。
pFunc	新しいスレッドを実行する際のコードの開始アドレス。
pData	新しいスレッドへ渡されるデータへのポインタ。

戻り値

成功した場合、WD_STATUS_SUCCESS (0) を返します。その他の場合、対応するエラーコードを返します [A.7]。

A.6.4 ThreadWait()

目的

- 終了するスレッドを待機します。

プロトタイプ

```
void ThreadWait(HANDLE hThread);
```

パラメータ

名前	型	入出力
➢ hThread	HANDLE	入力

説明

名前	説明
hThread	終了するのを待たれているスレッドへのハンドル。

戻り値

なし。

A.6.5 OsEventCreate()

目的

- イベント オブジェクトを生成します。

プロトタイプ

```
DWORD OsEventCreate(HANDLE *phOsEvent);
```

パラメータ

名前	型	入出力
➢ phOsEvent	HANDLE*	出力

説明

名前	説明
phOsEvent	新しく生成したイベント オブジェクトへのハンドルを受信する変数へのポインタ。

戻り値

成功した場合、WD_STATUS_SUCCESS (0) を返します。その他の場合、対応するエラーコードを返します[A.7]。

A.6.6 OsEventClose()

目的

- イベント オブジェクトへのハンドルを閉じます。

プロトタイプ

```
void OsEventClose(HANDLE hOsEvent);
```

パラメータ

名前	型	入出力
➢ hOsEvent	HANDLE	入力

説明

名前	説明
hOsEvent	閉じられるイベント オブジェクトへのハンドル。

戻り値

なし。

A.6.7 OsEventWait()

目的

- 指定したイベント オブジェクトがシグナル状態になるかまたはタイムアウトになるまで待機します。

プロトタイプ

```
DWORD OsEventWait(
    HANDLE hOsEvent,
    DWORD dwSecTimeout);
```

パラメータ

名前	型	入出力
➢ hOsEvent	HANDLE	入力
➢ dwSecTimeout	DWORD	入力

説明

名前	説明
hOsEvent	イベント オブジェクトへのハンドル。
dwSecTimeout	イベントのタイムアウト インターバル (秒単位)。 ゼロ (0) に設定すると、無限に待ちます。

戻り値

成功した場合、WD_STATUS_SUCCESS (0) を返します。その他の場合、対応するエラーコードを返します [A.7]。

A.6.8 OsEventSignal()

目的

- 指定したイベント オブジェクトをシグナル状態に設定します。

プロトタイプ

```
DWORD OsEventSignal(HANDLE hOsEvent);
```

パラメータ

名前	型	入出力
➢ hOsEvent	HANDLE	入力

説明

名前	説明
hOsEvent	イベント オブジェクトへのハンドル。

戻り値

成功した場合、WD_STATUS_SUCCESS (0) を返します。その他の場合、対応するエラーコードを返します [A.7]。

A.6.9 OsEventReset()

目的

- 指定したイベント オブジェクトを非シグナル状態に設定します。

プロトタイプ

```
DWORD OsEventReset(HANDLE hOsEvent);
```

パラメータ

名前	型	入出力
➢ hOsEvent	HANDLE	入力

説明

名前	説明
hOsEvent	イベント オブジェクトへのハンドル。

戻り値

成功した場合、WD_STATUS_SUCCESS (0) を返します。その他の場合、対応するエラーコードを返します[A.7]。

A.6.10 OsMutexCreate()

目的

- mutex オブジェクトを生成します。

プロトタイプ

```
DWORD OsMutexCreate(HANDLE *phOsMutex);
```

パラメータ

名前	型	入出力
➢ phOsMutex	HANDLE*	出力

説明

名前	説明
phOsMutex	新たに生成した mutex オブジェクトへのハンドルを受信する変数へのポインタ。

戻り値

成功した場合、WD_STATUS_SUCCESS (0) を返します。その他の場合、対応するエラーコードを返します [A.7]。

A.6.11 OsMutexClose()

目的

- mutex オブジェクトへのハンドルを閉じます。

プロトタイプ

```
void OsMutexClose(HANDLE hOsMutex);
```

パラメータ

名前	型	入出力
➢ hOsMutex	HANDLE	入力

説明

名前	説明
hOsMutex	閉じられる mutex オブジェクトへのハンドル。

戻り値

なし。

A.6.12 OsMutexLock()

目的

- 指定した mutex オブジェクトをロックします。

プロトタイプ

```
DWORD OsMutexLock(HANDLE hOsMutex);
```

パラメータ

名前	型	入出力
➢ hOsMutex	HANDLE	入力

説明

名前	説明
hOsMutex	ロックされる mutex オブジェクトへのハンドル。

戻り値

成功した場合、WD_STATUS_SUCCESS (0) を返します。その他の場合、対応するエラーコードを返します [A.7]。

A.6.13 OsMutexUnlock()

目的

- ロックした mutex オブジェクトを解放 (アンロック) します。

プロトタイプ

```
DWORD OsMutexUnlock(HANDLE hOsMutex);
```

パラメータ

名前	型	入出力
➢ hOsMutex	HANDLE	入力

説明

名前	説明
hOsMutex	アンロックされる mutex オブジェクトへのハンドル。

戻り値

成功した場合、WD_STATUS_SUCCESS (0) を返します。その他の場合、対応するエラーコードを返します [A.7]。

A.6.14 PrintDbgMessage()

目的

- Debug Monitor へデバッグ メッセージを送信します。

プロトタイプ

```
void PrintDbgMessage(
    DWORD dwLevel,
    DWORD dwSection,
    const char *format
    ...);
```

パラメータ

名前	型	入出力
➢ dwLevel	DWORD	入力
➢ dwSection	DWORD	入力
➢ format	const char*	入力
➢ argument		入力

説明

名前	説明
dwLevel	Debug Monitor (データを宣言します) のレベルを指定します。0 の場合、D_ERROR を宣言します。 詳細は **windrvr.h** の DEBUG_LEVEL を参照してください。
dwSection	Debug Monitor (データを宣言します) のレベルを指定します。0 の場合、S_MISC を宣言します。 詳細は **windrvr.h** の DEBUG_SECTION を参照してください。
format	書式を管理する文字列
argument	オプション引数、最大 256 バイト。

戻り値

なし。

A.6.15 WD_LogStart()

目的

- ログ ファイルを開きます。

プロトタイプ

```
DWORD WD_LogStart(
    const char *sFileName,
    const char *sMode);
```

パラメータ

名前	型	入出力
➢ sFileName	const char*	入力
➢ sMode	const char*	入力

説明

名前	説明
sFileName	開くログ ファイル名。
sMode	アクセスの種類。 たとえば、NULL または **w** の場合、書き込み用の空のファイルを開きます。指定したファイルが存在する場合、その内容を壊します。 **a** の場合、ファイルの終わりに書き込む (append: 追加する) ように開きます。

戻り値

成功した場合、WD_STATUS_SUCCESS (0) を返します。その他の場合、対応するエラーコードを返します [A.7]。

注釈

- ログ ファイルを開くと、すべての API 呼び出しをこのファイルに記録します。
  `WD_LogAdd()` [A.6.17] を呼んで、ログ ファイルへ出力内容を追加します。

A.6.16 WD_LogStop()

目的

- ログ ファイルを閉じます。

プロトタイプ

```
VOID WD_LogStop(void);
```

戻り値

なし。

A.6.17 WD_LogAdd()

目的

- ログ ファイルに出力内容を追加します。

プロトタイプ

```
VOID DLLCALLCONV WD_LogAdd(
    const char *sFormat
    ...);
```

パラメータ

名前	型	入出力
➢ sFormat	const char*	入力
➢ argument		入力

説明

名前	説明
sFormat	書式を管理する文字列。
argument	書式文字列用のオプション引数

戻り値

成功した場合、WD_STATUS_SUCCESS (0) を返します。その他の場合、対応するエラーコードを返します [A.7]。

A.7 WinDriver ステータス コード

A.7.1 はじめに

多くの WinDriver 関数はステータス コードを返します。正常終了した場合は 0 (`WD_STATUS_SUCCESS`) を、異常終了した場合は 0 以外の値を返します。
指定したステータス コードの意味を調べるのに `Stat2Str()` を使用します。ステータス コードとその説明を以下に示します。

A.7.2 WinDriver が返すステータス コード

ステータス コード	説明
WD_STATUS_SUCCESS	Success (成功)
WD_STATUS_INVALID_WD_HANDLE	Invalid WinDriver handle (無効な WinDriver のハンドル)
WD_WINDRIVER_STATUS_ERROR	Error (エラー)
WD_INVALID_HANDLE	Invalid handle (無効なハンドル)
WD_INVALID_PIPE_NUMBER	Invalid pipe number (無効なパイプ番号)
WD_READ_WRITE_CONFLICT	Conflict between read and write operations (読み込みと書き込み処理の競合)

WD_ZERO_PACKET_SIZE	Packet size is zero (パケット サイズが 0)
WD_INSUFFICIENT_RESOURCES	Insufficient resources (不十分なリソース)
WD_UNKNOWN_PIPE_TYPE	Unknown pipe type (未知のパイプの種類)
WD_SYSTEM_INTERNAL_ERROR	Internal system error (内部システム エラー)
WD_DATA_MISMATCH	Data mismatch (データの不整合)
WD_NO_LICENSE	No valid license (有効なライセンスがありません)
WD_NOT_IMPLEMENTED	Function not implemented (実装されていない関数)
WD_FAILED_ENABLING_INTERRUPT	Failed enabling interrupt (失敗した有効な割り込み)
WD_INTERRUPT_NOT_ENABLED	Interrupt not enabled (有効ではない割り込み)
WD_RESOURCE_OVERLAP	Resource overlap (リソースの重複)
WD_DEVICE_NOT_FOUND	Device not found (デバイスが見つかりません)
WD_WRONG_UNIQUE_ID	Wrong unique ID (間違った任意の ID)
WD_OPERATION_ALREADY_DONE	Operation already done (処理済の処理)
WD_USB_DESCRIPTOR_ERROR	USB descriptor error (USB 記述子のエラー)
WD_SET_CONFIGURATION_FAILED	Set configuration operation failed (失敗した設定処理を設定)
WD_CANT_OBTAIN_PDO	Cannot obtain PDO (PDO の取得ができません)
WD_TIME_OUT_EXPIRED	Timeout expired (期限切れのタイムアウト)
WD_IRP_CANCELED	IRP operation cancelled (キャンセルされた IRP 処理)
WD_FAILED_USER_MAPPING	Failed to map in user space (ユーザー スペースへの割り当ての失敗)
WD_FAILED_KERNEL_MAPPING	Failed to map in kernel space (カーネル スペースへの割り当ての失敗)
WD_NO_RESOURCES_ON_DEVICE	No resources on the device (デバイスにリソースがありません)
WD_NO_EVENTS	No events (イベントがありません)
WD_INVALID_PARAMETER	Invalid parameter (不正な引数)
WD_INCORRECT_VERSION	Incorrect WinDriver version installed (不正な WinDriver のバージョンがインストールされています)
WD_TRY_AGAIN	Try again (再試行)
WD_INVALID_IOCTL	Received an invalid IOCTL (不正な IOCTL を受信しました)

A.7.3 USBD が返すステータス コード

以下に、WinDriver ステータス コードは USB スタックドライバから返される USBD_XXX ステータス コードに対応します。

ステータス コード	説明
USBD ステータスの種類	
WD_USBD_STATUS_SUCCESS	USBD: Success (成功)
WD_USBD_STATUS_PENDING	USBD: Operation pending (処理待ち)
WD_USBD_STATUS_ERROR	USBD: Error (エラー)
WD_USBD_STATUS_HALTED	USBD: Halted (停止)
USBD ステータス コード (**注意:** 上記のステータスの種類とエラー コードで構成されています。たとえば、0XXYYYYYYL の X はステータスの種類、Y はエラーコード。同じエラー コードが、他のステータスの種類で現れる場合があります。)	
HC (ホスト コントローラ) ステータス コード (注意: WD_USBD_STATUS_HALTED のステータスの種類を使用します。)	
WD_USBD_STATUS_CRC	HC status: CRC
WD_USBD_STATUS_BTSTUFF	HC status: Bit stuffing (ビット スタッフ)
WD_USBD_STATUS_DATA_TOGGLE_MISMATCH	HC status: Data toggle mismatch (データトグルの不整合)
WD_USBD_STATUS_STALL_PID	HC status: PID stall (PID 停止)
WD_USBD_STATUS_DEV_NOT_RESPONDING	HC status: Device not responding (デバイスからの応答がありません)
WD_USBD_STATUS_PID_CHECK_FAILURE	HC status: PID check failed (PID のチェック失敗)
WD_USBD_STATUS_UNEXPECTED_PID	HC status: Unexpected PID (予期せぬ PID)
WD_USBD_STATUS_DATA_OVERRUN	HC status: Data overrun (データの超過)
WD_USBD_STATUS_DATA_UNDERRUN	HC status: Data underrun (データの不足)
WD_USBD_STATUS_RESERVED1	HC status: Reserved1 (予約 1)
WD_USBD_STATUS_RESERVED2	HC status: Reserved2 (予約 2)
WD_USBD_STATUS_BUFFER_OVERRUN	HC status: Buffer overrun (バッファの超過)
WD_USBD_STATUS_BUFFER_UNDERRUN	HC status: Buffer Underrun (バファの不足)
WD_USBD_STATUS_NOT_ACCESSED	HC status: Not accessed (未アクセス)
WD_USBD_STATUS_FIFO	HC status: FIFO
Windows の場合のみ:	
WD_USBD_STATUS_XACT_ERROR	HC status: The host controller has set the Transaction Error (XactErr) bit in the transfer descriptor's status field (転送記述子のステータス フィールドに、ホスト コントローラが Transaction Error (XacctErr) ビットを設定しまし

