INTERFACE CO., LTD.

ITF MIDDLEWARE SERIES

# ITF-EasyHost

初版　　2005 年　7 月　11 日
第十版　　2013 年　2 月　16 日

# User's Manual

インターフェイス株式会社

ご注意

1．本製品及び本書はインターフェイス株式会社の著作物です。
したがって、本製品及び本書の一部または全部を無断で複製、複写、転載、改変することは法律で禁じられています。

2．本製品及び本書の内容については、改良のために予告なく変更することがございます。

3．本製品を運用した結果の他への影響については、責任を負いかねますのでご了承ください。

4．本製品は、外国為替及び外国貿易法の規定により戦略物資等輸出規制品に該当する場合があります。
国外に持ち出す場合には、日本国政府の輸出許可申請などの手続きが必要となる場合があります。

5．本製品は、医療機器、航空宇宙機器など人命にかかわる設備や機器での使用は考慮しておりません。
これらの設備や機器に本製品を使用され、本製品の故障により、人身事故、火災などが発生しても、弊社では責任を負いかねます。

6．本製品は国内仕様です。本製品を国外で使用された場合の不具合などについては対応できませんのであらかじめご了承ください。

【目次】

# 1 はじめに

このたびは ITF ミドルウェアシリーズ ITF-EasyHost をお買い上げいただきまことにありがとうございます。

本製品は、USB1.1 ホストドライバの ITF-EasyHost I、USB2.0 ホストドライバの ITF-EasyHost II、オプションのマウスクラスドライバ、キーボードクラスドライバ、マスストレージクラスドライバから構成されます。本マニュアルでは特に製品構成を区別せずに記載されています。

本ミドルウェアにより、USB1.1 または USB2.0 仕様に準拠したホスト側デバイスの開発において下記のような利点があります。

また、HUB クラスを持たない IC を使うことで、小規模なミドルウェアを実現しております。

- 本製品は、USB ホストコントローラを使用したシステム開発において、USB ホストコントローラの使用方法などを意識することなく、USB デバイスと通信が行えるためのファームウェアをご提供いたします。
- オプションのクラスドライバを組み込むことにより、より幅広いデバイスに対応が可能となります。
- データ転送は Full-Speed（最大 12Mbps）に対応、ITF-EasyHostⅡは High-Speed（最大 480Mbps）にも対応。スムーズなデータ転送を実現できます。
- ITF-EasyHost I は、Low-Speed（最大 1.5Mbps）デバイスにも対応。キーボード、マウスの動作もイベントにより動作可能です。（ご使用になるコントローラにより制限される場合があります。ITF-EasyHostⅡは Low-Speed に対応しておりません。）
- USB の各転送方式（コントロール転送、バルク転送、インタラプト転送）に対応。幅広いデバイスの転送方式で転送が可能です。（ご使用になるコントローラにより制限される場合があります。）

USB についての詳細は、参考書、インターネット等をご参照ください。
ご使用になる前に、本マニュアルの内容をよくご理解いただき有効にご利用下さい。

## 2 本書について

本マニュアルは、ITF-EasyHost の概要および取り扱いについて述べたものです。
本書は以下の章から構成されています。

1 はじめに
2 本書について
3 製品内容
4 ソフトウェア仕様
5 機能
6 API 仕様
7 動作
8 注意･制限事項
9 一覧

なお，本書の内容は予告なしに変更することがあります。本書の内容について，ご不明な点がありましたら当社へお問合せください。

# 3 製品内容

## 3.1 ディレクトリ構成

本ライブラリは、次のディレクトリ構成となっています。

| | |
|---|---|
| +【DOC】 | 各種マニュアル |
| +【SRC】 | ソース |
| +--【IITFSingleHost】 | ITF-EasyHost　ルート |
| +--【Include】 | インクルード |
| +--【Library】 | ライブラリ格納フォルダ |
| +--【Object】 | オブジェクト格納フォルダ |
| +--【Sample】 | サンプル |
| +--【Application】 | アプリケーション |
| +--【Object】 | オブジェクト格納フォルダ |
| +--【Driver】 | ドライバ |
| +--【MB90330】 | MB90330/MB91660/MB9BF506 用ドライバ |
| +--【M66596】 | M66596 用ドライバ |
| +--【Source】 | ソース |
| +--【Depend】 | ハード・システム依存部 |
| +--【Include】 | インクルード(ローカル) |
| +--【Io】 | API |
| +--【ITFLib】 | ITF 共通ライブラリ　ルート |
| +--【Include】 | 共通インクルード |
| : (以下省略) | |
| : | |

