JUNGO

WinDriver

PCI / ISA / PCMCIA / CardBus
API 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

〒108-0073 東京都港区三田 3-9-9 森伝ビル 6F

TEL 03-5440-7875 FAX 03-5440-7876

E-MAIL: xlsoftkk@xlsoft.com

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

Rev. 10.10 - 12/2009

目次

付録 A
API 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 の .NET、Visual Basic および Delphi コードを 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 WDC ライブラリの概要

WinDriver カード "**WDC**" API は、便利なユーザーモード ラッパー関数を標準の WinDriver PCI / ISA / PCMCIA / CardBus WD_xxx API [A.4]へ提供します。

WDC ラッパーは PCI / ISA / PCMCIA / CardBus デバイスと通信するために WinDriver の使用法を簡素化するようにデザインされています。コードで基本的な WD_xxx PCI / PCMCIA / ISA WinDriver API を使用することができますが、代わりに高レベルの WDC API を使用することを推奨します。

注意: ほとんどの WDC API は、ユーザーモードおよびカーネルモード (WinDriver ユーザーズ ガイドの「Kernel PlugIn について」の章を参照してください) の両方から使用することができます。

DriverWizard で生成された PCI / PCMCIA / ISA 診断ドライバコード、PLX サンプルコードと同様に、**pci_diag**、Kernel PlugIn **kp_pci**、**pcmcia_diag** および **pci_dump** サンプルは WDC API を利用できます。

WDC API は **wdapi1010** DLL / 共有オブジェクトの一部で、**WinDriver¥redist¥wdapi1010.dll** (Windows 2000 / XP / Server 2003 / Server 2008 / Vista / 7 および Windows CE) / **WinDriver¥redist¥WINCE¥<TARGET_CPU>¥wdapi1010.dll** (Windows CE) / **WinDriver/lib/libwdapi1010.so** (Linux) に保存されています。WDC API のソース コードは、**WinDriver/src/wdapi** ディレクトリに保存されています。

WDC インターフェイスは、**wdc_lib.h** および **wdc_defs.h** ヘッダ ファイル (両ファイルとも **WinDriver/includes** ディレクトリに保存されています) から提供されます。

wdc_lib.h

高レベルの WDC API (型の定義、関数の宣言など) を宣言します。

wdc_defs.h

低レベルの WDC API を宣言します。このファイルは高レベルの **wdc_lib.h** ファイルによりカプセル化された定義と型の情報を含んでいます。

たとえば、WinDriver PCI / PCMCIA / ISA サンプルと DriverWizard で生成されたコードは、WDC API を利用し、特定のデバイス用のライブラリや診断アプリケーションで構成されます。高レベルの診断コードは **wdc_lib.h** API のみを利用しますが、ライブラリ コードは **wdc_defs.h** ファイルから低レベルの API も使用します。そのため希望するカプセル化のレベルを維持します。

以下のセクションでは WDC の 高レベルの API [A.2] および 低レベルの API [A.3] を説明します。

注意

- WinDriver の PCI API で **CardBus** デバイスを処理します。そのため、この章での **PCI** への言及には CardBus も含まれます。
- PCMCIA API (WDC ライブラリおよび低レベルの `WD_xxx` WinDriver API の両方とも) は Windows 2000 / XP / Server 2003 / Server 2008 / Vista / 7 でのみサポートされています。

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()` / `WDC_DriverOpen()` / `WDC_xxxDeviceOpen()` 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、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.2 WDC の高レベル API

このセクションでは **WinDriver/include/wdc_lib.h** ヘッダ ファイルで定義されている WDC API を説明します。

A.2.1 構造体、型、および一般的な定義

A.2.1.1 WDC_DEVICE_HANDLE

WDC デバイス情報構造体 [A.3.3] へのハンドル。

```
typedef void * WDC_DEVICE_HANDLE;
```

A.2.1.2 WDC_DRV_OPEN_OPTIONS の定義

```
typedef DWORD WDC_DRV_OPEN_OPTIONS;
```

WDC ライブラリ (`WDC_DriverOpen()` [A.2.2] を参照) へのハンドルが開かれる時に実行されるタスクを説明するフラグのプリプロセッサの定義。

# 定義値	説明
WDC_DRV_OPEN_CHECK_VER	コードで使用される WinDriver のソース ファイルのバージョンとロードされた WinDriver カーネルのバージョンを比べます。
WDC_DRV_OPEN_REG_LIC	WinDriver のライセンス登録文字列を登録します。

以下のプリプロセッサの定義は `WDC_DriverOpen()` [A.2.2] へ渡すことができる便利な WDC ドライバを開くオプションを提供します。

# 定義値	説明
WDC_DRV_OPEN_BASIC	主に WinDriver のカーネル モジュールへのハンドルを開く基本の WDC オープンのタスクのみを実行するように `WDC_Driveropen()` [A.2.2] へ指示します。

	注意: このオプションの値はゼロ (0) (<=> ドライバ オープンのフラグなし) であるため、このオプションはその他の WDC ドライバ オープン オプションと組み合わせることはできません。
WDC_DRV_OPEN_KP	Kernel PlugIn から `WDC_DriverOpen()` [A.2.2] を呼び出す時の便利なオプション。このオプションは Kernel PlugIn から WDC ライブラリへのハンドルを開くときに設定するオプションとして推奨される `WDC_DRV_OPEN_BASIC` フラグを設定するオプションに相当します。
WDC_DRV_OPEN_ALL	すべての基本的な WDC ドライバ オープン フラグ (`WDC_DRV_OPEN_CHECK_VER` および `WDC_DRV_OPEN_REG_REG_LIC`) の便利なマスク。 WinDriver のカーネル モジュールへのハンドルを開く基本的な機能は、常に `WDC_DriverOpen()` [A.2.2] により実行されます。そのため、`WDC_DRV_OPEN_BASIC` フラグを設定する必要はありません。
WDC_DRV_OPEN_DEFAULT	デフォルトの WDC オープン オプションを使用します。 • **ユーザーモード** アプリケーションの場合: `WDC_DRV_OPEN_ALL` の設定に相当します。 • **Kernel PlugIn** の場合: `WDC_DRV_OPEN_KP` の設定に相当します。

A.2.1.3 WDC_DIRECTION 列挙型

ドライバのアドレス / レジスタ アクセス方向の列挙型。

列挙値	説明
WDC_READ	アドレスからの読み取り
WDC_WRITE	アドレスへの書き込み
WDC_READ_WRITE	アドレスからの読み取りまたはアドレスへの書き込み。 この値は、レジスタのアクセス モード (読み取り可能 / 書き込み可能) を示すために、WinDriver サンプルや生成された DriverWizard 診断コードで使用されます。

A.2.1.4 WDC_ADDR_MODE 列挙型

メモリまたは I/O アドレス / レジスタの読み取り / 書き込みモードの列挙型。
この列挙値はメモリまたは I/O アドレス / レジスタが 8、16、32、または 64 ビット (1、2、4、または 8 バイト) の倍数で読み取り / 書き込みされているか確認するために使用されます。

列挙値	説明
WDC_MODE_8	8 ビット (1 バイト) モード
WDC_MODE_16	16 ビット(2 バイト) モード
WDC_MODE_32	32 ビット(4 バイト) モード
WDC_MODE_64	64 ビット(8 バイト) モード

A.2.1.5 WDC_ADDR_RW_OPTIONS 列挙型

メモリまたは I/O アドレスがどのように読み取り / 書き込みされているか確認するために使用されるフラグの列挙型。

列挙値	説明
WDC_ADDR_RW_DEFAULT	デフォルトの読み取り / 書き込みオプションを使用するように指示します。呼び出し元の処理からメモリ アドレスに直接アクセスします。ブロック転送は次のアドレスから実行されます (自動インクリメント)。 注意: このフラグの値はゼロ (0) (<=> 読み取り / 書き込みフラグなし) のため、ビット マスクでその他の読み取り / 書き込みフラグと**組み合わせることはできません。** このオプションは、`WDC_ReadAddr8/16/32/64()` 関数 [A.2.20] および `WDC_WriteAddr8/16/32/64()` 関数 [A.2.21] によって使用されます。
WDC_ADDR_RW_NO_AUTOINC	ブロック転送の読み取り / 書き込みアドレスを自動インクリメントしないようにします (ブロック (文字列) 転送のみ)。メモリまたは I/O アドレス ブロックの読み取り / 書き込みを行う際に、デバイスのアドレス定数を変更しないようにします。

A.2.1.6 WDC_ADDR_SIZE の定義

```
typedef DWORD WDC_ADDR_SIZE;
```

メモリおよび I/O アドレス / レジスタ サイズを表すプリプロセッサの定義。

# 定義値	説明
WDC_SIZE_8	8 ビット (1 バイト)
WDC_SIZE_16	16 ビット (2 バイト)
WDC_SIZE_32	32 ビット (4 バイト)
WDC_SIZE_64	64 ビット (8 バイト)

A.2.1.7 WDC_SLEEP_OPTIONS の定義

```
typedef DWORD WDC_SLEEP_OPTIONS;
```

`WDC_Sleep()` [A.2.55] へ渡されるスリープ オプションを表すプリプロセッサの定義。

# 定義値	説明
WDC_SLEEP_BUSY	CPU サイクルを消費して、実行を遅らせる (ビジー スリープ)
WDC_SLEEP_NON_BUSY	CPU サイクルを消費せずに、実行を遅らせる (ノンビジー スリープ) 注意: 最小 17,000 マイクロ秒。ビジー スリープよりも精度が落ちます。

A.2.1.8 WDC_DBG_OPTIONS の定義

```
typedef DWORD WDC_DBG_OPTIONS;
```

WDC_SetDebugOptions() [A.2.49] へ渡される WDC ライブラリの可能なデバッグ オプションを表すプリプロセッサの定義。

以下のフラグは WDC ライブラリのデバッグ メッセージ用の出力ファイルを決定します。

# 定義値	説明
WDC_DBG_OUT_DBM	WDC ライブラリからデバッグ メッセージをデバッグ モニタへ送ります。
WDC_DBG_OUT_FILE	WDC ライブラリからデバッグ メッセージをデバッグ ファイルへ送ります。デフォルトでは、異なったファイルが WDC_SetDebugOptions() 関数 [A.2.49] の **sDbgFile** パラメータで設定されない限り、デバッグ ファイルは **stderr** です。このオプションは、Kernel PlugIn とは反対にユーザーモードからのみサポートされます。

以下のフラグはデバッグ レベル (例: 表示する WDC デバッグ メッセージの種類など) を決定します。

# 定義値	説明
WDC_DBG_LEVEL_ERR	WDC エラー デバッグ メッセージのみ表示します。
WDC_DBG_LEVEL_TRACE	WDC エラー デバッグ メッセージおよび WDC トレースデバッグメッセージの両方を表示します。
WDC_DBG_NONE	WDC デバッグ メッセージを表示しません。

以下のプリプロセッサの定義は WDC_SetDebugOptions() [A.2.49] へ渡される便利なデバッグ フラッグの組み合わせを提供します。

- ユーザーモードおよび Kernel PlugIn の便利なデバッグ オプション。

# 定義値	説明
WDC_DBG_DEFAULT	WDC_DBG_OUT_DBM | WDC_DBG_LEVEL_TRACE : デフォルトのデバッグ オプションを使用します。WDC エラーおよびトレース メッセージをデバッグ モニタへ送ります。
WDC_DBG_DBM_ERR	WDC_DBG_OUT_DBM | WDC_DBG_LEVEL_ERR : WDC エラー デバッグ メッセージをデバッグ モニタへ送ります。
WDC_DBG_DBM_TRACE	WDC_DBG_OUT_DBM | WDC_DBG_LEVEL_TRACE : WDC エラーおよびトレース デバッグ メッセージをデバッグ モニタへ送ります。
WDC_DBG_FULL	フル WDC デバッグ: • **ユーザーモード** から WDC_DBG_OUT_DBM | WDC_DBG_OUT_FILE | WDC_DBG_LEVEL_TRACE : デバッグ モニタおよびデバッグ出力ファイル (デフォルトのファイル: **stderr**) の両方へ WDC エラー デバッグ メッセージおよび WDC トレースデバッグ メッセージを送ります。 • **Kernel PlugIn** から WDC_DBG_OUT_DBM | WDC_DBG_LEVEL_TRACE : WDC エラーおよびトレース メッセージをデバッグ モニターへ送ります。

- ユーザーモードのみの便利なデバッグ オプション。

# 定義値	説明
WDC_DBG_FILE_ERR	`WDC_DBG_OUT_FILE | WDC_DBG_LEVEL_ERR` : WDC エラー デバッグ メッセージをデバッグ ファイル (デフォルトのファイル: **stderr**) へ送ります。
WDC_DBG_FILE_TRACE	`WDC_DBG_OUT_FILE | WDC_DBG_LEVEL_TRACE` : WDC エラーおよびトレース デバッグ メッセージをデバッグ ファイル (デフォルトのファイル: **stderr**) へ送ります。
WDC_DBG_DBM_FILE_ERR	`WDC_DBG_OUT_DBM | WDC_DBG_OUT_FILE | WDC_DBG_LEVEL_ERR` : WDC エラー デバッグ メッセージをデバッグ モニタおよびデバッグ ファイル (デフォルトのファイル: **stderr**) へ送ります。
WDC_DBG_DBM_FILE_TRACE	`WDC_DBG_OUT_DBM | WDC_DBG_OUT_FILE | WDC_DBG_LEVEL_TRACE` : WDC エラーおよびデバッグ メッセージをデバッグ モニタおよびデバッグ ファイル (デフォルトのファイル: **stderr**) へ送ります。

A.2.1.9 WDC_SLOT_U の共用体

WDC PCI / PCMCIA デバイスのロケーション情報の共用体型。

フィールド	型	説明
➢ pciSlot	WD_PCI_SLOT	PCI デバイスのロケーション情報の構造体 [A.4.8]
➢ pcmciaSlot	WD_PCMCIA_SLOT	PCMCIA デバイスのロケーション情報の構造体 [A.4.9]

A.2.1.10 WDC_PCI_SCAN_RESULT 構造体

PCI バス スキャン (`WDC_PciScanDevices()` [A.2.4] を参照) の結果を保持する構造体型。

フィールド	型	説明
➢ dwNumDevices	DWORD	検索基準 (ベンダーおよびデバイス ID) に一致する PCI バスにあるデバイス数
➢ deviceId	WD_PCI_ID[WD_PCI_CARDS]	PCI バスにあるベンダーおよびデバイス ID と一致する配列 [A.4.6]
➢ deviceSlot	WD_PCI_SLOT[WD_PCI_CARDS]	検索基準に一致する PCI デバイスの場所に関する情報を保持する構造体 [A.4.8] の配列

A.2.1.11 WDC_PCMCIA_SCAN_RESULT 構造体

PCMCIA バス スキャン (`WDC_PcmciaScanDevices()` [A.2.6] を参照) の結果を保持するための構造体型。

フィールド	型	説明
➢ dwNumDevices	DWORD	検索基準 (製造元およびデバイス ID) と一致した PCMCIA であるデバイス数

➢ deviceId	WD_PCMCIA_ID[WD_PCMCIA_CARDS]	PCIMCIA バスにある一致するベンダーおよびデバイス ID の配列 [A.4.7]
➢ deviceSlot	WD_PCMCIA_SLOT[WD_PCMCIA_CARDS]	検索基準に一致する PCMCIA デバイスの場所に関する情報を保持する構造体 [A.4.9] の配列

A.2.2 WDC_DriverOpen()

目的

- WinDriver のカーネル モジュールへのハンドルを開き、保管します。オープン オプションに従って WDC ライブラリを初期化します。
  この関数は、その他の WDC API を呼び出す前に一度だけ呼び出します。

プロトタイプ

```
DWORD DLLCALLCONV WDC_DriverOpen(
    WDC_DRV_OPEN_OPTIONS openOptions,
    const CHAR *sLicense);
```

パラメータ

名前	型	入出力
➢ openOptions	WDC_DRV_OPEN_OPTIONS	入力
➢ sLicense	const CHAR*	入力

説明

名前	説明
openOptions	関数によって実行される初期化動作を決定するサポートされるオープン フラグのマスク [A.2.1.2]
sLicense	WinDriver ライセンス登録文字列。 この引数は、**openOptions**引数で WDC_DRV_OPEN_REG_LIC フラグ [A.2.1.2] が設定されていない場合、無視されます。 このパラメータが NULL ポインタまたは文字列の場合、この関数はデモの WinDriver 評価版ライセンスを登録しようとします。 WinDriver を評価しているときは、このパラメタに NULL ポインタを渡します。WinDriver ツールキットを登録した後、WinDriver ライセンス登録文字列を渡すコードを変更します。

戻り値

正常終了した場合、WD_STATUS_SUCCESS (0) を返します。そうでない場合、エラーコードを返します [A.8]。

A.2.3 WDC_DriverClose()

- (以前に `WDC_DriverOpen()` [A.2.2] への呼び出しにより取得され、保管された) WDC WinDriver ハンドルを閉じ、WDC ライブラリの終了処理を行います。

各 `WDC_DriverOpen()` の呼び出しには、WDC ライブラリを使用する必要がなくなった場合に実行する対応した `WDC_DriverClose()` の呼び出しがあります。

プロトタイプ

```
DWORD DLLCALLCONV WDC_DriverClose(void);
```

戻り値

正常終了した場合、WD_STATUS_SUCCESS (0) を返します。そうでない場合、エラーコードを返します [A.8]。

A.2.4 WDC_PciScanDevices()

目的

- 特定のベンダーおよびデバイス ID の組み合わせで全デバイス用の PCI バスをスキャンし、検出されたデバイスおよびそれらの場所に関する情報を返します。

注意: 稀に、正常に動作しないハードウェアの場合、関数のスキャン結果が、同じデバイスのインスタンスの繰り返しとなり、その結果、関数が正常なスキャン データを返さない可能性があります。その場合には、正常に動作しないデバイスを削除できない場合、WDC_PciScanDevicesByTopology() 関数を使用して、PCI バスをスキャンできます。

プロトタイプ

```
DWORD DLLCALLCONV WDC_PciScanDevices(
    DWORD dwVendorId,
    DWORD dwDeviceId,
    WDC_PCI_SCAN_RESULT *pPciScanResult);
```

パラメータ

名前	型	入出力
➢ dwVendorId	DWORD	入力
➢ dwDeviceId	DWORD	入力
➢ pPciScanResult	WDC_PCI_SCAN_RESULT*	出力

説明

名前	説明
dwVendorId	検索するベンダー ID (16 進数)。 ゼロ (0) はすべてのベンダー ID。
dwDeviceId	検索するデバイス ID (16 進数)。 ゼロ (0) はすべてのデバイス ID。

pPciScanResult	関数により、PCI バス スキャン [A.2.1.10] の結果でアップデートされる構造体へのポインタ。

戻り値

正常終了した場合、WD_STATUS_SUCCESS (0) を返します。そうでない場合、エラーコードを返します [A.8]。

注釈

- ベンダーおよびデバイス ID の両方をゼロ (0) に設定した場合、この関数は接続されたすべての PCI デバイスに関する情報を返します。

A.2.5 WDC_PciScanDevicesByTopology()

目的

- 特定のベンダーおよびデバイス ID の組み合わせですべてのデバイス用の PCI バスをスキャンし、検出されたデバイスおよびそれらの場所に関する情報を返します。この関数は、トポロジーによるスキャンを実行します。

注意: 複数のホスト コントローラを持つ場合、`WDC_PciScanDevicesByTopology()` 関数は最初のホスト コントローラのみをスキャンします。
デフォルトでは、`WDC_PciScanDevices()` 関数を使用して PCI バスをスキャンします。稀に、正常に動作しないハードウェアによって、同じデバイスのインスタンスを繰り返し返すために、`WDC_PciScanDevices()` 関数が失敗する場合に、`WDC_PciScanDevicesByTopology()` 関数を使用します。

プロトタイプ

```
DWORD DLLCALLCONV WDC_PciScanDevicesByTopology(
    DWORD dwVendorId,
    DWORD dwDeviceId,
    WDC_PCI_SCAN_RESULT *pPciScanResult);
```

パラメータ

名前	型	入出力
➢ dwVendorId	DWORD	入力
➢ dwDeviceId	DWORD	入力
➢ pPciScanResult	WDC_PCI_SCAN_RESULT*	出力

説明

名前	説明
dwVendorId	検索するベンダー ID (16 進数)。 ゼロ (0) はすべてのベンダー ID。
dwDeviceId	検索するデバイス ID (16 進数)。 ゼロ (0) はすべてのデバイス ID。

pPciScanResult	関数により、PCI バス スキャン [A.2.1.10] の結果でアップデートされる構造体へのポインタ。

戻り値

正常終了した場合、WD_STATUS_SUCCESS (0) を返します。そうでない場合、エラーコードを返します [A.8]。

注釈

- ベンダーおよびデバイス ID の両方をゼロ (0) に設定した場合、この関数は接続されたすべての PCI デバイスに関する情報を返します。

A.2.6 WDC_PcmciaScanDevices()

目的

- 特定の製造元およびデバイス ID の組み合わせで全デバイス用の PCMCIA をスキャンし、検出されたデバイスおよびそれらの場所に関する情報を返します。

プロトタイプ

```
DWORD DLLCALLCONV WDC_PcmciaScanDevices(
    WORD wManufacturerId,
    WORD wDeviceId,
    WDC_PCMCIA_SCAN_RESULT *pPcmciaScanResult);
```

パラメータ

名前	型	入出力
➢ wManufacturerId	WORD	入力
➢ wDeviceId	WORD	入力
➢ pPcmciaScanResult	WDC_PCMCIA_SCAN_RESULT*	出力

説明

名前	説明
wManufacturerId	検索する製造元 ID (16 進数)。 ゼロ (0) はすべての製造元 ID。
wDeviceId	検索するデバイス ID (16 進数)。 ゼロ (0) はすべてのデバイス ID。
pPcmciaScanResult	関数により、PCMCIA バス スキャン [A.2.1.11] の結果でアップデートされる構造体へのポインタ。

戻り値

正常終了した場合、WD_STATUS_SUCCESS (0) を返します。そうでない場合、エラーコードを返します [A.8]。

注釈

- ベンダーおよびデバイス ID の両方をゼロ (0) に設定した場合、この関数は接続されたすべての PCI デバイスに関する情報を返します。

A.2.7 WDC_PciGetDeviceInfo()

目的

- PCI デバイスのリソース情報 (メモリおよび I/O 範囲と割り込み情報) を取得します。

プロトタイプ

```
DWORD DLLCALLCONV WDC_PciGetDeviceInfo(
    WD_PCI_CARD_INFO *pDeviceInfo);
```

パラメータ

名前	型	入出力
➢ pDeviceInfo	WD_PCI_CARD_INFO*	入力 / 出力
☐ pciSlot	WD_PCI_SLOT	入力
☐ Card	WD_CARD	出力

説明

名前	説明
pDeviceInfo	PCI デバイス情報構造体へのポインタ [A.4.11]

戻り値

正常終了した場合、WD_STATUS_SUCCESS (0) を返します。そうでない場合、エラーコードを返します [A.8]。

注釈

- リソース 情報は情報が無い場合を除き、オペレーティング システムの Plug-and-Play マネージャから入手されます。情報が無い場合、PCI 設定レジスタから直接読み取られます。
  注意: Windows では、この関数を呼び出す前に WinDriver にデバイスを登録する **INF** ファイルをインストールする必要があります。(WinDriver での INF ファイルの作成に関する詳細は、WinDriver ユーザーズ ガイドのセクション 14.4 を参照)。
- 割り込み要求 (IRQ) 番号が Plug-and-Play マネージャから取得される場合、IRQ はマップされています。そのため物理的な IRQ の番号と異なる場合があります。

A.2.8 WDC_PcmciaGetDeviceInfo()

目的

- PCMCIA デバイスのリソース情報 (メモリと I/O 範囲および割り込み情報) を取得します。

プロトタイプ

```
DWORD DLLCALLCONV WDC_PcmciaGetDeviceInfo(
    WD_PCMCIA_CARD_INFO *pDeviceInfo);
```

パラメータ

名前	型	入出力
➢ pDeviceInfo	WD_PCMCIA_CARD_INFO*	入力 / 出力
☐ pcmciaSlot	WD_PCMCIA_SLOT	入力
☐ Card	WD_CARD	出力
☐ cVersion	CHAR	出力
☐ cManufacturer	CHAR [WD_PCMCIA_ MANUFACTURER_LEN]	出力
☐ cProductName	CHAR [WD_PCMCIA_ PRODUCTNAME_LEN]	出力
☐ wManufacturerId	WORD	出力
☐ wCardId	WORD	出力
☐ wFuncId	WORD	出力

説明

名前	説明
pDeviceInfo	PCMCIA デバイスの情報構造体へのポインタ [A.4.12]