	た)
WD_USBD_STATUS_BABBLE_DETECTED	HC status: Babble detected (バブルを検出しました)
WD_USBD_STATUS_DATA_BUFFER_ERROR	HC status: Data buffer error (データ バッファ エラー)
Windows CE の場合のみ:	
WD_USBD_STATUS_NOT_COMPLETE	USBD: Transfer not completed (転送が終了しませんでした)
WD_USBD_STATUS_CLIENT_BUFFER	USBD: Cannot write to buffer (バッファへ書き込みできません)
すべてのプラットフォームの場合のみ:	
WD_USBD_STATUS_CANCELED	USBD: Transfer cancelled (転送がキャンセルされました)
転送が停止されたエンドポイントへ送信された場合、HCD (ホスト コントロール ドライバ)が返します:	
WD_USBD_STATUS_ENDPOINT_HALTED	HCD: Transfer submitted to stalled endpoint (停止されたエンドポイントへ送信された転送)
ソフトウェア ステータス コード (**注意:** エラー ビットのみ設定):	
WD_USBD_STATUS_NO_MEMORY	USBD: Out of memory (メモリーがありません)
WD_USBD_STATUS_INVALID_URB_FUNCTION	USBD: Invalid URB function (無効な URB 関数)
WD_USBD_STATUS_INVALID_PARAMETER	USBD: Invalid parameter (無効な引数)
クライアントのドライバが、未解決の転送の設定またはエンドポイント / インターフェイスを閉じる場合に返されます。	
WD_USBD_STATUS_ERROR_BUSY	USBD: Attempted to close endpoint/interface/configuration with outstanding transfer (未解決の転送のエンドポイント / インターフェイス / 設定を閉じます)
URB の要求を完了できない場合に USBD に返されます。基本的に、これは特定のエラー コードを持つ URB のステータス項目 (IRQ が完成時) に返されます。IRQ ステータスは、WinDriver の Monitor Debug メッセージ (wddebug_gui) ツールで表示されます	
WD_USBD_STATUS_REQUEST_FAILED	USBD: URB request failed (URB の要求失敗)
WD_USBD_STATUS_INVALID_PIPE_HANDLE	USBD: Invalid pipe handle (無効なパイプ ハンドル)
要求したエンドポイントを開くのに十分な帯域幅が無い場合に返されます:	
WD_USBD_STATUS_NO_BANDWIDTH	USBD: Not enough bandwidth for endpoint (エンドポイントに十分な帯域幅がありません)
汎用 HC (ホスト コントローラ)エラー:	
WD_USBD_STATUS_INTERNAL_HC_ERROR	USBD: Host controller error (ホスト コントロール エラー)
短いパケットが転送を終了したときに返されます。たとえば、USBD_SHORT_TRANSFER_OK ビットが設定されていない場合:	

WD_USBD_STATUS_ERROR_SHORT_TRANSFER	USBD: Transfer terminated with short packet (短いパケットで終了された転送)
要求した開始フレームが、USB フレームの USBD_ISO_START_FRAME_RANGE 内に無い場合、返されます。(**注意:** 停止ビットが設定されます):	
WD_USBD_STATUS_BAD_START_FRAME	USBD: Start frame outside range (開始フレームが範囲外です)
等時性転送のすべてのパケットがエラーを起こして終了した場合、HCD (ホスト コントローラ デバイス)が返します:	
WD_USBD_STATUS_ISOCH_REQUEST_FAILED	HCD: Isochronous transfer completed with error (エラーを起こして終了した等時性転送)
指定した HC (ホスト コントローラ)のフレーム長コントロールが既に他のドライバに使用されている場合、USBD が返します:	
WD_USBD_STATUS_FRAME_CONTROL_OWNED	USBD: Frame length control already taken (既に使用されているフレーム長コントロール)
呼出し元が自分自身のフレーム長コントロールを持っておらず、HC フレーム長を解放または修正する場合に、USBD が返します:	
WD_USBD_STATUS_FRAME_CONTROL_NOT_OWNED	USBD: Attempted operation on frame length control not owned by caller (呼出し元が持っていないフレーム長コントロールの処理)
USB 2.0 用に追加したエラー コードです (Windows の場合のみ) :	
WD_USBD_STATUS_NOT_SUPPORTED	USBD: API not supported/implemented (サポートされていない / 実装されない API)
WD_USBD_STATUS_INAVLID_CONFIGURATION_DESCRIPTOR	USBD: Invalid configuration descriptor (不正な設定記述子)
WD_USBD_STATUS_INSUFFICIENT_RESOURCES	USBD: Insufficient resources (リソースが足りません)
WD_USBD_STATUS_SET_CONFIG_FAILED	USBD: Set configuration failed (設定に失敗しました)
WD_USBD_STATUS_BUFFER_TOO_SMALL	USBD: Buffer too small (バッファが小さ過ぎます)
WD_USBD_STATUS_INTERFACE_NOT_FOUND	USBD: Interface not found (インターフェイスが見つかりません)
WD_USBD_STATUS_INAVLID_PIPE_FLAGS	USBD: Invalid pipe flags (不正なパイプ フラグ)
WD_USBD_STATUS_TIMEOUT	USBD: Timeout (タイムアウト)
WD_USBD_STATUS_DEVICE_GONE	USBD: Device gone (デバイスがありません)
WD_USBD_STATUS_STATUS_NOT_MAPPED	USBD: Status not mapped (ステータスがマップされていません)
USBD から返る拡張等時性エラー コード。 これらのエラーは、等時性転送のパケット ステータス フィールドに表示します:	
WD_USBD_STATUS_ISO_NOT_ACCESSED_BY_HW	USBD: The controller did not access the TD associated with this packet (コントローラが、このパケットに関連付けた TD にアクセスしませんでした)

WD_USBD_STATUS_ISO_TD_ERROR	USBD: Controller reported an error in the TD (コントローラが TD でエラーを報告しました)
WD_USBD_STATUS_ISO_NA_LATE_USBPORT	USBD: The packet was submitted in time by the client but failed to reach the miniport in time (クライアントがパケットを送信しましたが、ミニポートに届きませんでした)
WD_USBD_STATUS_ISO_NOT_ACCESSED_LATE	USBD: The packet was not sent because the client submitted it too late to transmit (クライアントの転送が遅くて、パケットが届きませんでした)

付録 B
USB Device - Cypress EZ-USB FX2LP CY7C68013A API のリファレンス

B.1 ファームウェア ライブラリの API

このセクションでは、Cypress EZ-USB FX2LP CY7C68013A 開発ボード用の WinDriver USB Device ファームウェア ライブラリの API について説明します。このセクションで説明する関数および一般的な型と定義は、**FX2LP\include\wdf_cypress_lib .h** ヘッダー ファイルで宣言、定義され、DriverWizard で生成された **wdf_cypress_lib .c** ファイル (登録ユーザーの場合)、または **FX2LP\ lib\ wdf_cypress_fx2lp_eval.lib** 評価版ファームウェア ライブラリ (評価版ユーザー) で実装されます。詳細は、WinDriver ユーザーズ ガイドのセクション 16.3.5 を参照してください。

注意

登録ユーザーはライブラリのソース コードを変更することができます。コードを変更する場合、USB 規格および開発するボードのハードウェア要件を満たしているかどうかを確認してください (WinDriver ユーザーズ ガイドのセクション 16.4.3 を参照)。

B.1.1 ファームウェア ライブラリの型

このセクションで説明される API は **FX2LP\wdf_cypress_lib.h** で定義されています。

B.1.1.1 EP_DIR 列挙型

エンドポイント方向の列挙型:

列挙値	説明
DIR_OUT	OUT 方向 (ホストからデバイスへの書き込み)
DIR_IN	IN 方向 (デバイスからホストへの読み取り)

B.1.1.2 EP_TYPE 列挙型

エンドポイント タイプの列挙型。
エンドポイント タイプは、エンドポイント上で実行される転送の種類 (バルク転送、割り込み転送、または等時性 (アイソクロナス) 転送) を割り出します。

列挙値	説明
ISOCHRONOUS	等時性エンドポイント
BULK	バルク エンドポイント
INTERRUPT	割り込みエンドポイント

B.1.1.3 EP_BUFFERING 列挙型

エンドポイント バッファリングの種類の列挙型:

列挙値	説明
DOUBLE_BUFFERING	ダブル バッファリング
TRIPLE_BUFFERING	トリプル バッファリング
QUAD_BUFFERING	クアドラプル (四重) バッファリング

B.1.2 ファームウェア ライブラリの関数

このセクションで説明される関数は **FX2LP\wdf_cypress_lib.h** で定義されています。

B.1.2.1 WDF_EP1INConfig() / WDF_EP1OUTConfig()

目的

- IN 転送 (`WDF_EP1INConfig()`) または OUT 転送 (`WDF_EPOUTConfig()`) へエンドポイント 1 を設定します。

プロトタイプ

```
void WDF_EP1INConfig(EP_TYPE type);
void WDF_EP1OUTConfig(EP_TYPE type);
```

パラメータ

名前	型	入出力
➢ type	EP_TYPE	入力

説明

名前	説明
type	エンドポイントの転送の型 [B.1.1.2]

戻り値

なし。

B.1.2.2 WDF_EP2Config / WDF_EP6Config()

注意

WDF_EP2Config() および **WDF_EP6Config()** のプロトタイプおよび説明は、エンドポイント番号を除いて同じです。以下のセクションではエンドポイント 2 について説明していますが、"2" を "6" に変換すると WDF_EP6Config() の説明になります。

目的

- エンドポント 2 を設定します。

プロトタイプ

```
void WDF_EP2Config(
    EP_DIR dir,
    EP_TYPE type,
    EP_BUFFERING buffering,
    int size,
    int nPacketPerMF);
```

パラメータ

名前	型	入出力
➢ dir	EP_DIR	入力
➢ type	EP_TYPE	入力
➢ buffering	EP_BUFFERING	入力
➢ size	int	入力
➢ nPacketPerMF	int	入力

説明

名前	説明
dir	エンドポイントの方向 [B.1.1.1]
type	エンドポイントの転送の種類 [B.1.1.2]
buffering	エンドポイントのバッファリングの種類 [B.1.1.3]
size	エンドポイントの FIFO バッファのバイト単位でのサイズ
nPacketPerMF	マイクロフレームごとのパケット数

戻り値

なし。

B.1.2.3 WDF_EP4Config / WDF_EP8Config()

注意

`WDF_EP4Config()` および **`WDF_EP8Config()`** のプロトタイプおよび説明は、エンドポイント番号を除いて同じです。以下のセクションではエンドポイント 4 について説明していますが、"4" を "8" に変換するだけで `WDF_EP8Config()` の説明になります。

目的

- エンドポント 4 を設定します。

プロトタイプ

```
void WDF_EP4Config(
    EP_DIR dir,
    EP_TYPE type);
```

パラメータ

名前	型	入出力
➢ dir	EP_DIR	入力
➢ type	EP_TYPE	入力

説明

名前	説明
dir	エンドポイントの方向 [B.1.1.1]
type	エンドポイントの転送の種類 [B.1.1.2]

戻り値

なし。

B.1.2.4 WDF_FIFOReset()

目的

- エンドポイントの FIFO (First In First Out) バッファをデフォルト状態に戻します。

プロトタイプ

```
void WDF_FIFOReset(int ep);
```

パラメータ

名前	型	入出力
➢ ep	int	入力

説明

名前	説明
ep	エンドポイント番号 (アドレス)

戻り値

なし。

B.1.2.5 WDF_SkipOutPacket()

目的

- 受け取った OUT パケットを無視するシグナルおよびエンドポイントの FIFO (First In First Out) バッファ。

プロトタイプ

```
void WDF_SkipOutPacket(int ep);
```

パラメータ

名前	型	入出力
➢ ep	int	入力

説明

名前	説明
ep	エンドポイント番号 (アドレス)

戻り値

なし。

B.1.2.6 WDF_FIFOWrite()

目的

- エンドポイントの FIFO (First In First Out) バッファへデータを書き込みます。
  この関数は `WDF_SetEPByteCount()` [B.1.2.10] を呼び出した後に呼び出します。

プロトタイプ

```
void WDF_FIFOWrite(
    int ep,
    BYTE buf,
    int size);
```

パラメータ

名前	型	入出力
➢ ep	int	入力
➢ buf	BYTE[ ]	入力
➢ size	int	入力

説明

名前	説明
ep	エンドポイント番号 (アドレス
buf	書き込むデータ バッファ
size	書き込むバイト数

戻り値

なし。

B.1.2.7 WDF_FIFORead()

目的

- エンドポイントの FIFO (First In First Out) バッファからデータを読み取ります。
  読み取るバイト数を割り出すために、この関数は `WDF_GetEPByteCount()` [B.1.2.11] を呼び出す前に呼び出す必要があります。

プロトタイプ

```
void WDF_FIFORead(
    int ep,
    BYTE buf,
    int size);
```

パラメータ

名前	型	入出力
➢ ep	int	入力
➢ buf	BYTE[ ]	出力
➢ size	int	入力

説明

名前	説明
ep	エンドポイント番号 (アドレス)
buf	読み取るデータを保持するバッファ
size	FIFO バッファから読み取るバイト数

戻り値

なし。

B.1.2.8 WDF_FIFOFull()

目的

- エンドポイントの FIFO (First In First Out) バッファがフルかどうかを確認します。

プロトタイプ

```
BOOL WDF_FIFOFull(int ep);
```

パラメータ

名前	型	入出力
➢ ep	int	入力

説明

名前	説明
ep	エンドポイント番号 (アドレス)

戻り値

エンドポイントの FIFO (First In First Out) バッファがフルの場合、TRUE を返します。フルでない場合、FALSE を返します。

B.1.2.9 WDF_FIFOEmpty()

目的

- エンドポイントの FIFO (First In First Out) バッファが空でないかを確認します。

プロトタイプ

```
BOOL WDF_FIFOEmpty(int ep);
```

パラメータ

名前	型	入出力
➢ ep	int	入力

説明

名前	説明
ep	エンドポイント番号 (アドレス)

戻り値

エンドポイントの FIFO (First In First Out) バッファが空の場合、TRUE を返します。空でない場合、FALSE を返します。

B.1.2.10 WDF_SetEPByteCount()

目的

- FIFO (First In First Out) バッファのバイト カウントを設定します。
  エンドポイント FIFO バッファをホストへ転送するデータでアップデートするために、この関数は `WDF_FIFOWrite()` [B.1.2.6] を呼び出す前に呼び出します。

プロトタイプ

```
void WDF_SetEPByteCount(
    int ep,
    WORD bytes_count);
```

パラメータ

名前	型	入出力
➢ ep	int	入力
➢ bytes_count	WORD	入力

説明

名前	説明
ep	エンドポイント番号 (アドレス)
bytes_count	設定するバイト カウント

戻り値

なし。

B.1.2.11 WDF_GetEPByteCount()

目的

- FIFO (First In First Out) バッファの現在のバイト カウントを取得します。
  読み取るバイトの量を割り出すために、この関数は `WDF_FIFORead()` [B.1.2.7] を呼び出す前に呼び出す必要があります。