※　【ITFLib】については、「ITF 共通ライブラリ User's Manual」を参照して下さい。

### 3.2 ファイル構成

| ファイル名 | 内容 | 場所 |
|---|---|---|
| **ITF-EasyHost コンフィグレーションファイル** | | |
| singlehostconfig.h | ITF-EasyHost コンフィグレーションの宣言 | ¥ITFSingleHost¥Include |
| **USBHost ドライバ** | | |
| singlehost.h | ITF-EasyHost ヘッダファイル | ¥ITFSingleHost¥Include |
| singlehostapi.h | ITF-EasyHost API 宣言 | 〃 |
| singlehostcode.h | ITF-EasyHost イベントコード | 〃 |
| singlehosterror.h | ITF-EasyHost エラーコード | 〃 |
| singlehostresource.h | ITF-EasyHost リソースの宣言 | |
| singlehosthidapi.h | ITF-EasyHost HID API 宣言 | 〃 |
| singlehostscdapi.h | ITF-EasyHost ストレージクラス API 宣言 | 〃 |
| device.c | デバイス管理モジュール | ¥ITFSingleHost¥Source |
| interface.c | インターフェイス管理モジュール | 〃 |
| usb.c | バスドライバモジュール | 〃 |
| usb_event.c | イベント管理モジュール | 〃 |
| usb_pool.c | メモリ管理プール | 〃 |
| usb_timer.c | タイマーモジュール | 〃 |
| usbkbd.c | キーボードクラス | 〃 |
| usbmouse.c | マウスクラス | 〃 |
| usbscd.c | ストレージクラス | 〃 |
| usbscdmgr.c | ストレージクラス リソース管理モジュール | 〃 |
| device.h | デバイス管理モジュール | ¥ITFSingleHost¥Source ¥Include |
| hcd_if.h | コントローラドライバインターフェース | 〃 |
| interface.h | インーフェイス管理モジュール | 〃 |
| usb.h | バスドライバモジュール | 〃 |
| usb_event.h | イベント管理モジュール | 〃 |
| usb_pool.h | メモリ管理プール | 〃 |
| usb_timer.h | タイマーモジュール | 〃 |
| usbkbd.h | キーボードクラス | 〃 |
| usbmouse.h | マウスクラス | 〃 |
| usbscd.h | ストレージクラス | 〃 |
| usbscdmgr.h | ストレージクラス リソース管理モジュール | 〃 |
| singlehostapi.c | USB Host API | ¥ITFSingleHost¥Source ¥Io |
| singlehostkbdapi.c | キーボードクラス API | 〃 |
| singlehostscdapi.c | ストレージクラス API | 〃 |

| ハード・システム依存部(MB90330/MB91660/MB9BF506) (※1) | | | |
|---|---|---|---|
| dma.c | DMA 転送モジュール | ¥ITFSingleHost¥Sample¥Driver¥MB90330 | |
| dma.h | DMA 転送モジュールヘッダ | 〃 | |
| hcd.h | ドライバ内部定義 | 〃 | |
| hcd_bus.c | ドライババス検出/操作モジュール | 〃 | |
| hcd_bus.h | ドライババス検出/操作モジュール | 〃 | |
| hcd_config.h | ドライバユーザ定義ヘッダ | 〃 | |
| hcd_endpoint.c | ドライバエンドポイント管理モジュール | 〃 | |
| hcd_endpoint.h | ドライバエンドポイント管理モジュールヘッダ | 〃 | |
| hcd_event.c | ドライバイベントモジュール | 〃 | |
| hcd_event.h | ドライバイベントモジュールヘッダ | 〃 | |
| hcd_hw.c | ハードウェア依存処理モジュール | 〃 | |
| hcd_hw.h | ハードウェア依存処理モジュールヘッダ | 〃 | |
| hcd_if.c | ドライバインターフェース | 〃 | |
| hcd_init.c | ドライバコントローラ初期化/終了 | 〃 | |
| hcd_init.h | ドライバコントローラ初期化/終了ヘッダ | 〃 | |
| hcd_interrupt.c | ドライバ割込ハンドラ | 〃 | |
| hcd_interrupt.h | ドライバ割込ハンドラヘッダ | 〃 | |
| hcd_pool.c | ドライバ構造体領域確保/開放 | 〃 | |
| hcd_pool.h | ドライバ構造体領域確保/開放ヘッダ | 〃 | |
| hcd_reg.h | ドライバレジスタ定義 | 〃 | |
| hcd_transaction.c | ドライバトランザクション管理 | 〃 | |
| hcd_transaction.h | ドライバトランザクション管理ヘッダ | 〃 | |