戻り値

正常終了した場合、WD_STATUS_SUCCESS (0) を返します。そうでない場合、エラーコードを返します [A.8]。

注釈

- リソース 情報は情報が無い場合を除き、オペレーティング システムの Plug-and-Play マネージャから入手されます。情報が無い場合、PCMCIA 設定レジスタから直接読み取られます。
  注意: Windows では、この関数を呼び出す前に WinDriver にデバイスを登録する **INF** ファイルをインストールする必要があります。(WinDriver での INF ファイルの作成に関する詳細は、WinDriver ユーザーズ ガイドのセクション 14.4 を参照)。
- 割り込み要求 (IRQ) 番号が Plug-and-Play マネージャから取得される場合、IRQ はマップされています。そのため物理的な IRQ の番号と異なる場合があります。

A.2.9 WDC_PciDeviceOpen()

目的

- WDC PCI デバイス構造体を割り当て、初期化します。そして、デバイスを WinDriver へ登録し、デバイスへのハンドルを返します。

この関数で実行される動作

- デバイス上の共有できないメモリまたは I/O リソースが排他的に登録されていないことを確認します。
- デバイス上にある物理的メモリ範囲をカーネルモードおよびユーザーモードの両方のアドレス空間へマップし、将来使用するために、デバイス構造体にマップされたアドレスを保存します。
- デバイスとの通信をサポートするために必要なデバイスのリソース情報を保存します。
  たとえば、割り込み要求 (IRQ) 番号、割り込みの種類を保存し、後でユーザーが関数を呼び出しデバイスの割り込み処理を行う際に使用する割り込みハンドルも取得、保存します。
- 呼び出し元がデバイスとの通信に Kernel PlugIn ドライバを使用する場合、ドライバへのハンドルを開き、将来使用するために保存します。

プロトタイプ

```
DWORD DLLCALLCONV WDC_PciDeviceOpen(
    WDC_DEVICE_HANDLE *phDev,
    const WD_PCI_CARD_INFO *pDeviceInfo,
    const PVOID pDevCtx,
    PVOID reserved,
    const CHAR *pcKPDriverName,
    PVOID pKPOpenData);
```

パラメータ

名前	型	入出力
➢ phDev	WDC_DEVICE_HANDLE*	出力
➢ pDeviceInfo	const WD_PCI_CARD_INFO*	入力
☐ pciSlot	WD_PCI_SLOT	入力
☐ Card	WD_CARD	入力
◆ dwItems	DWORD	入力
◆ Item	WD_ITEMS[WD_CARD_ITEMS]	入力
◇ item	DWORD	入力
◇ fNotSharable	DWORD	入力
◇ I	union	入力
✦ Mem	struct	入力
→ dwPhysicalAddr	DWORD	入力
→ dwBytes	DWORD	入力
→ dwTransAddr	DWORD	N/A
→ dwUserDirectAddr	DWORD	N/A

→ dwCpuPhysicalAddr	DWORD	N/A
→ dwBar	DWORD	入力
✦ IO	struct	入力
→ dwAddr	DWORD	入力
→ dwBytes	DWORD	入力
→ dwBar	DWORD	入力
✦ Int	struct	入力
→ dwInterrupt	DWORD	入力
→ dwOptions	DWORD	入力
→ hInterrupt	DWORD	N/A
✦ Bus	struct	入力
→ dwBusType	WD_BUS_TYPE	入力
→ dwBusNum	DWORD	入力
→ dwSlotFunc	DWORD	入力
✦ Val	struct	N/A
➢ pDevCtx	const PVOID	入力
➢ reserved	PVOID	
➢ pcKPDriverName	const CHAR*	入力
➢ pKPOpenData	PVOID	入力

説明

名前	説明
phDev	この関数により割り当てられた WDC デバイスのハンドルへのポインタ
pDeviceInfo	PCI デバイスのリソース情報構造体へのポインタ [A.4.11]、オープンするデバイスに関する情報が含まれます。
pDevCtx	デバイス構造体に保存されるデバイスのコンテキスト情報へのポインタ
Reserved	Reserved for future use (予約)
pcKPDriverName	Kernel PlugIn ドライバ名。 アプリケーションが Kernel PlugIn を使用していない場合は、この引数に NULL ポインタを渡します。
pKPOpenData	`WD_KernelPlugInOpen()` へ渡される Kernel PlugIn ドライバのオープン データ。 アプリケーションが Kernel PlugIn ドライバを使用していない場合、この引数に NULL ポインタを渡します。

戻り値

正常終了した場合、WD_STATUS_SUCCESS (0) を返します。そうでない場合、エラーコードを返します [A.8]。

注釈

- この関数は**ユーザーモードからのみ**呼び出すことができます。
- カードがカーネルの仮想アドレス空間へ完全にマップできない広いメモリ範囲を持つ場合、`WDC_PciGetDeviceInfo()` [A.2.7] から入手したカードのリソース情報構造体で、このリソースに関連するアイテムを変更し、情報構造体 (`pDeviceInfo`) が `WDC_PciDeviceOpen()` へ渡される前にアイテムの `dwOptions` オプション フィールド (`pDeviceInfo->Card.Item[i].dwOptions`) で `WD_ITEM_DO_NOT_MAP_KERNEL` フラグを設定します。`WD_ITEM_DO_NOT_MAP_KERNEL` フラグは、カーネルのアドレス空間ではなくユーザーモードの仮想アドレス空間だけに関連したメモリ範囲をマップするよう関数に指示します。

  注意: このフラグを設定した場合、この関数により作成されるデバイスの情報構造体は、このリソースのカーネルのマップ アドレスを保持しません。(関連したメモリ範囲の `WDC_DEVICE`構造体 [A.3.3] で `pAddrDesc[i].kptAddr` はアップデートされません)。このため WinDriver API への呼び出しでこのマップに依存することはできません。Kernel PlugIn ドライバからメモリへアクセスする時も依存できません。

A.2.10 WDC_PcmciaDeviceOpen()

目的

- WDC PCMCIA デバイス構造体を割り当て、初期化します。デバイスを WinDriver へ登録し、デバイスへのハンドルを返します。

この関数で実行される動作

- デバイス上の共有できないメモリまたは I/O リソースが排他的に登録されていないことを確認します。
- デバイス上にある物理的メモリ範囲をカーネルモードおよびユーザーモードの両方のアドレス空間へマップし、将来使用するために、デバイス構造体にマップされたアドレスを保存します。
- デバイスとの通信をサポートするために必要なデバイスのリソース情報を保存します。
  たとえば、割り込み要求 (IRQ) 番号、割り込みの種類を保存し、後でユーザーが関数を呼び出しデバイスの割り込み処理を行う際に使用する割り込みハンドルも取得、保存します。
- 呼び出し元がデバイスとの通信に Kernel PlugIn ドライバを使用する場合、ドライバへのハンドルを開き、将来使用するために保存します。

プロトタイプ

```
DWORD DLLCALLCONV WDC_PcmciaDeviceOpen(
    WDC_DEVICE_HANDLE *phDev,
    const WD_PCMCIA_CARD_INFO *pDeviceInfo,
    const PVOID pDevCtx,
    PVOID reserved,
    const CHAR *pcKPDriverName,
    PVOID pKPOpenData);
```

パラメータ

名前	型	入出力
➢ phDev	WDC_DEVICE_HANDLE*	出力
➢ pDeviceInfo	const WD_PCMCIA_CARD_INFO*	入力
☐ pcmciaSlot	WD_PCMCIA_SLOT	入力
☐ Card	WD_CARD	入力
◆ dwItems	DWORD	入力
◆ Item	WD_ITEMS[WD_CARD_ITEMS]	入力
◇ item	DWORD	入力
◇ fNotSharable	DWORD	入力
◇ I	union	入力
✦ Mem	struct	入力
→ dwPhysicalAddr	DWORD	入力
→ dwBytes	DWORD	入力
→ dwTransAddr	DWORD	N/A
→ dwUserDirectAddr	DWORD	N/A
→ dwCpuPhysicalAddr	DWORD	N/A
→ dwBar	DWORD	入力
✦ IO	struct	入力
→ dwAddr	DWORD	入力
→ dwBytes	DWORD	入力
→ dwBar	DWORD	入力
✦ Int	struct	N/A
→ dwInterrupt	DWORD	入力
→ dwOptions	DWORD	入力
→ hInterrupt	DWORD	N/A
✦ Bus	struct	入力
→ dwBusType	WD_BUS_TYPE	入力
→ dwBusNum	DWORD	入力
→ dwSlotFunc	DWORD	入力
✦ Val	struct	N/A
☐ cVersion	CHAR	入力
☐ cManufacturer	CHAR [WD_PCMCIA_ MANUFACTURER_LEN]	入力
☐ cProductName	CHAR [WD_PCMCIA_	入力

	PRODUCTNAME_LEN]	
☐ wManufacturerId	WORD	入力
☐ wCardId	WORD	入力
☐ wFuncId	WORD	入力
➢ pDevCtx	const PVOID	入力
➢ reserved	PVOID	
➢ pcKPDriverName	const CHAR*	入力
➢ pKPOpenData	PVOID	入力

説明

名前	説明
phDev	この関数により割り当てられた WDC デバイスのハンドルへのポインタ
pDeviceInfo	PCMCIA デバイスのリソース情報構造体へのポインタ [A.4.12]、オープンするデバイスに関する情報が含まれます。
pDevCtx	デバイス構造体に保存されるデバイスのコンテキスト情報へのポインタ
reserved	Reserved for future use (予約)
pcKPDriverName	Kernel PlugIn ドライバ名。 アプリケーションが Kernel PlugIn を使用していない場合、この引数に NULL ポインタを渡します。
pKPOpenData	`WD_KernelPlugInOpen()` へ渡される Kernel PlugIn ドライバのオープン データ。 アプリケーションが Kernel PlugIn を使用していない場合、この引数に NULL ポインタを渡します。

戻り値

正常終了した場合、WD_STATUS_SUCCESS (0) を返します。そうでない場合、エラーコードを返します [A.8]。

注釈

- この関数は**ユーザーモードからのみ**呼び出すことができます。
- カードがカーネルの仮想アドレス空間へ完全にマップできない広いメモリ範囲を持つ場合、`WDC_PcmciaGetDeviceInfo()` [A.2.8] から入手したカードのリソース情報構造体で、このリソースに関連するアイテムを変更し、情報構造体 (`pDeviceInfo`) が `WDC_PciDeviceOpen()` へ渡される前にアイテムの `dwOptions` オプション フィールド (`pDeviceInfo->Card.Item[i].dwOptions`) で `WD_ITEM_DO_NOT_MAP_KERNEL` フラグを設定します。
  `WD_ITEM_DO_NOT_MAP_KERNEL` フラグは、カーネルのアドレス空間ではなくユーザーモードの仮想アドレス空間だけに関連したメモリ範囲をマップするよう関数に指示します。

  注意: このフラグを設定した場合、この関数により作成されるデバイスの情報構造体は、このリソースのカーネルのマップ アドレスを保持しません。(関連したメモリ範囲の `WDC_DEVICE`構造体 [A.3.3] で `pAddrDesc[i].kptAddr` はアップデートされません)。このため WinDriver

API への呼び出しでこのマップに依存することはできません。Kernel PlugIn ドライバからメモリへアクセスする時も依存できません。

A.2.11 WDC_IsaDeviceOpen()

目的

- WDC ISA デバイス構造体を割り当て、開始します。デバイスを WinDriver へ登録し、デバイスへのハンドルを返します。

この関数で実行される動作

- デバイス上の共有できないメモリまたは I/O リソースが排他的に登録されていないことを確認します。
- デバイス上にある物理的メモリ範囲をカーネルモードおよびユーザーモードの両方のアドレス空間へマップし、将来使用するために、デバイス構造体にマップされたアドレスを保存します。
- デバイスとの通信をサポートするために必要なデバイスのリソース情報を保存します。
  たとえば、割り込み要求 (IRQ) 番号、割り込みの種類を保存し、後でユーザーが関数を呼び出しデバイスの割り込み処理を行う際に使用する割り込みハンドルも取得、保存します。
- 呼び出し元がデバイスとの通信に Kernel PlugIn ドライバを使用する場合、ドライバへのハンドルを開き、将来使用するために保存します。

プロトタイプ

```
DWORD DLLCALLCONV WDC_IsaDeviceOpen(
    WDC_DEVICE_HANDLE *phDev,
    const WD_CARD *pDeviceInfo,
    const PVOID pDevCtx,
    PVOID reserved,
    const CHAR *pcKPDriverName,
    PVOID pKPOpenData);
```

パラメータ

名前	型	入出力
➢ phDev	WDC_DEVICE_HANDLE*	出力
➢ pDeviceInfo	const WD_CARD*	入力
☐ dwItems	DWORD	入力
☐ Item	WD_ITEMS[WD_CARD_ITEMS]	入力
◆ item	DWORD	入力
◆ fNotSharable	DWORD	入力
◆ dwOptions	DWORD	入力
◆ I	union	入力
◇ Mem	struct	入力
✦ dwPhysicalAddr	DWORD	入力
✦ dwBytes	DWORD	入力

✦ dwTransAddr	DWORD	N/A
✦ dwUserDirectAddr	DWORD	N/A
✦ dwCpuPhysicalAddr	DWORD	N/A
✦ dwBar	DWORD	入力
◇ IO	struct	入力
✦ dwAddr	DWORD	入力
✦ dwBytes	DWORD	入力
✦ dwBar	DWORD	入力
◇ Int	struct	入力
✦ dwInterrupt	DWORD	入力
✦ dwOptions	DWORD	入力
✦ hInterrupt	DWORD	N/A
◇ Bus	struct	入力
✦ dwBusType	WD_BUS_TYPE	入力
✦ dwBusNum	DWORD	入力
✦ dwSlotFunc	DWORD	入力
◇ Val	struct	N/A
➢ pDevCtx	const PVOID	入力
➢ reserved	PVOID	N/A
➢ pcKPDriverName	const CHAR*	入力
➢ pKPOpenData	PVOID	入力

説明

名前	説明
phDev	この関数により割り当てられた WDC デバイスのハンドルへのポインタ
pDeviceInfo	カード情報構造体へのポインタ [A.4.10]、オープンするデバイスに関する情報が含まれます。
pDevCtx	デバイス構造体に保存されるデバイスのコンテキスト情報へのポインタ
reserved	Reserved for future use (予約)
pcKPDriverName	Kernel PlugIn ドライバ名。 アプリケーションが Kernel PlugIn を使用していない場合、この引数に NULL ポインタを渡します。
pKPOpenData	`WD_KernelPlugInOpen()` へ渡される Kernel PlugIn ドライバのオープン データ。 アプリケーションが Kernel PlugIn を使用していない場合、この引数に NULL ポインタを渡します。

戻り値

正常終了した場合、WD_STATUS_SUCCESS (0) を返します。そうでない場合、エラーコードを返します [A.8]。

注釈

- この関数は**ユーザーモードからのみ**呼び出すことができます。
- カードがカーネルの仮想アドレス空間へ完全にマップできない広いメモリ範囲を持つ場合、カーネルのアドレス空間ではなくユーザーモードの仮想アドレス空間だけに関連したメモリ範囲をマップするよう関数に指示するために、関連したメモリの `WD_ITEMS` 構造体の WD_ITEM_OPTIONS フラグを設定することができます (`pDeviceInfo->Card.Item[i].dwOptions`)。

  注意: このフラグを設定した場合、この関数により作成されるデバイスの情報構造体は、このリソースのカーネルのマップ アドレスを保持しません。(関連したメモリ範囲の `WDC_DEVICE`構造体 [A.3.3] で `pAddrDesc[i].kptAddr` はアップデートされません)。このため WinDriver API への呼び出しでこのマップに依存することはできません。Kernel PlugIn ドライバからメモリへアクセスする時も依存できません。

A.2.12 WDC_PciDeviceClose()

目的

- WDC PCI デバイスの終了処理を実行し、割り当てられたメモリを解放します。

プロトタイプ

```
DWORD DLLCALLCONV WDC_PciDeviceClose(WDC_DEVICE_HANDLE hDev);
```

パラメータ

名前	型	入出力
➢ hDev	WDC_DEVICE_HANDLE	入力

説明

名前	説明
hDev	`WDC_PciDeviceOpen()` [A.2.9] により返される WDC PCI デバイスの構造体へのハンドル

戻り値

正常終了した場合、WD_STATUS_SUCCESS (0) を返します。そうでない場合、エラーコードを返します [A.8]。

注釈

- この関数は**ユーザーモードからのみ**呼び出すことができます。

A.2.13 WDC_PcmciaDeviceClose()

目的

- WDC PCMCIA デバイスの終了処理を実行し、割り当てられたメモリを解放します。

プロトタイプ

```
DWORD DLLCALLCONV WDC_PcmciaDeviceClose(WDC_DEVICE_HANDLE
hDev);
```

パラメータ

名前	型	入出力
➢ hDev	WDC_DEVICE_HANDLE	入力

説明

名前	説明
hDev	`WDC_PcmciaDeviceOpen()` [A.2.10] により返される WDC PCMCIA デバイスの構造体へのハンドル

戻り値

正常終了した場合、WD_STATUS_SUCCESS (0) を返します。そうでない場合、エラーコードを返します [A.8]。

注釈

- この関数は**ユーザーモードからのみ**呼び出すことができます。

A.2.14 WDC_IsaDeviceClose()

目的

- WDC ISA デバイスの終了処理を実行し、割り当てられたメモリを解放します。

プロトタイプ

```
DWORD DLLCALLCONV WDC_IsaDeviceClose(WDC_DEVICE_HANDLE hDev);
```

パラメータ

名前	型	入出力
➢ hDev	WDC_DEVICE_HANDLE	入力

説明

名前	説明
hDev	`WDC_IsaDeviceOpen()` [A.2.11] により返される WDC ISA デバイスの構造体へのハンドル

戻り値

正常終了した場合、WD_STATUS_SUCCESS (0) を返します。そうでない場合、エラーコードを返します [A.8]。

注釈

- この関数は**ユーザーモードからのみ**呼び出すことができます。

A.2.15 WDC_CardCleanupSetup()

目的

- 次の場合に、指定したカードで実行されるクリーンアップ転送コマンドのリストを設定します。
  - アプリケーションが異常終了した場合
  - カードを終了せずに、アプリケーションが正常終了した場合
  - `bForceCleanup` パラメータが `TRUE` に設定されている場合 (カード終了時に、実行されます)

プロトタイプ

```
DWORD WDC_CardCleanupSetup(
    WDC_DEVICE_HANDLE hDev,
    WD_TRANSFER *Cmd,
    DWORD dwCmds,
    BOOL bForceCleanup);
```

パラメータ

名前	型	入出力
➢ hDev	WDC_DEVICE_HANDLE	入力
➢ Cmd	WD_TRANSFER*	入力
➢ dwCmds	DWORD	入力
➢ bForceCleanup	BOOL	入力

説明

名前	説明
hDev	`WDC_xxxDeviceOpen()` (PCI [A.2.9] / PCMCIA [A.2.10] / ISA [A.2.11]) により返される WDC へのハンドル

Cmd	実行されるクリーンアップ転送コマンド配列へのポインタ [A.4.14]
dwCmds	Cmd 配列のクリーンアップ コマンドの数
bForceCleanup	**FALSE** に設定すると、クリーンアップ転送コマンド (Cmd) は、次のいずれかの場合に実行されます。 • アプリケーションが異常終了した場合 • WDC_xxxDeviceClose() 関数 (PCI [A.2.12] / PCMCIA [A.2.13] / ISA [A.2.14]) を呼び出してカードを終了せずに、アプリケーションが正常終了した場合 **TRUE** に設定すると、クリーンアップ転送コマンドは、上記の 2 つの場合に加え、次の場合にも実行されます。 • カードに対して WD_xxxDeviceClose() 関数が呼び出された場合

戻り値

正常終了した場合、WD_STATUS_SUCCESS (0) を返します。そうでない場合、エラーコードを返します [A.8]。

A.2.16 WDC_KernelPlugInOpen()

目的

- Kernel PlugIn ドライバへのハンドルを開きます。

プロトタイプ

```
DWORD DLLCALLCONV WDC_KernelPlugInOpen(
    WDC_DEVICE_HANDLE hDev,
    const CHAR *pcKPDriverName,
    PVOID pKPOpenData);
```

パラメータ

名前	型	入出力
➢ HDev	WDC_DEVICE_HANDLE	入力 / 出力
➢ pcKPDriverName	const CHAR*	入力
➢ pKPOpenData	PVOID	入力

説明

名前	説明
hDev	WDC_xxxDeviceOpen() (PCI [A.2.9] / PCMCIA [A.2.10] / ISA [A.2.11]) により返される WDC デバイスへのハンドル
pcKPDriverName	Kernel PlugIn ドライバの名前
pKPOpenData	WD_KernelPlugInOpen() へ渡される Kernel PlugIn ドライバのオープン データ

戻り値

正常終了した場合、WD_STATUS_SUCCESS (0) を返します。そうでない場合、エラーコードを返します [A.8]。

注釈

- `WDC_xxxDeviceOpen()` 関数 (PCI [A.2.9] / PCMCIA [A.2.10] / ISA [A.2.11]) の説明のとおり、デバイスへのハンドルを開く際にKernel PlugIn ドライバへのハンドルも開くことができるため、通常この関数を呼び出す必要はありません。
  この関数は、.NET アプリケーションから Kernel PlugIn へのハンドルを開く場合に使用します。WinDriver Kernel PlugIn サンプルは、割り当てられるデバイス ハンドルのアドレス (例: オープン関数の **`phDev`** パラメータおよび Kernel PlugIn の **`pKPOpenData`** オープン データ パラメータ) を渡します。.NET ではこれがサポートされていないため、Kernel PlugIn へのハンドルは個別の関数呼び出しで開かれます。

A.2.17 WDC_CallKerPlug()

目的

- ユーザーモード アプリケーションから Kernel PlugIn ドライバへメッセージを送ります。この関数はアプリケーションから指定のメッセージ ID の処理を実装する Kernel PlugIn の `KP_Call()` [A.5.4] 関数へメッセージ ID を渡し、Kernel PlugIn からユーザーモード アプリケーションへ結果を返します。

プロトタイプ

```
DWORD DLLCALLCONV WDC_CallKerPlug(
    WDC_DEVICE_HANDLE hDev,
    DWORD dwMsg,
    PVOID pData,
    PDWORD pdwResult);
```

パラメータ

名前	型	入出力
➢ hDev	WDC_DEVICE_HANDLE	入力
➢ dwMsg	DWORD	入力
➢ pData	PVOID	入力 / 出力
➢ pdwResult	pdwResult	出力

説明

名前	説明
Hdev	`WDC_xxxDeviceOpen()` (PCI [A.2.9] / PCMCIA [A.2.10] / ISA [A.2.11]) により返される WDC デバイスへのハンドル
DwMsg	Kernel PlugIn ドライバ (`KP_Call()` [A.5.4]) へ渡すメッセージ ID
pData	Kernel PlugIn ドライバとユーザーモード アプリケーション間で渡すデータへのポインタ

pdwResult	Kernel PlugIn ドライバ (KP_Call()) により返される、送られたメッセージに対してカーネルで実行された処理の結果

戻り値

正常終了した場合、WD_STATUS_SUCCESS (0) を返します。そうでない場合、エラーコードを返します [A.8]。

A.2.18 WDC_ReadMemXXX()

目的

- WDC_ReadMem8/16/32/64() は、それぞれ 1 バイト (8 ビット) / 2 バイト (16 ビット) / 4 バイト (32 ビット) / 8 バイト (64 ビット) を指定のメモリ アドレスから読み取ります。このアドレスは、呼び出しコンテキスト (ユーザーモード / カーネルモード) で直接読み取られます。

プロトタイプ

```
BYTE WDC_ReadMem8(addr, off);
WORD WDC_ReadMem16(addr, off);
UINT32 WDC_ReadMem32(addr, off);
UINT64 WDC_ReadMem64(addr, off);
```

注意: WDC_ReadMemXXX API はマクロとして実装されます。上記のプロトタイプでは、予測される戻り値を強調するために関数宣言構文を使用しています。

パラメータ

名前	型	入出力
➢ addr	DWORD	入力
➢ off	DWORD	入力

説明

名前	説明
addr	読み取るメモリ アドレス領域
off	読み取る指定アドレス領域 (addr) の初めからのオフセット

戻り値

特定のアドレスから読み取られたデータを返します。

A.2.19 WDC_WriteMemXXX()

目的

- WDC_WriteMem8/16/32/64() は、それぞれ 1 バイト (8 ビット) / 2 バイト (16 ビット) / 4 バイト (32 ビット) / 8 バイト (64 ビット) を指定のメモリ アドレスへ書き込みます。このアドレスは、呼び出しコンテキスト (ユーザーモード / カーネルモード) で直接書き込まれます。

プロトタイプ

```
void WDC_WriteMem8(addr, off, val);
void WDC_WriteMem16(addr, off, val);
void WDC_WriteMem32(addr, off, val);
void WDC_WriteMem64(addr, off, val);
```

注意: WDC_WriteMemXXX API はマクロとして実装されます。上記のプロトタイプでは、予測される戻り値を強調するために関数宣言構文を使用しています。

パラメータ

名前	型	入出力
➢ Addr	DWORD	入力
➢ Off	DWORD	入力
➢ Val	BYTE / WORD / UINT32 / UINT64	入力

説明

名前	説明
addr	書き込むメモリ アドレス領域
off	書き込む指定アドレス領域 (addr) の初めからのオフセット
val	指定のアドレスへ書き込むデータ