プロトタイプ

```
WORD WDF_GetEPByteCount(int ep);
```

パラメータ

名前	型	入出力
➢ ep	int	入力

説明

名前	説明
ep	エンドポイント番号 (アドレス)

戻り値

エンドポイントの FIFO (First In First Out)のバイト カウントを返します。

B.1.2.12 WDF_I2CInit()

目的

- I2C バスを初期化します。

プロトタイプ

```
void WDF_I2CInit(void);
```

戻り値

なし。

B.1.2.13 WDF_SetDigitLed()

目的

- 指定されたデジットで開発ボードのデジット LED を表示します。

プロトタイプ

```
void WDF_SetDigitLed(int digit);
```

パラメータ

名前	型	入出力
➢ digit	int	入力

説明

名前	説明
digit	表示するデジット

戻り値

なし。

B.1.2.14 WDF_I2CWrite()

目的

- I2C バス上の指定されたアドレスにデータを書き込みます。

プロトタイプ

```
BOOL WDF_I2CWrite(
    BYTE addr,
    BYTE len,
    BYTE xdata *dat);
```

パラメータ

名前	型	入出力
➢ addr	BYTE	入力
➢ len	BYTE	入力
➢ dat	xdata*	入力

説明

名前	説明
addr	書き込むアドレス
len	書き込むバイト数
dat	書き込むデータを保持するバッファへのポインタ

戻り値

正常に書き込み処理が行われた場合は TRUE を返します。そうでない場合は、FALSE を返します。

B.1.2.15 WDF_I2CRead()

目的

- I2C バス上の指定されたアドレスからデータを読み取ります。

プロトタイプ

```
BOOL WDF_I2CRead(
    BYTE addr,
    BYTE len,
    BYTE xdata *dat);
```

パラメータ

名前	型	入出力
➢ addr	BYTE	入力
➢ len	BYTE	入力
➢ dat	xdata*	出力

説明

名前	説明
addr	読み取るアドレス
len	読み取るバイト数
Dat	読み取るデータを保持するバッファへのポインタ

戻り値

正常に読み取り処理が行われた場合は TRUE を返します。そうでない場合は、FALSE を返します。

B.1.2.16 WDF_I2CWaitForEEPROMWrite()

目的

- 指定された I2C バス アドレスへの現在の書き込み処理が終了するのを待機します。

プロトタイプ

```
void WDF_I2CWaitForEEPROMWrite(BYTE addr);
```

パラメータ

名前	型	入出力
➢ addr	BYTE	入力

説明

名前	説明
addr	待機する I2C バス アドレス

戻り値

なし。

B.1.2.17 WDF_I2CGetStatus()

目的

- I2C バスの現在のステータスを取得します。

プロトタイプ

```
int WDF_I2CGetStatus(void);
```

戻り値

I2C バスのステータスを返します。

B.1.2.18 WDF_I2CClearStatus()

目的

- I2C バスの エラー/NAK ステータスを消去します。

プロトタイプ

```
void WDF_I2CClearStatus(void);
```

戻り値

なし。

B.2 DriverWizard で生成されたファームウェアの API

このセクションでは、DriverWizard で生成された Cypress EZ-USB FX2LP CY7C68013A 開発ボード用の WinDriver USB Device ファームウェア ライブラリの API について説明します。このセクションで説明されている関数は **FX2LP\ include\ periph.h** ヘッダー ファイルで宣言され、ウィザードで定義されたデバイスの設定情報に従って DriverWizard で生成された **periph.c** ファイルで実装されます。

このファームウェアのエントリー ポイント(**main.c** の `main()` (ソース コードは登録ユーザーのみに提供されます)) は、周辺のデバイスと通信するために **periph.h** で宣言された `WDF_xxx()` 関数 (**periph.c** で実装される) を呼び出す **Task Dispatcher (タスク ディスパッチャー)** を実装します。

注意

生成されたコードを変更する場合、USB 規格および開発するボードのハードウェア要件を満たしているかどうかを確認してください (WinDriver ユーザーズ ガイドのセクション 16.4.3 を参照)。

B.2.1 WDF_Init()

目的

- デバイスを初期化します。
  デバイスとの通信を有効にする必要な初期設定を実行するために、この関数は自動的にファームウェアの `main()` 関数から呼び出されます。

プロトタイプ

```
void WDF_Init(void);
```

戻り値

なし。

B.2.2 WDF_Poll()

目的

- FIFO データ用のデバイスをポーリングします。
  Task Dispatcher はデバイスが動作していない間、この関数を繰り返し呼び出します。

プロトタイプ

```
void WDF_Poll(void);
```

戻り値

なし。

B.2.3 WDF_Suspend()

目的

- この関数は、デバイスがサスペンド モードになる前に Task Dispatcher により呼び出されます。

プロトタイプ

```
BOOL WDF_Suspend(void);
```

戻り値

成功した場合、TRUE を返します。失敗した場合、FALSE を返します。

B.2.4 WDF_Resume()

目的

- この関数は、デバイスが活動を再開した後 Task Dispatcher により呼び出されます。

プロトタイプ

```
BOOL WDF_Resume(void);
```

戻り値

成功した場合、TRUE を返します。失敗した場合、FALSE を返します。

B.2.5 WDF_GetDescriptor()

目的

- この関数は、GET DESCRIPTOR コマンドを受け取ったとき Task Dispatcher により呼び出されます。

プロトタイプ

```
BOOL WDF_GetDescriptor(void);
```

戻り値

成功した場合、TRUE を返します。失敗した場合、FALSE を返します。

B.2.6 WDF_SetConfiguration()

目的

- この関数は、SET CONFIGURATION コマンドを受け取ったとき Task Dispatcher により呼び出されます。

プロトタイプ

```
BOOL WDF_SetConfiguration(BYTE config);
```

パラメータ

名前	型	入出力
➢ config	BYTE	入力

説明

名前	説明
config	設定する設定番号

戻り値

成功した場合、TRUE を返します。失敗した場合、FALSE を返します。

B.2.7 WDF_GetConfiguration()

目的

- この関数は、GET CONFIGURATION コマンドを受け取ったとき Task Dispatcher により呼び出されます。

プロトタイプ

```
BOOL WDF_GetConfiguration(void);
```

戻り値

成功した場合、TRUE を返します。失敗した場合、FALSE を返します。

B.2.8 WDF_SetInterface()

目的

- この関数は、SET INTERFACE コマンドを受け取ったとき Task Dispatcher により呼び出されます。

プロトタイプ

```
BOOL WDF_SetInterface(
    BYTE ifc,
    BYTE alt_set);
```

パラメータ

名前	型	入出力
➢ ifc	BYTE	入力
➢ alt_set	BYTE	入力

説明

名前	説明
ifc	設定するインターフェイス番号
alt_set	控えの設定番号

戻り値

成功した場合、TRUE を返します。失敗した場合、FALSE を返します。

B.2.9 WDF_GetInterface()

目的

- この関数は、GET INTERFACE コマンドを受け取ったとき Task Dispatcher により呼び出されます。

プロトタイプ

```
BOOL WDF_GetInterface(BYTE ifc);
```

パラメータ

名前	型	入出力
➢ ifc	BYTE	入力

説明

名前	説明
ifc	インターフェイス番号

戻り値

成功した場合、TRUE を返します。失敗した場合、FALSE を返します。

B.2.10 WDF_GetStatus()

目的

- この関数は、GET STATUS コマンドを受け取ったとき Task Dispatcher により呼び出されます。

プロトタイプ

```
BOOL WDF_GetStatus(void);
```

戻り値

成功した場合、TRUE を返します。失敗した場合、FALSE を返します。

B.2.11 WDF_ClearFeature()

目的

- この関数は、CLEAR FEATURE コマンドを受け取ったとき Task Dispatcher により呼び出されます。

プロトタイプ

```
BOOL WDF_ClearFeature(void);
```

戻り値

成功した場合、TRUE を返します。失敗した場合、FALSE を返します。

B.2.12 WDF_SetFeature()

目的

- この関数は、SET FEATURE コマンドを受け取ったとき Task Dispatcher により呼び出されます。

プロトタイプ

```
BOOL WDF_SetFeature(void);
```

戻り値

成功した場合、TRUE を返します。失敗した場合、FALSE を返します。

B.2.13 WDF_VendorCmnd()

目的

- この関数は、ベンダー特有のコマンドを受け取ったとき Task Dispatcher により呼び出されます。

プロトタイプ

```
BOOL WDF_VendorCmnd(void);
```

戻り値

成功した場合、TRUE を返します。失敗した場合、FALSE を返します。

付録 C
USB Device - Microchip PIC18F4550 API のリファレンス

C.1 ファームウェア ライブラリの API

このセクションでは、Microchip PIC18F4550 開発ボード用の WinDriver USB Device ファームウェア ライブラリの API について説明します。このセクションで説明する関数、マクロ、一般的な型と定義は、それぞれ **18F4550\ include\ wdf_microchip_lib .h**、**18F4550\ include\ types.h** および **18F4550\ include\ wdf_usb9.h** ヘッダー ファイルで宣言、定義され、DriverWizard で生成された **wdf_microchip_lib .c** ファイルおよび **wdf_usb9.c** ファイル (登録ユーザーの場合)、または **18F4550\ lib\ wdf_microchip_18f4550_eval.lib** 評価版ファームウェア ライブラリ (評価版ユーザー) で実装されます。詳細は、WinDriver ユーザーズ ガイドのセクション 16.3.5 を参照してください。

Microchip PIC18F4550 ボードの **Mass Storage** ファームウェア ライブラリには、このセクションで説明する標準ファームウェア ライブラリのファームウェア ファイルおよび API と同じものが含まれています。さらに、**Mass Storage** ライブラリでは、セクション [C.2] で説明する大容量記憶装置用の API も定義しています。

注意

登録ユーザーはライブラリのソース コードを変更することができます。コードを変更する場合、USB 規格および開発するボードのハードウェア要件を満たしているかどうかを確認してください(WinDriver ユーザーズ ガイドのセクション 16.4.3 を参照)。

C.1.1 ファームウェア ライブラリの型

このセクションで説明するデータ型は、**18F4550\ include\ types.h** ヘッダー ファイルで定義されています。

C.1.1.1 EP_DIR 列挙値

エンドポイント方向の列挙型:

列挙値	説明
OUT	OUT 方向 (ホストからデバイスへの書き込み)
IN	IN 方向 (デバイスからホストへの読み取り)

C.1.1.2 EP_TYPE 列挙型

エンドポイン タイプの列挙型。
エンドポイント タイプは、エンドポイント上で実行される転送の種類 (バルク転送、割り込み転送、または等時性 (アイソクロナス) 転送) を割り出します。

列挙値	説明
ISOCHRONOUS	等時性エンドポイント
BULK	バルク エンドポイント
INTERRUPT	割り込みエンドポイント
CONTROL	コントロール エンドポイント

C.1.1.3 WDF_TRIGGER_OPTIONS 列挙型

トリガー オプションの列挙型。

列挙値	説明
TRIGGER_NO_TOGGLE_DTS	データトグル同期ビット (DTS) を切り替えません
TRIGGER_IGNORE_DTS	DTS の値を無視します
TRIGGER_DAT0	DTS を DAT0 に切り替えます
TRIGGER_DAT1	DTS を DAT1 に切り替えます

C.1.1.4 BD_STAT 共有体

エンドポイント バッファ記述子のステータスの共有体型。

名前	型	説明
➢ _byte	byte	
➢	struct	
☐ BC8	bit field (1)	エンドポイントの最終転送バイトのビット 8
☐ BC9	bit field (1)	エンドポイントの最終転送バイトのビット 9 (MSB)
☐ BSTALL	bit field (1)	バッファのストールが有効です
☐ DTSEN	bit field (1)	データトグル同期化が有効です
☐ INCDIS	bit field (1)	アドレスのインクリメントが無効です
☐ KEN	bit field (1)	バッファ記述子の保持が有効です
☐ DTS	bit field (1)	データトグル同期化の値
☐ UOWN	bit field (1)	USB の所有権
➢	struct	
☐ BC8	bit field (1)	エンドポイントの最終転送バイトのビット 8
☐ BC9	bit field (1)	エンドポイントの最終転送バイトのビット 9 (MSB)
☐ PID0	bit field (1)	パケット識別子のビット 0
☐ PID1	bit field (1)	パケット識別子のビット 1
☐ PID2	bit field (1)	パケット識別子のビット 2

☐ PID3	bit field (1)	パケット識別子のビット 3
☐	bit field (1)	Reserved (予約)
☐ UOWN	bit field (1)	USB の所有権
➢	struct	
☐	bit field (2)	Reserved (予約)
☐ PID	bit field (4)	パケット識別子
☐	bit field (2)	Reserved (予約)

C.1.1.5 BDT 共有体

エンドポイント バッファ記述子テーブルの共有体。

名前	型	説明
➢	struct	
☐ Stat	BD_STAT	バッファ記述子のステータス [C.1.1.4]
☐ Cnt	byte	エンドポイントの最終転送バイト。バイトの MSB (最上位のビット) は、BD_STAT 共有体 (**Stat**) の BC8 フィールドおよび BC9 フィールドに格納されます。
☐ ADRL	byte	バッファ アドレスの下位
☐ ADRH	byte	バッファ アドレスの上位
➢	struct	
☐	bit field (8)	Reserved (予約)
☐	bit field (8)	Reserved (予約)
☐ ADR	byte*	バッファ アドレスへのポインタ

C.1.1.6 WORD 共有体

2 バイト アクセスの共有体型。

名前	型	説明
➢ _word	word	word 型
➢	struct	2 バイト配列の構造体
☐ v	byte[2]	2 バイトの配列

C.1.1.7 DWORD 共有体

4 バイト アクセスの共有体型。

名前	型	説明
➢ _dword	dword	dword 型
➢	struct	4 バイトの構造体
☐ byte0	byte	第 1 バイト
☐ byte1	byte	第 2 バイト
☐ byte2	byte	第 3 バイト
☐ byte3	byte	第 4 バイト
➢	struct	2 ワードの構造体
☐ word0	word	第 1 ワード
☐ word1	word	第 2 ワード
➢	struct	2 WORD の構造体
☐ Word0	WORD	第 1 WORD [C.1.1.6]
☐ Word1	WORD	第 2 WORD [C.1.1.6]
➢	struct	4 バイト配列の構造体
☐ v	byte[4]	4 バイトの配列