| サンプルアプリケーション(MB90330/MB91660/MB9BF506) (※1) | | | |
|---|---|---|---|
| ITFFile_MB90330.dat | Softune 用 ITF-FILE プロジェクトファイル | ¥ITFFile | (※2) |
| ITFFile_MB90330.prj | Softune 用 ITF-FILE プロジェクトファイル | 〃 | (※2) |
| ITFFile_MB91660.dat | Softune 用 ITF-FILE プロジェクトファイル | 〃 | (※2) |
| ITFFile_MB91660.prj | Softune 用 ITF-FILE プロジェクトファイル | 〃 | (※2) |
| ITFFile_MB9BF506.ewd | IAR Embedded Workbench 用 ITF-FILE プロジェクトファイル | 〃 | (※2) |
| ITFFile_MB9BF506.ewp | IAR Embedded Workbench 用 ITF-FILE プロジェクトファイル | 〃 | (※2) |
| ITFLib_MB90330.dat | Softune 用 ITF-Lib プロジェクトファイル | ¥ITFLib | (※3) |
| ITFLib_MB90330.prj | Softune 用 ITF-Lib プロジェクトファイル | 〃 | (※3) |
| ITFLib_MB91660.dat | Softune 用 ITF-Lib プロジェクトファイル | 〃 | (※3) |
| ITFLib_MB91660.prj | Softune 用 ITF-Lib プロジェクトファイル | 〃 | (※3) |
| ITFLib_MB9BF506.ewd | IAR Embedded Workbench 用 ITF-Lib プロジェクトファイル | 〃 | (※4) |
| ITFLib_MB9BF506.ewp | IAR Embedded Workbench 用 ITF- Lib プロジェクトファイル | 〃 | (※4) |

| | | |
|---|---|---|
| ITFSingleHost_MB90330.dat | Softune 用<br>ITF-EasyHost プロジェクトファイル | ¥ITFSingleHost |
| ITFSingleHost_MB90330.prj | Softune 用<br>ITF-EasyHost プロジェクトファイル | 〃 |
| ITFSingleHost_MB91660.dat | Softune 用<br>ITF-EasyHost プロジェクトファイル | 〃 |
| ITFSingleHost_MB91660.prj | Softune 用<br>ITF-EasyHost プロジェクトファイル | 〃 |
| ITFSingleHost_MB9BF506.ewd | IAR Embedded Workbench 用<br>ITF-EasyHost プロジェクトファイル | 〃 |
| ITFSingleHost_MB9BF506.ewp | IAR Embedded Workbench 用<br>ITF-EasyHost プロジェクトファイル | 〃 |
| MB90330.dat | サンプルプロジェクトファイル | ¥ITFSingleHost¥Sample¥Application |
| MB90330.prj | サンプルプロジェクトファイル | 〃 |
| MB90330.wsp | サンプルアプリケーションワークスペース | 〃 |
| MB91660.dat | サンプルプロジェクトファイル | 〃 |
| MB91660.prj | サンプルプロジェクトファイル | 〃 |
| MB91660.wsp | サンプルアプリケーションワークスペース | 〃 |
| MB9BF506.ewd | サンプルプロジェクトファイル | 〃 |
| MB9BF506.ewp | サンプルプロジェクトファイル | 〃 |
| MB9BF506.eww | サンプルアプリケーションワークスペース | 〃 |
| MB9BF506_Flash.icf | セクション定義ファイル | 〃 |
| MB9BF506_Flash.mac | デバッガ用マクロファイル | 〃 |
| sample.c | サンプルアプリケーションソース | 〃 |
| MB90330_serial.c | CPU 内蔵シリアルドライバソース | 〃 |
| MB91660_serial.c | CPU 内蔵シリアルドライバソース | 〃 |
| MB9BF506_serial.c | CPU 内蔵シリアルドライバソース | 〃 |
| serial.h | CPU 内蔵シリアルドライバヘッダ | 〃 |
| MB90330_startup.asm | スタートアップアセンブラ | 〃 |
| MB91660_startup.asm | スタートアップアセンブラ | 〃 |
| MB9BF506_startup.s | スタートアップアセンブラ | 〃 |
| MB90330_timer.c | CPU 内蔵タイマードライバソース | 〃 |
| MB91660_timer.c | CPU 内蔵タイマードライバソース | 〃 |
| MB9BF506_timer.c | CPU 内蔵タイマードライバソース | 〃 |
| timer.h | CPU 内蔵タイマードライバヘッダ | 〃 |