戻り値

なし

A.2.20 WDC_ReadAddrXXX()

目的

- WDC_ReadAddr8/16/32/64() は、それぞれ 1 バイト (8 ビット) / 2 バイト (16 ビット) / 4 バイト (32 ビット) / 8 バイト (64 ビット) を指定のメモリ アドレス、または I/O アドレスから読み取ります。

プロトタイプ

```
DWORD DLLCALLCONV WDC_ReadAddr8(WDC_DEVICE_HANDLE hDev,
    DWORD dwAddrSpace, KPTR dwOffset, BYTE *val);
```

```
DWORD DLLCALLCONV WDC_ReadAddr16(WDC_DEVICE_HANDLE hDev,
    DWORD dwAddrSpace, KPTR dwOffset, WORD *val);

DWORD DLLCALLCONV WDC_ReadAddr32(WDC_DEVICE_HANDLE hDev,
    DWORD dwAddrSpace, KPTR dwOffset, UINT32 *val);

DWORD DLLCALLCONV WDC_ReadAddr64(WDC_DEVICE_HANDLE hDev,
    DWORD dwAddrSpace, KPTR dwOffset, UINT64 *val);
```

パラメータ

名前	型	入出力
➢ hDev	WDC_DEVICE_HANDLE	入力
➢ dwAddrSpace	DWORD	入力
➢ dwOffset	KPTR	入力
➢ val	BYTE* / WORD* / UINT32* / UINT64*	出力

説明

名前	説明
hDev	`WDC_xxxDeviceOpen()` (PCI [A.2.9] / PCMCIA [A.2.10] / ISA [A.2.11]) により返される WDC デバイスへのハンドル
dwAddrSpace	読み取るメモリまたは I/O アドレス領域
dwOffset	読み取る指定アドレス領域 (`dwAddrSpace`) の初めからのオフセット
val	指定のアドレスから読み取られるデータで埋められるバッファへのポインタ

戻り値

正常終了した場合、WD_STATUS_SUCCESS (0) を返します。そうでない場合、エラーコードを返します [A.8]。

A.2.21 WDC_WriteAddrXXX()

目的

- `WDC_WriteAddr8/16/32/64()` は、それぞれ 1 バイト (8 ビット) / 2 バイト (16 ビット) / 4 バイト (32 ビット) / 8 バイト (64 ビット) を指定のメモリ アドレス、または I/O アドレスへ書き込みます。

プロトタイプ

```
DWORD DLLCALLCONV WDC_WriteAddr8(WDC_DEVICE_HANDLE hDev,
    DWORD dwAddrSpace, KPTR dwOffset, BYTE val)
```

```
DWORD DLLCALLCONV WDC_WriteAddr16(WDC_DEVICE_HANDLE hDev,
    DWORD dwAddrSpace, KPTR dwOffset, WORD val);

DWORD DLLCALLCONV WDC_WriteAddr32(WDC_DEVICE_HANDLE hDev,
    DWORD dwAddrSpace, KPTR dwOffset, UINT32 val);

DWORD DLLCALLCONV WDC_WriteAddr64(WDC_DEVICE_HANDLE hDev,
    DWORD dwAddrSpace, KPTR dwOffset, UINT64 val);
```

パラメータ

名前	型	入出力
➢ hDev	WDC_DEVICE_HANDLE	入力
➢ dwAddrSpace	DWORD	入力
➢ dwOffset	KPTR	入力
➢ val	BYTE / WORD / UINT32 / UINT64	入力

説明

名前	説明
hDev	`WDC_xxxDeviceOpen()` (PCI [A.2.9] / PCMCIA [A.2.10] / ISA [A.2.11]) により返される WDC デバイスへのハンドル
dwAddrSpace	書き込むメモリまたは I/O アドレス領域
dwOffset	書き込む指定アドレス領域 (`dwAddrSpace`) の初めからのオフセット
Val	指定のアドレスへ書き込むデータ

戻り値

正常終了した場合、WD_STATUS_SUCCESS (0) を返します。そうでない場合、エラーコードを返します [A.8]。

A.2.22 WDC_ReadAddrBlock()

目的

- デバイスからデータ ブロックを読み取ります。

プロトタイプ

```
DWORD DLLCALLCONV WDC_ReadAddrBlock(
    WDC_DEVICE_HANDLE hDev,
    DWORD dwAddrSpace,
    KPTR dwOffset,
    DWORD dwBytes,
```

```
    PVOID pData,
    WDC_ADDR_MODE mode,
    WDC_ADDR_RW_OPTIONS options);
```

パラメータ

名前	型	入出力
➢ hDev	WDC_DEVICE_HANDLE	入力
➢ dwAddrSpace	DWORD	入力
➢ dwOffset	KPTR	入力
➢ dwBytes	DWORD	入力
➢ pData	PVOID	出力
➢ mode	WDC_ADDR_MODE	入力
➢ options	WDC_ADDR_RW_OPTIONS	入力

説明

名前	説明
hDev	`WDC_xxxDeviceOpen()` (PCI [A.2.9] / PCMCIA [A.2.10] / ISA [A.2.11]) により返される WDC デバイスへのハンドル
dwAddrSpace	読み取るメモリまたは I/O アドレス領域
dwOffset	読み取る指定アドレス領域 (`dwAddrSpace`) の初めからのオフセット
dwBytes	読み取るバイト数
pData	デバイスから読み取られるデータで埋められるバッファへのポインタ
mode	リード アクセス モード (`WDC_ADDR_MODE` [A.2.1.4] を参照)
options	データがどのように読み取られるかを決定するビット マスク (`WDC_ADDR_RW_OPTIONS` [A.2.1.5] を参照)。 この関数は、自動的に `WDC_RW_BLOCK` フラグを設定します。

戻り値

正常終了した場合、WD_STATUS_SUCCESS (0) を返します。そうでない場合、エラーコードを返します [A.8]。

A.2.23 WDC_WriteAddrBlock()

目的

- データ ブロックをデバイスへ書き込みます。

プロトタイプ

```
DWORD DLLCALLCONV WDC_WriteAddrBlock(
```

```
    WDC_DEVICE_HANDLE hDev,
    DWORD dwAddrSpace,
    KPTR dwOffset,
    DWORD dwBytes,
    PVOID pData,
    WDC_ADDR_MODE mode,
    WDC_ADDR_RW_OPTIONS options);
```

パラメータ

名前	型	入出力
➢ hDev	WDC_DEVICE_HANDLE	入力
➢ dwAddrSpace	DWORD	入力
➢ dwOffset	KPTR	入力
➢ dwBytes	DWORD	入力
➢ pData	PVOID	入力
➢ mode	WDC_ADDR_MODE	入力
➢ options	WDC_ADDR_RW_OPTIONS	入力

説明

名前	説明
hDev	`WDC_xxxDeviceOpen()` (PCI [A.2.9] / PCMCIA [A.2.10] / ISA [A.2.11]) により返される WDC デバイスへのハンドル
dwAddrSpace	書き込むメモリまたは I/O アドレス領域
dwOffset	書き込む指定アドレス領域 (`dwAddrSpace`) の初めからのオフセット
dwBytes	書き込むバイト数
pData	デバイスへ書き込むデータを保持するバッファへのポインタ
mode	書き込みアクセス モード (`WDC_ADDR_MODE` [A.2.1.4] を参照)
options	データがどのように書き込まれるかを決定するビット マスク (`WDC_ADDR_RW_OPTIONS` [A.2.1.5] を参照)。 この関数は、自動的に `WDC_RW_BLOCK` フラグを設定します。

戻り値

正常終了した場合、WD_STATUS_SUCCESS (0) を返します。そうでない場合、エラーコードを返します [A.8]。

A.2.24 WDC_MultiTransfer()

目的

- メモリ グループおよび / または I/O グループの読み取り / 書き込み転送を実行します。

プロトタイプ

```
DWORD DLLCALLCONV WDC_MultiTransfer(
    WD_TRANSFER *pTrans,
    DWORD dwNumTrans);
```

パラメータ

名前	型	入出力
➢ pTrans	WD_TRANSFER*	
➢ dwNumTrans	DWORD	入力

説明

名前	説明
pTrans	転送コマンドの情報構造体 [A.4.14] の配列へのポインタ
dwNumTrans	`pTrans`配列の転送コマンド数

戻り値

正常終了した場合、WD_STATUS_SUCCESS (0) を返します。そうでない場合、エラーコードを返します [A.8]。

注釈

- この転送は、カーネルで指定したアドレスを読み取る / 書き込む低レベルの `WD_MultiTransfer()` WinDriver 関数を使用して実行されます。
- メモリ アドレスは (I/O アドレスのように)、カーネルで読み込み/書き込みされます。ユーザーモードでは直接読み込み / 書き込みされません。そのため、この関数に渡されるポート アドレス (メモリ、I/O アドレスともに) は、デバイスの構造体 [A.3.3] に保存されている物理的なアドレスのカーネルモードのマップである必要があります。

A.2.25 WDC_AddrSpaceIsActive()

目的

- 指定したメモリまたは I/O アドレス空間がアクティブかどうかを確認します (たとえば、サイズがゼロ (0) でないか)。

プロトタイプ

```
BOOL DLLCALLCONV WDC_AddrSpaceIsActive(
    WDC_DEVICE_HANDLE hDev,
```

```
    DWORD dwAddrSpace);
```

パラメータ

名前	型	入出力
➢ hDev	WDC_DEVICE_HANDLE	入力
➢ dwAddrSpace	DWORD	入力

説明

名前	説明
hDev	`WDC_xxxDeviceOpen()` (PCI [A.2.9] / PCMCIA [A.2.10] / ISA [A.2.11]) により返される WDC デバイスへのハンドル
dwAddrSpace	検索するメモリまたは I/O アドレス空間

戻り値

指定したアドレス空間がアクティブの場合、TRUE を返します。そうでない場合 FALSE を返します。

A.2.26 WDC_PciReadCfgBySlot()

目的

- PCI デバイスの設定空間または PCI Express デバイスの拡張設定空間で指定したオフセットからデータを読み取ります。
  このデバイスは PCI バス上の場所によって識別されます。

Windows と Linux では、PCI Express の拡張設定空間へのアクセスを対象のプラットフォームでサポートします。以下の説明において “PCI” へのすべての言及には PCI Express も含まれています。

プロトタイプ

```
DWORD DLLCALLCONV WDC_PciReadCfgBySlot(
    WD_PCI_SLOT *pPciSlot,
    DWORD dwOffset,
    PVOID pData,
    DWORD dwBytes);
```

パラメータ

名前	型	入出力
➢ pPciSlot	WD_PCI_SLOT*	入力
➢ dwOffset	DWORD	入力
➢ pData	PVOID	出力
➢ dwBytes	DWORD	入力

説明

名前	説明
pPciSlot	`WDC_PciScanDevices()` [A.2.4] の呼び出しにより取得することができるデバイスの場所情報構造体 [A.4.8] へのポインタ
dwOffset	読み取る PCI 設定空間の初めからのオフセット
pData	PCI 設定空間から読み取られるデータで埋められるバッファへのポインタ
dwBytes	読み取るバイト数

戻り値

正常終了した場合、WD_STATUS_SUCCESS (0) を返します。そうでない場合、エラーコードを返します [A.8]。

A.2.27 WDC_PciWriteCfgBySlot()

目的

- PCI デバイスの設定空間または PCI Express デバイスの拡張設定空間で指定したオフセットへデータを書き込みます。
  このデバイスは PCI バス上の場所によって識別されます。

Windows と Linux では、PCI Express の拡張設定空間へのアクセスを対象のプラットフォームでサポートします。以下の説明において “PCI” へのすべての言及には PCI Express も含まれています。

プロトタイプ

```
DWORD DLLCALLCONV WDC_PciWriteCfgBySlot(
    WD_PCI_SLOT *pPciSlot,
    DWORD dwOffset,
    PVOID pData,
    DWORD dwBytes);
```

パラメータ

名前	型	入出力
➢ pPciSlot	WD_PCI_SLOT*	入力
➢ dwOffset	DWORD	入力
➢ pData	PVOID	入力
➢ dwBytes	DWORD	入力

説明

名前	説明
pPciSlot	`WDC_PciScanDevices()` [A.2.4] の呼び出しにより取得すること

	ができるデバイスの場所情報構造体 [A.4.8] へのポインタ
dwOffset	書き込む PCI 設定空間の初めからのオフセット
pData	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
dwBytes	書き込むバイト数

戻り値

正常終了した場合、WD_STATUS_SUCCESS (0) を返します。そうでない場合、エラーコードを返します [A.8]。

A.2.28 WDC_PciReadCfg()

目的

- PCI デバイスの設定空間または PCI Express デバイスの拡張設定空間で指定したオフセットからデータを読み取ります。
  Windows と Linux では、PCI Express の拡張設定空間へのアクセスを対象のプラットフォームでサポートします。以下の説明において “PCI” へのすべての言及には PCI Express も含まれています。

プロトタイプ

```
DWORD DLLCALLCONV WDC_PciReadCfg(
    WDC_DEVICE_HANDLE hDev,
    DWORD dwOffset,
    PVOID pData,
    DWORD dwBytes);
```

パラメータ

名前	型	入出力
➢ hDev	WDC_DEVICE_HANDLE	入力
➢ dwOffset	DWORD	入力
➢ pData	PVOID	出力
➢ dwBytes	DWORD	入力

説明

名前	説明
hDev	`WDC_PciDeviceOpen()` [A.2.9] により返される WDC PCI デバイスの構造体へのハンドル
dwOffset	読み取る PCI 設定空間の初めからのオフセット
pData	PCI 設定空間から読み取られるデータで埋められるバッファへのポインタ

dwBytes	読み取るバイト数

戻り値

正常終了した場合、WD_STATUS_SUCCESS (0) を返します。そうでない場合、エラーコードを返します [A.8]。

A.2.29 WDC_PciWriteCfg()

目的

- PCI デバイスの設定空間または PCI Express デバイスの拡張設定空間で指定したオフセットへデータを書き込みます。

Windows と Linux では、PCI Express の拡張設定空間へのアクセスを対象のプラットフォームでサポートします。以下の説明において “PCI” へのすべての言及には PCI Express も含まれています。

プロトタイプ

```
DWORD DLLCALLCONV WDC_PciWriteCfg(
    WDC_DEVICE_HANDLE hDev,
    DWORD dwOffset,
    PVOID pData,
    DWORD dwBytes);
```

パラメータ

名前	型	入出力
➢ hDev	WDC_DEVICE_HANDLE	入力
➢ dwOffset	DWORD	入力
➢ pData	PVOID	入力
➢ dwBytes	DWORD	入力

説明

名前	説明
hDev	`WDC_PciDeviceOpen()` [A.2.9] により返される WDC PCI デバイスの構造体へのハンドル
dwOffset	書き込む PCI 設定空間の初めからのオフセット
pData	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
dwBytes	書き込むバイト数

戻り値

正常終了した場合、WD_STATUS_SUCCESS (0) を返します。そうでない場合、エラーコードを返します [A.8]。

A.2.30 WDC_PciReadCfgBySlotXXX()

目的

- `WDC_PciReadCfgBySlot8/16/32/64()` は、それぞれ 1 バイト (8 ビット) / 2 バイト (16 ビット) / 4 バイト (32 ビット) / 8 バイト (64 ビット) を指定の PCI デバイスの設定空間または PCI Express デバイスの拡張設定空間でのオフセットから読み取ります。
  このデバイスは PCI バス上の場所によって識別されます。

Windows と Linux では、PCI Express の拡張設定空間へのアクセスを対象のプラットフォームでサポートします。以下の説明において “PCI” へのすべての言及には PCI Express も含まれています。

プロトタイプ

```
DWORD DLLCALLCONV WDC_PciReadCfgRegBySlot8(
    WD_PCI_SLOT *pPciSlot, DWORD dwOffset, BYTE *val);

DWORD DLLCALLCONV WDC_PciReadCfgReg1BySlot6(
    WD_PCI_SLOT *pPciSlot, DWORD dwOffset, WORD *val);

DWORD DLLCALLCONV WDC_PciReadCfgReg32BySlot(
    WD_PCI_SLOT *pPciSlot, DWORD dwOffset, UINT32 *val);

DWORD DLLCALLCONV WDC_PciReadCfgReg64BySlot(
    WD_PCI_SLOT *pPciSlot, DWORD dwOffset, UINT64 *val);
```

パラメータ

名前	型	入出力
➢ pPciSlot	WD_PCI_SLOT*	入力
➢ dwOffset	DWORD	入力
➢ val	BYTE* / WORD* / UINT32* / UINT64*	出力

説明

名前	説明
pPciSlot	`WDC_PciScanDevices()` [A.2.4] の呼び出しにより取得することができるデバイスの場所情報構造体 [A.4.8] へのポインタ
dwOffset	読み取る PCI 設定空間の初めからのオフセット
val	PCI 設定空間から読み取られるデータで埋められるバッファへのポインタ

戻り値

正常終了した場合、WD_STATUS_SUCCESS (0) を返します。そうでない場合、エラーコードを返します [A.8]。

A.2.31 WDC_PciWriteCfgBySlotXXX()

目的

- `WDC_PciWriteCfgBySlot8/16/32/64()` は、それぞれ 1 バイト (8 ビット) / 2 バイト (16 ビット) / 4 バイト (32 ビット) / 8 バイト (64 ビット) を指定の PCI デバイスの設定空間または PCI Express デバイスの拡張設定空間でのオフセットへ書き込みます。
  このデバイスは PCI バス上の場所によって識別されます。

Windows と Linux では、PCI Express の拡張設定空間へのアクセスを対象のプラットフォームでサポートします。以下の説明において “PCI” へのすべての言及には PCI Express も含まれています。

プロトタイプ

```
DWORD DLLCALLCONV WDC_PciWriteCfgRegBySlot8(
    WD_PCI_SLOT *pPciSlot, DWORD dwOffset, BYTE val);

DWORD DLLCALLCONV WDC_PciWriteCfgRegBySlot16(
    WD_PCI_SLOT *pPciSlot, DWORD dwOffset, WORD val);

DWORD DLLCALLCONV WDC_PciWriteCfgRegBySlot32(
    WD_PCI_SLOT *pPciSlot, DWORD dwOffset, UINT32 val);

DWORD DLLCALLCONV WDC_PciWriteCfgRegBySlot64(
    WD_PCI_SLOT *pPciSlot, DWORD dwOffset, UINT64 val);
```

パラメータ

名前	型	入出力
➢ pPciSlot	WD_PCI_SLOT*	入力
➢ dwOffset	DWORD	入力
➢ val	BYTE / WORD / UINT32 / UINT64	入力

説明

名前	説明
pPciSlot	`WDC_PciScanDevices()` [A.2.4] の呼び出しにより取得することができるデバイスの場所情報構造体 [A.4.8] へのポインタ
dwOffset	書き込む PCI 設定空間の初めからのオフセット
val	PCI 設定空間へ書き込むデータ

戻り値

正常終了した場合、WD_STATUS_SUCCESS (0) を返します。そうでない場合、エラーコードを返します [A.8]。

A.2.32 WDC_PciReadCfgXXX()

目的

- `WDC_PciReadCfg8/16/32/64()` は、それぞれ 1 バイト (8 ビット) / 2 バイト (16 ビット) / 4 バイト (32 ビット) / 8 バイト (64 ビット) を指定の PCI デバイスの設定空間または PCI Express デバイスの拡張設定空間でのオフセットから読み取ります。

Windows と Linux では、PCI Express の拡張設定空間へのアクセスを対象のプラットフォームでサポートします。以下の説明において “PCI” へのすべての言及には PCI Express も含まれています。

プロトタイプ

```
DWORD DLLCALLCONV WDC_PciReadCfgReg8(WDC_DEVICE_HANDLE hDev,
    DWORD dwOffset, BYTE *val);

DWORD DLLCALLCONV WDC_PciReadCfgReg16(WDC_DEVICE_HANDLE hDev,
    DWORD dwOffset, WORD *val);

DWORD DLLCALLCONV WDC_PciReadCfgReg32(WDC_DEVICE_HANDLE hDev,
    DWORD dwOffset, UINT32 *val);

DWORD DLLCALLCONV WDC_PciReadCfgReg64(WDC_DEVICE_HANDLE hDev,
    DWORD dwOffset, UINT64 *val);
```

パラメータ

名前	型	入出力
➢ hDev	WDC_DEVICE_HANDLE	入力
➢ dwOffset	DWORD	入力
➢ val	BYTE* / WORD* / UINT32* / UINT64*	出力

説明

名前	説明
hDev	`WDC_PciDeviceOpen()` [A.2.9] により返される WDC PCI デバイスの構造体へのハンドル
dwOffset	読み取る PCI 設定空間の初めからのオフセット
val	PCI 設定空間から読み取られるデータで埋められるバッファへのポインタ

戻り値

正常終了した場合、WD_STATUS_SUCCESS (0) を返します。そうでない場合、エラーコードを返します [A.8]。

A.2.33 WDC_PciWriteCfgXXX()

目的

- `WDC_PciWriteCfg8/16/32/64()` は、それぞれ 1 バイト (8 ビット) / 2 バイト (16 ビット) / 4 バイト (32 ビット) / 8 バイト (64 ビット) を指定の PCI デバイスの設定空間または PCI Express デバイスの拡張設定空間でのオフセットへ書き込みます。

Windows と Linux では、PCI Express の拡張設定空間へのアクセスを対象のプラットフォームでサポートします。以下の説明において “PCI” へのすべての言及には PCI Express も含まれています。

プロトタイプ

```
DWORD DLLCALLCONV WDC_PciWriteCfgReg8(WDC_DEVICE_HANDLE hDev,
    DWORD dwOffset, BYTE val);

DWORD DLLCALLCONV WDC_PciWriteCfgReg16(WDC_DEVICE_HANDLE hDev,
    DWORD dwOffset, WORD val);

DWORD DLLCALLCONV WDC_PciWriteCfgReg32(WDC_DEVICE_HANDLE hDev,
    DWORD dwOffset, UINT32 val);

DWORD DLLCALLCONV WDC_PciWriteCfgReg64(WDC_DEVICE_HANDLE hDev,
    DWORD dwOffset, UINT64 val);
```

パラメータ

名前	型	入出力
➢ hDev	WDC_DEVICE_HANDLE	入力
➢ dwOffset	DWORD	入力
➢ val	BYTE / WORD / UINT32 / UINT64	入力

説明

名前	説明
hDev	`WDC_PciDeviceOpen()` [A.2.9] により返される WDC PCI デバイスの構造体へのハンドル
dwOffset	書き込む PCI 設定空間の初めからのオフセット
val	PCI 設定空間へ書き込むデータ

戻り値

正常終了した場合、WD_STATUS_SUCCESS (0) を返します。そうでない場合、エラーコードを返します [A.8]。

A.2.34 WDC_PcmciaReadAttribSpace()

目的

- PCMCIA デバイスの属性空間で指定したオフセットからデータを読み取ります。

プロトタイプ

```
DWORD DLLCALLCONV WDC_PcmciaReadAttribSpace(
    WDC_DEVICE_HANDLE hDev,
    DWORD dwOffset,
    PVOID pData,
    DWORD dwBytes);
```

パラメータ

名前	型	入出力
➢ hDev	WDC_DEVICE_HANDLE	入力
➢ dwOffset	DWORD	入力
➢ pData	PVOID	出力
➢ dwBytes	DWORD	入力

説明

名前	説明
hDev	`WDC_PcmciaDeviceOpen()` [A.2.10] により返される WDC PCMCIA デバイスの構造体へのハンドル
dwOffset	読み取る PCMCIA の属性空間の初めからのオフセット
pData	PCMCIA 属性空間から読み取られるデータで埋められるバッファへのポインタ
dwBytes	読み取るバイト数

戻り値

正常終了した場合、WD_STATUS_SUCCESS (0) を返します。そうでない場合、エラーコードを返します [A.8]。

A.2.35 WDC_PcmciaWriteAttribSpace()

目的

- PCMCIA デバイスの属性空間で指定したオフセットへデータを書き込みます。

プロトタイプ

```
DWORD DLLCALLCONV WDC_PcmciaWriteAttribSpace(
```

```
    WDC_DEVICE_HANDLE hDev,
    DWORD dwOffset,
    PVOID pData,
    DWORD dwBytes);
```

パラメータ

名前	型	入出力
➢ hDev	WDC_DEVICE_HANDLE	入力
➢ dwOffset	DWORD	入力
➢ pData	PVOID	入力
➢ dwBytes	DWORD	入力

説明

名前	説明
hDev	`WDC_PcmciaDeviceOpen()` [A.2.10] により返される WDC PCMCIA デバイスの構造体へのハンドル
dwOffset	書き込む PCMCIA の属性空間の初めからのオフセット
pData	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
dwBytes	書き込むバイト数

戻り値

正常終了した場合、WD_STATUS_SUCCESS (0) を返します。そうでない場合、エラーコードを返します [A.8]

A.2.36 WDC_PcmciaSetWindow()