C.1.1.8 POINTER 共有体

ポインターの共有体型。

名前	型	説明
➢ _pFunc	typedef void(*pFunc)(void);	関数ポインタ
➢ bRam	byte*	RAM バイト ポインタ: 1 バイトのデータをポイントする 2 バイトのポインタ
➢ wRam	word*	RAM ワード ポインタ: 2 バイトのデータをポイントする 2 バイトのポインタ
➢ bRom	byte*	ROM バイト ポインタ。サイズは、コンパイラの設定により異なります。
➢ wRom	word*	ROM ワードポインタ。サイズは、コンパイラの設定により異なります。

C.1.1.9 USB_DEVICE_STATUS 共有体

USB デバイス ステータスの共有体型。

名前	型	説明
➢ _byte	Byte	デバイスのステータス情報
➢	struct	ステータス情報の構造体
☐ remote_wakeup	bit field (1)	デバイスのリモート ウェイクアップ ステータス。 • 0: リモート ウェイクアップは有効です • 1: リモート ウェイクアップは無効です
☐ self_powered	bit field (1)	デバイスのセフル パワー ステータス。 • 0: デバイスはバス パワーです • 1: デバイスはセルフ パワーです
☐ ctr_trf_mem	bit field (1)	アクティブ コントロール転送のデータの場所。 • 0: RAM • 1: ROM

C.1.1.10 CTRL_TRF_SETUP 共有体

コントロール転送のセットアップ パケット情報の共有体型。

名前	型	説明
➢	struct	
☐ bmRequestType	Byte	要求の種類: *ビット 7*: 要求方向 (0=ホストからデバイスへ – Out、1=デバイスからホストへ - In) *ビット 5-6*: 要求の種類 (0=スタンダード、1=クラス、2=ベンダー、3=予約) *ビット 0-4*: 受信箇所 (0=デバイス、1=インターフェイス、2=エンドポイント、3=その他)
☐ bRequest	byte	特定の要求
☐ wValue	word	要求によって異なるワード サイズの値。通常、この値はエンドポイントまたはインターフェイスを特定するために使用されます。
☐ wIndex	word	要求によって異なるワード サイズの値
☐ wLength	word	要求のデータ セグメントのバイト単位での長さ。例えば、データ ステージがある場合の転送するバイト数など。
➢	struct	
☐	bit field (8)	
☐	bit field (8)	
☐ W_Value	WORD	
☐ W_Index	WORD	
☐ W_Length	WORD	

➢	struct	
☐ Recipient	bit field (5)	要求の受信箇所: デバイス / インターフェイス / エンドポイント / その他
☐ RequestType	bit field (2)	要求の種類: スタンダード / クラス / ベンダー / 予約
☐ DataDir	bit field (1)	転送方向: • 0 – ホストからデバイスへ (OUT) • 1 – デバイスからホストへ (IN)
☐	bit field (8)	
☐ bFeature	byte	
☐	bit field (8)	
☐	bit field (8)	
☐	bit field (8)	
☐	bit field (8)	
☐	bit field (8)	
➢	struct	
☐	bit field (8)	
☐	bit field (8)	
☐ bDscIndex	byte	記述子のインデックス (設定および文字記述子にのみ関連)
☐ bDescType	byte	記述子の種類 – デバイス / 設定 / 文字列
☐ wLangID	word	言語 ID (文字列記述子に関連)
☐	bit field (8)	
☐ bFeature	byte	
☐	bit field (8)	
➢	struct	
☐	bit field (8)	
☐	bit field (8)	
☐ bDevADR	byte	デバイス アドレス (0 - 127)
☐ bDevADRH	byte	0 でなければなりません
☐	bit field (8)	
☐	bit field (8)	
☐	bit field (8)	
☐	bit field (8)	
➢	struct	
☐	bit field (8)	
☐	bit field (8)	
☐ bCfgValue	byte	設定番号 (0 - 255)
☐ bCfgRSD	byte	0 でなければなりません (予約)

☐	bit field (8)	
☐	bit field (8)	
☐	bit field (8)	
☐	bit field (8)	
➢	struct	
☐	bit field (8)	
☐	bit field (8)	
☐ bAltID	byte	代替設定番号 (0 - 255)
☐ bAltID_H	byte	0 でなければなりません
☐ bIntfID	byte	インターフェイス番号 (0 - 255)
☐ bIntfID_H	byte	0 でなければなりません
☐	bit field (8)	
☐	bit field (8)	
➢	struct	
☐	bit field (8)	
☐	bit field (8)	
☐ bEPID	byte	エンドポイント ID – 番号および方向
☐ bEPID_H	byte	0 でなければなりません
☐	bit field (8)	
☐	bit field (8)	
☐	bit field (8)	
☐	bit field (8)	
➢	struct	
☐	bit field (8)	
☐	bit field (8)	
☐	bit field (8)	
☐	bit field (8)	
☐ EPNum	bit field (4)	エンドポイント番号 (0 - 15)
☐	bit field (3)	
☐ EPDir	bit field (1)	エンドポイント方向: • 0 - OUT • 1 - IN
☐	bit field (8)	
☐	bit field (8)	
☐	bit field (8)	

C.1.1.11 EP_DATA 構造体

エンドポイント データの構造体型。

名前	型	説明
number	byte	エンドポイント番号
reg	near byte*	UEPn レジスタ アドレス
max_packet	word	エンドポイントのバイト単位での最大パケット サイズ
e_bdt	BDT*	エンドポイントの偶数バッファ記述子テーブル [C.1.1.5] へのポインタ
o_bdt	BDT*	エンドポイントの奇数バッファ記述子テーブル [C.1.1.5] へのポインタ
e_buffer	byte*	エンドポイントの偶数データ バッファへのポインタ
o_buffer	byte*	エンドポイントの奇数データ バッファへのポインタ

C.1.1.12 USB_DEVICE_CTX 構造体

USB デバイスのコンテキスト データの構造体型。

名前	型	説明
ep0_out	EP_DATA	コントロール パイプ (エンドポイント 0) OUT 要求用のエンドポイント データの構造体
ep0_in	EP_DATA	コントロール パイプ (エンドポイント 0) IN 要求用のエンドポイント データの構造体
pSrc	POINTER	アクティブ コントロール転送用のデータ ソース ポインタ [C.1.1.8]
pDst	POINTER	アクティブ コントロール転送用のデータ デスティネーション ポインタ [C.1.1.8]
wCount	WORD	アクティブ コントロール転送のデータ ステージ用のバイト単位でのデータのサイズ
usb_stat	USB_DEVICE_STATUS	USB デバイスのステータス情報 [C.1.1.9]
usb_device_state	byte	デバイスの状態
usb_active_cfg	byte	アクティブなデバイス設定の数
ctrl_trf_state	byte	アクティブ コントロール転送の状態

C.1.2 wdf_microchip_lib .h 関数およびマクロ

このセクションでは、ファームウェア ライブラリの一般的な関数およびマクロについて説明します。

このセクションで説明する関数およびマクロは、**wdf_microchip_lib .h** ヘッダー ファイルで宣言、定義され、DriverWizard で生成された **wdf_microchip_lib .c** ファイル (登録ユーザーの場合)、または **18F4550\ lib wdf_microchip_18f4550_eval.lib** 評価版ファームウェア ライブラリ (評価版ユーザー) で実装されます。詳細は、WinDriver ユーザーズ ガイドのセクション 16.3.5 を参照してください。

C.1.2.1 WDF_EPConfig()

目的

- 指定された USB 転送用のエンドポイントを設定し、有効にします。

プロトタイプ

```
void WDF_EPConfig(
    EP_DATA *ep_data,
    byte ep_num,
    EP_DIR dir,
    EP_TYPE type,
    word max_packet,
    near byte *reg,
    BDT *e_bdt,
    byte *e_buffer,
    BDT *o_bdt,
    byte *o_buffer);
```

パラメータ

名前	型	入出力
➢ ep_data	EP_DATA*	入力/出力
➢ ep_num	byte	入力
➢ dir	EP_DIR	入力
➢ type	EP_TYPE	入力
➢ max_packet	word	入力
➢ reg	near byte*	入力
➢ e_bdt	BDT*	入力
➢ e_buffer	byte*	入力
➢ o_bdt	BDT*	入力
➢ o_buffer	byte*	入力

説明

名前	説明
ep_data	エンドポイント データの構造体 [C.1.1.11] へのポインタ
ep_num	エンドポイントの数
dir	エンドポイントの方向 [C.1.1.1]
type	エンドポイントの転送の種類 [C.1.1.2]
max_packet	エンドポイントのバイト単位での最大パケット サイズ
reg	エンドポイントの UEPn レジスタへのポインタ

e_bdt	エンドポイントの偶数バッファ記述子テーブル [C.1.1.5] へのポインタ
e_buffer	エンドポイントの偶数データ バッファへのポインタ
o_bdt	エンドポイントの奇数バッファ記述子テーブル [C.1.1.5] へのポインタ ダブル バッファリングが無効の場合は NULL にすることができます。
o_buffer	エンドポイントの奇数データ バッファ ダブル バッファリングが無効の場合は NULL にすることができます。

戻り値

なし。

C.1.2.2 WDF_EPWrite()

目的

- RAM バッファから指定されたエンドポイントへデータを書き込みます。

この関数を呼び出した後に、`WDF_TriggerWriteTransfer()` [C.1.2.9] または `WDF_TriggerOptionWriteTransfer()` [C.1.2.10] を呼び出す必要があります。

プロトタイプ

```
void WDF_EPWrite(
    EP_DATA *ep_data,
    byte *buffer,
    word len);
```

パラメータ

名前	型	入出力
➢ ep_data	EP_DATA*	入力
➢ buffer	byte*	入力
➢ len	word	入力

説明

名前	説明
ep_data	エンドポイント データの構造体 [C.1.1.11] へのポインタ
buffer	書き込むデータを保持するバッファへのポインタ
len	書き込むバイト数

戻り値

なし。

C.1.2.3 WDF_EPWriteRom()

目的

- ROM バッファから指定されたエンドポイントへデータを書き込みます。

この関数を呼び出した後に、`WDF_TriggerWriteTransfer()` [C.1.2.9] または `WDF_TriggerOptionWriteTransfer()` [C.1.2.10] を呼び出す必要があります。

プロトタイプ

```
void WDF_EPWriteRom(
    EP_DATA *ep_data,
    rom byte *buffer,
    word len);
```

パラメータ

名前	型	入出力
➢ ep_data	EP_DATA*	入力
➢ buffer	byte*	入力
➢ len	word	入力

説明

名前	説明
ep_data	エンドポイント データの構造体 [C.1.1.11] へのポインタ
buffer	書き込むデータを保持する ROM バッファへのポインタ
len	書き込むバイト数

戻り値

なし。

C.1.2.4 WDF_EPWriteNoCopy()

目的

- インプット RAM バッファを使用して、指定されたエンドポイントへデータを書き込みます。
  データは、エンドポイントのデータ構造体 (**`ep_data`**) で保持されるデータ バッファへコピーされず、関数のインプット データ バッファ アドレス(**`addr`**) から直接書き込まれます。注意: この関数への呼び出しでインプット データ **`buffer`** は、0x400 から 0x7FF の間の 1KB RAM アドレス空間になければなりません。

この関数を呼び出した後に、WDF_TriggerWriteTransfer() [C.1.2.9] または WDF_TriggerOptionWriteTransfer() [C.1.2.10] を呼び出す必要があります。

プロトタイプ

```
void WDF_EPWriteNoCopy(
    EP_DATA *ep_data,
    byte *buffer,
    word len);
```

パラメータ

名前	型	入出力
➢ ep_data	EP_DATA*	入力
➢ buffer	rom byte*	入力
➢ len	word	入力

説明

名前	説明
ep_data	エンドポイント データの構造体 [C.1.1.11] へのポインタ
buffer	書き込むデータを保持するバッファへのポインタ
len	書き込むバイト数

戻り値

なし。

C.1.2.5 WDF_EPRead()

目的

- 指定されたエンドポイントから RAM へデータを読み取ります。

この関数を呼び出した後に、WDF_TriggerReadTransfer() [C.1.2.11] または WDF_TriggerOptionReadTransfer() [C.1.2.12] を呼び出す必要があります。

プロトタイプ

```
word WDF_EPRead(
    EP_DATA *ep_data,
    byte *buffer,
    word len);
```

パラメータ

名前	型	入出力
➢ ep_data	EP_DATA*	入力
➢ buffer	byte*	出力
➢ len	word	入力

説明

名前	説明
ep_data	エンドポイント データの構造体 [C.1.1.11] へのポインタ
buffer	読み取ったデータで更新するバッファへのポインタ
len	読み取るバイト数

戻り値

読み取ったバイト数を返します。

C.1.2.6 WDF_IsEPStall()

目的

- 指定されたエンドポイントが現在ストールされているかどうかを確認します。

プロトタイプ

```
BOOL WDF_IsEPStall(EP_DATA *ep_data);
```

パラメータ

名前	型	入出力
➢ ep_data	EP_DATA*	入力

説明

名前	説明
ep_data	エンドポイント データの構造体 [C.1.1.11] へのポインタ

戻り値

エンドポイントが現在ストールされている場合は TRUE を返します。そうでない場合は FALSE を返します。

C.1.2.7 WDF_IsEPBusy()

目的

- 指定されたエンドポイントが現在ビジーかどうかを確認します。

プロトタイプ

```
WDF_IsEPBusy(ep_data)
```

パラメータ

名前	型	入出力
➢ ep_data	EP_DATA*	入力

説明

名前	説明
ep_data	エンドポイント データの構造体 [C.1.1.11] へのポインタ

戻り値

エンドポイントが現在ビジーな場合は TRUE を返します。そうでない場合は FALSE を返します。

C.1.2.8 WDF_IsEPDataReady()

目的

- 指定されたエンドポイントにホストから受け取ったデータが含まれているかどうかを確認します。

プロトタイプ

```
WDF_IsEPDataReady(ep_data)
```

パラメータ

名前	型	入出力
➢ ep_data	EP_DATA*	入力

説明

名前	説明
ep_data	エンドポイント データの構造体 [C.1.1.11] へのポインタ

戻り値

エンドポイントにホストからのデータが含まれている場合は TRUE を返します。そうでない場合は FALSE を返します。

C.1.2.9 WDF_TriggerWriteTransfer()

目的

- 指定されたエンドポイントのデータ書き込み転送をトリガーし、関連するバッファ記述子の USB の所有権を SIE へ転送します。

プロトタイプ

```
WDF_TriggerWriteTransfer(ep_data)
```

パラメータ

名前	型	入出力
➢ ep_data	EP_DATA*	入力

説明

名前	説明
ep_data	エンドポイント データの構造体 [C.1.1.11] へのポインタ

戻り値

なし。

C.1.2.10 WDF_TriggerOptionWriteTransfer()