目的

- PCMCIA バス コントローラのメモリ ウィンドウの設定を変更します。

プロトタイプ

```
DWORD DLLCALLCONV WDC_PcmciaSetWindow(
    WDC_DEVICE_HANDLE hDev,
    WD_PCMCIA_ACC_SPEED speed,
    WD_PCMCIA_ACC_WIDTH width,
    DWORD dwCardBase);
```

パラメータ

名前	型	入出力
➢ hDev	WDC_DEVICE_HANDLE	入力

➢ speed	WD_PCMCIA_ACC_SPEED	入力
➢ width	WD_PCMCIA_ACC_WIDTH	入力
➢ dwCardBase	DWORD	入力

説明

名前	説明
hDev	`WDC_PcmciaDeviceOpen()` [A.2.10] により返される WDC PCMCIA デバイスの構造体へのハンドル
speed	CMCIA バスへのアクセス スピード(`WD_PCMCIA_ACC_SPEED` 列挙型 [A.4.3] を参照)
width	PCMCIA バスの幅 (`WD_PCMCIA_ACC_WIDTH` 列挙型 [A.4.4] を参照)
dwCardBase	メモリ マップが始まる PCMCIA デバイスのメモリでのオフセット

戻り値

正常終了した場合、WD_STATUS_SUCCESS (0) を返します。そうでない場合、エラーコードを返します [A.8]。

A.2.37 WDC_PcmciaSetVpp()

目的

- PCIMCIA バス コントローラのボルテージ パワー ピン (Vpp) のパワー レベルを変更します。

プロトタイプ

```
DWORD DLLCALLCONV WDC_PcmciaSetVpp(
    WDC_DEVICE_HANDLE hDev,
    WD_PCMCIA_VPP vpp);
```

パラメータ

名前	型	入出力
➢ hDev	WDC_DEVICE_HANDLE	入力
➢ vpp	WD_PCMCIA_VPP	入力

説明

名前	説明
hDev	`WDC_PcmciaDeviceOpen()` [A.2.10] により返される WDC PCMCIA デバイスの構造体へのハンドル
vpp	PCIMCIA バス コントローラのボルテージ パワー ピン (Vpp) のパ

	ワー レベル (WD_PCMCIA_VPP 列挙型 [A.4.5] を参照)

戻り値

正常終了した場合、WD_STATUS_SUCCESS (0) を返します。そうでない場合、エラーコードを返します [A.8]。

A.2.38 WDC_DMAContigBufLock()

目的

- DMA バッファを割り当て、物理メモリにロックし、物理アドレス空間およびユーザーモードとカーネルモードの仮想アドレス空間へ割り当てられたバッファのマップを返します。

プロトタイプ

```
DWORD DLLCALLCONV WDC_DMAContigBufLock(
    WDC_DEVICE_HANDLE hDev,
    PVOID *ppBuf,
    DWORD dwOptions,
    DWORD dwDMABufSize,
    WD_DMA **ppDma);
```

パラメータ

名前	型	入出力
➢ hDev	WDC_DEVICE_HANDLE	入力
➢ ppBuf	PVOID*	出力
➢ dwOptions	DWORD	入力
➢ dwDMABufSize	DWORD	入力
➢ ppDma	WD_DMA**	出力

説明

名前	説明
hDev	WDC_xxxDeviceOpen( ) (PCI [A.2.9] / PCMCIA [A.2.10] / ISA [A.2.11]) により返される WDC デバイスへのハンドル
ppBuf	関数により、割り当てられた DMA バッファのユーザ モードのマップアドレスで埋められるポインタへのポインタ
dwOptions	以下の (**windrvr.h** で列挙型に定義された) いずれかのフラグのビット マスク。 • **DMA_FROM_DEVICE**: デバイスからメモリへ転送するために、DMA バッファを同期させます。 • **DMA_TO_DEVICE**:メモリからデバイスへ転送するために、DMA バッファを同期させます。 • **DMA_TO_FROM_DEVICE**: 両方向 (デバイスからメモリ:

	DMA_FROM_DEVICE /メモリからデバイス: DMA_TO_DEVICE) の転送用に DMAバッファを同期させます。 • **DMA_ALLOW_CACHE**: メモリのキャッシュを許可します。 **DMA_KBUF_BELOW_16M**: メイン メモリの下位 16 MB 内で物理 DMA バッファを割り当てます。 • **DMA_ALLOW_64BIT_ADDRESS**: (ターゲット プラットフォームでサポートされる場合) 64 ビットの DMA アドレスの割り当てを許可します。このフラグは、Windows と Linux でサポートされています。
dwDMABufSize	DMA バッファのバイト単位でのサイズ
ppDma	この関数により割り当てられた DMA バッファの情報構造体 [A.4.13] へのポインタへのポインタ。 DMA バッファが不要になった時、この構造体 (*ppDma) へのポインタは WDC_DMABufUnlock() [A.2.40] へ渡します。

戻り値

正常終了した場合、WD_STATUS_SUCCESS (0) を返します。そうでない場合、エラーコードを返します [A.8]。

注釈

- この関数を呼び出すとき、DMA_KERNEL_BUFFER_ALLOC フラグを設定する**必要はありません。** この関数はこのフラグを自動的に設定します。
- 現在この関数はユーザーモードからのみサポートされています。
- Windows x86 と x86_64 プラットフォームでは、通常、DMA_ALLOW_CACHE フラグを DMA オプション ビットマスク パラメータ (dwOptions) に設定してください。

A.2.39 WDC_DMASGBufLock()

目的

- DMA の割り当て済みユーザーモード メモリ バッファをロックし、ロックされた DMA ページの物理マップを返します。Windows 2000 / XP / Server 2003 / Server 2008 / Vista / 7 では、バッファのカーネル モード マップも返します。

プロトタイプ

```
DWORD DLLCALLCONV WDC_DMASGBufLock(
    WDC_DEVICE_HANDLE hDev,
    PVOID pBuf,
    DWORD dwOptions,
    DWORD dwDMABufSize,
    WD_DMA **ppDma);
```

パラメータ

名前	型	入出力
➢ hDev	WDC_DEVICE_HANDLE	入力
➢ pBuf	PVOID	入力
➢ dwOptions	DWORD	入力
➢ dwDMABufSize	DWORD	入力
➢ ppDma	WD_DMA**	出力

説明

名前	説明
hDev	`WDC_xxxDeviceOpen()` (PCI [A.2.9] / PCMCIA [A.2.10] / ISA [A.2.11]) により返される WDC デバイスへのハンドル
pBuf	割り当てられた物理 DMA バッファへマップされるユーザーモードバッファへのポインタ。
dwOptions	以下の (**windrvr.h** で列挙型に定義された) いずれかのフラグのビット マスク。 • **`DMA_FROM_DEVICE`**: デバイスからメモリへ転送するために、DMA バッファを同期させます。 • **`DMA_TO_DEVICE`**: メモリからデバイスへ転送するために、DMA バッファを同期させます。 • **`DMA_TO_FROM_DEVICE`**: 両方向 (デバイスからメモリ: `DMA_FROM_DEVICE` /メモリからデバイス: `DMA_TO_DEVICE`) の転送用に DMAバッファを同期させます。 • **`DMA_ALLOW_CACHE`**: メモリのキャッシュを許可します。 • **`DMA_ALLOW_64BIT_ADDRESS`**: (ターゲット プラットフォームでサポートされる場合) 64 ビットの DMA アドレスの割り当てを許可します。このフラグは、Windows と Linux でサポートされています。
dwDMABufSize	DMA バッファのバイト単位でのサイズ
ppDma	この関数により割り当てられた DMA バッファの情報構造体 [A.4.13] へのポインタへのポインタ。 DMA バッファが不要になった時、この構造体 (`*ppDma`) へのポインタは `WDC_DMABufUnlock()` [A.2.40] へ渡します。

戻り値

正常終了した場合、WD_STATUS_SUCCESS (0) を返します。そうでない場合、エラーコードを返します [A.8]。

注釈

- 大きいバッファ (> 1MB) を割り当てるためにこの関数を呼び出すとき、低レベル WinDriver `WD_DMALock()` 関数を使用して、大きいスキャッタ / ギャザー DMA バッファの割り当てに使用する `DMA_LARGE_BUFFER` フラグを設定する**必要はありません**。`WDC_DMASGBufLock()` がこの処理を行います。

- 現在この関数はユーザーモードからのみサポートされています。
- Windows x86 と x86_64 プラットフォームでは、通常、DMA_ALLOW_CACHE フラグを DMA オプション ビットマスク パラメータ (dwOptions) に設定してください。

A.2.40 WDC_DMABufUnlock()

目的

- 以前に WDC_DMAContigBufLock() [A.2.38] または WDC_DMASGBufLock() [A.2.39] への呼び出しにより DMA バッファへ割り当てられたメモリをアンロックし、解放します。

プロトタイプ

```
DWORD DLLCALLCONV WDC_DMABufUnlock(WD_DMA *pDma);
```

パラメータ

名前	型	入出力
➢ pDma	WD_DMA*	入力

説明

名前	説明
pDma	以前に行った WDC_DMAContigBufLock() [A.2.38] (連続 DMA バッファ) または WDC_DMASGBufLock() [A.2.39] (スキャッタ / ギャザー DMA バッファ) への呼び出しから受け取った DMA 情報構造体 [A.4.13] へのポインタ *ppDma はこれらの関数より返されます。

戻り値

正常終了した場合、WD_STATUS_SUCCESS (0) を返します。そうでない場合、エラーコードを返します [A.8]。

注釈

- 現在この関数はユーザーモードからのみサポートされています。

A.2.41 WDC_DMASyncCpu()

目的

- CPU キャッシュからデータをフラッシュすることにより、DMA バッファとすべての CPU のキャッシュを同期化します。

注意: この関数は DMA 転送が実行される前に呼び出します (注釈を参照)。

プロトタイプ

```
DWORD DLLCALLCONV WDC_DMASyncCpu(WD_DMA *pDma);
```

パラメータ

名前	型	入出力
➢ pDma	WD_DMA*	入力

説明

名前	説明
pDma	以前に行った `WDC_DMAContigBufLock()` [A.2.38] (連続 DMA バッファ) または `WDC_DMASGBufLock()` [A.2.39] (スキャッタ / ギャザー DMA バッファ) への呼び出しから受け取った DMA 情報構造体 [A.4.13] へのポインタ `*ppDma` はこれらの関数により返されます。

戻り値

正常終了した場合、WD_STATUS_SUCCESS (0) を返します。そうでない場合、エラーコードを返します [A.8]。

注釈

- 非同期的な DMA の読み取りまたは書き込み動作は、メモリ内のデータにアクセスし、CPU とホストの物理メモリ間に存在するプロセッサ (CPU) キャッシュ内のデータへはアクセスしません。読み取り転送の直前に `WDC_DMASyncCpu()` を呼び出して CPU キャッシュをフラッシュした場合を除き、後で CPU キャッシュをフラッシュすると、DMA 動作でシステム メモリへ転送されるデータは、古くなったデータで上書きされる場合があります。書き込み転送の直前に `WDC_DMASyncCpu()`を呼び出して CPU キャッシュをフラッシュしないかぎり、CPU キャッシュ内のデータのほうがメモリにコピーされたものより新しいもの (アップデートされたもの) になる場合があります。
- 現在この関数はユーザーモードからのみサポートされています。

A.2.42 WDC_DMASyncIo()

目的

- I/O キャッシュからデータをフラッシュし、CPU キャッシュをアップデートすることにより、DMA バッファと I/O キャッシュを同期化します。

注意: この関数は DMA 転送が実行された後に呼び出します (注釈を参照)。

プロトタイプ

```
DWORD DLLCALLCONV WDC_DMASyncIo(WD_DMA *pDma);
```

パラメータ

名前	型	入出力
➢ pDma	WD_DMA*	入力

説明

名前	説明
pDma	以前に行った WDC_DMAContigBufLock() [A.2.38] (連続 DMA バッファ) または WDC_DMASGBufLock() [A.2.39] (スキャッタ / ギャザー DMA バッファ) への呼び出しから受け取った DMA 情報構造体 [A.4.13] へのポインタ *ppDma はこれらの関数により返されます。

戻り値

正常終了した場合、WD_STATUS_SUCCESS (0) を返します。そうでない場合、エラーコードを返します [A.8]。

注釈

- DMA 転送の完了後、データはホストの物理メモリとバスマスタ DMA デバイス間に存在する I/O キャッシュにあり、ホストのメイン メモリにはまだありません。CPU がメモリにアクセスした場合、CPU キャッシュから誤ったデータを読み取る可能性があります。CPU 用のメモリを常に同期されたデータにするには、I/O キャッシュからデータをフラッシュし、新しいデータで CPU キャッシュをアップデートするために DMA 転送後に WDC_DMASyncIo() を呼び出します。この関数は、バス エクステンダやバス ブリッジに関連したキャッシュなど、デバイスとメモリ間のキャッシュおよびバッファもフラッシュします。
- 現在この関数はユーザーモードからのみサポートされています。

A.2.43 WDC_SharedBufferAlloc()

目的

- ユーザーモードとカーネルモード間で共有できるメモリ バッファを割り当てます (**“shared buffer”**)。
- 割り当てたバッファのユーザーモードとカーネルモードの仮想アドレス空間のマップを返します。

注意: この関数はユーザーモードのアプリケーションと Kernel PlugIn ドライバ間のデータを共有するために有効な方法を提供します。

プロトタイプ

```
DWORD DLLCALLCONV WDC_SharedBufferAlloc(
    PVOID *ppUserAddr,
    KPTR *ppKernelAddr,
    DWORD dwBufSize,
    DWORD dwOptions,
    HANDLE *phBuf);
```

パラメータ

名前	型	入出力
➢ *ppUserAddr	PVOID*	出力
➢ *ppKernelAddr	KPTR*	出力

➢ dwBufSize	DWORD	入力
➢ dwOptions	DWORD	入力
➢ phBuf	HANDLE*	出力

説明

名前	説明
ppUserAddr	この関数を使用して、割り当てたバッファのユーザーモードのマップ アドレスで満たされたポインタへのポインタ
ppKernelAddr	この関数を使用して、割り当てたバッファのカーネルモードのマップ アドレスで満たされたポインタへのポインタ
dwBufSize	割り当てるバッファのバイト サイズ
dwOptions	後で使用するために予約。このフィールドを 0 に初期化します。
phBuf	この関数で満たされる割り当てたバッファのハンドルへのポインタ。バッファが必要ない場合、バッファのハンドル (*phBuf) を WDC_SharedBufferFree() へ渡してください。

戻り値

正常終了した場合、WD_STATUS_SUCCESS (0) を返します。そうでない場合、エラーコードを返します [A.8]。

注釈

- この関数は**ユーザーモードからのみ**呼び出すことができます。

A.2.44 WDC_SharedBufferFree()

目的

- WDC_SharedBufferAlloc() への以前の呼び出しで割り当てられた共有バッファーを開放します。

プロトタイプ

```
#define WDC_SharedBufferFree(hBuf)
```

パラメータ

名前	型	入出力
➢ hBuf	HANDLE	入力

説明

名前	説明
hBuf	WDC_SharedBufferAlloc() への以前の呼び出しの

	*phBuf パラメータ内で受信した、共有バッファへのハンドル

戻り値

正常終了した場合、WD_STATUS_SUCCESS (0) を返します。そうでない場合、エラーコードを返します [A.8]。

注釈

- この関数は**ユーザーモードからのみ**呼び出すことができます。

A.2.45 WDC_IntEnable()

目的

- デバイスの割り込み処理を有効にします。
  Linux、Windows Vista およびそれ以降では、MSI-X (Extended Message-Signaled Interrupts) または MSI (Message-Signaled Interrupts) をサポート (および Windows Vista およびそれ以降では関連する INF ファイルをインストール) する PCI デバイスの割り込みを有効にする際、この関数は初めに MSI-X または MSI を有効にします。失敗した場合、または対象の OS が MSI / MSI-X をサポートしていない場合、この関数はレガシーなレベル センシティブ割り込みを有効にします (デバイスで対応している場合)。その他のハードウェアのタイプの場合 (MSI / MSI-X をサポートしていない PCI / PCMCIA / ISA)、この関数はデバイスで対応しているレガシーな割り込みのタイプ (レベル センシティブまたはエッジ トリガー) を有効にします。
  注意: この関数はユーザーモードのアプリケーションと Kernel PlugIn ドライバ間のデータを共有するために有効な方法を提供します
  Kernel PlugIn ドライバ (`fUseKP=TRUE`) を使用して割り込みを有効にする場合、Kernel PlugIn 関数を使用して、デバイスに対して有効にした割り込みのタイプから派生した割り込みを処理します。MSI / MSI-X の場合、`KP_IntAtIrqlMSI()` と `KP_IntAtDpcMSI()` 関数を使用します。その他の場合、`KP_IntAtIrql()` と `KP_IntAtDpc()` 関数を使用します。

- 呼び出し元が **Kernel PlugIn** ドライバを使用してカーネルで割り込み処理を実行する場合、高い IRQ (割り込み要求) レベルで実行する Kernel PlugIn の `KP_IntAtIrql()` 関数 [A.5.8] (レガシー割り込み) または `KP_IntAtIrqlMSI()` 関数 (MSI / MSI-X) は、割り込みを受け取った直後に呼び出されます。

- 割り込みが受け取られると、この関数は、WinDriver によりカーネルにおいて高い IRQ レベルでで実行される転送コマンド情報を受け取ります。割り込みの処理に **Kernel PlugIn** ドライバが使用されている場合、呼び出し元で設定された転送コマンドは、Kernel PlugIn の `KP_IntAtDpc()` 関数または `KP_IntAtDpcMSI()` 関数の実行が完了した後で、WinDriver により実行されます。
  Kernel PlugIn ドライバを使用せずに、**ユーザーモード**からのレベル センシティブな割り込み (レガシーな PCI 割り込みなど) を処理する場合、割り込みを認識させるための転送コマンド情報を関数に渡す必要があります。**Kernel PlugIn** ドライバを使用している場合、割り込みを認識させる情報は Kernel PlugIn の `KP_IntAtIrql()` 関数 [A.5.8] で実装します。そのため `WDC_IntEnable()` への呼び出しの転送コマンドは不要です。

- この関数は、カーネルモードの割り込み処理が完了した後で WinDriver に呼び出されるユーザーモード割り込み処理ルーチンを受け取ります。
  Kernel PlugIn ドライバを使用して割り込みが処理される場合、Kernel PlugIn の遅延型割り込み処理関数 (`KP_IntAtDpc()` [A.5.9] (レガシー割り込み) または `KP_IntAtDpcMSI()` (MSI / MSI-X)) の戻り値は、ユーザーモードの割り込み処理が何回呼ばれるか (Kernel PlugIn の `KP_IntAtIrql()` [A.5.8] 関数または `KP_IntAtIrqlMSI()` 関数の戻り値により割り出される `KP_IntAtDpc()` または `KP_IntAtDpcMSI()` も実行されることを前提として) を割り出します。

プロトタイプ

```
DWORD DLLCALLCONV WDC_IntEnable(
    WDC_DEVICE_HANDLE hDev,
    WD_TRANSFER *pTransCmds,
    DWORD dwNumCmds,
    DWORD dwOptions,
    INT_HANDLER funcIntHandler,
    PVOID pData,
    BOOL fUseKP);
```

パラメータ

名前	型	入出力
➢ hDev	WDC_DEVICE_HANDLE	入力
➢ pTransCmds	WD_TRANSFER*	入力
➢ dwNumCmds	DWORD	入力
➢ dwOptions	DWORD	入力
➢ funcIntHandler	typedef void (*INT_HANDLER)(PVOID pData);	入力
➢ pData	PVOID	入力
➢ fUseKP	BOOL	入力

説明

名前	説明
hDev	`WDC_xxxDeviceOpen()` (PCI [A.2.9] / PCMCIA [A.2.10] / ISA [A.2.11]) により返される WDC デバイスへのハンドル
pTransCmds	割り込みが検出された際、カーネル レベルで実行される処理を定義する転送コマンド情報構造体の配列 (転送コマンドが不要の場合は NULL です)。 注意: • 転送コマンド用に割り当てたメモリは、割り込みが無効になるまで、利用可能にしておく必要があります。 • Kernel PlugIn を使用せずにレベル センシティブな割り込みを処理する場合 (レガシー PCI 割り込みなど)、この配列を使用して、割り込みを受け取った直後に、カーネルで割り込みを認識するハードウェア固有のコマンドを定義する必要があります。詳細は、WinDriver ユーザーズ ガイドのセクション 9.2 を参照してください。 転送コマンドの設定方法については、セクション A.4.14 の `WD_TRANSFER` の説明を参照してください。
dwNumCmds	`pTransCmds` 配列の転送コマンド数
dwOptions	割り込み処理フラグのビット マスク。 オプションが無い場合は、ゼロ (0)。または、

	• **INTERRUPT_CMD_COPY**: 設定されている場合、WinDriver は読み取り転送コマンドの結果としてカーネルで読み取ったデータをコピーし、関連した転送コマンド構造体内でユーザーへ返します。 ユーザーモードの割り込みハンドラー ルーチン (**funcIntHandler**) からデータにアクセスできます。
funcIntHandler	カーネルで割り込みが受け取られ、処理された後に実行されるユーザーモード割り込み処理コールバック関数 (割り込み処理 INT_HANDLER のプロトタイプは、**windrvr_int_thread.h** で定義されています)。
pData	ユーザーモードの割り込み処理コールバック ルーチン (funcIntHandler) 用のデータ
fUseKP	TRUE の場合、高い IRQ (割り込み要求) レベルで実行するデバイスの Kernel PlugIn ドライバの KP_IntAtIrql() 関数 [A.5.8] または KP_IntAtIrqlMSI() 関数は、割り込みを受け取った直後に実行されます。(デバイスで使用される Kernel PlugIn ドライバは WDC_xxxDeviceOpen() へ渡され、WDC デバイス構造体へ保存されます)。 呼び出し元が転送コマンドも関数 (**pTransCmds**) へ渡した場合、これらのコマンドは、KP_IntAtIrql() または KP_IntAtIrqlMSI() の実行完了後、カーネルにおいて高い IRQ レベルで WinDriver により実行されます。 高い IRQL ハンドラーが TRUE を返した場合、Kernel PlugIn の引き継いだ割り込み処理ルーチン KP_IntAtDpc() [A.5.9] または KP_IntAtDpcMSI() を呼び出します。この関数の戻り値は、コントロールがユーザーモードへ戻った時点でユーザーモードの割り込み処理 (**funcIntHandler**) が何回実行されるかを割り出します。FALSE の場合、割り込みを受け取ると、**pTransCmds** でユーザーが設定した転送コマンドは、カーネルにおいて高い IRQ レベルで WinDriver により実行されます。ユーザーモードの割り込み処理ルーチン (**funcIntHandler**) はコントロールがユーザーモードへ戻ったときに実行されます。

戻り値

正常終了した場合、WD_STATUS_SUCCESS (0) を返します。そうでない場合、エラーコードを返します [A.8]。

注釈

- この関数は**ユーザーモードからのみ**呼び出すことができます。
- この関数はソフトウェア内の割り込み処理を有効にします。この関数の呼び出しが成功した後、(コードからデバイスへ書き込むことにより) ハードウェア内で割り込みの生成を物理的に有効する必要があります。
- この関数の呼び出しが成功した後に、コードで WDC_IntDisable() を呼び出し、割り込みを無効にする必要があります。
  デバイスの割り込みが有効の場合、WDC_xxxDriverClose() 関数 (PCI: [A.2.12]、PCMCIA: [A.2.13]、ISA: [A.2.14]) は WDC_IntDisable() を呼び出します。
- 割り込みを有効にする前にデバイスのドライバとして WinDriver を OS に登録する必要があります。Windows プラットフォームの Plug-and-Play ハードウェア (PCI / PCI-Express / PCMCIA) の場合、デバイスの INF ファイルをインストールすることによって、関連付けします。INF ファイルを

インストールしない場合、WDC_IntEnable() は WD_NO_DEVICE_OBJECT エラーで失敗します。

A.2.46 WDC_IntDisable()

目的

- 以前に呼び出した WDC_IntEnable() [A.2.43] に従ったデバイスの割り込み処理を無効にします。

プロトタイプ

```
DWORD DLLCALLCONV WDC_IntDisable(WDC_DEVICE_HANDLE hDev);
```

パラメータ

名前	型	入出力
➢ hDev	WDC_DEVICE_HANDLE	入力

説明

名前	説明
hDev	WDC_xxxDeviceOpen() (PCI [A.2.9] / PCMCIA [A.2.10] / ISA [A.2.11]) により返される WDC デバイスへのハンドル

戻り値

正常終了した場合、WD_STATUS_SUCCESS (0) を返します。そうでない場合、エラーコードを返します [A.8]。

注釈

- この関数は**ユーザーモードからのみ**呼び出すことができます。

A.2.47 WDC_IntIsEnabled()

目的

- デバイスの割り込みが現在有効であるかどうかを確認します。

プロトタイプ

```
BOOL DLLCALLCONV WDC_IntIsEnabled(WDC_DEVICE_HANDLE hDev);
```

パラメータ

名前	型	入出力
➢ hDev	WDC_DEVICE_HANDLE	入力

説明

名前	説明
hDev	`WDC_xxxDeviceOpen()` (PCI [A.2.9] / PCMCIA [A.2.10] / ISA [A.2.11]) により返される WDC デバイスへのハンドル