目的

- 指定されたエンドポイントのデータ書き込み転送をトリガーし、関連するバッファ記述子のUSB の所有権を SIE へ転送します。
  この関数では、書き込み転送のトリガー オプション [C.1.1.3] を設定することができます。

プロトタイプ

```
void WDF_TriggerOptionWriteTransfer(
    EP_DATA *ep_data,
    byte options);
```

パラメータ

名前	型	入出力
➢ ep_data	EP_DATA*	入力
➢ options	byte	入力

説明

名前	説明
ep_data	エンドポイント データの構造体 [C.1.1.11] へのポインタ
options	トリガー オプション – 0 (オプションなし) または WDF_TRIGGER_OPTIONS 列挙値 [C.1.1.3] を設定できます

戻り値

なし。

C.1.2.11 WDF_TriggerReadTransfer()

目的

- 指定されたエンドポイントのデータ読み取り転送をトリガーし、関連するバッファ記述子の USB の所有権を SIE へ転送します。

プロトタイプ

```
WDF_TriggerReadTransfer(ep_data)
```

パラメータ

名前	型	入出力
➢ ep_data	EP_DATA*	入力

説明

名前	説明
ep_data	エンドポイント データの構造体 [C.1.1.11] へのポインタ

戻り値

なし。

C.1.2.12 WDF_TriggerOptionReadTransfer()

目的

- 指定されたエンドポイントのデータ読み取り転送をトリガーし、関連するバッファ記述子のUSB の所有権を SIE へ転送します。
  この関数では、読み取り転送のトリガー オプション [C.1.1.3] を設定することができます。

プロトタイプ

```
void WDF_TriggerOptionReadTransfer(
    EP_DATA *ep_data,
    byte options);
```

パラメータ

名前	型	入出力
➢ ep_data	EP_DATA*	入力
➢ options	byte	入力

説明

名前	説明
ep_data	エンドポイント データの構造体 [C.1.1.11] へのポインタ
options	トリガー オプション – 0 (オプションなし) または WDF_TRIGGER_OPTIONS 列挙値 [C.1.1.3] を設定できます

戻り値

なし。

C.1.2.13 WDF_TriggerReadTransferNoCopy()

目的

指定されたエンドポイントのデータ読み取り転送をトリガーし、関連するバッファ記述子の USB の所有権を SIE へ転送します。
読み取ったデータは、エンドポイントのデータ構造体 (**ep_data**) で保持されるデータ バッファではなく、関数のインプット データ バッファ アドレス (**addr**) にコピーされます。
注意: 関数のインプット データ バッファ アドレス (**addr**) は、0x400 から 0x7FF の間の 1KB RAM アドレス空間になければなりません。

この関数を呼び出した後、読み取ったデータは、呼び出し元のデータ バッファ (**addr**) で利用できるようになります。そのため、WDF_EPRead( ) [C.1.2.5] への呼び出しは必要ありません。

プロトタイプ

```
void WDF_TriggerReadTransferNoCopy(
    EP_DATA *ep_data,
    byte *addr);
```

パラメータ

名前	型	入出力
➢ ep_data	EP_DATA*	入力
➢ addr	byte*	入力

説明

名前	説明
ep_data	エンドポイント データの構造体 [C.1.1.11] へのポインタ
addr	読み取ったデータがコピーされるデータ バッファの初めへのポインタ

戻り値

なし。

C.1.2.14 WDF_GetReadBytesCount()

目的

- 指定されたエンドポイントの読み取りバッファの現在のバイト カウントを取得します。`WDF_EPRead()` [C.1.2.5] を呼び出してエンドポイントから読み取る前に、この関数を呼び出し、読み取るバイト数を割り出す必要があります。

プロトタイプ

```
WORD WDF_GetReadBytesCount(EP_DATA *ep_data);
```

パラメータ

名前	型	入出力
➢ ep_data	EP_DATA*	入力

説明

名前	説明
ep_data	エンドポイント データの構造体 [C.1.1.11] へのポインタ

戻り値

エンドポイントの読み取りバッファのバイト数を返します。

C.1.2.15 WDF_DisableEP1to15()

目的

- エンドポイント 1 から 15 を無効にします。

プロトタイプ

```
void WDF_DisableEP1to15(void);
```

戻り値

なし。

C.1.2.16 WDF_DisableEP()

目的

- 指定されたエンドポイントを無効にします。

プロトタイプ

```
void WDF_DisableEP(EP_DATA *ep);
```

パラメータ

名前	型	入出力
➢ ep_data	EP_DATA*	入力

説明

名前	説明
ep_data	エンドポイント データの構造体 [C.1.1.11] へのポインタ

戻り値

なし。

C.1.3 wdf_usb9.h 関数

このセクションでは、USB 2.0 規格をサポートするファームウェア ライブラリの USB 記述子関数について説明します。

このセクションで説明する関数は、**wdf_usb9.h** ヘッダー ファイルで宣言され、DriverWizard で生成された **wdf_usb9.c** ファイル (登録ユーザーの場合)、または **18F4550\ lib wdf_microchip_18f4550_eval.lib** 評価版ファームウェア ライブラリ (評価版ユーザー) で実装されます。詳細は、WinDriver ユーザーズ ガイドのセクション 16.3.5 を参照してください。

C.1.3.1 WDF_USBCheckStdRequest()

目的

- 指定された標準 USB コントロール要求が、USB 2.0 規格に準拠しているかどうかを確認し、準拠している場合は要求を処理します。

プロトタイプ

```
BOOL USBCheckStdRequest(
    USB_DEVICE_CTX *device_ctx,
    CTRL_TRF_SETUP *setup,
    byte *data_buffer);
```

パラメータ

名前	型	入出力
➢ device_ctx	USB_DEVICE_CTX*	入力/出力
➢ setup	CTRL_TRF_SETUP*	入力
➢ data_buffer	byte*	出力

説明

名前	説明
device_ctx	デバイスのコンテキスト情報の構造体 [C.1.1.12] へのポインタ
setup	コントロール転送のセットアップ パケット情報の共有体 [C.1.1.10] へのポインタ
data_buffer	転送用データ バッファへのポインタ

戻り値

要求が有効な場合は TRUE を返します。そうでない場合は FALSE を返します。

C.2 Mass Storage ファームウェア ライブラリの API

このセクションでは、WinDriver USB Device Mass Storage ファームウェア ライブラリの大容量記憶装置用 API について説明します。このセクションで説明する関数は **18F4550\include\ class\ msd\ wdf_msd.h** ヘッダー ファイルで宣言され、DriverWizard で生成された **wdf_msd.c** ファイル (登録ユーザーの場合)、または **18F4550\ lib wdf_microchip_msd_18f4550_eval.lib** 評価版ファームウェア ライブラリ (評価版ユーザー) で実装されます。詳細は、WinDriver ユーザーズ ガイドのセクション 16.3.5 を参照してください。

注意

登録ユーザーはライブラリのソース コードを変更することができます。コードを変更する場合、USB および大容量記憶装置の規格、開発するボードのハードウェア要件を満たしているかどうかを確認してください(WinDriver ユーザーズ ガイドのセクション 16.4.3 を参照)。

C.2.1 WDF_MSD_Init()

目的

- 大容量記憶装置デバイスを初期化します。

プロトタイプ

```
void WDF_MSD_Init(
    EP_DATA *ep_in,
    EP_DATA *ep_out,
    byte max_lun,
    byte interface,
    byte alternate_setting);
```

パラメータ

名前	型	入出力
➢ ep_data_in	EP_DATA*	入力
➢ ep_data_out	EP_DATA*	入力
➢ max_lun	byte	入力
➢ interface	byte	入力
➢ alternate_setting	byte	入力

説明

名前	説明
ep_data_in	大容量記憶装置デバイスのIN 用エンドポイントのデータ構造体 [C.1.1.11] へのポインタ
ep_data_out	大容量記憶装置デバイスの OUT 用エンドポイントのデータ構造体 [C.1.1.11] へのポインタ
max_lun	デバイス (0 ベース) 上の最後のサポートされた論理ユニットの数
interface	大容量記憶装置デバイスのインターフェイスの数
alternate_setting	大容量記憶装置デバイスの代替設定の数

戻り値

なし。

C.2.2 WDF_MSD_USBCheckMSDRequest()

目的

- 指定された USB コントロール要求が、USB Mass Storage Class の規格に準拠しているかどうかを確認し、準拠している場合は要求を処理します。

プロトタイプ

```
BOOL WDF_MSD_USBCheckMSDRequest(
    USB_DEVICE_CTX *device_ctx,
    CTRL_TRF_SETUP *setup,
    byte *data_buffer);
```

パラメータ

名前	型	入出力
➢ device_ctx	USB_DEVICE_CTX*	入力/出力
➢ setup	CTRL_TRF_SETUP*	入力
➢ data_buffer	byte*	出力

説明

名前	説明
device_ctx	デバイスのコンテキスト情報の構造体 [C.1.1.12] へのポインタ
setup	コントロール転送のセットアップ パケット情報の共有体 [C.1.1.10] へのポインタ
data_buffer	転送用のデータ バッファへのポインタ

戻り値

要求が有効な場合は TRUE を返します。そうでない場合は FALSE を返します。

C.2.3 WDF_MSD_ProcessIO()

目的

- 大容量記憶装置の SCSI コマンドを処理します。

プロトタイプ

```
void WDF_MSD_ProcessIO(void);
```

戻り値

なし。

C.3 DriverWizard で生成されたファームウェアの API

このセクションでは、WinDriver USB Device の DriverWizard によって生成された Microchip PIC18F4550 開発ボード用ファームウェア ライブラリの API について説明します。このセクションで説明する関数は、**18F4550\ include\ periph.h** ヘッダー ファイルで宣言され、DriverWizard で生成された **periph.c** ファイルにおいて、ウィザードで定義されたデバイス設定情報に従って実装されます。

このファームウェアのエントリー ポイント(**main.c** の `main()` (ソース コードは登録ユーザーのみに提供されます)) は、周辺のデバイスと通信するために **periph.h** で宣言された `WDF_xxx()` 関数 (**periph.c** で実装される) を呼び出す **Task Dispatcher (タスク ディスパッチャー)** を実装します。

生成された Microchip PIC18F4550 ボードの大容量記憶装置用コードには、このセクションで説明するファイルおよび API に加え、セクション [C.4] で説明する大容量記憶装置用の特別な API も含まれています。

注意

コードを変更する場合、USB 規格および開発するボードのハードウェア要件を満たしているかどうかを確認してください(WinDriver ユーザーズ ガイドのセクション 16.4.3 を参照)。

C.3.1 WDF_Init()

目的

- デバイスを初期化します。
  デバイスとの通信を有効にする必要な初期設定を実行するために、この関数は自動的にファームウェアの `main()` 関数から呼び出されます。

プロトタイプ

```
void WDF_Init(void);
```

戻り値

なし。

C.3.2 WDF_Poll()

目的

- FIFO データ用のデバイスをポーリングします。
  Task Dispatcher はデバイスが動作していない間、この関数を繰り返し呼び出します。

プロトタイプ

```
void WDF_Poll(void);
```

戻り値

なし。

C.3.3 WDF_SOFHandler()

目的

- フレームの開始割り込み処理関数。

プロトタイプ

```
void WDF_SOFHandler(void);
```

戻り値

成功した場合、TRUE を返します。失敗した場合、FALSE を返します。

C.3.4 WDF_Suspend()

目的

- この関数は、デバイスがサスペンド モードになる前に Task Dispatcher により呼び出されます。

プロトタイプ

```
BOOL WDF_Suspend(void);
```

戻り値

成功した場合、TRUE を返します。失敗した場合、FALSE を返します。

C.3.5 WDF_Resume()

目的

- この関数は、デバイスが活動を再開した後 Task Dispatcher により呼び出されます。

プロトタイプ

```
BOOL WDF_Resume(void);
```

戻り値

成功した場合、TRUE を返します。失敗した場合、FALSE を返します。

C.3.6 WDF_ErrorHandler()

目的

- USB エラー割り込み処理関数。

プロトタイプ

```
void WDF_ErrorHandler(void);
```

戻り値

なし。

C.3.7 WDF_SetConfiguration()

目的

- この関数は、SET CONFIGURATION コマンドを受け取ったとき Task Dispatcher により呼び出されます。

プロトタイプ

```
void WDF_SetConfiguration(byte config);
```

パラメータ

名前	型	入出力
➢ config	byte	入力

説明

名前	説明
config	設定する設定番号

戻り値

なし。

C.3.8 WDF_SetInterface()

目的

- この関数は、SET INTERFACE コマンドを受け取ったとき Task Dispatcher により呼び出されます。

プロトタイプ

```
void WDF_SetInterface(
    byte ifc,
    byte alt_set);
```

パラメータ

名前	型	入出力
➢ ifc	byte	入力
➢ alt_set	byte	入力

説明

名前	説明
ifc	設定するインターフェイス番号
alt_set	控えの設定番号

戻り値

なし。

C.3.9 WDF_GetInterface()

目的

- この関数は、GET INTERFACE コマンドを受け取ったとき Task Dispatcher により呼び出されます。

プロトタイプ

```
byte WDF_GetInterface(byte ifc);
```

パラメータ

名前	型	入出力
➢ ifc	byte	入力

説明

名前	説明
ifc	インターフェイス番号

戻り値

指定されたインターフェイスのアクティブな代替設定の数を返します。

C.3.10 WDF_VendorCmnd()

目的

- この関数は、ベンダー特有のコマンドを受け取ったとき Task Dispatcher により呼び出されます。

プロトタイプ

```
BOOL WDF_VendorCmnd(
    byte bRequest,
    word wValue,
    word wIndex,
    word wLength);
```

パラメータ

名前	型	入出力
➢ bRequest	byte	入力
➢ wValue	word	入力
➢ wIndex	word	入力
➢ wLength	word	入力

説明

名前	説明
bRequest	実際の要求
wValue	要求の `wValue` フィールド
wIndex	要求の `wIndex` フィールド
wLength	転送するバイト数 (データ ステージがある場合)

戻り値

成功した場合、TRUE を返します。失敗した場合、FALSE を返します。

C.3.11 WDF_ClearFeature()