戻り値

デバイスの割り込みが有効の場合 TRUE を返します。無効な場合 FALSE を返します。

A.2.48 WDC_EventRegister()

目的

- デバイス用の Plug-and-Play およびパワー マネージメント イベントの通知を受け取るアプリケーションを登録します。

プロトタイプ

```
DWORD DLLCALLCONV WDC_EventRegister(
    WDC_DEVICE_HANDLE hDev,
    DWORD dwActions,
    EVENT_HANDLER funcEventHandler,
    PVOID pData,
    BOOL fUseKP);
```

パラメータ

名前	型	入出力
➢ hDev	WDC_DEVICE_HANDLE	入力
➢ dwActions	DWORD	入力
➢ funcEventHandler	typedef void (*EVENT_HANDLER)(WD_EVENT *pEvent, void *pData);	入力
➢ pData	PVOID	入力
➢ fUseKP	BOOL	入力

説明

名前	説明
hDev	`WDC_PciDeviceOpen()` [A.2.9] または `WDC_PcmciaDeviceOpen()` [A.2.10] により返される Plug-and-Play WDC デバイスへのハンドル
dwActions	登録するイベントを示すフラグのビット マスク。 **Plug-and-Play イベント:** • **`WD_INSERT`** - 挿入されたデバイス

	• **WD_REMOVE** - 取り除かれたデバイス **デバイスのパワー状況を変更するイベント:** • **WD_POWER_CHANGED_D0** – フル パワー • **WD_POWER_CHANGED_D1** - ロー スリープ • **WD_POWER_CHANGED_D2** – ミディアム スリープ • **WD_POWER_CHANGED_D3** - フル スリープ • **WD_POWER_SYSTEM_WORKING** - オン **システムのパワー状況:** • **WD_POWER_SYSTEM_SLEEPING1** - オンでスリープ状態 • **WD_POWER_SYSTEM_SLEEPING2** - CPU オフ、メモリ オン、PCI / PCMCIA オン • **WD_POWER_SYSTEM_SLEEPING3** - CPU オフ、メモリをリフレッシュ、PCI / PCMCIA は補助パワー • **WD_POWER_SYSTEM_HIBERNATE** - OS はコンテキストをシャットダウンする前に保存する • **WD_POWER_SYSTEM_SHUTDOWN** - コンテクストを保存しない
funcEventHandler	呼び出し元が通知を受け取るよう登録しているイベント (`dwActions` を参照) が発生したときに呼ばれるユーザーモードのイベント処理コールバック関数。(このイベント処理 `EVENT_HANDLER` のプロトタイプは **`windrvr_events.h`** で定義されています)。
pData	ユーザーモードのイベント処理コールバック ルーチン (`funcEventHandler`) 用のデータ
fUseKP	TRUE の場合、呼び出し元が通知を受け取るよう登録しているイベント (`dwActions`) が発生したときに、デバイスの Kernel PlugIn ドライバの `KP_Event()` 関数 [A.5.5] が呼び出されます。(デバイスで使用される Kernel PlugIn ドライバは `WDC_xxxDeviceOpen()` へ渡され、WDC デバイス構造体へ保存されます)。 この関数が TRUE を返した場合、ユーザーモードのイベント処理コールバック関数 (`funcEventHandler`) は、カーネルモードのイベント処理が完了したとき呼び出されます。 FALSE の場合、呼び出し元が通知を受け取るよう登録しているイベント (`dwActions`) が発生したときに、ユーザーモードのイベント処理コールバック関数が呼び出されます。

戻り値

正常終了した場合、WD_STATUS_SUCCESS (0) を返します。そうでない場合、エラーコードを返します [A.8]。

注釈

- この関数は**ユーザーモードからのみ**呼び出すことができます。
- この関数の呼び出しが成功した後に、コードで `WDC_EventUnregister()` [A.2.47] を呼び出し、デバイスからの Plug-and-Play およびパワー マネージメントの通知受け取り登録を取り消す必要があります。

A.2.49 WDC_EventUnregister()

目的

- 以前に呼び出した `WDC_EventRegister()` [A.2.46] に従ったデバイスからの Plug-and-Play およびパワー マネージメントの通知受け取り登録を取り消します。

プロトタイプ

```
DWORD DLLCALLCONV WDC_EventUnregister(WDC_DEVICE_HANDLE hDev);
```

パラメータ

名前	型	入出力
➢ hDev	WDC_DEVICE_HANDLE	入力

説明

名前	説明
hDev	`WDC_PciDeviceOpen()` [A.2.9] または `WDC_PcmciaDeviceOpen()` [A.2.10] により返される Plug-and-Play WDC デバイスへのハンドル

戻り値

正常終了した場合、WD_STATUS_SUCCESS (0) を返します。そうでない場合、エラーコードを返します [A.8]。

注釈

- この関数は**ユーザーモードからのみ**呼び出すことができます。

A.2.50 WDC_EventIsRegistered()

目的

- デバイス用の Plug-and-Play およびパワー マネージメントの通知を受け取るように、アプリケーションが登録されているかどうかを確認します。

プロトタイプ

```
BOOL DLLCALLCONV WDC_EventIsRegistered(WDC_DEVICE_HANDLE
hDev);
```

パラメータ

名前	型	入出力
➢ hDev	WDC_DEVICE_HANDLE	入力

説明

名前	説明
hDev	WDC_PciDeviceOpen() [A.2.9] または WDC_PcmciaDeviceOpen() [A.2.10] により返される Plug-and-Play WDC デバイスへのハンドル

戻り値

デバイス用の Plug-and-Play およびパワー マネージメントの通知を受け取るようにアプリケーションが登録されている場合は TRUE を返します。登録されていない場合は FALSE を返します。

A.2.51 WDC_SetDebugOptions()

目的

- WDC ライブラリ用のデバッグ オプションを設定します。設定可能なデバッグ オプションに関する詳細は WDC_DBG_OPTIONS [A.2.1.8] の記述を参照してください。
- この関数は通常アプリケーションの最初 (WDC_DriverOpen() [A.2.2] を呼び出した後) に呼び出されます。またデバッグ設定を変更するために、この関数は WDC ライブラリを使用している間 (WDC_DriverClose() [A.2.3] が呼び出されるまで) はいつでも呼び出すことができます。
- この関数が呼び出されるまで WDC ライブラリはデフォルトのデバッグ オプションを使用します (WDC_DEBG_DEFAULT [A.2.1.8] を参照)。

この関数が再度呼び出された場合、以前のデバッグ設定をクリーンナップし、呼び出し元により指定される新しいオプションを設定する前にデフォルトのデバッグ オプションを設定します。

プロトタイプ

```
DWORD DLLCALLCONV WDC_SetDebugOptions(
    WDC_DBG_OPTIONS dbgOptions,
    const CHAR *sDbgFile);
```

パラメータ

名前	型	入出力
➢ dbgOptions	WDC_DBG_OPTIONS	入力
➢ sDbgFile	const CHAR*	入力

説明

名前	説明
dbgOptions	希望するデバッグ設定を示すフラグのビット マスク (WDC_DBG_OPTIONS [A.2.1.8] を参照) このパラメータをゼロ (0) に設定した場合、デフォルトのデバッグ オプションが使用されます (WDC_DBG_DEFAULT [A.2.1.8] を参照)
sDbgFile	WDC デバッグ出力ファイル。 このパラメータは、WDC_DBG_OUT_FILE フラグが (直接または便

	利なデバッグ オプションの組み合わせの 1 つにより) デバッグ オプション (dbgOptions) で設定されている場合のみ使用されます (WDC_DBG_OPTIONS [A.2.1.8] を参照)。 WDC_DBG_OUT_FILE デバッグ フラグが設定され、sDbgFile が NULL の場合、WDC のデバッグ メッセージはデフォルトのデバッグ ファイル (**stderr**) へ記述されます。

戻り値

正常終了した場合、WD_STATUS_SUCCESS (0) を返します。そうでない場合、エラーコードを返します [A.8]。

A.2.52 WDC_Err()

目的

- WDC のデバッグ オプションに従ってデバッグ エラー メッセージを表示します (WDC_DBG_OPTIONS [A.2.1.8] および WDC_SetDebugOptions() [A.2.49] を参照)。

プロトタイプ

```
void DLLCALLCONV WDC_Err(
    const CHAR *format
     ...);
```

パラメータ

名前	型	入出力
➢ format	const CHAR*	入力
➢ argument		入力

説明

名前	説明
format	表示するエラー メッセージを含む書式を管理する文字列。文字列は 256 文字 (CHAR) までです。
argument	書式文字列用のオプションの引数

戻り値

なし

A.2.53 WDC_Trace()

目的

- WDC のデバッグ オプションに従ってデバッグ トレース メッセージを表示します (WDC_DBG_OPTIONS [A.2.1.8] および WDC_SetDebugOptions() [A.2.49] を参照)。

プロトタイプ

```
void DLLCALLCONV WDC_Trace(
    const CHAR *format
     ...);
```

パラメータ

名前	型	入出力
➢ format	const CHAR*	入力
➢ argument		入力

説明

名前	説明
format	表示するトレース メッセージを含む書式を管理する文字列。文字列は 256 文字 (CHAR) までです。
argument	書式文字列用のオプションの引数

戻り値

なし

A.2.54 WDC_GetWDHandle()

目的

- 基本的な WD_xxx WinDriver PCI / PCMCIA / ISA API (「WinDriver PCI Low-Level API Reference」を参照) で必要とされる WinDriver のカーネル モジュールへのハンドルを返します (注釈を参照)。

プロトタイプ

```
HANDLE DLLCALLCONV WDC_GetWDHandle(void);
```

戻り値

WinDriver のカーネル モジュールへのハンドルを返します。失敗した場合 INVALID_HANDLE_VALUE を返します。

注釈

- WDC API のみ使用する場合、WDC ライブラリは WinDriver へのハンドルをカプセル化するため、WinDriver へのハンドルを取得する必要はありません。この関数により、WDC ライブラリで使用される WinDriver のハンドルを入手し、コードから使用される低レベルの WD_xxx API へ渡すことができます。このような場合、(WD_Close() を使用して) 受け取ったハンドルを**閉じないよう**に注意してください。このハンドルは、WDC_DriverClose() [A.2.3] を使用して WDC ライブラリが閉じられる際に閉じられます。

A.2.55 WDC_GetDevContext()

目的

- デバイスのユーザー コンテキスト情報を返します。

プロトタイプ

```
PVOID DLLCALLCONV WDC_GetDevContext(WDC_DEVICE_HANDLE hDev);
```

パラメータ

名前	型	入出力
➢ hDev	WDC_DEVICE_HANDLE	入力

説明

名前	説明
hDev	WDC_xxxDeviceOpen() (PCI [A.2.9] / PCMCIA [A.2.10] / ISA [A.2.11]) により返される WDC デバイスへのハンドル

戻り値

デバイスのユーザー コンテキストへのポインタを返します。コンテキストが設定されていない場合、NULL を返します。

A.2.56 WDC_GetBusType()

目的

- デバイスのバスの種類 (WD_BUS_PCI、WD_BUS_PCMCIA、WD_BUS_ISA または WD_BUS_UNKNOWN) を返します。

プロトタイプ

```
WD_BUS_TYPE DLLCALLCONV WDC_GetBusType(WDC_DEVICE_HANDLE
hDev);
```

パラメータ

名前	型	入出力
➢ hDev	WDC_DEVICE_HANDLE	入力

説明

名前	説明
hDev	`WDC_xxxDeviceOpen()` (PCI [A.2.9] / PCMCIA [A.2.10] / ISA [A.2.11]) により返される WDC デバイスへのハンドル

戻り値

デバイスのバスの種類 [A.4.1] を返します。

A.2.57 WDC_Sleep()

目的

- 実行を指定した時間分 (マイクロ秒単位) 遅らせます。
  デフォルトでは、この関数は CPU サイクルを消費するビジー スリープを実行します。

プロトタイプ

```
DWORD DLLCALLCONV WDC_Sleep(
    DWORD dwMicroSecs,
    WDC_SLEEP_OPTIONS options);
```

パラメータ

名前	型	入出力
➢ dwMicroSecs	DWORD	入力
➢ options	WDC_SLEEP_OPTIONS	入力

説明

名前	説明
dwMicroSecs	スリープするマイクロ秒数
options	スリープのオプション [A.2.1.7]

戻り値

正常終了した場合、WD_STATUS_SUCCESS (0) を返します。そうでない場合、エラーコードを返します [A.8]。

A.2.58 WDC_Version()

目的

- WDC ライブラリにより使用される 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 DLLCALLCONV WDC_Version(
    CHAR *sVersion,
    DWORD *pdwVersion);
```

パラメータ

名前	型	入出力
➢ sVersion	CHAR*	出力
➢ pdwVersion	DWORD*	出力

説明

名前	説明
sVersion	ドライバのバージョン情報文字列を保持するために、あらかじめ割り当てられたバッファへのポインタ。 バージョン文字列バッファのサイズは、128 バイト (文字) 以上でなければなりません。
pdwVersion	WinDriver カーネル モジュールのバージョン番号へのポインタ

戻り値

正常終了した場合、WD_STATUS_SUCCESS (0) を返します。そうでない場合、エラーコードを返します [A.8]。

A.3 WDC 低レベル API

このセクションでは、**WinDriver/include/wdc_defs.h** ヘッダー ファイルで定義された WDC の種類およびプリプロセッサの定義について説明します。

A.3.1 WDC_ID_U の共用体

WDC のデバイス ID 情報共用体の型 (PCI と PCMCIA デバイスで使用)。

名前	型	説明
➢ PciId	WD_PCI_ID	PCI デバイス ID 情報構造体 [A.4.6]
➢ pcmciaId	WD_PCMCIA_ID	PCMCIA デバイス ID 情報構造体 [A.4.7]

A.3.2 WDC_ADDR_DESC 構造体

PCI / PCMCIA / ISA デバイスのメモリまたは I/O アドレス空間の情報構造体の型。

名前	型	説明
➢ dwAddrSpace	DWORD	アドレス空間番号
➢ fIsMemory	BOOL	TRUE - メモリ アドレス空間 FALSE - I/O アドレス空間
➢ dwItemIndex	DWORD	関連した WDC デバイス情報構造体 [A.3.3] 内の `cardReg.Card.Item` 配列で `WDC_xxxDeviceOpen()` により取得、保存されるアドレス空間用の `WD_ITEMS` 構造体のインデックス
➢ dwBytes	DWORD	アドレス空間のバイト単位でのサイズ
➢ kptAddr	KPTR	アドレス空間の物理的なベース アドレスのカーネルモード マップ。このアドレスは、`WD_Transfer()` または `WD_MultiTransfer()` API を使用してメモリまたは I/O 領域へアクセスするために、またはカーネルでメモリ アドレスへ直接アクセスするときに、WDC API により使用されます。
➢ dwUserDirectMemAddr	UPTR	アドレス空間の物理的なベース アドレスのユーザーモード マップ。このアドレスは、ユーザーモードから直接メモリ アドレスへアクセスするために使用されます。

A.3.3 WDC_DEVICE 構造体

PCI / PCMCIA / ISA デバイスの情報構造体の型。
`WDC_xxxDeviceOpen()` 関数 (PCI: [A.2.9] / PCMCIA: [A.2.10] / ISA: [A.2.11]) は、この型のデバイス構造体を割り当て、返します。

名前	型	説明
➢ id	WDC_ID_U	デバイス ID 情報共用体 (PCI および PCMCIA デバイス に関連)。[A.3.1] を参照。
➢ slot	WDC_SLOT_U	デバイスのロケーション情報構造体。セクション [A.2.1.9] の `WDC_SLOT_U` の記述を参照。
➢ dwNumAddrSpaces	DWORD	デバイス上で検出されたアドレス空間数
➢ pAddrDesc	WDC_ADDR_DESC*	メモリおよび I/O アドレス空間の情報構造体 [A.3.2] の配列。
➢ cardReg	WD_CARD_REGISTER	`WDC_xxxDeviceOpen()`関数により呼び出される低レベルの `WD_CardRegister()` 関数から返される WinDriver のデバイス リソース情報構造体
➢ kerPlug	WD_KERNEL_PLUGIN	Kernel PlugIn ドライバ情報構造体 [A.6.1]。 呼び出し元が Kernel PlugIn ドライバを使用する場合 (使用しない場合は、この構造体は使用されません)、`WDC_xxxDeviceOpen()` 関数により埋められ、WDC ライブラリにより保持されています。
➢ Int	WD_INTERRUPT	割り込み情報構造体。 この構造体は、割り込みを持つデバイス用の

		WDC_xxxDeviceOpen() 関数により埋められ、WDC ライブラリにより保持されます。
➢ hIntThread	DWORD	割り込みが有効になると生じる割り込みスレッド このハンドルは、WDC により低レベルの WinDriver 割り込み API に渡されます。WDC API を使用する場合、このハンドルに直接アクセスする必要はありません。
➢ Event	WD_EVENT	WinDriver の Plug-and-Play およびパワー マネージメント イベントの情報構造体。詳細は、EventRegister() に関する説明を参照してください。
➢ hEvent	HANDLE	WinDriver EventRegister() / EventUnregister() 関数により使用されるハンドル WDC API を使用する場合、直接このハンドルにアクセスする必要はありません。
➢ pCtx	PVOID	デバイスのコンテキスト情報。 この情報は、WDC_xxxDeviceOpen() 関数によりパラメータとして受け取られ、呼び出し元アプリケーションで将来使用するために、デバイスの構造体に保存されます (オプション)。

A.3.4 PWDC_DEVICE

WDC_DEVICE 構造体 [A.3.3] の型へのポインタ。

```
typedef WDC_DEVICE *PWDC_DEVICE
```

A.3.5 WDC_MEM_DIRECT_ADDR マクロ

目的

- ポインタを返すユーティリティ マクロです。呼び出し処理のコンテキストから、指定したメモリ アドレス空間へ直接アクセスするのに使用できます。

プロトタイプ

```
WDC_MEM_DIRECT_ADDR(pAddrDesc)
```

パラメータ

名前	型	入出力
➢ pAddrDesc	WDC_ADDR_DESC*	入力

説明

名前	説明
pAddrDesc	WDC メモリ アドレス空間の情報構造体へのポインタ [A.3.2]

戻り値

ユーザーモードから呼び出された場合、物理メモリ アドレスのユーザーモード マップを返します (`pAddrDesc->dwUserDirectMemAddr`)。
カーネルモードから呼び出された場合、物理メモリ アドレスのカーネルモード マップを返します (`pAddrDesc->kptAddr`)。
返されたポインタはユーザーモードまたはカーネルモードから直接メモリへアクセスするために使用することができます。

A.3.6 WDC_ADDR_IS_MEM マクロ

目的

- 与えられたアドレス空間がメモリまたは I/O アドレス空間かどうかを確認するユーティリティ マクロ。

プロトタイプ

```
WDC_ADDR_IS_MEM(pAddrDesc)
```

パラメータ

名前	型	入出力
➢ pAddrDesc	WDC_ADDR_DESC*	入力

説明

名前	説明
pAddrDesc	WDC メモリ アドレス空間の情報構造体へのポインタ [A.3.2]

戻り値

`pAddrDesc->fIsMemory` を返します。メモリ アドレス空間の場合、TRUE を返します。そうでない場合、FALSE を返します。

A.3.7 WDC_GET_ADDR_DESC マクロ

目的

- 指定したアドレス空間番号へ従う WDC アドレス空間の情報構造体 (`WDC_ADDR_DESC` [A.3.2]) を取得するユーティリティ マクロ。

プロトタイプ

```
WDC_GET_ADDR_DESC(
    pDev,
    dwAddrSpace)
```

パラメータ

名前	型	入出力
➢ pDev	PWDC_DEVICE	入力
➢ dwAddrSpace	DWORD	入力

説明

名前	説明
pDev	WDC デバイス情報構造体へのポインタ [A.3.4]
dwAddrSpace	アドレス空間番号

戻り値

指定したアドレス空間番号 (`pDev->pAddrDesc[dwAddrSpace]`) のためのデバイスのアドレス情報構造体 (`WDC_ADDR_DESC` [A.3.2]) へのポインタを返します。

A.3.8 WDC_GET_ENABLED_INT_TYPE マクロ

目的

- WDC デバイスの **`dwEnabledIntType`** `WD_INTERRUPT` フィールドの値を取得するためのユーティリティ マクロ。以下のマクロの戻り値の説明のとおり、`WDC_IntEnable()` でこのフィールドをアップデートし、デバイスの有効な割り込みタイプを示します。

プロトタイプ

```
WDC_GET_ENABLED_INT_TYPE(pDev)
```

パラメータ

名前	型	入出力
➢ PDev	PWDC_DEVICE	入力

説明

名前	説明
pDev	WDC デバイス情報構造体へのポインタ [A.3.4]

戻り値

以下のデバイスの有効な割り込みタイプを返します:

- **INTERRUPT_MESSAGE_X**: MSI-X (Extended Message-Signaled Interrupts)
- **INTERRUPT_MESSAGE**: MSI (Message-Signaled Interrupts)
- **INTERRUPT_LEVEL_SENSITIVE**: レガシーなレベル センシティブ割り込み

- **INTERRUPT_LATCHED**: レガシーなエッジ トリガー割り込み
  このフラグの値は 0 で、他の割り込みフラグが設定されていない場合のみ適用されます。

注釈

- **Windows** API は MSI と MSI-X の違いは区別しません。従って、Windows OS では、WinDriver の関数は、MSI と MSI-X の両方に対して、**INTERRUPT_MESSAGE** フラグを設定します。
- `WDC_IntEnable()` を呼んで、対象の PCI カードで割り込みを有効にした後にのみ、このマクロを呼びます。
- このマクロは、通常、1 つ以上の割り込みのタイプをサポートする PCI デバイスの場合にのみ関連します。
- 戻り値を `WDC_INT_IS_MSI` マクロへ渡して、MSI または MSI-X が有効か、確認します。

A.3.9 WDC_GET_INT_OPTIONS マクロ

目的

- WDC デバイスの割り込みオプションの値を取得するためのユーティリティ マクロ。以下のマクロの戻り値の説明のとおり、デバイスで対応する割り込みのタイプを示します。

プロトタイプ

```
WDC_GET_INT_OPTIONS(pDev)
```

パラメータ

名前	型	入出力
➢ PDev	PWDC_DEVICE	入力

説明

名前	説明
pDev	WDC デバイス情報構造体へのポインタ [A.3.4]

戻り値

以下のデバイスで対応する割り込みのタイプを示すビット マスクを返します:

- **INTERRUPT_MESSAGE_X**: MSI-X (Extended Message-Signaled Interrupts)
- **INTERRUPT_MESSAGE**: MSI (Message-Signaled Interrupts)
- **INTERRUPT_LEVEL_SENSITIVE**: レガシーなレベル センシティブ割り込み
- **INTERRUPT_LATCHED**: レガシーなエッジ トリガー割り込み
  このフラグの値は 0 で、他の割り込みフラグが設定されていない場合のみ適用されます。

注釈

- 戻ったオプションを `WDC_INT_IS_MSI` マクロへ渡して、そのオプションが MSI (`INTERRUPT_MESSAGE`) または MSI-X (`INTERRUPT_MESSAGE_X`) フラグを含むかどうか確認します。

A.3.10 WDC_INT_IS_MSI マクロ

目的

- 指定した割り込みタイプのビット マスクが、MSI (Message-Signaled Interrupts) または MSI-X (Extended Message-Signaled Interrupts) 割り込みタイプ フラグを含むかどうか確認するユーティリティ マクロ。

プロトタイプ

```
WDC_INT_IS_MSI(dwIntType)
```

パラメータ

名前	型	入出力
➢ dwIntType	DWORD	入力

説明

名前	説明
dwIntType	割り込みタイプのビット マスク

戻り値

指定した割り込みタイプのビット マスクが MSI (**INTERRUPT_MESSAGE**) または MSI-X (**INTERRUPT_MESSAGE_X**) フラグを含む場合、TRUE を返します。それ以外の場合、FALSE を返します。

A.3.11 WDC_GET_ENABLED_INT_LAST_MSG マクロ

目的

- デバイスの有効な MSI (Message-Signaled Interrupts) または MSI-X (Extended Message-Signaled Interrupts) の最後に受信した割り込みのメッセージ データを取得するユーティリティ マクロ (Windows Vista およびそれ以降)。

プロトタイプ

```
WDC_GET_ENABLED_INT_LAST_MSG(pDev)
```

パラメータ

名前	型	入出力
➢ pDev	PWDC_DEVICE	入力

説明

名前	説明
pDev	WDC デバイス情報構造体へのポインタ [A.3.4]

戻り値

指定したデバイスに対して、MSI または MSI-X が有効な場合、マクロは、デバイスの割り込みの最後に受信したメッセージのメッセージ データを返します。有効でない場合、0 を返します。

A.3.12 WDC_IS_KP マクロ

目的

- WDC デバイスが Kernel PlugIn ドライバを使用しているかどうかを確認するユーティリティ マクロ。

プロトタイプ

```
WDC_IS_KP(pDev)
```

パラメータ

名前	型	入出力
➢ PDev	PWDC_DEVICE	入力

説明

名前	説明
pDev	WDC デバイス情報構造体へのポインタ [A.3.4]

戻り値

デバイスが Kernel PlugIn ドライバを使用している場合は TRUE を返します。使用していない場合は FALSE を返します。

A.4 WD_xxx の構造体、型、および一般的な定義

このセクションでは、WDC_xxx API によって使用される基本の WD_xxx の構造体および型について説明します。このセクションで説明する API は、**WinDriver/include/windrvr.h** ヘッダーファイルで定義されます

A.4.1 WD_BUS_TYP 列挙型

バスの種類の列挙型。

列挙値	説明
WD_BUS_USB	ユニバーサル シリアル バス (USB)
WD_BUS_UNKNOWN	不明なバス
WD_BUS_ISA	ISA バス
WD_BUS_EISA	EISA (ISA Plug-and-Play) バス

WD_BUS_PCI	PCI バス
WD_BUS_PCMCIA	PCMCIA バス

A.4.2 ITEM_TYPE 列挙型

カード アイテムの種類の列挙型。

列挙値	説明
ITEM_NONE	不明なアイテム
ITEM_INTERRUPT	割り込みアイテム
ITEM_MEMORY	メモリ アイテム
ITEM_IO	I/O アイテム
ITEM_BUS	バス アイテム

A.4.3 WD_PCMCIA_ACC_SPEED 列挙型

PCMCIA バス アクセス スピードの列挙型。

列挙値	説明
WD_PCMCIA_ACC_SPEED_DEFAULT	デフォルトの PCMCIA バス アクセス スピードを使用する
WD_PCMCIA_ACC_SPEED_250NS	250 *ns*
WD_PCMCIA_ACC_SPEED_200NS	200 *ns*
WD_PCMCIA_ACC_SPEED_150NS	150 *ns*
WD_PCMCIA_ACC_SPEED_1000NS	100 *ns*

A.4.4 WD_PCMCIA_ACC_WIDTH 列挙型

PCMCIA バス幅の列挙型。

列挙値	説明
WD_PCMCIA_ACC_WIDTH_DEFAULT	デフォルトの PCMCIA バス幅を使用する
WD_PCMCIA_ACC_WIDTH_8BIT	8 ビット
WD_PCMCIA_ACC_WIDTH_16BIT	16 ビット

A.4.5 WD_PCMCIA_VPP 列挙型

PCMCIA コントローラのボルテージ パワー ピン (Vpp) のパワー レベルの列挙型。

列挙値	説明
WD_PCMCIA_VPP_DEFAULT	デフォルトの PCMCIA Vpp のパワー レベルを使用する
WD_PCMCIA_VPP_OFF	Vpp のボルテージを 0 (無効) に設定する
WD_PCMCIA_VPP_ON	Vpp のボルテージを 12V (有効) に設定する
WD_PCMCIA_VPP_AS_VSS	Vpp のボルテージを Vcc と同じにする

A.4.6 WD_PCI_ID 構造体

PCI デバイス ID 情報の構造体。

名前	型	説明
➢ dwVendorId	DWORD	ベンダー ID
➢ dwDeviceId	DWORD	デバイス ID

A.4.7 WD_PCMCIA_ID 構造体

PCMCIA デバイス ID 情報の構造体。

名前	型	説明
➢ wManufacturerId	WORD	製造元 ID
➢ wCardId	WORD	デバイス ID

A.4.8 WD_PCI_SLOT 構造体

PCI デバイスのロケーション情報の構造体。

名前	型	説明
➢ dwBus	DWORD	PCI バス番号 (0 ベース)
➢ dwSlot	DWORD	スロット番号 (0 ベース)
➢ dwFunction	DWORD	関数番号 (0 ベース)

A.4.9 WD_PCMCIA_SLOT 構造体

PCMCIA デバイスのロケーション情報の構造体。

名前	型	説明
➢ uBus	BYTE	PCMCIA バス番号 (0 ベース)
➢ uSocket	BYTE	ソケット番号 (0 ベース)
➢ uFunction	BYTE	関数番号 (0 ベース)

A.4.10 WD_ITEMS 構造体

カード リソース情報の構造体。

名前	型	説明
➢ item	DWORD	アイテムの種類 (ITEM_TYPE 列挙型 [A.4.2] を参照)。 このフィールドは、WDC_XXXGetDeviceInfo() 関数 (PCI: [A.2.7]、PCMCIA: [A.2.8]) または低レベルの WD_PciGetCardInfo() 関数および WD_PcmciaGetCardInfo() 関数によって更新されます。
➢ fNotSharable	DWORD	TRUE の場合、一度に 1 つのアプリケーションだけがメモリまたは I/O 範囲へアクセスするか、またはデバイスの割り込みをモニタします。 このフィールドは、WDC_XXXGetDeviceInfo() 関数 (PCI: [A.2.7]、PCMCIA: [A.2.8]) または低レベルの WD_PciGetCardInfo() 関数および WD_PcmciaGetCardInfo() 関数によって更新されます。
➢ dwOptions	DWORD	WDC_xxxDeviceOpen() (PCI [A.2.9] / PCMCIA [A.2.10] / ISA [A.2.11]) または低レベルの WD_CardRegister() 関数を呼び出す際に適用されるアイテム登録フラグのビットマスク。マスクは、WD_ITEM_OPTIONS 列挙型の値で構成されます。 • WD_ITEM_DO_NOT_MAP_KERNEL: このフラグは、メモリ アドレスの範囲をカーネルの仮想アドレス空間へマップせずに、ユーザーモードの仮想アドレス空間へのみマップするよう関数に指示します。 詳細は、注釈を参照してください。 注意: このフラグは、メモリ アイテムにのみ適用されます。 • WD_ITEM_ALLOW_CACHE (Windows および CE のみ): キャッシュとしてアイテムの物理メモリをマップします (I.Mem.dwPhysicalAddr)。 注意: このフラグは、カード上のローカル メモリではなく、ホストの RAM に関係するメモリ アイテムにのみ適用されます。
➢ I	union	アイテムの種類 (**item**) を基にしたリソース データの共有体
☐ Mem	struct	メモリ アイテム (item = ITEM_MEMORY) の情報
◆ dwPhysicalAddr	DWORD	物理メモリ範囲の最初のアドレス。 このフィールドは、WDC_XXXGetDeviceInfo() 関数 (PCI: [A.2.7]、PCMCIA: [A.2.8]) または低レベルの WD_PciGetCardInfo() 関数および WD_PcmciaGetCardInfo() 関数によって更新されます。
◆ dwBytes	DWORD	メモリ範囲のバイト単位での長さ。 このフィールドは、WDC_XXXGetDeviceInfo() 関数 (PCI: [A.2.7]、PCMCIA: [A.2.8]) または低レベルの WD_PciGetCardInfo() 関数および

		WD_PcmciaGetCardInfo() 関数によって更新されます。
◆ dwTransAddr	DWORD	メモリ範囲の物理的なベース アドレス (**dwPhysicalAddr**) のカーネルモード マップ。このフィールドは、高レベルの WDC_xxxDeviceOpen() 関数 (PCI [A.2.9] / PCMCIA [A.2.10] / ISA [A.2.11]) によって呼び出される WD_CardRegister() によって更新されます。
◆ dwUserDirectAddr	DWORD	メモリ範囲の物理的なベース アドレス (**dwPhysicalAddr**) のユーザーモード マップ。このフィールドは、高レベルの WDC_xxxDeviceOpen() 関数 (PCI [A.2.9] / PCMCIA [A.2.10] / ISA [A.2.11]) によって呼び出される WD_CardRegister() によって更新されます。
◆ dwCpuPhysicalAddr	DWORD	カードの物理メモリのベース アドレス (**dwPhysicalAddr**) を、バス固有の値から CPU 値に変換します。このフィールドは、高レベルの WDC_xxxDeviceOpen() 関数 (PCI [A.2.9] / PCMCIA [A.2.10] / ISA [A.2.11]) によって呼び出される WD_CardRegister() によって更新されます。
◆ dwBar	DWORD	ベース アドレス レジスタ (BAR) 番号。このフィールドは、WDC_XXXGetDeviceInfo() 関数 (PCI: [A.2.7]、PCMCIA: [A.2.8]) または低レベルの WD_PciGetCardInfo() 関数および WD_PcmciaGetCardInfo() 関数によって更新されます。
☐ IO	struct	I/O アイテム (item = ITEM_IO) の情報
◆ dwAddr	DWORD	I/O 範囲の最初のアドレス。このフィールドは、WDC_XXXGetDeviceInfo() 関数 (PCI: [A.2.7]、PCMCIA: [A.2.8]) または低レベルの WD_PciGetCardInfo() 関数および WD_PcmciaGetCardInfo() 関数によって更新されます。
◆ dwBytes	DWORD	I/O 範囲のバイト単位での長さ。このフィールドは、WDC_XXXGetDeviceInfo() 関数 (PCI: [A.2.7]、PCMCIA: [A.2.8]) または低レベルの WD_PciGetCardInfo() 関数および WD_PcmciaGetCardInfo() 関数によって更新されます。
◆ dwBar	DWORD	ベース アドレス レジスタ (BAR) 番号。このフィールドは、WDC_XXXGetDeviceInfo() 関数 (PCI: [A.2.7]、PCMCIA: [A.2.8]) または低レベルの WD_PciGetCardInfo() 関数および WD_PcmciaGetCardInfo() 関数によって更新されます。
☐ Int	struct	割り込みアイテム (item = ITEM_INTERRUPT) の情報
◆ dwInterrupt	DWORD	物理的な割り込み要求 (IRQ) 番号。このフィールドは、WDC_XXXGetDeviceInfo() 関

		数 (PCI: [A.2.7]、PCMCIA: [A.2.8]) または低レベルの `WD_PciGetCardInfo()` 関数および `WD_PcmciaGetCardInfo()` 関数によって更新されます。
◆ dwOptions	DWORD	以下のフラグで構成される割り込みビットマスク。 • `INTERRUPT_MESSAGE_X` – MSI-X (Extended Message-Signaled Interrupts) をサポートするハードウェアを示します。 このオプションは、Linux で PCI カードに対してのみ適用されます。 • `INTERRUPT_MESSAGE` – Linux では、MSI (Message-Signaled Interrupts) をサポートするハードウェアを示します。 Windows では、MSI または MSI-X をサポートするハードウェアを示します。 このオプションは、Linux、Windows Vista およびそれ以降で PCI カードに対してのみ適用されます。 • `INTERRUPT_LEVEL_SENSITIVE` – レベル センシティブ割り込みをサポートするハードウェアを示します。 • `INTERRUPT_LATCHED` – レガシーなエッジ トリガー割り込みをサポートするデバイスを示します。 このフラグの値は 0 で、従って、他の割り込みフラグが設定されていない場合のみ適用されます。 注意: • Plug-and-Plau ハードウェア (PCI / PCMCIA / ISA) の場合、WinDriver の `WDC_PciGetDeviceInfo()` (PCI) または `WDC_PcmciaGetDeviceInfo()` (PCMCIA) 関数を使用して (または低レベル `WD_PciGetCardInfo()` または `WD_PcmciaGetCardInfo()`)、サポートする割り込みタイプを含む Plug-and-Play のハードウェア情報を取得します。 その他の割り込みオプション: • `INTERRUPT_CE_INT_ID` - Windows CE では (その他のオペレーティング システムとは異なり)、物理的な割り込み番号を論理的な割り込み番号へ抽出します。このビットを設定することにより、**`dwInterrupt`** の値を論理的な割り込み番号として参照し、物理的な割り込み番号に変換するよう WinDriver に指示します。
◆ hInterrupt	DWORD	低レベル `WD_xxx()` WinDriver割り込み API で必要な内部の WinDriver 割り込み構造体へのハンドル。 このフィールドは、高レベル `WDC_xxxDeviceOpen()` 関数 (PCI [A.2.9] / PCMCIA [A.2.10] / ISA [A.2.11]) によって呼び出される `WD_CardRegister()` によって更新されます。
☐ Bus	WD_BUS	バス アイテム (`item = ITEM_BUS`) のデータ
◆ dwBusType	WD_BUS_TYPE	デバイスのバスの種類 (`WD_BUS_TYPE` 列挙型 [A.4.1] を参照)

◆ dwBusNum	DWORD	バス番号
◆ dwSlotFunc	DWORD	デバイスのスロット (ソケット) および関数情報。下位 3 ビットは関数番号を、残りのビットはスロット / ソケット番号を示します。例えば、0x80 (<=> バイナリでは 10000000) の場合、関数番号は 0 (下位 3 ビット: 000) で、スロット / ソケット番号は 0x10 (残りのビット: 10000) です。 このフィールドは、`WDC_XXXGetDeviceInfo()` 関数 (PCI: [A.2.7]、PCMCIA: [A.2.8]) または低レベルの `WD_PciGetCardInfo()` 関数および `WD_PcmciaGetCardInfo()` 関数によって更新されます。
➢ Val	struct	内部使用のために予約

A.4.11 WD_CARD 構造体

カード情報の構造体。

名前	型	説明
➢ dwItems	DWORD	カード上のアイテム (リソース) の数
➢ Item	WD_ITEMS	カードのリソース (アイテム) 情報構造体の配列

A.4.12 WD_PCI_CARD_INFO 構造体

PCI カード情報の構造体。

名前	型	説明
➢ pciSlot	WD_PCI_SLOT	`WDC_PciScanDevices()` [A.2.4] (または低レベル `WD_PciScanCards()` 関数) の呼び出しにより取得可能な PCI デバイスのロケーション情報の構造体 [A.4.8]
➢ Card	WD_CARD	カード情報の構造体 [A.4.10]

A.4.13 WD_PCMCIA_CARD_INFO 構造体

PCMCIA カード情報の構造体。

名前	型	説明
➢ pcmciaSlot	WD_PCMCIA_SLOT	`WDC_PcmciaScanDevices()` [A.2.6] (または低レベルの `WD_PcmciaScanCards()` 関数の呼び出しにより取得可能な PCMCIA デバイスのロケーション情報の構造体 [A.4.9]
➢ Card	WD_CARD	カード情報の構造体 [A.4.10]
➢ cVersion	CHAR	バージョン (文字列)
➢ cManufacturer	CHAR [WD_PCMCIA_	製造元 (文字列)

	MANUFACTURER_LEN]	
➢ cProductName	CHAR [WD_PCMCIA_ PRODUCTNAME_LEN]	製品 (文字列)
➢ wManufacturerId	WORD	製造元 ID
➢ wCardId	WORD	デバイス ID
➢ wFuncId	WORD	関数 ID

A.4.14 WD_DMA 構造体

DMA (Direct Memory Access) 情報の構造体。

名前	型	説明
➢ hDma	DWORD	DMA バッファのハンドル (割り当てに失敗した場合は 0)。このハンドルは、`WDC_DMAContigBufLock()` [A.2.38] および `WDC_DMASGBufLock()` [A.2.39] (または低レベルの `WD_DMALock()` 関数) により返されます。
➢ pUserAddr	PVOID	DMA バッファのユーザーモードのマップ アドレス。このマップは、`WDC_DMAContigBufLock()` [A.2.38] および `WDC_DMASGBufLock()` [A.2.39] (この関数では、呼び出し元より提供されたユーザーモード バッファ (`pBuf`) が使用されます)、または 低レベルの `WD_DMALock()` 関数により返されます。 注意: DMA オプションのビットマスク フィールド (**`dwOptions`**) で `DMA_KERNEL_ONLY` フラグが設定されている場合、このフィールドは更新されません。
➢ pKernelAddr	KPTR	DMA バッファのカーネルモードのマップ アドレス。このマップは、`WDC_DMAContigBufLock()` [A.2.38] および `WDC_DMASGBufLock()` [A.2.39] (Windows 2000 / XP / Server 2003 / Server 2008 / Vista / 7 の場合)、または低レベルの `WD_DMALock()` 関数 (Windows 2000 / XP / Server 2003 / Server 2008 / Vista / 7 上の連続 DMA バッファおよびスキャッタ / ギャザー DMA の場合) により返されます。
➢ dwBytes	DWORD	DMA バッファのバイト単位でのサイズ
➢ dwOptions	DWORD	以下の列挙値で構成される DMA オプションのビットマスク。 **注意:** (以下の説明に従って) `WDC_DMASGBufLock()` 関数および `WDC_DMAContigBufLock()` 関数にも適用されるオプションは、これらの関数の `dwOptions` パラメータ内で設定されなければなりません。これらの関数により返される `WD_DMA` 構造体の `dwOptions` フィールドは、適宜更新されます。 DMA フラグ: • **`DMA_FROM_DEVICE`**: デバイスからメモリへ転送するために、DMA バッファを同期させます。 • **`DMA_TO_DEVICE`**: メモリからデバイスへ転送するために、DMA バッファを同期させます。 • **`DMA_TO_FROM_DEVICE`**: 両方向 (デバイスからメモリ: DMA_FROM_DEVICE /メモリからデバイス: DMA_TO_DEVICE) の転送用に DMA バッファを同期させ

		ます。 • **DMA_KERNEL_BUFFER_ALLOC**: 物理メモリの連続 DMA バッファを割り当てます。 デフォルト (このフラグが設定されていない場合) では、スキャッタ / ギャザー DMA バッファを割り当てます。 低レベルの `WD_DMALock()` 関数を呼び出して連続 DMA バッファを割り当てる場合、このフラグを設定します。WDC API を使用する場合、このフラグは `WDC_DMAContigBufLock()` [A.2.38] により自動的に設定され、このフラグが適用されないスキャッタ / ギャザー DMA バッファを割り当てる `WDC_DMASGBufLock()` [A.2.39] が使用されるため、このフラグを設定する必要はありません。 • **DMA_KBUF_BELOW_16M**: メイン メモリの最初の 16 MB 内に物理的な DMA バッファを割り当てます。 このフラグは、連続 DMA バッファ (`WDC_DMAContigBufLock()` 関数 [A.2.38] または `DMA_KERNEL_BUFFER_ALLOC` フラグと共に低レベルの `WD_DMALock()` 関数を呼び出す場合) にのみ適用されます。 • **DMA_LARGE_BUFFER**: 大容量の DMA バッファのロックを可能にします (`dwBytes > 1MB`)。 このフラグは、スキャッタ / ギャザー DMA にのみ適用されます。 低レベルの `WD_DMALock()` 関数を呼び出して大容量の DMA バッファを割り当てる場合、このフラグを設定します。WDC API を使用する場合、大容量の DMA バッファを割り当てるために `WDC_DMASGBufLock()` [A.2.39] が呼び出されるとこのフラグは自動的に設定され、このフラグが適用されない連続 DMA バッファを割り当てる `WDC_DMASGBufLock()` [A.2.39] が使用されるため、このフラグを設定する必要はありません。 • **DMA_ALLOW_CACHE**: DMA バッファのキャッシュを許可します。 • **DMA_KERNEL_ONLY_MAP**: 割り当てられた DMA バッファをユーザーモードにマップしません (カーネルモードにのみマップします)。 このフラグは、`DMA_KERNEL_BUFFER_ALLOC` フラグが適用される場合のみ適用されます (上記を参照)。 • **DMA_ALLOW_64BIT_ADDRESS**: (ターゲット プラットフォームでサポートされる場合) 64 ビットの DMA アドレスの割り当てを許可します。このフラグは、Windows と Linux でサポートされています。
➢ dwPages	DWORD	割り当てられたバッファに使用される物理メモリブロックの数。 連続 DMA バッファの場合、このフィールドは常に 1 に設定されます。
➢ hCard	DWORD	(`WD_CardRegister()` を呼び出し) `WDC_xxxDeviceOpen()` により取得され、WDC デバイス構造体に格納された低レベルの WinDriver カードのハンドル
➢ Page	WD_DMA_PAGE	物理メモリのページ情報構造体の配列。 連続 DMA バッファの場合、この配列は常に 1 つの要素のみ保持します (`dwPages` を参照)。
☐ pPhysicalAddr	KPTR	ページの物理アドレス

☐ dwBytes	DWORD	ページのバイト単位でのサイズ

A.4.15 WD_TRANSFER 構造体

メモリ/IO の読み取り / 書き込み転送コマンド情報の構造体。

名前	型	説明
➢ cmdTrans	DWORD	実行する転送の種類を示す値 (**windrvr.h** の `WD_TRANSFER_CMD` 列挙型の定義を参照)。 転送コマンドは次のいずれかです。 • 次の形式の読み取り / 書き込み転送コマンド: **`<dir><p>_[S]<size>`** 説明: **`<dir>`**: **`R`** (読み取り)、**`W`** (書き込み) **`<p>`**: **`P`** (I/O)、**`M`** (メモリ) **`<S>`**: シングル転送とは対照的である、文字列 (ブロック) 転送を示します。 **`<size>`**: BYTE、WORD、DWORD または QWORD • **`CMD_MASK`**: このコマンドは、割り込み転送コマンドを割り込みを有効にする関数 (`WDC_IntEnable()` [A.2.43]、低レベルの `InterruptEnable()` または `WD_IntEnable()` 関数) へ渡す際に適用されます。 `CMD_MASK` は、割り込み元を決定する割り込みマスク コマンドです。このコマンドが設定されていると、カーネルで割り込みを受け取った際に、WinDriver は `WD_TRANSFER`コマンド配列内の以前の読み取りコマンドの値と、マスク転送コマンドの `Data` フィールド共有体のメンバーで設定されたマスクを比較します。たとえば、`pTransCmds WD_TRANSFER` 配列では、`pTransCmds[i-1].cmdTrans` が `RM_BYTE` の場合、WinDriver は `pTransCmds[i-1].Data.Byte & pTransCmds[i].Data.Byte` を実行します。値が一致した場合、ドライバは割り込みの所有権を要求し、ユーザーモードにコントロールが戻ると、割り込みを有効にする関数に渡された割り込み処理ルーチンが呼び出されます。そうでない場合、ドライバは割り込みの所有権を拒否し、割り込み処理ルーチンは呼び出されず、配列内の後に続く転送コマンドは実行されません。 (割り込みの承認と拒否は、レガシーな割り込みを処理する場合のみ関連します。MSI / MSI-X 割り込みは共有されないので、WinDriver は常にこの割り込みのコントロールを承認します) 注意: `CMD_MASK` コマンドは、読み取り転送コマンド (`RM_XXX / RP_XXX`) よりも先に実行する必要があります。
➢ dwPort	KPTR	デバイス (`WDC_DEVICE` [A.3.3]) に格納された I/O ポート アドレスまたはカーネル マップ仮想メモリアドレス: `dev.pAddrDesc[i].kptAddr` (*i* はアドレス空間のインデックス)。(低レベルの `WD_xxx()` API を使用する場合、これらの値は `cardReg.Card.Item[i]` アイテム内の `dwAddr` (I/O) フィールドおよび `dwTransAddr` (メモリ) フィールドに格納されています。
➢ dwBytes	DWORD	転送するバイト数
➢ fAutoinc	DWORD	文字列 (ブロック) 転送にのみに適用されます。 TRUE の場合、転送される各ブロックの後に I/O またはメモリ ポート / アドレスはインクリメントされます。 FALSE の場合、すべてのデータは同じポート/アドレスから、または同じポート / アドレスへ転送されます。
➢ dwOptions	DWORD	0 でなければなりません

➢ Data	union	転送されるデータ バッファ (書き込みコマンドの場合は入力、読み取りコマンドの場合は出力)
☐ Byte	BYTE	8 ビット転送に使用されます。
☐ Word	WORD	16 ビット転送に使用されます。
☐ Dword	UINT32	32 ビット転送に使用されます。
☐ Qword	UINT64	64 ビット転送に使用されます。
☐ pBuffer	PVOID	文字列 (ブロック) 転送に使用されます (転送されるデータ バッファへのポインタ)。

A.5 Kernel PlugIn – カーネルモード関数

次に Kernel PlugIn ドライバに組み込むコールバック関数を説明します。これらはその「呼び出す」イベントが発生すると呼び出されます。例えば、KP_Init() [A.5.1] はドライバをロードしたときに呼び出されるコールバック関数です。ロードした際に実行するコードはこの関数に追加してください。

KP_Init() はドライバ名と KP_Open() 関数を設定します。

KP_Open() はドライバのコールバック関数の残りを設定します。

例:

```
kpOpenCall->funcClose = KP_Close;
kpOpenCall->funcCall = KP_Call;
kpOpenCall->funcIntEnable = KP_IntEnable;
kpOpenCall->funcIntDisable = KP_IntDisable;
kpOpenCall->funcIntAtIrql = KP_IntAtIrql;
kpOpenCall->funcIntAtDpc = KP_IntAtDpc;
kpOpenCall->funcIntAtIrqlMSI = KP_IntAtIrqlMSI;
kpOpenCall->funcIntAtDpcMSI = KP_IntAtDpcMSI;
kpOpenCall->funcEvent = KP_Event;
```

注意

このリファレンスでは便宜上、Kernel PlugIn コールバック関数を KP_XXX() として表示します。つまり、KP_Open()、KP_Call() など。ただし、Kernel PlugIn で実装するコールバック関数を作成した場合、KP_Init() 以外で、作成した Kernel PlugIn コールバック関数に任意の名前を付けても構いません。たとえば、DriverWizard で生成された Kernel PlugIn コードでは、コールバック関数名で、選択したドライバ名を使用します (たとえば、<MyKP> ドライバの場合、KP_MyKP_Open()、KP_MyKP_Call() など)。

A.5.1 KP_Init()

目的

- Kernel PlugIn ドライバをロードた際に、呼び出されます。
  Kernel PlugIn ドライバの名前と KP_Open() [A.5.2] コールバック関数を設定します。

プロトタイプ