目的

- この関数は、CLEAR FEATURE コマンドを受け取ったとき Task Dispatcher により呼び出されます。

プロトタイプ

```
BOOL WDF_ClearFeature(void);
```

戻り値

成功した場合、TRUE を返します。失敗した場合、FALSE を返します。

C.3.12 WDF_SetFeature()

目的

- この関数は、SET FEATURE コマンドを受け取ったとき Task Dispatcher により呼び出されます。

プロトタイプ

```
BOOL WDF_SetFeature(void);
```

戻り値

成功した場合、TRUE を返します。失敗した場合、FALSE を返します。

C.4 DriverWizard により生成された Mass Storage ファームウェアの API

このセクションでは、Microchip PIC18F4550 開発ボード ベースのデバイス上の記憶媒体にアクセスするために、WinDriver USB Device の DriverWizard によって生成された Mass Storage ファームウェアの API について説明します。

このセクションで説明する関数は、**18F4550\ include\class\msd\ wdf_disk.h** ヘッダー ファイルで宣言され、DriverWizard で生成された **wdf_xxx_hw.c** ファイルで実装されます。

このファイルには、ユーザーによって記述される必要のある PIC18F4550 ボードと共に使用する記憶媒体用の関数を実装するスタブが含まれています。
WinDriver\ wdf\ microchip\18F4550\ samples\msd\ sdcard.c 大容量記憶装置サンプル ファイルには、SD カード用の記憶媒体にアクセスする関数の実装例が含まれています。

注意

コードを変更する場合、USB および大容量記憶装置の規格、開発するボードのハードウェア要件を満たしているかどうかを確認してください (WinDriver ユーザーズ ガイドのセクション 16.4.3 を参照)。

C.4.1 生成された Mass Storage ファームウェアの型

C.4.1.1 WDF_DISK_STATUS 列挙型

デバイスの記憶媒体上で実行される処理のステータス コードの列挙型。

列挙値	説明
DISK_STATUS_SUCCESS	ステータス OK
DISK_INIT_COMM_FAILURE	記憶媒体との通信は確立されていません
DISK_INIT_TIMEDOUT	記憶媒体の初期化はタイム アウトしました
DISK_TYPE_INVALID	記憶媒体の種類の定義に失敗しました
DISK_INVALID_COMMAND	コマンドは記憶媒体によって認識されませんでした
DISK_TIMEDOUT	記憶媒体は読み取り、書き込み、消去シーケンス中にタイム アウトしま

	した
DISK_CRC_ERROR	読み取り転送中に CRC エラーが発生しました。読み取ったデータを無効にする必要があります。
DISK_DATA_REJECTED	記憶媒体の CRC は送信されたデータの CRC と一致しませんでした

C.4.1.2 DISK_STATE 共有体

記憶媒体のステータスの共有体型。

名前	型	説明
➢	struct	
☐ is_initialized	bit field (1)	媒体の初期化に成功後 1 に設定します
➢ _byte	byte	

C.4.2 生成された Mass Storage ファームウェア関数

C.4.2.1 WDF_DISK_MediaInitialize()

目的

- デバイスの記憶媒体を初期化します。

プロトタイプ

```
WDF_DISK_STATUS WDF_DISK_MediaInitialize(void);
```

戻り値

記憶媒体の初期化 [C.4.1.1] の結果を返します。

C.4.2.2 WDF_DISK_SectorRead()

目的

- 指定されたセクターをデバイスの記憶媒体から読み取ります。

プロトタイプ

```
WDF_DISK_STATUS WDF_DISK_SectorRead(
    dword sector_addr,
    byte *buffer);
```

パラメータ

名前	型	入出力
➢ sector_addr	dword	入力
➢ buffer	byte*	出力

説明

名前	説明
sector_addr	読み取るセクターのアドレス
buffer	読み取ったデータで更新されるバッファへのポインタ

戻り値

読み取り結果 [C.4.1.1] を返します。

C.4.2.3 WDF_DISK_SectorWrite()

目的

- 指定されたセクターをデバイスの記憶媒体へ書き込みます。

プロトタイプ

```
WDF_DISK_STATUS WDF_DISK_SectorWrite(
    dword sector_addr,
    byte *buffer);
```

パラメータ

名前	型	入出力
➢ sector_addr	dword	入力
➢ buffer	byte*	入力

説明

名前	説明
sector_addr	書き込むセクターのアドレス
buffer	書き込むデータを保持しているバッファへのポインタ

戻り値

書き込み結果 [C.4.1.1] を返します。

C.4.2.4 WDF_DISK_Detect()

目的

- デバイス上に記憶媒体があるかどうかをチェックします。

プロトタイプ

```
BOOL WDF_DISK_Detect(void);
```

戻り値

記憶媒体がある場合は TRUE を返します。そうでない場合は FALSE を返します。

C.4.2.5 WDF_DISK_IsWriteProtected()

目的

- デバイス上の記憶媒体が書き込み禁止かどうかをチェックします。

プロトタイプ

```
BOOL WDF_DISK_IsWriteProtected(void);
```

戻り値

デバイス上の記憶媒体が書き込み禁止の場合は TRUE を返します。そうでない場合は FALSE を返します。

C.4.2.6 WDF_DISK_GetCapacity()

目的

- デバイスの記憶媒体の容量を取得します。

プロトタイプ

```
void WDF_DISK_GetCapacity(
    dword *last_lba,
    dword *block_len);
```

パラメータ

名前	型	入出力
➢ last_lba	dword*	出力
➢ block_len	dword*	出力

説明

名前	説明
last_lba	記憶媒体の最終論理ブロック アドレス (LBA)
block_len	記憶媒体のバイト単位でのブロック長

戻り値

なし。

付録 D
USB Device - Philips PDIUSBD12 API のリファレンス

D.1 ファームウェア ライブラリの API

このセクションでは、Philips PDIUSBD12 開発ボード用の WinDriver USB Device ファームウェア ライブラリの API について説明します。このセクションで説明する関数、一般的な型と定義は、それぞれ **d12\ include\ d12_lib .h** および **d12\ include\ types.h** ヘッダー ファイルで宣言、定義され、DriverWizard で生成された **d12_lib .c** ファイル (登録ユーザーの場合)、または **d12\ lib\ d12_eval.lib** 評価版ファームウェア ライブラリ (評価版ユーザー) で実装されます。詳細は、WinDriver ユーザーズ ガイドのセクション 16.3.5 を参照してください。

注意

登録ユーザーはライブラリのソース コードを変更することができます。コードを変更する場合、USB 規格および開発するボードのハードウェア要件を満たしているかどうかを確認してください(WinDriver ユーザーズ ガイドのセクション 16.4.3 を参照)。

D.1.1 ファームウェア ライブラリの型

このセクションで説明するデータ型は、**d12\ include\ types.h** ヘッダー ファイルで定義されています。

D.1.1.1 WDF_ENDPOINTS 列挙型

PDIUSBD12 エンドポイントの列挙型。

列挙値	説明
EP0_OUT	コントロール (エンドポイント 0) OUT エンドポイント
EP0_IN	コントロール (エンドポイント 0) IN エンドポイント
EP1_OUT	汎用 OUT エンドポイント
EP1_IN	汎用 IN エンドポイント
EP2_OUT	メイン OUT エンドポイント
EP2_IN	メイン IN エンドポイント

D.1.1.2 D12_MODES 列挙型

PDIUSBD12 モードの列挙型。

列挙値	説明
NO_ISO	メイン エンドポイントは両方とも等時性ではありません
ISO_IN	メイン IN エンドポイントは等時性です
ISO_OUT	メイン OUT エンドポイントは等時性です
ISO_INOUT	メイン エンドポイントは両方とも等時性です

D.1.1.3 DMA_DIRECTION 列挙型

直接メモリ アクセス (DMA) の方向の列挙型。

列挙値	説明
DMA_IN	DMA IN 方向 – デバイス メモリから PDIUSBD12 へ (そこからホストへ)
DMA_OUT	DMA OUT 方向 - (ホストから受け取り) PDIUSBD12 からデバイス メモリへ

D.1.2 ファームウェア ライブラリの関数

セクション D.1.2.1 から D.1.2.16 で説明する関数は、**d12\ d12_lib .h** ヘッダー ファイルで宣言されています。

セクション D.1.2.17 から D.1.2.18 で説明する関数は、**d12\ d12_io.h** で宣言され、ライブラリのハードウェア アブストラクション レイヤを提供します。デフォルトの実装では、ISA カードを使用して PDIUSBD12 ベースのボードと x86 PC の接続をサポートする D12-ISA (PC) Eval Kit バージョン 1.4 をターゲットとしています。登録ユーザーは、その他のマイクロコントローラをサポートするために、DriverWizard で生成された **d12_io.c** ファイルおよび **d12_io.h** ヘッダー ファイルのハードウェア固有の定義で、これらの関数の実装を変更することができます。

D.1.2.1 WDF_Exit()

目的

- ファームウェア ライブラリを終了します。

プロトタイプ

```
void WDF_Exit(void);
```

戻り値

なし。

D.1.2.2 WDF_ConnectUSB()

目的

- デバイスと USB バス間の通信を確立します。

プロトタイプ

```
void WDF_ConnectUSB(D12_MODES mode);
```

パラメータ

名前	型	入出力
➢ mode	D12_MODES	入力

説明

名前	説明
mode	PDIUSBD12 モード [D.1.1.2]

戻り値

なし。

D.1.2.3 WDF_DisconnectUSB()

目的

- デバイスと USB バス間の通信を切断します。

プロトタイプ

```
void WDF_DisconnectUSB(void);
```

戻り値

なし。

D.1.2.4 WDF_ReconnectUSB()

目的

- デバイスと USB バス間の通信を切断し (`WDF_DisconnectUSB()`)、再接続します (`WDF_ConnectUSB()`)。

プロトタイプ

```
void WDF_ReconnectUSB(D12_MODES mode);
```

パラメータ

名前	型	入出力
➢ mode	D12_MODES	入力

説明

名前	説明
mode	PDIUSBD12 モード [D.1.1.2]

戻り値

なし。

D.1.2.5 WDF_EnableAllEP()

目的

- デバイスのエンドポイントをすべて有効にします。

プロトタイプ

```
void WDF_EnableAllEP(void);
```

戻り値

なし。

D.1.2.6 WDF_DisableEP1AND2()

目的

- デバイスの汎用エンドポイント (EP1) およびメイン エンドポイント (EP2) を無効にします。

プロトタイプ

```
void WDF_DisableEP1AND2(void);
```

戻り値

なし。

D.1.2.7 WDF_StallEP0()

目的

- デバイスのコントロール エンドポイント (エンドポイント 0) をストールします。

プロトタイプ

```
void WDF_StallEP0(void);
```

戻り値

なし。

D.1.2.8 WDF_EPoutFull()

目的

- 指定された汎用またはメイン OUT エンドポイントのデータ バッファが、ホストからのデータを保持しているかどうかをチェックします。

プロトタイプ

```
unsigned char WDF_EPoutFull(WDF_ENDPOINTS ep);
```

パラメータ

名前	型	入出力
➢ ep	WDF_ENDPOINTS	入力

説明

名前	説明
ep	チェックするエンドポイント [D.1.1.1]

戻り値

エンドポイントのデータ バッファにホストからのデータが保持されている場合は 1 を返します。そうでない場合は 0 を返します。エラーの場合 (**ep** が `EP1_OUT` または `EP2_OUT` でない場合) は `GENERR` を返します。

D.1.2.9 WDF_EPinFull()

目的

- 指定された汎用またはメイン IN エンドポイントで、ファームウェアからデータを受け取る準備ができているかどうかをチェックします (受け取ったデータは、後でホストに転送されます)。

プロトタイプ

```
unsigned char WDF_EPinFull(WDF_ENDPOINTS ep);
```

パラメータ

名前	型	入出力
➢ ep	WDF_ENDPOINTS	入力

説明

名前	説明
ep	チェックするエンドポイント [D.1.1.1]

戻り値

エンドポイントでデータを受け取る準備ができている場合は 1 を返します。そうでない場合は 0 を返します。エラーの場合 (**ep** が EP1_OUT または EP2_OUT でない場合) は GENERR を返します。

D.1.2.10 WDF_EPWrite()

目的

- 指定されたエンドポイントにデータを書き込みます。

プロトタイプ

```
unsigned char WDF_EPWrite(
    WDF_ENDPOINTS ep,
    unsigned char code *pData,
    unsigned short len);
```

パラメータ

名前	型	入出力
➢ ep	WDF_ENDPOINTS	入力
➢ pData	unsigned char code*	入力
➢ len	unsigned short	入力

説明

名前	説明
ep	書き込むエンドポイント [D.1.1.1]
pData	書き込むデータを保持するバッファへのポインタ
len	書き込むバイト数

戻り値

無効なパラメータの場合は 0 または GENERR を返します。

D.1.2.11 WDF_EPRead()

目的

- 指定されたエンドポイントからデータを読み取ります。

プロトタイプ

```
unsigned char WDF_EPRead(
    WDF_ENDPOINTS ep,
    unsigned char code *pData,
    unsigned short len);
```

パラメータ

名前	型	入出力
➢ ep	WDF_ENDPOINTS	入力
➢ pData	unsigned char code*	出力
➢ len	unsigned short	入力

説明

名前	説明
ep	読み取るエンドポイント [D.1.1.1]
pData	読み取ったデータを格納するバッファへのポインタ
len	読み取るバイト数

戻り値

無効なパラメータの場合は 0 または GENERR を返します。

D.1.2.12 WDF_DMASetup()

目的

- 直接メモリアクセス (DMA) 転送のセットアップを行います。

プロトタイプ

```
void WDF_DMASetup(
    DMA_DIRECTION direction,
    unsigned char dmaFlags,
    void *pUserData);
```

パラメータ

名前	型	入出力
➢ direction	DMA_DIRECTION	入力
➢ dmaFlags	unsigned char	入力
➢ pUserData	void*	入力/出力

説明

名前	説明
direction	DMA 転送の方向 [D.1.1.3]
dmaFlags	以下のいずれかの DMA フラグ。 • D12_DMASINGLE - DMA シングル モード • D12_BURST_4 - DMA バースト モード 4 • D12_BURST_8 - DMA バースト モード 8 • D12_BURST_16 - DMA バースト モード 16
pUserData	DMA バッファへのポインタ

戻り値

なし。

D.1.2.13 WDF_DMARunning()

目的

- 現在アクティブな DMA 転送があるかどうかをチェックします。

プロトタイプ

```
unsigned char WDF_DMARunning(void);
```

戻り値

現在アクティブな DMA 転送がある場合は 1 を返します。そうでない場合は 0 を返します。

D.1.2.14 WDF_DMAStop()

目的

- アクティブな DMA 転送を停止します。

プロトタイプ

```
void WDF_DMAStop(void);
```

戻り値

なし。

D.1.2.15 WDF_SetLEDStatus()

目的

- PDIUSBD12 評価版ボード上の指定された LED のステータスを設定します。
  D12-ISA (PC) Eval Kit に特有な関数です。

プロトタイプ

```
void WDF_SetLEDStatus(
    unsigned char ledNum,
    unsigned char status);
```

パラメータ

名前	型	入出力
➢ ledNum	unsigned char	入力
➢ status	unsigned char	入力

説明

名前	説明
ledNum	LED 番号
status	LED ステータス

戻り値

なし

D.1.2.16 WDF_GetKeyStatus()

目的

- PDIUSBD12 評価版ボード上の指定されたキーのステータスを取得します。
  D12-ISA (PC) Eval Kit に特有な関数です。

プロトタイプ

```
char WDF_GetKeyStatus(void);
```

戻り値

キーのステータスを返します。

D.1.2.17 outportb()

目的

- 指定されたポートにバイトを書き込みます。

プロトタイプ

```
void outportb(unsigned short port, unsigned char val);
```

名前	型	入出力
➢ port	unsigned short	入力
➢ val	unsigned char	入力

説明

名前	説明
port	書き込むポート
val	書き込むバイト

戻り値

なし。

D.1.2.18 inportb()

目的

- 指定されたポートからバイトを読み取ります。

プロトタイプ

```
unsigned char inportb(unsigned short port);
```

名前	型	入出力
➢ port	unsigned short	入力

説明

名前	説明
port	読み取るポート

戻り値

指定されたポートから読み取ったバイトを返します。

D.2 DriverWizard で生成されたファームウェアの API

このセクションでは、WinDriver USB Device の DriverWizard によって生成された Philips PDIUSBD12 開発ボード用ファームウェア ライブラリの API について説明します。このセクションで説明する関数は、**d12\ include\ periph.h** ヘッダー ファイルで宣言され、DriverWizard で生成された **periph.c** ファイルにおいて、ウィザードで定義されたデバイス設定情報に従って実装されます。

このファームウェアのエントリー ポイント(**main.c** の `main()` (ソース コードは登録ユーザーのみに提供されます)) は、周辺のデバイスと通信するために **periph.h** で宣言された `WDF_xxx()` 関数 (**periph.c** で実装される) を呼び出す **Task Dispatcher (タスク ディスパッチャー)** を実装します。

注意

コードを変更する場合、USB 規格および開発するボードのハードウェア要件を満たしているかどうかを確認してください(WinDriver ユーザーズ ガイドのセクション 16.4.3 を参照)。

D.2.1 WDF_Init()

目的

- ユーザー固有のデバイスを初期化し、デバイスの代替設定モードを設定します。

この関数は、ファームウェア ライブラリを初期化する際に、ファームウェアの `main()` 関数から自動的に呼び出されます。

プロトタイプ

```
void WDF_Init(void);
```

戻り値

なし。

D.2.2 WDF_Uninit()

目的

- ユーザー固有のデバイスの終了処理を実行します。

この関数は、ファームウェア ライブラリの終了処理を実行する際に、ファームウェアの `main()` 関数から自動的に呼び出されます。

プロトタイプ

```
void WDF_Uninit(void);
```

戻り値

なし。

D.2.3 WDF_SuspendChange()

目的

- この関数は、デバイスがサスペンド モードになる前に、またはサスペンド モードから戻る際に、Task Dispatcher (タスク ディスパッチャ) により呼び出されます。

プロトタイプ

```
void WDF_SuspendChange(void);
```

戻り値

成功した場合、TRUE を返します。失敗した場合、FALSE を返します。

D.2.4 WDF_Poll()

目的

- データ用のデバイスをポーリングします。
  Task Dispatcher (タスク ディスパッチャ) は、この関数を繰り返し呼び出します。

プロトタイプ

```
void WDF_Poll(void);
```

戻り値

なし。

D.2.5 WDF_BusReset()

目的

- バスのリセットが発生した際に、Task Dispatcher (タスク ディスパッチャ) により呼び出されます。

プロトタイプ

```
void WDF_BusReset(void);
```

戻り値

なし。

D.2.6 WDF_SetConfiguration()

目的

- この関数は、GET CONFIGURATION コマンドを受け取ったとき Task Dispatcher により呼び出されます。

プロトタイプ

```
void WDF_SetConfiguration(void);
```

戻り値

なし。

D.2.7 WDF_SetInterface()

目的

- この関数は、SET INTERFACE コマンドを受け取ったとき Task Dispatcher により呼び出されます。

プロトタイプ

```
void WDF_SetInterface(void);
```

戻り値

なし。

D.2.8 WDF_GetInterface()

目的

- この関数は、GET INTERFACE コマンドを受け取ったとき Task Dispatcher により呼び出されます。

プロトタイプ

```
void WDF_GetInterface(void);
```

戻り値

なし。

D.2.9 WDF_VendorRequest()

目的

- この関数は、ベンダー特有のコマンドを受け取ったとき Task Dispatcher により呼び出されます。

プロトタイプ

```
char WDF_VendorRequest(
    unsigned char bRequest,
    unsigned short wValue,
    unsigned short wIndex,
    unsigned short wLength,
    unsigned char *pData,
    unsigned short *pwRetLen,
    unsigned char **ppRetData);
```

パラメータ

名前	型	入出力
➢ bRequest	unsigned char	入力
➢ wValue	unsigned short	入力
➢ wIndex	unsigned short	入力
➢ wLength	unsigned short	入力
➢ pData	unsigned char*	入力
➢ pwRetLen	unsigned short*	出力
➢ ppRetData	unsigned char**	出力

説明

名前	説明
bRequest	実際の要求
wValue	要求の `wValue` フィールド
wIndex	要求の `wIndex` フィールド
wLength	転送するバイト数 (データ ステージがある場合)
pData	ホストから受け取ったデータ ステージのデータへのポインタ (OUT 要求の場合のみ)
pwRetLen	返されたデータ (`*ppRetData`) の長さ (byte)
ppRetData	データ ステージのホストへ送信するデータを保持するバッファへのポインタをポイントします (IN 要求の場合のみ)

戻り値

不正なベンダー要求の場合は 0 を返します。そうでない場合は 0 以外の値を返します。

付録 E
USB Device - Silicon Laboratories C8051F320 API のリファレンス

E.1 ファームウェア ライブラリの API

このセクションでは、Silicon Laboratories C8051F320 開発ボード用の WinDriver USB Device ファームウェア ライブラリの API について説明します。このセクションで説明する関数および一般的な型と定義は、**F320\include\wdf_silabs_lib .h** ヘッダー ファイルで宣言、定義され、DriverWizard で生成された **wdf_silabs_lib .c** ファイル (登録ユーザーの場合)、または **F320\ lib\ wdf_silabs_f320_eval.lib** 評価版ファームウェア ライブラリ (評価版ユーザー) で実装されます。詳細は、WinDriver ユーザーズ ガイドのセクション 16.3.5 を参照してください。

注意

登録ユーザーはライブラリのソース コードを変更することができます。コードを変更する場合、USB 規格および開発するボードのハードウェア要件を満たしているかどうかを確認してください(WinDriver ユーザーズ ガイドのセクション 16.4.3 を参照)。

E.1.1 wdf_silabs_lib.h の型

このセクションで定義されている API は **F320\ wdf_silabs_lib.h** で定義されています。

E.1.1.1 EP_DIR 列挙型

エンドポイント方向の列挙型:

列挙値	説明
DIR_OUT	OUT 方向 (ホストからデバイスへの書き込み)
DIR_IN	IN 方向 (デバイスからホストへの読み取り)

E.1.1.2 EP_TYPE 列挙型

エンドポイン タイプの列挙型。
エンドポイント タイプは、エンドポイント上で実行される転送の種類 (バルク転送、割り込み転送、または等時性 (アイソクロナス) 転送) を割り出します。

列挙値	説明
ISOCHRONOUS	等時性エンドポイント
BULK	バルク エンドポイント
INTERRUPT	割り込みエンドポイント

E.1.1.3 EP_BUFFERING 列挙型

エンドポイント バッファリングの型の列挙型:

列挙値	説明
NO_BUFFERING	バッファリングなし
DOUBLE_BUFFERING	ダブル バッファリング

E.1.1.4 EP_SPLIT の列挙型

エンドポイントの FIFO (First In First Out) バッファ スプリット モードの列挙型

列挙値	説明
NO_SPLIT	エンドポイントの FIFO バッファをスプリットしません
SPLIT	エンドポイントの FIFO バッファをスプリットします

E.1.2 c8051f320.h の型および一般的な定義

このセクションで定義されている API は **F320\ c8051f320.h** で定義されています。

E.1.2.1 エンドポイント アドレスの定義

以下のプリプロセッサの定義はエンドポイントのアドレス (番号) を示します。

名前	説明
EP1_IN	エンドポイント 1、IN 方向 - アドレス 0x81
EP1_OUT	エンドポイント 1、OUT 方向 - アドレス 0x01
EP2_IN	エンドポイント 2、IN 方向 - アドレス 0x82
EP2_OUT	エンドポイント 2、OUT 方向 - アドレス 0x02
EP3_IN	エンドポイント 3、IN 方向 - アドレス 0x83
EP3_OUT	エンドポイント 3、OUT 方向 - アドレス 0x03

E.1.2.2 エンドポイントの状況の定義

以下のプリプロセッサの定義はエンドポイントの状態を示します。

名前	説明
EP_IDLE	エンドポイントは作動していません。
EP_TX	エンドポイントはデータを転送しています。
EP_ERROR	エンドポイントでエラーが発生しました。
EP_HALTED	エンドポイントは停止されました。
EP_RX	エンドポイントはデータを受信しています。
EP_NO_CONFIGURED	エンドポイントは設定されていません。

E.1.2.3 EP_INT_HANDLER 関数のポインタ

エンドポイント割り込み処理関数のポインタの型。

```
typedef void (*EP_INT_HANDLER)(PEP_STATUS);
```

E.1.2.4 EP0_COMMAND 構造体

エンドポイント (パイプ 0) ホスト コマンドの情報構造体の型を管理します。

名前	型	説明
bmRequestType	BYTE	要求の種類: **ビット 7**: 要求方向 (0= ホストからデバイスへ – Out、1= デバイスからホストへ – In) **ビット 5–6**: 要求の種類 (0= スタンダード、1= クラス、2= ベンダー、3= リザーブ) **ビット 0–4**: 受信箇所 (0= デバイス、1= インターフェイス、2= エンドポイント、3= その他)
bRequest	BYTE	特定の要求
wValue	WORD	要求によって異なる WORD サイズの値
wIndex	WORD	要求によって異なる WORD サイズの値。通常、この値はエンドポイントまたはインターフェイスを特定するために使用されます。
wLength	WORD	要求用のデータ セグメントのバイト単位での長さ。例えば、データ ステージがある場合の転送するバイト数など。

E.1.2.5 EP_STATUS 構造体

IN、OUT、およびエンドポイント 0 (コントロール) 要求で使用されるエンドポイントの状況情報構造体の型。

名前	型	説明
bEp	BYTE	エンドポイントのアドレス [E.1.2.1]
uNumBytes	UINT	転送に利用できるバイト数
uMaxP	UINT	最大のパケット サイズ
bEpState	BYTE	エンドポイントの状態
pData	void*	エンドポイントへ/からの転送データに使用されるデータ バッファへのポインタ
wData	WORD	小さいデータ パケット用のストレージ
pfIsr	EP_INT_HANDLER	割り込みサービス ルーチン (ISR) [E.1.2.3]

E.1.2.6 PEP_STATUS 構造体のポインタ

EP_STATUS構造体 [E.1.2.5] へのポインタ。

E.1.2.7 IF_STATUS 構造体

インターフェイスの状況構造体の型。

名前	型	説明
bNumAlts	BYTE	インターフェイスのための代替設定の数
bCurrentAlt	BYTE	現在アクティブなインターフェイスのための代替設定
bIfNumber	BYTE	インタフェース番号

E.1.2.8 PIF_STATUS 構造体のポインタ

IF_STATUS 構造体へのポインタ

E.1.3 ファームウェア ライブラリの関数

このセクションで説明されている関数は **F320\ wdf_silabs_lib.h** で定義されています。

E.1.3.1 WDF_EPINConfig()

目的

- IN 転送用のエンドポイント 1-3 を設定します。

プロトタイプ

```
void WDF_EPINConfig(
    PEP_STATUS pEpStatus,
    BYTE address,
    EP_TYPE type,
    int maxPacketSize,
    EP_INT_HANDLER pfIsr,
    EP_BUFFERING buffering,
    EP_SPLIT splitMode);
```

パラメータ

名前	型	入出力
➢ pEpStatus	PEP_STATUS	出力
➢ address	BYTE	入力
➢ type	EP_TYPE	入力
➢ maxPacketSize	int	入力
➢ pfIsr	EP_INT_HANDLER	入力
➢ buffering	EP_BUFFERING	入力
➢ splitMode	EP_SPLIT	入力

説明

名前	説明
pEpStatus	エンドポイントの状況情報構造体 [E.1.2.6] へのポインタ。この関数は関連した状況情報で構造体をアップデートします。
address	エンドポイントのアドレス [E.1.2.1]
type	エンドポイントの転送の種類 [E.1.1.2]
maxPacketSize	エンドポイントの最大のパケット サイズ
pfIsr	エンドポイントの割り込みハンドラ [E.1.2.3]
buffering	エンドポイントのバッファリングの種類 [E.1.1.3]
splitMode	エンドポイントのスプリット モード [E.1.1.4]

戻り値

なし。

E.1.3.2 WDF_EPOUTConfig()

目的

- OUT 転送用のエンドポイント 1-3 を設定します。

プロトタイプ

```
void WDF_EPOUTConfig(
    PEP_STATUS pEpStatus,
    BYTE address,
    EP_TYPE type,
    int maxPacketSize,
    EP_INT_HANDLER pfIsr,
    EP_BUFFERING buffering);
```

パラメータ

名前	型	入出力
➢ pEpStatus	PEP_STATUS	出力
➢ address	BYTE	入力
➢ type	EP_TYPE	入力
➢ maxPacketSize	int	入力
➢ pfIsr	EP_INT_HANDLER	入力
➢ buffering	EP_BUFFERING	入力