```
BOOL __cdecl KP_Init(KP_INIT *kpInit);
```

パラメータ

名前	型	入出力
➢ kpInit	KP_INIT*	
☐ dwVerWD	DWORD	出力
☐ cDriverName	CHAR[12]	出力
☐ funcOpen	KP_FUNC_OPEN	出力

説明

名前	説明
kpInit	Kernel PlugIn の初期化情報の構造体 [A.6.4] へのポインタ
dwVerWD	WinDriver Kernel PlugIn ライブラリのバージョン。
cDriverName	デイバスドライバ名 (最大 12 文字)。
funcOpen	`WD_KernelPlugInOpen()`を呼んだ際に実行される `KP_Open()` コールバック関数。これらの関数が、`pcKPDriverName` パラメータで設定されている有効な Kernel PlugIn ドライバで呼び出されると、`WD_KernelPlugInOpen()` は `WDC_xxxDeviceOpen()` 関数 (PCI [A.2.9] / PCMCIA [A.2.10] / ISA [A.2.11]) から呼び出されます。

戻り値

正常に終了すると TRUE を返します。そうでない場合、FALSE を返します。

注釈

- コードに `KP_Init()` 関数を定義して Kernel PlugIn ドライバと WinDriver をリンクする必要があります。`KP_Init()` はドライバがロードされたときに呼び出されます。ロード時に実行するコードはこの関数に含まれなければなりません。

例

```
BOOL __cdecl KP_Init(KP_INIT *kpInit)
{
  /* Check if the version of the WinDriver Kernel
     PlugIn library is the same version
     as windrvr.h and wd_kp.h */
  if (kpInit->dwVerWD != WD_VER)
  {
      /* You need to re-compile your Kernel PlugIn
         with the compatible version of the WinDriver
         Kernel PlugIn library, windrvr.h and wd_kp.h */
     return FALSE;
  }

  kpInit->funcOpen = KP_Open;
  strcpy (kpInit->cDriverName, "KPDriver");  /* Up to 12 chars */

  return TRUE;
}
```

A.5.2 KP_Open()

目的

- ユーザーモードで WD_KernelPlugInOpen() が呼び出された際に呼び出されます。
  これらの関数が、pcKPDriverName パラメータで設定されている有効な Kernel PlugIn ドライバで呼び出されると、WD_KernelPlugInOpen() は WDC_xxxDeviceOpen() 関数 (PCI [A.2.9] / PCMCIA [A.2.10] / ISA [A.2.11]) から自動的に呼び出されます。

この関数は Kernel PlugIn コールバック関数 (KP_Call() [A.5.4]、KP_IntEnable() [A.5.6] など) の残りを設定しその他の初期化を実行します (ドライバ コンテキストのメモリの割り当て、ユーザーモードから渡されたデータで書き込みなど)。
返されたドライバ コンテキスト (*ppDrvContext) を Kernel PlugIn コールバック関数の残りに渡されます。

プロトタイプ

```
BOOL __cdecl KP_Open(
    KP_OPEN_CALL *kpOpenCall,
    HANDLE hWD,
    PVOID pOpenData,
    PVOID *ppDrvContext);
```

パラメータ

名前	型	入出力
➢ kpOpenCall	KP_OPEN_CALL	入力
➢ hWD	HANDLE	入力
➢ pOpenData	PVOID	入力
➢ ppDrvContext	PVOID*	出力

説明

名前	説明
kpOpenCall	KP_xxx() コールバック関数のアドレスを格納する構造体 [A.6.5]。
hWD	WD_KernelPlugInOpen() が呼び出されたときの WinDriver のハンドル。
pOpenData	ユーザーモードから渡されたデータへのポインタ。
ppDrvContext	KP_Close() [A.5.3]、KP_Call() [A.5.4]、KP_IntEnable() [A.5.6]、および KP_Event() [A.5.5] 関数が呼ばれた際の、ドライバ コンテキスト データへのポインタ。これを使用してドライバ固有の情報を保存します。この情報をこれらのコールバック関数で共有します。

戻り値

正常に終了すると TRUE を返します。FALSE の場合、ユーザーモードから呼ばれた WD_KernelPlugInOpen()に失敗します。

例

```
BOOL __cdecl KP_Open(KP_OPEN_CALL *kpOpenCall, HANDLE hWD,
            PVOID pOpenData, PVOID *ppDrvContext)
{
    kpOpenCall->funcClose = KP_Close;
    kpOpenCall->funcCall = KP_Call;
    kpOpenCall->funcIntEnable = KP_IntEnable;
    kpOpenCall->funcIntDisable = KP_IntDisable;
    kpOpenCall->funcIntAtIrql = KP_IntAtIrql;
    kpOpenCall->funcIntAtDpc = KP_IntAtDpc;
    kpOpenCall->funcIntAtIrqlMSI = KP_IntAtIrqlMSI;
    kpOpenCall->funcIntAtDpcMSI = KP_IntAtDpcMSI;
    kpOpenCall->funcEvent = KP_Event;

    /* You can allocate driver context memory here: */
    *ppDrvContext = malloc(sizeof(MYDRV_STRUCT));
    return *ppDrvContext!=NULL;
}
```

A.5.3 KP_Close()

目的

- ユーザーモードから `WD_KernelPlugInClose()` が呼ばれた場合に呼び出されます。Kernel PlugIn で開かれたデバイス (`pcKPDriverName` パラメータで設定されている有効な Kernel PlugIn ドライバで `WDC_xxxDeviceOpen()` (PCI [A.2.9] / PCMCIA [A.2.10] / ISA [A.2.11]) が呼び出された場合) では、`WDC_xxxDeviceClose()` 関数 (PCI [A.2.12] / PCMCIA [A.2.13] / ISA [A.2.14]) は、Kernel PlugIn ドライバへのハンドルを閉じるために `WD_KernelPlugInClose()` を自動的に呼び出します。

この関数は Kernel PlugIn の除去を実行するのに使用します (以前にドライバ コンテキストに対して割り当てられたメモリの解放など)。

プロトタイプ

```
void __cdecl KP_Close(PVOID pDrvContext);

KP_FUNC_ CLOSE Kernel PlugnIn callback function type.
```

パラメータ

名前	型	入出力
➢ pDrvContext	PVOID	入力

説明

名前	説明
pDrvContext	`KP_Open()` [A.5.2] でセットされたドライバ コンテキスト データ。

戻り値

なし。

例

```
void __cdecl KP_Close(PVOID pDrvContext)
{
   if (pDrvContext)
        free(pDrvContext); /* Free allocated driver context memory */
}
```

A.5.4 KP_Call()

目的

- ユーザーモード アプリケーションが WDC_CallKerPlug() [A.2.17] (または、低レベルの WD_KernelPlugInCall()) 関数を呼び出した際に呼び出されます。

この関数は、ユーティリティ関数のメッセージ ハンドラです。

プロトタイプ

```
void __cdecl KP_Call(
    PVOID pDrvContext,
    WD_KERNEL_PLUGIN_CALL
    *kpCall,
    BOOL fIsKernelMode);

KP_FUNC_ CALL Kernel PlugnIn callback function type.
```

パラメータ

名前	型	入出力
➢ pDrvContext	PVOID	入力 / 出力
➢ kpCall	WD_KERNEL_PLUGIN_CALL	
☐ dwMessage	DWORD	入力
☐ pData	PVOID	入力 / 出力
☐ dwResult	DWORD	出力
➢ fIsKernelMode	BOOL	入力

説明

名前	説明
pDrvContext	KP_Open() [A.5.2] セットされたドライバ コンテキスト データで、KP_Close() [A.5.3]、KP_IntEnable() [A.5.6]、および KP_Event() [A.5.5] に渡されます。
kpCall	WDC_CallKerPlug() [A.2.17] (または、低レベルの WD_KernelPlugInCall()) 関数からのユーザーモード情報を持つ構造体、および / またはユーザーモードへ戻す情報を持つ構造体[A.6.3]。

fIsKernelMode	Windriver のカーネルにより渡されるパラメータ (注釈を参照 [A.5.4])。

戻り値

なし。

注釈

- ユーザーモードで `WDC_CallKerPlug()` [A.2.17] (または、低レベルの `WD_KernelPlugInCall()`) 関数を呼んで、カーネルモードで作成した `KP_Call()` [A.5.4] コールバック関数を呼びます。Kernel PlugIn の `KP_Call()` 関数は、渡されるメッセージによって、どのルーテンを実行するか決定します。
- `fIsKernelMode` パラメータは、Windriver のカーネルにより `KP_Call()` ルーチンに渡されます。パラメータについて変更する必要はありません。しかし、このパラメータがマクロを正確に機能させるために必要な `COPY_TO_USER_OR_KERNEL` にどのように渡されるのかを注意する必要があります。詳細はセクション A.5.10 を参照してください。

例

```
void __cdecl KP_Call(PVOID pDrvContext,
   WD_KERNEL_PLUGIN_CALL *kpCall, BOOL fIsKernelMode)
{
    kpCall->dwResult = MY_DRV_OK;
    switch (kpCall->dwMessage)
    {
        /* In this sample we implement a GetVersion message */
        case MY_DRV_MSG_VERSION:
            {
              DWORD dwVer = 100;
              MY_DRV_VERSION *ver = (MY_DRV_VERSION *)kpCall->pData;
              COPY_TO_USER_OR_KERNEL(&ver->dwVer, &dwVer,
                  sizeof(DWORD), fIsKernelMode);
              COPY_TO_USER_OR_KERNEL(ver->cVer, "My Driver V1.00",
                  sizeof("My Driver V1.00")+1, fIsKernelMode);

              kpCall->dwResult = MY_DRV_OK;
            }
            break;
        /* You can implement other messages here */
        default:
            kpCall->dwResult = MY_DRV_NO_IMPL_MESSAGE;
    }
}
```

A.5.5 KP_Event()

目的

- デバイスの Plug-and-Play またはパワー マネージメント イベントを受け取った際に呼び出されます。ユーザーモードのアプリケーションが最初に、`fUseKP = TRUE` で `WDC_EventRegister()` [A.2.46] を呼び出したこと (または、Kernel PlugIn ハンドルで低レベルの `EventRegister()` 関数を呼び出したこと) を前提としています (注釈を参照)。

プロトタイプ

```
BOOL __cdecl KP_Event(
    PVOID pDrvContext,
    WD_EVENT *wd_event);

KP_FUNC_ EVENT Kernel PlugnIn callback function type.
```

パラメータ

名前	型	入出力
➢ pDrvContext	PVOID	入力 / 出力
➢ wd_event	WD_EVENT*	入力

説明

名前	説明
pDrvContext	KP_Open() [A.5.2] によって設定されるドライバ コンテキスト データで、KP_Close() [A.5.3]、KP_IntEnable() [A.5.6]、および KP_Call() [A.5.4] へ渡されます。
wd_event	ユーザーモードから受信した PnP / パワー マネージメント イベント情報へのポインタ。

戻り値

イベントについてユーザーに TRUE を伝える。

注釈

- ユーザーモードの処理が、fUseKP = TRUE で WDC_EventRegister() [A.2.46] を呼び出した場合 (または、Kernel PlugIn ハンドルで低レベルの EventRegister() 関数を呼び出した場合)、KP_Event() は呼び出されます。

例

```
BOOL __cdecl KP_Event(PVOID pDrvContext, WD_EVENT *wd_event)
{
    /* Handle the event here */
    return TRUE; /* Return TRUE to notify the user about the event */
}
```

A.5.6 KP_IntEnable()

目的

- Kernel PlugIn ハンドルを持つユーザーモードから WD_IntEnable() が呼び出された際に呼び出されます。
  WD_IntEnable() は、WDC_IntEnable() [A.2.43] および InterruptEnable() から自動的に呼び出されます。

この関数によって設定される割り込みコンテキスト (*ppIntContext) は、残りの Kernel PlugIn 割り込み関数へ渡されます。

プロトタイプ

```
BOOL __cdecl KP_IntEnable (
    PVOID pDrvContext,
    WD_KERNEL_PLUGIN_CALL *kpCall,
    PVOID *ppIntContext);

KP_FUNC_ INT_ENABLE Kernel PlugnIn callback function type.
```

パラメータ

名前	型	入出力
➢ pDrvContext	PVOID	入力 / 出力
➢ kpCall	WD_KERNEL_PLUGIN_CALL	入力
☐ dwMessage	DWORD	入力
☐ pData	PVOID	入力 / 出力
☐ dwResult	DWORD	出力
➢ ppIntContext	PVOID*	入力 / 出力

説明

名前	説明
pDrvContext	KP_Open() [A.5.2] でセットされたドライバ コンテキスト データで、KP_Close() [A.5.3]、KP_Call() [A.5.4]、および KP_Event() [A.5.5] に渡されます。
kpCall	WD_IntEnable() [A.6.3] からの情報を持つ構造体。
ppIntContext	KP_IntDisable() [A.5.7] と Kernel PlugIn の割り込み処理関数へ渡される割り込みコンテキスト データへのポインタ。割り込み関連の特定情報を保持するのに使用します。

戻り値

正常に終了すると TRUE を返します。そうでない場合、FALSE を返します。

注釈

- この関数には、Kernel PlugIn 割り込み処理に必要な初期化をすべて含む必要があります。

例

```
BOOL __cdecl KP_IntEnable(PVOID pDrvContext,
     WD_KERNEL_PLUGIN_CALL *kpCall, PVOID *ppIntContext)
{
    DWORD *pIntCount;
    /* You can allocate specific memory for each interrupt
```

```
       in *ppIntContext */
    *ppIntContext = malloc(sizeof (DWORD));
    if (!*ppIntContext)
        return FALSE;
    /* In this sample the information is a DWORD used to
       count the incoming interrupts */
    pIntCount = (DWORD *) *ppIntContext;
    *pIntCount = 0; /* Reset the count to zero */
    return TRUE;
}
```

A.5.7 KP_IntDisable()

目的

- Kernel PlugIn で有効になった割り込みに対して、ユーザーモードから `WD_IntDisable()` を呼び出す際に呼び出されます。
  `WD_IntDisable()` は、`WDC_IntDisable()` [A.2.44] および `InterruptDisable()` から自動的に呼び出されます。
- この関数は、`KP_IntEnable()` [A.5.6] で割り当てたメモリを解放します。

プロトタイプ

```
void __cdecl KP_IntDisable(PVOID pIntContext);

KP_FUNC_ INT_DISABLE Kernel PlugIn callback function type.
```

パラメータ

名前	型	入出力
➢ pIntContext	PVOID	入力

説明

名前	説明
pIntContext	`KP_IntEnable()` [A.5.6] で設定した割り込みコンテキスト データ。

戻り値

なし。

例

```
void __cdecl KP_IntDisable(PVOID pIntContext)
{
  /* You can free the interrupt specific memory
     allocated to pIntContext here */
  free(pIntContext);
}
```

A.5.8 KP_IntAtIrql()

目的

- 高い割り込み要求レベルで実行される優先度の高い割り込み処理ルーチン。この関数は、Kernel PlugIn ドライバを使用して有効にされた割り込みを受け取った際に呼び出されます (`WDC_IntEnable()` [A.2.43]、低レベル `InterruptEnable()` 関数および `WD_IntEnable()` 関数を参照)。

プロトタイプ

```
BOOL __cdecl KP_IntAtIrql(
    PVOID pIntContext,
    BOOL *pfIsMyInterrupt);

KP_FUNC_ INT_AT_IRQL Kernel PlugIn callback function type.
```

パラメータ

名前	型	入出力
➢ pIntContext	PVOID	入力 / 出力
➢ pfIsMyInterrupt	BOOL*	出力

説明

名前	説明
pIntContext	`KP_IntEnable()` [A.5.6] で設定された割り込みコンテキスト データへのポインタで、`KP_IntAtDpc()` [A.5.9] (実行した場合) および `KP_IntDisable()` [A.5.7] へ渡されます。
pfIsMyInterrupt	このドライバの場合、`*pfIsMyInterrupt` を TRUE に設定します。それ以外の場合は、同じ割り込みが呼ばれる他のドライバのための割り込みサービス ルーチンを可能するために FALSE に設定します。

戻り値

DPC (deferred interrupt processing) を実行する場合は、TRUE を返します。そうでない場合、FALSE を返します。

注釈

- IRQL で実行されるコードは高い優先度の割り込みでないと割り込めません。
- 高い IRQL で実行されるコードは、次の制限があります:
  - 非ページ メモリに対してのみアクセス可能です。
  - 次の関数 (または、これらの関数を呼び出すラッパー関数) のみ呼び出し可能です:
    - アドレスまたは設定空間の読み取り / 書き込みを行う `WDC_xxx` 関数。
    - `WDC_MultiTransfer()` [A.2.24]、低レベル `WD_Transfer()`、`WD_MultiTransfer()` または `WD_DebugAdd()` 関数。

        - 高い割り込み要求レベルで呼び出し可能な OS 固有のカーネル関数 (WDK 関数など)。OS 固有のカーネル関数を使用すると、他の OS への互換性が損なわれることがある点に注意してください。
    - `malloc()`、`free()`、または前述以外の `WDC_xxx()` または `WD_xxx()` API は呼び出せません。
- 優先度が高いため、高い割り込み要求レベルで実行されるコードは最小限にする必要があります (たとえば、レベル センシティブな割り込みの検知のみなど)。残りのコードを `KP_IntAtDpc()` [A.5.9] で記述する必要があり、遅延 DISPATCH レベルで実行し、上記の制限に従いません。

例

```
BOOL __cdecl KP_IntAtIrql(PVOID pIntContext,
    BOOL *pfIsMyInterrupt)
{
    DWORD *pdwIntCount = (DWORD *) pIntContext;

    /* Check your hardware here to see if the interrupt belongs to you.
       If it does, you must set *pfIsMyInterrupt to TRUE.
       Otherwise, set *pfIsMyInterrupt to FALSE. */
    *pfIsMyInterrupt = FALSE;

    /* In this example we will schedule a DPC
       once in every 5 interrupts */
    (*pdwIntCount) ++;
    if (*pdwIntCount==5)
    {
        *pdwIntCount = 0;
        return TRUE;
    }

    return FALSE;
}
```

A.5.9 KP_IntAtDpc()

目的

- 引継ぎ処理のレガシーな割り込み処理ルーチン。
  `KP_IntAtIrql()` [A.5.8] が TRUE を返した場合、優先度の高いレガシーな割り込み処理の完了時に実行されます。

プロトタイプ

```
DWORD __cdecl KP_IntAtDpc(
    PVOID pIntContext,
    DWORD dwCount);

KP_FUNC_ INT_AT_DPC Kernel PlugnIn callback function type.
```

パラメータ

名前	型	入出力
➢ pIntContext	PVOID	入力 / 出力

➢ dwCount	DWORD	入力

説明

名前	説明
pIntContext	KP_IntEnable() [A.5.6] で設定した割り込みコンテキスト データで、KP_IntAtIrql() [A.5.8] および KP_IntDisable() [A.5.7] へ渡されます。
dwCount	最後の DPC 呼び出しから KP_IntAtIrql() [A.5.8] が TRUE を返した回数。dwCount が 1 の場合、最後の DPC 呼び出しから KP_IntAtIrql() が DPC を一度だけ要求したことを表します。値が 1 より大きい場合、KP_IntAtIrql() が DPC を既に数回要求したことを表します。しかし、その間隔が短かいために、各 DPC の要求に対し、KP_IntAtDpc() が呼び出されなかったことを示します。

戻り値

ユーザーモードに通知する回数を返します (WD_IntWait() の戻り値)。

注釈

- ほとんどの割り込み処理は、優先度の高い KP_IntAtIrql() [A.5.8] 割り込み処理ではなく、この関数内で実装されます。
- KP_IntAtDpc() の戻り値が 0 より大きい場合、WD_IntWait() は戻り、ユーザーモード割り込みハンドラは、KP_IntAtDpc() の戻り値に設定されている回数分呼び出されます。ユーザーモード割り込みハンドラを実行したくない場合、KP_IntAtDpc() は 0 を返す必要があります。

例

```
DWORD __cdecl KP_IntAtDpc(PVOID pIntContext, DWORD dwCount)
{
  /* Return WD_IntWait as many times as KP_IntAtIrql
     scheduled KP_IntAtDpc */
  return dwCount;
}
```

A.5.10 KP_IntAtIrqlMSI()

目的

- 高い割り込み要求レベルで実行される優先度の高い MSI (Message-Signaled Interrupts) / MSI-X (Extended Message-Signaled Interrupts) 処理ルーチン。この関数は、Kernel PlugIn を使用して有効にされた MSI / MSI-X を受け取った際に呼び出されます (WDC_IntEnable() [A.2.43]、低レベル InterruptEnable() 関数および WD_IntEnable() 関数を参照)。

プロトタイプ

```
BOOL __cdecl KP_PCI_IntAtIrqlMSI(
    PVOID pIntContext,
```

```
    ULONG dwLastMessage,
    DWORD dwReserved);

KP_FUNC_ INT_AT_IRQL_MSI Kernel PlugnIn callback function type.
```

パラメータ

名前	型	入出力
➢ pIntContext	PVOID	入力 / 出力
➢ dLastMessage	DWORD	入力
➢ dwReserved	DWORD	入力

説明

名前	説明
pIntContext	`KP_IntEnable()` [A.5.6] で設定された割り込みコンテキスト データへのポインタで、`KP_IntAtDpcMSI()` (実行した場合) および `KP_IntDisable()` [A.5.7] へ渡されます。
dLastMessage	最後に受信した割り込みのメッセージ データ (Windows Vista およびそれ以降でのみ適用)。
dwReserved	後で使用するために予約。このパラメータは使用しないでください。

戻り値

DPC (deferred MSI / MSI-X processing) を実行する場合は、TRUE を返します。そうでない場合、FALSE を返します。

注釈

- IRQL で実行されるコードは高い優先度の割り込みでないと割り込めません。
- 高い IRQL で実行されるコードは、次の制限があります:
  - 非ページ メモリに対してのみアクセス可能です。
  - 次の関数 (または、これらの関数を呼び出すラッパー関数) のみ呼び出し可能です:
    - アドレスまたは設定空間の読み取り / 書き込みを行う `WDC_xxx` 関数。
    - `WDC_MultiTransfer()` [A.2.24]、低レベル `WD_Transfer()`、`WD_MultiTransfer()` または `WD_DebugAdd()` 関数。
    - 高い割り込み要求レベルで呼び出し可能な OS 固有のカーネル関数 (WDK 関数など)。OS 固有のカーネル関数を使用すると、他の OS への互換性が損なわれることがある点に注意してください。
  - `malloc()`、`free()`、または前述以外の `WDC_xxx()` または `WD_xxx()` API は呼び出せません。
- 優先度が高いため、高い割り込み要求レベルで実行されるコードは最小限にする必要があります (たとえば、レベル センシティブな割り込みの検知のみなど)。残りのコードを `KP_IntAtDpcMSI()` で記述する必要があり、遅延 DISPATCH レベルで実行し、上記の制限に従いません。

例

```
BOOL __cdecl KP_PCI_IntAtIrqlMSI(PVOID pIntContext,
    ULONG dwLastMessage, DWORD dwReserved)
{
    return TRUE;
}
```

A.5.11 KP_IntAtDpcMSI()

目的

- 引継ぎ処理の MSI (Message-Signaled Interrupts) / MSI-X (Extended Message-Signaled Interrupts) レガシーな割り込み処理ルーチン。
  `KP_IntAtIrqlMSI()` が TRUE を返した場合、優先度の高い MSI / MSI-X 処理の完了時に実行されます。

プロトタイプ

```
DWORD __cdecl KP_IntAtDpcMSI(
  PVOID pIntContext,
  DWORD dwCount,
  ULONG dwLastMessage,
  DWORD dwReserved);

KP_FUNC_ INT_AT_DPC_MSI Kernel PlugnIn callback function type.
```

パラメータ

名前	型	入出力
➢ pIntContext	PVOID	入力 / 出力
➢ dwCount	DWORD	入力
➢ dwLastMessage	DWORD	入力
➢ dwReserved	DWORD	入力

説明

名前	説明
pIntContext	`KP_IntEnable()` [A.5.6] で設定した割り込みコンテキスト データで、`KP_IntAtIrqlMSI()` および `KP_IntDisable()` [A.5.7] へ渡されます。
dwCount	最後の DPC 呼び出しから `KP_IntAtIrqlMSI()` が TRUE を返した回数。dwCount が 1 の場合、最後の DPC 呼び出しから `KP_IntAtIrqlMSI()` が DPC を一度だけ要求したことを表します。値が 1 より大きい場合、`KP_IntAtIrqlMSI()` が DPC を既に数回要求したことを表します。しかし、その間隔が短いために、各 DPC の要求に対し、`KP_IntAtDpcMSI()` が呼び出されなかったことを示します。

dLastMessage	最後に受信した割り込みのメッセージ データ (Windows Vista およびそれ以降でのみ適用)。
dwReserved	後で使用するために予約。このパラメータは使用しないでください。

戻り値

ユーザーモードに通知する回数を返します (`WD_IntWait()` の戻り値)。

注釈

- ほとんどの MSI / MSI-X 処理は、優先度の高い `KP_IntAtIrqlMSI()` 割り込み処理ではなく、この関数内で実装されます。
- `KP_IntAtDpcMSI()` の戻り値が 0 より大きい場合、`WD_IntWait()` は戻り、ユーザーモード割り込みハンドラは、`KP_IntAtDpcMSI()` の戻り値に設定されている回数分呼び出されます。ユーザーモード割り込みハンドラを実行したくない場合、`KP_IntAtDpcMSI()` は 0 を返す必要があります。

例

```
DWORD __cdecl KP_IntAtDpcMSI(PVOID pIntContext, DWORD dwCount,
  ULONG dwLastMessage, DWORD dwReserved)
{
  /* Return WD_IntWait as many times as KP_IntAtIrqlMSI
     scheduled KP_IntAtDpcMSI */
  return dwCount;
}
```

A.5.12 COPY_TO_USER_OR_KERNEL、COPY_FROM_USER_OR_KERNEL

目的

- データをユーザーモードから Kernel PlugIn へコピーまたは Kernel PlugIn からユーザーモードへコピーするマクロ。

注釈

- `COPY_TO_USER_OR_KERNEL` および `COPY_FROM_USER_OR_KERNEL` は、Kernel PlugIn 内でユーザーモード メモリ アドレスなどへ / からアクセスする (必要がある) ときにデータのコピーを行うためのマクロです。ユーザーモード処理が I/O オペレーションの途中で変更されても、データをコピーすることで、ユーザーモード アドレスを正しく使用することができます。ユーザーモードの処理が変更するような長いオペレーションで使用します。マクロを使用したコピーは、サポートするすべてのオペレーティング システムで有効です。
- Kernel PlugIn の関数内からユーザーモードにアクセスする場合、カーネルモードの割り込み処理ルーチンを実行する前に Kernel PlugIn の変数にデータをコピーしてください。
- `COPY_TO_USER_OR_KERNEL` および `COPY_FROM_USER_OR_KERNEL` マクロは **`WinDriver¥include¥kpstdlib.h`** ヘッダー ファイルで定義されています。
- `COPY_TO_USER_OR_KERNEL` マクロを使用した例については、**`WinDriver/samples/pci_diag/kp_pci /kp_pci .c`** にあるサンプル Kernel PlugIn ファイルの `KP_Call()` [A.5.4] の実装 (`KP_PCI_Call()`) を参照してください)。

- ユーザーモードおよび Kernel PlugIn ルーチン (例、`KP_IntAtIrql()` [A.5.8] および `KP_IntAtDpc()` [A.5.9]) 間で安全なデータ バッファの共有についての詳細は、Web サイトの テクニカル ドキュメント #41「Kernel PlugIn と DMA またはその他の目的のユーザーモード プロジェクト間で、メモリ バッファをどのように共有しますか？」を参照してください。

A.5.13 Kernel PlugIn の同期 API

このセクションでは、Kernel PlugIn の同期 API について説明します。
これらの API では、次の同期メカニズムをサポートしています:

- シングルまたはマルチ CPU システム上のスレッド間の同期に使用されるスピンロック [A.5.11.2 - A.5.11.5]

  注意

  Kernel PlugIn のスピンロック関数は、高い割り込み要求レベルとは別に、任意のコンテキストから呼び出すことができます。そのため、`KP_IntAtIrql()` [A.5.8] と `KP_IntAtIrqlMSI()` を除く、任意の Kernel PlugIn 関数から呼び出すことができます (`KP_IntAtDpc()` [A.5.9] と `KP_IntAtDpcMSI()` から呼び出すこともできます)。

- 複数のスレッドで共有され、複雑なアトミック処理が実行される変数へのアクセスの同期に使用されるインターロック操作 [A.5.11.6 - A.5.11.7]。

  注意

  Kernel PlugIn のインターロック関数は、高い割り込み要求レベルを含む、Kernel PlugIn の任意のコンテキストから呼び出すことができます。そのため、Kernel PlugIn 割り込み処理関数を含む、任意の Kernel PlugIn 関数から呼び出すことができます。

A.5.13.1 Kernel PlugIn の同期の型

Kernel PlugIn の同期 API では、次の型を使用しています:

- **`KP_SPINLOCK`** - Kernel PlugIn のスピンロック オブジェクトの構造体:

  ```
  typedef struct _KP_SPINLOCK KP_SPINLOCK;
  ```

  `_KP_SPINLOCK` は、ユーザーには不透明な、内部 WinDriver スピンロック オブジェクトの構造体です。

- **`KP_INTERLOCKED`** - Kernel PlugIn のインターロック操作カウンタ:

  ```
  typedef volatile int KP_INTERLOCKED;
  ```

A.5.13.2 kp_spinlock_init()

目的

- 新しい Kernel PlugIn スピンロック オブジェクトを初期化します。

プロトタイプ

```
KP_SPINLOCK * kp_spinlock_init(void);
```

戻り値

成功した場合は、新しい Kernel PlugIn スピンロック オブジェクト [A.5.11.1] へのポインタを返します。そうでない場合は NULL を返します。

A.5.13.3 kp_spinlock_wait()

目的

- Kernel PlugIn スピンロック オブジェクトを待機します。

プロトタイプ

```
void kp_spinlock_wait(KP_SPINLOCK *spinlock);
```

パラメータ

名前	型	入出力
➢ spinlock	KP_SPINLOCK*	入力

説明

名前	説明
Spinlock	待機する Kernel PlugIn スピンロック オブジェクト [A.5.11.1] へのポインタ

戻り値

なし。

A.5.13.4 kp_spinlock_release()

目的

- Kernel PlugIn スピンロック オブジェクトを解放します。

プロトタイプ

```
void kp_spinlock_release(KP_SPINLOCK *spinlock);
```

パラメータ

名前	型	入出力
➢ spinlock	KP_SPINLOCK*	入力

説明

名前	説明
spinlock	解放する Kernel PlugIn スピンロック オブジェクト [A.5.11.1] へのポインタ

戻り値

なし。

A.5.13.5 kp_spinlock_uninit()

目的

- Kernel PlugIn スピンロック オブジェクトの終了処理を実行します。

プロトタイプ

```
void kp_spinlock_uninit(KP_SPINLOCK *spinlock);
```

パラメータ

名前	型	入出力
➢ spinlock	KP_SPINLOCK*	入力

説明

名前	説明
spinlock	終了処理を実行する Kernel PlugIn スピンロック オブジェクト [A.5.11.1] へのポインタ

戻り値

なし。

A.5.13.6 kp_interlocked_init()

目的

- Kernel PlugIn インターロック カウンタを初期化します。

プロトタイプ

```
void kp_interlocked_init(KP_INTERLOCKED *target);
```

パラメータ

名前	型	入出力
➢ target	KP_INTERLOCKED*	入力 / 出力

説明

名前	説明
target	初期化する Kernel PlugIn インターロック カウンタ [A.5.11.1] へのポインタ

戻り値

なし。

A.5.13.7 kp_interlocked_uninit()

目的

- Kernel PlugIn インターロック カウンタの終了処理を実行します。

プロトタイプ

```
void kp_interlocked_uninit(KP_INTERLOCKED *target);
```

パラメータ

名前	型	入出力
➢ target	KP_INTERLOCKED*	入力 / 出力

説明

名前	説明
target	終了処理を実行する Kernel PlugIn インターロック カウンタ [A.5.11.1] へのポインタ

戻り値

なし。

A.5.13.8 kp_interlocked_increment()

目的

- Kernel PlugIn インターロック カウンタの値を 1 だけ増分 (インクリメント) します。

プロトタイプ

```
int kp_interlocked_increment(KP_INTERLOCKED *target);
```

パラメータ

名前	型	入出力
➢ target	KP_INTERLOCKED*	入力 / 出力

説明

名前	説明
target	増分 (インクリメント) する Kernel PlugIn インターロック カウンタ [A.5.11.1] へのポインタ

戻り値

インターロック カウンタの新しい値 (`target`) を返します。

A.5.13.9 kp_interlocked_decrement()

目的

- Kernel PlugIn インターロック カウンタの値を 1 だけ減少 (デクリメント) します。

プロトタイプ

```
int kp_interlocked_decrement(KP_INTERLOCKED *target);
```

パラメータ

名前	型	入出力
➢ target	KP_INTERLOCKED*	入力 / 出力

説明

名前	説明
target	減少 (デクリメント) する Kernel PlugIn インターロック カウンタ [A.5.11.1] へのポインタ

戻り値

インターロック カウンタの新しい値 (`target`) を返します。

A.5.13.10 kp_interlocked_add()

目的

- Kernel PlugIn インターロック カウンタの現在値に指定した値を加えます。

プロトタイプ

```
int kp_interlocked_add(
    KP_INTERLOCKED *target,
    int val);
```

パラメータ

名前	型	入出力
➢ target	KP_INTERLOCKED*	入力 / 出力
➢ val	val	入力

説明

名前	説明
target	増分する Kernel PlugIn インターロック カウンタ [A.5.11.1] へのポインタ
val	インターロック カウンタ (`target`) に加える値

戻り値

インターロック カウンタの新しい値 (`target`) を返します。

A.5.13.11 kp_interlocked_read()

目的

- Kernel PlugIn インターロック カウンタの値を読み取ります。

プロトタイプ

```
int kp_interlocked_read(KP_INTERLOCKED *target);
```

パラメータ

名前	型	入出力
➢ target	KP_INTERLOCKED*	入力

説明

名前	説明
target	読み取る Kernel PlugIn インターロック カウンタ [A.5.11.1] へのポインタ

戻り値

インターロック カウンタの値 (`target`) を返します。

A.5.13.12 kp_interlocked_set()

目的

- Kernel PlugIn インターロック カウンタを指定した値に設定します。

プロトタイプ

```
void kp_interlocked_set(
    KP_INTERLOCKED *target,
    int val);
```

パラメータ

名前	型	入出力
➢ target	KP_INTERLOCKED*	入力 / 出力
➢ val	val	入力

説明

名前	説明
target	設定する Kernel PlugIn インターロック カウンタ [A.5.11.1] へのポインタ
val	インターロック カウンタ (`target`) に設定する値

戻り値

なし。

A.5.13.13 kp_interlocked_exchange()

目的

- Kernel PlugIn インターロック カウンタを指定した値に設定し、変更前の値を返します。

プロトタイプ

```
int kp_interlocked_exchange(
    KP_INTERLOCKED *target,
    int val);
```

パラメータ

名前	型	入出力
➢ target	KP_INTERLOCKED*	入力 / 出力
➢ val	val	入力

説明

名前	説明
target	変更する Kernel PlugIn インターロック カウンタ [A.5.11.1] へのポインタ
val	インターロック カウンタ (`target`) に設定する値

戻り値

インターロック カウンタ (`target`) の変更前の値を返します。

A.6 Kernel PlugIn ー 構造体リファレンス

ここでは、Kernel PlugIn のさまざまな構造体に関する詳細情報を説明します。ユーザーモード関数では **`WD_XXX`** 構造体を使用し、カーネルモード関数では **`KP_XXX`** 構造体を使用します。

Kernel PlugIn の同期の型については、セクション A.5.11.1 を参照してください。

A.6.1 WD_KERNEL_PLUGIN

Kernel PlugIn open コマンドを定義します。

この関数は、低レベルの `WD_KernelPlugInOpen()` および `WD_KernelPlugInClose()` 関数で使用します。

名前	型	説明
hKernelPlugIn	DWORD	Kernel PlugIn へのハンドル
pcDriverName	PCHAR	Kernel PlugIn ドライバの名前。12 文字以下でなければなりません。VXD、SYS 拡張子は含みません。
pcDriverPath	PCHAR	この項目には、NULL を設定してください。WinDriver は OS のドライバ / モジュール ディレクトリのドライバを検索します。
pOpenData	PVOID	Kernel PlugIn の `KP_Open()` [A.5.2] コールバックに渡されるデータ。

A.6.2 WD_INTERRUPT

割り込み情報の構造体です。

この構造体は、低レベルの InterruptEnable()、InterruptDisable()、WD_IntEnable()、WD_IntDisable()、WD_IntWait()、および WD_IntCount() 関数で使用します。WDC_IntEnable() [A.2.43] は、WD_IntEnable(), WD_IntWait() および WD_IntCount() を呼び出す InterruptEnable() を呼び出します。WDC_IntDisable() [A.2.44] は、WD_IntDisable() を呼び出す InterruptDisable() を呼び出します。

名前	型	説明
kpCall	WD_KERNEL_PLUGIN_CALL	Kernel PlugIn のメッセージ情報の構造体 [A.6.3] です。この構造体は Kernel PlugIn へのハンドルの他に、カーネルモードの割り込みハンドラへ渡す情報も保持しています。Kernel PlugIn ハンドルが 0 の場合、Kernel PlugIn 割り込みハンドラなしで割り込みをインストールします。有効な Kernel PlugIn ハンドラが設定されると、この構造体は引数として KP_IntEnable() [A.5.6] Kernel PlugIn コールバック関数へ渡されます。

その他の WD_INTERRUPT のメンバに関する詳細は、InterruptEnable() の説明を参照してください。

A.6.3 WD_KERNEL_PLUGIN_CALL

Kernel PlugIn のメッセージ情報の構造体です。この構造体には、ユーザーモードの処理と Kernel PlugIn の間で引き渡される情報が含まれ、Kernel PlugIn へメッセージを受け渡したり、Kernel PlugIn の割り込みをインストールする際に使用されます。

この構造体はパラメータとして Kernel PlugIn の KP_Call() [A.5.4] および KP_IntEnable() [A.5.6] コールバック関数へ引き渡され、低レベルの WD_KernelPlugInCall()、InterruptEnable()、WD_IntEnable() 関数により使用されます。WD_KernelPlugInCall() は、高レベルの WDC_CallKerPlug() 関数 [A.2.17] から呼び出されます。(WD_IntEnable() を呼び出す) InterruptEnable() は、高レベルの WDC_IntEnable() 関数 [A.2.43] から呼び出されます。

名前	型	説明
hKernelPlugIn	DWORD	Kernel PlugIn を使用してデバイスを開く際に WDC_xxxDeviceOpen() 関数から呼び出される WD_KernelPlugInOpen() によって返される Kernel PlugIn へのハンドル
dwMessage	DWORD	Kernel PlugIn へ渡すメッセージ ID。
pData	PVOID	Kernel PlugIn へ渡すデータへのポインタ。
dwResult	DWORD	ユーザーモードへ返す Kernel PlugIn がセットした値。

A.6.4 KP_INIT

この構造体は Kernel PlugIn の KP_Init() 関数 [A.5.1] で使用されます。この構造体は主に アプリケーションが WD_KernelPlugInOpen() を呼び出す際にドライバ名と、どのカーネルモード関数を呼び出すかを WinDriver に通知する際に使用します。
WD_KernelPlugInOpen() は、これらの関数が pcKPDriverName パラメータで設定される有効な Kernel PlugIn ドライバで呼び出された際に、高レベルの WDC_xxxDeviceOpen() 関数 (PCI [A.2.9] / PCMCIA [A.2.10] / ISA [A.2.11]) から呼び出されます。

名前	型	説明
dwVerWD	DWORD	WinDriver の Kernel PlugIn ライブラリのバージョン。
cDriverName	CHAR[12]	デバイスドライバの名前。最大 12 文字です。
funcOpen	KP_FUNC_OPEN	ユーザーモードから `WD_KernelPlugInOpen()` を呼び出す際に、WinDriver が呼び出す `KP_Open()` [A.5.2] カーネルモード関数。`WD_KernelPlugInOpen()` は、これらの関数が `pcKPDriverName` パラメータで設定される有効な Kernel PlugIn ドライバで呼び出された際に、高レベルの `WDC_xxxDeviceOpen()` 関数 (PCI [A.2.9] / PCMCIA [A.2.10] / ISA [A.2.11]) から呼び出されます。

A.6.5 KP_OPEN_CALL

Kernel PlugIn がコールバック関数名 (`KP_Open()` 以外) を定義する構造体です。構造体でコールバックを設定する `KP_Open()` [A.5.2] Kernel PlugIn 関数から使用されます。

Kernel PlugIn は、`KP_Open()` [A.5.2] 以外のコールバック関数を実装可能です:

funcClose – ユーザーモードの処理がドライバのインスタンスを終了したときに呼び出されます。

funcCall –ユーザーモードの処理が、低レベルの `WD_KernelPlugInCall()` 関数を呼び出す際に呼び出されます。低レベルの `WD_KernelPlugInCall()` 関数は、`DC_CallKerPlug()` [A.2.17]、または `WDC_CallKerPlug()` から呼び出されます。これは一般的な関数で、カーネル モードで実行される任意の関数 (特別な場合である割り込み処理を除く) を実装するのに使用できます。`funcCall` コールバックは、ユーザーモードから渡されたメッセージを基に、実行する関数を決定します。

funcIntEnable – ユーザーモード処理が Kernel PlugIn ハンドルを持つ `WD_IntEnable()`を呼び出す際に呼び出されます。`WD_IntEnable()` は、高レベルの `WDC_IntEnable()` 関数 [A.2.43] から呼び出される `InterruptEnable()` により呼び出されます。`fUseKP = TRUE` で `WDC_IntEnable()` を呼び出した場合、この関数は Kernel PlugIn ハンドルで `InterruptEnable()` を呼び出します。
このコールバック関数は、割り込みを有効にする際に必要な初期化を実行します。

funcIntDisable –割り込みクリーンアップ関数です。Kernel PlugIn ドライバを使用して割り込みを有効にした後、ユーザーモードの処理が `WD_IntDisable()` を呼び出す際に呼び出されます。`WD_IntDisable()` は、`WDC_IntDisable()` [A.2.44] により呼び出される `InterruptDisable()` から呼び出されます。

funcIntAtIrql – 優先度の高いカーネルモードのレガシーな割り込み処理です。このコールバック関数は、WinDriver がこの Kernel PlugIn に割り当てられた割り込みを処理する際に、高い割り込み要求レベルで呼び出されます。この関数の戻り値が 0 より大きい場合、`funcIntAtDpc()` コールバックが DPC (Deferred Procedure Call) として呼び出されます。

funcIntAtDpc – レガシーな割り込みハンドラのコードの大部分はこのコールバックに記述します。`funcIntAtIrql()` が 0 より大きい値を返したときに DPC (Deferred Procedure Call) として呼び出されます。

funcIntAtIrqlMSI – 優先度の高いカーネルモードの PCI MSI (Message-Signaled Interrupts) と MSI-X (Extended Message-Signaled Interrupt) 処理です。このコールバック関数は、WinDriver がこの Kernel PlugIn に割り当てられた MSI / MSI-X を処理する際に、高い割り込み要求レベルで呼び出されます。この関数の戻り値が 0 より大きい場合、`funcIntAtDpcMSI()` コールバックが DPC (Deferred Procedure Call) として呼び出されます。
注意: Linux、Windows Vista およびそれ以降で MSI / MSI-X をサポートしています。

funcIntAtDpcMSI – カーネルモードの PCI MSI (Message-Signaled Interrupts) と MSI-X (Extended Message-Signaled Interrupt) ハンドラのコードの大部分はこのコールバックに記述します。

`funcIntAtIrqlMSI()` が 0 より大きい値を返したときに DPC (Deferred Procedure Call) として呼び出されます。
注意: Linux、Windows Vista およびそれ以降で MSI / MSI-X をサポートしています。

funcEvent – ユーザーモードの処理が最初に、`fUseKP = TRUE` で `WDC_EventRegister()` [A.2.46] を呼び出した場合 (または、Kernel PlugIn ハンドルで低レベルの `EventRegister()` 関数をを呼び出した場合)、Plug-and-Play またはパワー マネージメント イベントが発生した際に呼び出されます。このコールバック関数は、プラグ アンド プレイおよびパワー マネージメント イベントをカーネルが処理するように実装します。

名前	型	説明
funcClose	KP_FUNC_CLOSE	カーネル内の `KP_Close()` [A.5.3] 関数名。
funcCall	KP_FUNC_CALL	カーネル内の `KP_Call()` [A.5.4] 関数名。
funcIntEnable	KP_FUNC_INT_ENABLE	カーネル内の `KP_IntEnable()` [A.5.6] 関数名。
funcIntDisable	KP_FUNC_INT_DISABLE	カーネル内の `KP_IntDisable()` [A.5.7] 関数名。
funcIntAtIrql	KP_FUNC_INT_AT_IRQL	カーネル内の `KP_IntAtIrql()` [A.5.8] 関数名。
funcIntAtDpc	KP_FUNC_INT_AT_DPC	カーネル内の `KP_IntAtDpc()` [A.5.9] 関数名。
funcIntAtIrqlMSI	KP_FUNC_INT_AT_IRQL_MSI	カーネル内の `KP_IntAtIrqlMSI()` 関数名。 注意: Linux、Windows Vista およびそれ以降で MSI / MSI-X をサポートしています。
funcIntAtDpcMSI	KP_FUNC_INT_AT_DPC_MSI	カーネル内の `KP_IntAtDpcMSI()` 関数名。 注意: Linux、Windows Vista およびそれ以降で MSI / MSI-X をサポートしています。
funcEvent	KP_FUNC_EVENT	カーネル内の `KP_Event()` [A.5.5] 関数名。

A.7 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.7.1 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.8 を参照してください。

A.7.2 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.7.3 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	Typedef void (*HANDLER_FUNC)(void *pData);	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 (関数のプロトタイプ – HANDLER_FUNC – は **utils.h** で定義されています)。
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

正常終了した場合、WD_STATUS_SUCCESS (0) を返します。そうでない場合、エラーコードを返します [A.8]。

A.7.4 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	終了するのを待っているスレッドへのハンドル。

戻り値

なし。

A.7.5 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

正常終了した場合、WD_STATUS_SUCCESS (0) を返します。そうでない場合、エラーコードを返します [A.8]。

A.7.6 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.7.7 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)。 タイムアウトを INFINITE に設定すると、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。そうでない場合、エラーコードを返します [A.8]。

A.7.8 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

正常終了した場合、WD_STATUS_SUCCESS (0) を返します。そうでない場合、エラーコードを返します [A.8]。

A.7.9 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	Input

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

正常終了した場合、WD_STATUS_SUCCESS (0) を返します。そうでない場合、エラーコードを返します [A.8]。

A.7.10 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

正常終了した場合、WD_STATUS_SUCCESS (0) を返します。そうでない場合、エラーコードを返します [A.8]。

A.7.11 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.7.12 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

正常終了した場合、WD_STATUS_SUCCESS (0) を返します。そうでない場合、エラーコードを返します [A.8]。

A.7.13 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

正常終了した場合、WD_STATUS_SUCCESS (0) を返します。そうでない場合、エラーコードを返します [A.8]。

A.7.14 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) のセクションを指定します。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.7.15 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、その内容は破棄されます。 **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

正常終了した場合、WD_STATUS_SUCCESS (0) を返します。そうでない場合、エラーコードを返します [A.8]。

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] を呼び出して、ログ ファイルへ出力内容を追加することもできます。

A.7.16 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.7.17 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

正常終了した場合、WD_STATUS_SUCCESS (0) を返します。そうでない場合、エラーコードを返します [A.8]。

A.8 WinDriver ステータス コード

A.8.1 はじめに

多くの WinDriver API 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()` 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.8.2 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_KERPLUG_FAILURE	Kernel PlugIn failure (KernelPlugIn の失敗)
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_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)
WD_OPERATION_FAILED	Operation failed (操作が失敗しました)
WD_INVALID_32BIT_APP	Received an invalid 32-bit IOCTL (不正な 32 ビット IOCTL を受信しました)
WD_TOO_MANY_HANDLES	No room to add handle (ハンドルを追加する場所がありません)

WD_NO_DEVICE_OBJECT	Driver not installed (ドライバがインストールされていません)

付録 B
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 C
評価版 (Evaluation Version) の制限

C.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。

C.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 / Server 2008 / Vista / 7 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。

C.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 (モジュールのアンロードおよびロード) します。

  注意: root 権限で以下のコマンドを実行する必要があります。

アンロードするには:
/sbin/modprobe –r windrvr6

ロードするには:
<path to wdreg>/wdreg windrvr6

wdreg は **WinDriver/util/** ディレクトリ以下にあります。

付録 D
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 - 0073
東京都港区三田 3-9-9 森伝ビル 6F
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 E
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.pdf** ファイルを参照してください。

付録 F
その他のドキュメント

最新マニュアル

最新版の WinDriver 日本語ユーザーズ ガイドは、下記 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

2009 年 12 月 10 日

発行　**エクセルソフト株式会社**

〒108-0073 東京都港区三田3-9-9 森伝ビル6F

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