説明

名前	説明
pEpStatus	エンドポイントの状況情報構造体 [E.1.2.6] へのポインタ。この関数は関連した状況情報で構造体をアップデートします。
address	エンドポイントのアドレス [E.1.2.1]
type	エンドポイントの転送の種類 [E.1.1.2]
maxPacketSize	エンドポイントの最大のパケット サイズ
pfIsr	エンドポイントの割り込みハンドラ [E.1.2.3]
buffering	エンドポイントのバッファリングの種類 [E.1.1.3]

戻り値

なし。

E.1.3.3 WDF_HaltEndpoint()

目的

- エンドポイントを停止します。

プロトタイプ

```
BYTE WDF_HaltEndpoint(PEP_STATUS pEpStatus);
```

パラメータ

名前	型	入出力
➢ pEpStatus	PEP_STATUS	入力/出力

説明

名前	説明
pEpStatus	エンドポイントの状況情報構造体 [E.1.2.6] へのポインタ

戻り値

エンドポイントの状態 [E.1.2.2] を返します。

E.1.3.4 WDF_EnableEndpoint()

目的

- エンドポイントを有効にします。

プロトタイプ

```
BYTE WDF_EnableEndpoint(PEP_STATUS pEpStatus);
```

パラメータ

名前	型	入出力
➢ pEpStatus	PEP_STATUS	入力/出力

説明

名前	説明
pEpStatus	エンドポイントの状況情報構造体 [E.1.2.6] へのポインタ

戻り値

エンドポイントの状態 [E.1.2.2] を返します。

E.1.3.5 WDF_SetEPByteCount()

目的

- FIFO (First In First Out) バッファのバイト カウントを設定します。
  エンドポイント FIFO バッファをホストへ転送するデータでアップデートするために、この関数は `WDF_FIFOWrite()` [E.1.3.10] を呼び出す前に呼び出します。

プロトタイプ

```
void WDF_SetEPByteCount(
    BYTE bEp,
    UINT bytes_count);
```

パラメータ

名前	型	入出力
➢ bEp	BYTE	入力
➢ bytes_count	UINT	入力

説明

名前	説明
bEp	エンドポイントのアドレス [E.1.2.1]
bytes_count	設定するバイト カウント

戻り値

なし。

E.1.3.6 WDF_GetEPByteCount()

目的

- FIFO (First In First Out) バッファの現在のバイト カウントを取得します。
  読み取るバイトの量を割り出すために、この関数は `WDF_FIFORead()` [E.1.3.11] を呼び出す前に呼び出す必要があります。

プロトタイプ

```
UINT WDF_GetEPByteCount(BYTE bEp);
```

パラメータ

名前	型	入出力
➢ bEp	BYTE	入力

説明

名前	説明
bEp	エンドポイントのアドレス [E.1.2.1]

戻り値

エンドポイントの FIFO バイト カウントを返します。

E.1.3.7 WDF_FIFOClear()

目的

- エンドポイントの FIFO (First In First Out) バッファを空にします。

プロトタイプ

```
void WDF_FIFOClear(BYTE bEp);
```

パラメータ

名前	型	入出力
➢ bEp	BYTE	入力

説明

名前	説明
bEp	エンドポイントのアドレス [E.1.2.1]

戻り値

なし。

E.1.3.8 WDF_FIFOFull()

目的

- エンドポイントの FIFO (First In First Out) バッファがフルかどうかを確認します。

プロトタイプ

```
BOOL WDF_FIFOFull(BYTE bEp);
```

パラメータ

名前	型	入出力
➢ bEp	BYTE	入力

説明

名前	説明
bEp	エンドポイントのアドレス [E.1.2.1]

戻り値

エンドポイントの FIFO (First In First Out) バッファがフルの場合、TRUE を返します。フルでない場合、FALSE を返します。

E.1.3.9 WDF_FIFOEmpty()

目的

- エンドポイントの FIFO (First In First Out) バッファが空でないかを確認します。

プロトタイプ

```
BOOL WDF_FIFOEmpty(BYTE bEp);
```

パラメータ

名前	型	入出力
➢ bEp	BYTE	入力

説明

名前	説明
bEp	エンドポイントのアドレス [E.1.2.1]

戻り値

エンドポイントの FIFO (First In First Out) バッファが空の場合、TRUE を返します。空でない場合、FALSE を返します。

E.1.3.10 WDF_FIFOWrite()

目的

- エンドポイントの FIFO (First In First Out) バッファへデータを書き込みます。
  この関数は`WDF_SetEPByteCount()` [E.1.3.5] を呼び出した後に呼び出します。

プロトタイプ

```
void WDF_FIFOWrite(
    BYTE bEp,
    UINT uNumBytes,
    BYTE *pData);
```

パラメータ

名前	型	入出力
➢ bEp	BYTE	入力
➢ pData	BYTE*	入力
➢ uNumBytes	UINT	入力

説明

名前	説明
bEp	エンドポイントのアドレス [E.1.2.1]
pData	書き込むデータ バッファ
uNumBytes	書き込むバイト数

戻り値

なし。

E.1.3.11 WDF_FIFORead()

目的

- エンドポイントの FIFO (First In First Out) バッファからデータを読み取ります。
  読み取るバイト数を割り出すために、この関数は `WDF_GetEPByteCount()` [E.1.3.6] を呼び出す前に呼び出す必要があります。

プロトタイプ

```
void WDF_FIFORead(
    BYTE bEp,
    UINT uNumBytes,
    BYTE *pData);
```

パラメータ

名前	型	入出力
➢ bEp	BYTE	入力
➢ pData	BYTE*	出力
➢ uNumBytes	UINT	入力

説明

名前	説明
bEp	エンドポイントのアドレス [E.1.2.1]
pData	読み取るデータを保持するバッファ
uNumBytes	FIFO バッファから読み取るバイト数

戻り値

なし。

E.1.3.12 WDF_GetEPStatus()

目的

- エンドポイントの状況情報を取得します。

プロトタイプ

```
PEP_STATUS WDF_GetEPStatus(BYTE bEp);
```

パラメータ

名前	型	入出力
➢ bEp	BYTE	入力

説明

名前	説明
bEp	エンドポイントのアドレス [E.1.2.1]

戻り値

エンドポイントの状況情報 [E.1.2.6] を保持する構造体へのポインタを返します。

E.2 DriverWizard で生成されたファームウェアの API

このセクションでは、WinDriver USB Device の DriverWizard によって生成された Silicon Laboratories C8051F320 開発ボード用ファームウェア ライブラリの API について説明します。このセクションで説明する関数は、**F320\ include\periph.h** ヘッダー ファイルで宣言され、DriverWizard で生成された **periph.c** ファイルにおいて、ウィザードで定義されたデバイス設定情報に従って実装されます。

注意

コードを変更する場合、USB 規格および開発するボードのハードウェア要件を満たしているかどうかを確認してください(WinDriver ユーザーズ ガイドのセクション 16.4.3 を参照)。

E.2.1 WDF_USBReset()

目的

- デバイスの状況情報をゼロ (0) に初期化し、すべてのエンドポイントをリセットします。

プロトタイプ

```
void WDF_USBReset(void);
```

戻り値

なし。

E.2.2 WDF_SetAddressRequest()

目的

- SET ADDRESS 要求を処理します。

プロトタイプ

```
void WDF_SetAddressRequest(void);
```

戻り値

なし。

E.2.3 WDF_SetFeatureRequest()

目的

- SET ADDRESS 要求を処理します。

プロトタイプ

```
void WDF_SetFeatureRequest(void);
```

戻り値

なし。

E.2.4 WDF_ClearFeatureRequest()

目的

- CLEAR FEATURE 要求を処理します。

プロトタイプ

```
void WDF_ClearFeatureRequest(void);
```

戻り値

なし。

E.2.5 WDF_SetConfigurationRequest()

目的

- SET CONFIGURATION 要求を処理します。

プロトタイプ

```
void WDF_SetConfigurationRequest(void);
```

戻り値

なし。

E.2.6 WDF_SetDescriptorRequest()

目的

- SET DESCRIPTOR 要求を処理します。

プロトタイプ

```
void WDF_SetDescriptorRequest(void);
```

戻り値

なし。

E.2.7 WDF_SetInterfaceRequest()

目的

- SET INTERFACE 要求を処理します。

プロトタイプ

```
void WDF_SetInterfaceRequest(void);
```

戻り値

なし。

E.2.8 WDF_GetStatusRequest()

目的

- GET STATUS 要求を処理します。

プロトタイプ

```
void WDF_GetStatusRequest(void);
```

戻り値

なし。

E.2.9 WDF_GetDescriptorRequest()

目的

- GET DESCRIPTOR 要求を処理します。

プロトタイプ

```
void WDF_GetDescriptorRequest(void);
```

戻り値

なし。

E.2.10 WDF_GetConfigurationRequest()

目的

- GET CONFIGURATION 要求を処理します。

プロトタイプ

```
void WDF_GetConfigurationRequest(void);
```

戻り値

なし。

E.2.11 WDF_GetInterfaceRequest()

目的

- GET INTERFACE 要求を処理します。

プロトタイプ

```
void WDF_GetInterfaceRequest(void);
```

戻り値

なし。

付録 F
トラブルシューティングとサポート

開発者向けの技術情報が Web サイト http://www.xlsoft.com/jp/products/windriver/products.html より参照できます。以下の文書がありますので、参考にしてください。

- テクニカル ドキュメント
- FAQ
- サンプル コード
- クイック スタート ガイド

付録 G
評価版 (Evaluation Version) の制限

G.1 WinDriver Windows

- 毎回 WinDriver を起動すると評価版であることを示すメッセージが表示されます。
- DriverWizard を使用する際に、評価版を起動していることを知らせるメッセージのダイアログ ボックスが、ハードウェアと相互作用するたびに表示されます。
- DriverWizard:
  - 毎回 DriverWizard を起動すると評価版であることを示すメッセージが表示されます。
  - DriverWizard を使用してハードウェアと相互作用するたびに、評価版であることを示すメッセージが表示されます。
- WinDriver は最初のインストールから 30 日間だけ使用可能です。

G.2 WinDriver Windows CE

- 毎回 WinDriver を起動すると評価版であることを示すメッセージが表示されます。
- WinDriver CE Kernel (**windrvr6.dll**) は 1 度に 60 分間まで動作します。
- DriverWizard (Windows 2000/XP/Server 2003/Vista PC ホストで使用する場合):
  - 毎回 DriverWizard を起動すると評価版であることを示すメッセージが表示されます。
  - DriverWizard を使用してハードウェアと相互作用するたびに、評価版であることを示すメッセージが表示されます。
- Windows 2000/XP/Server 2003/Vista 上の WinDriver CE エミュレータは 30 日後に動作しなくなります。

G.3 WinDriver Linux

- 毎回 WinDriver を起動すると評価版であることを示すメッセージが表示されます。
- DriverWizard:
  - 毎回 DriverWizard を起動すると評価版であることを示すメッセージが表示されます。
  - DriverWizard を使用してハードウェアと相互作用するたびに、評価版であることを示すメッセージが表示されます。
- WinDriver のカーネル モジュールは 1 度に 60 分間まで動作します。
- WinDriver を継続して使用するには、次のコマンドを使用して WinDriver カーネル モジュールをリロード (モジュールの削除および挿入) します。

  削除するには:
  `/sbin/rmmod windrvr6`

  挿入するには:
  `/sbin/modprobe windrvr6`

付録 H
WinDriver の購入

Windows の [スタート] メニューの [プログラム] - [WinDriver] - [Order Form] にある申込用紙に記入し、電子メール、ファックス、または郵送してください。

WinDriver のライセンスを電子メール、またはファックスで返信致します。

お問い合わせ先:

エクセルソフト株式会社

〒108 - 0014
東京都港区芝 5 - 1 - 9 ブゼンヤビル 4F
Phone: 03 - 5440 - 7875 Fax: 03 - 5440 - 7876
メール: xlsoftkk@xlsoft.com
ホームページ: http://www.xlsoft.com/

付録 I
ドライバの配布 - 法律問題

WinDriver は、開発者ごとにライセンスされます。WinDriver の Node-Locked ライセンスは一台のマシンで一人の開発者が無制限の数のドライバを開発し、ロイヤリティなしで作成したドライバを配布することを許可します。

windrvr.h ファイル は配布できません。WinDriver の機能を説明した如何なるソース ファイルの配布もできません。WinDriver のライセンス契約書については、**WinDriver/docs/license.txt** ファイルを参照してください。

Node-Locked ライセンスの他に Floating ライセンスを用意しています。WinDriver の Floating ライセンスは、Node-Locked ライセンスとは異なり、一台のマシンに限定されず、複数のマシンで WinDriver を使用できる非常にフレキシブルなシングル ユーザー ライセンスです。開発環境を頻繁に変更する開発者や、複数のマシンを使用してドライバの開発を行う開発者に最適なライセンスです。

付録 J
その他のドキュメント

最新マニュアル

最新版の WinDriver ユーザーズ ガイドは、Jungo 社の Web サイトより入手可能です: 最新版の WinDriver 日本語ユーザーズ ガイドは、下記Jungo 社の Web サイトより入手可能です:
http://www.xlsoft.com/jp/products/download/download.html

バージョン履歴

WinDriver のバージョン履歴は、http://www.xlsoft.com/jp/products/windriver/wdversion.html を参照してくださいこの Web サイトでは、WinDriver の各バージョンで追加されたすべての新機能、強化および修正リストを参照することができます。

テクニカル ドキュメント

次の Web サイトから、テクニカル ドキュメント データベースも利用可能です:
http://www.xlsoft.com/jp/products/windriver/support/tech_docs_indexes/main_index.html
テクニカル ドキュメント データベースには、WinDriver の機能、ユーティリティ、API とその正しい使用方法についての詳細な説明、一般的な問題のトラブルシューティング、役立つヒント、よくある質問が含まれています。

WinDriver

ユーザーズ ガイド

2007 年 6 月 5 日

発行　**エクセルソフト株式会社**

〒108-0014 東京都港区芝5-1-9 ブゼンヤビル4F

TEL 03-5440-7875　FAX 03-5440-7876

E-MAIL: xlsoftkk@xlsoft.com

ホームページ: http://www.xlsoft.com/

Translated by

米国 XLsoft Corporation

12K Mauchly

Irvine, CA 92618 USA

URL: http://www.xlsoft.com/

E-Mail: sales@xlsoft.com