| ハード・システム依存部(M66596) (※1) | | |
|---|---|---|
| dma.c | M66596/MS7751R DMA 転送モジュール | ¥ITFSingleHost¥Sample¥Driver¥M66596 |
| dma.h | M66596/MS7751R DMA 転送モジュールヘッダ | 〃 |
| hcd.h | ドライバ内部定義 | 〃 |
| hcd_bus.c | ドライババス検出/操作モジュール | 〃 |
| hcd_bus.h | ドライババス検出/操作モジュール | 〃 |
| hcd_config.h | ドライバユーザ定義ヘッダ | 〃 |
| hcd_endpoint.c | ドライバエンドポイント管理モジュール | 〃 |
| hcd_endpoint.h | ドライバエンドポイント管理モジュールヘッダ | 〃 |
| hcd_event.c | ドライバイベントモジュール | 〃 |
| hcd_event.h | ドライバイベントモジュールヘッダ | 〃 |
| hcd_hw.c | ハードウェア依存処理モジュール | 〃 |
| hcd_hw.h | ハードウェア依存処理モジュールヘッダ | 〃 |
| hcd_if.c | ドライバインターフェース | 〃 |
| hcd_init.c | ドライバコントローラ初期化/終了 | 〃 |
| hcd_init.h | ドライバコントローラ初期化/終了ヘッダ | 〃 |
| hcd_interrupt.c | ドライバ割込ハンドラ | 〃 |
| hcd_interrupt.h | ドライバ割込ハンドラヘッダ | 〃 |
| hcd_pool.c | ドライバ構造体領域確保/開放 | 〃 |
| hcd_pool.h | ドライバ構造体領域確保/開放ヘッダ | 〃 |
| hcd_reg.h | ドライバレジスタ定義 | 〃 |
| hcd_wait.h | ウェイト定義ヘッダ | 〃 |

| サンプルアプリケーション(M66596) (※1) | | |
|---|---|---|
| ITFFile_sh7751.hwp | HEW 用 ITF-FILE プロジェクトファイル | ¥ITFFile (※2) |
| ITFSingleHost_SH7751R_M66596.hwp | HEW 用 ITF-EasyHost プロジェクトファイル | ¥ITFSingleHost |
| SH7751R_ST.hwp | サンプルプロジェクトファイル | ¥ITFSingleHost¥Sample¥Application |
| SH7751R_ST.hws | サンプルアプリケーションワークスペース | 〃 |
| Sample.c | サンプルアプリケーションソース | 〃 |
| SH7751R_serial.c | CPU 内蔵シリアルドライバソース | 〃 |
| serial.h | CPU 内蔵シリアルドライバヘッダ | 〃 |
| SH7751R_timer.c | CPU 内蔵タイマードライバソース | 〃 |
| timer.h | CPU 内蔵タイマードライバヘッダ | 〃 |
| iodefine.h | I/O レジスタ定義 | 〃 |
| vect.inc | ベクタ定義 | 〃 |
| intprg.src | 割込プログラム | 〃 |
| vecttbl.src | ベクタテーブル | 〃 |
| vhandler.src | リセット/割込ハンドラ | 〃 |

(※1) ハード・システム依存部及びサンプルアプリケーションは、ご購入時にご指定いただいた構成のみ付属します。
(※2) ITF-FILE はオプションとなります。ITF-FILE のプロジェクトファイルは、ITF-FILE に付属します。
(※3) ITF-Lib は ITF-EasyHost 及び ITF-FILE のそれぞれに付属しますが、Softune 用 ITF-Lib プロジェクトファイルは ITF-EasyHost 付属の ITF-Lib のみに含まれます。
(※4) ITF-Lib は ITF-EasyHost 及び ITF-FILE のそれぞれに付属しますが、IAR Embedded Workbench 用 ITF-Lib プロジェクトファイルは ITF-EasyHost 付属の ITF-Lib のみに含まれます。

### 3.3 動作環境

本ライブラリの動作環境は次の通りです。

| 仕様 | 説明 |
|---|---|
| ＯＳ | 非 OS 環境 |
| | NORTi Ver4 (SH7751+M66596 のみ) |
| 対応ＣＰＵ | MB90F334(内蔵 USB Host コントローラ) |
| | MB91F662(内蔵 USB Host コントローラ) |
| | MB9BF506(内蔵 USB Host コントローラ) |
| | SH7751(M66596) |
| 対応ホストコントローラ | MB90F334　　USB1.1 準拠 |
| | MB91F662　　USB1.1 準拠 |
| | MB9BF506　　USB1.1 準拠 |
| | M66596　　USB2.0 準拠 |
| ＲＯＭ容量(※1)(※2) | 標準 50K バイト(MB90F334) |
| | 標準 45K バイト(MB91F662) |
| | 標準 19K バイト(MB9BF506) |
| | 標準 40K バイト(SH7751(M66596)) |
| ＲＡＭ容量(※1)(※2) | 標準 8K バイト(MB90F334) |
| | 標準 8K バイト(MB91F662) |
| | 標準 8K バイト(MB9BF506) |
| | 標準 8K バイト(SH7751(M66596)) |
| 対応ＵＳＢクラス | ＨＩＤクラス（キーボード） |
| | ＨＩＤクラス（マウス） |
| | マスストレージクラス |

(※1)　本書に記載の ROM/RAM 容量は、標準の構成での値となります。実際の ROM/RAM 使用量は、ユーザ設定により増減します。

(※2)　本書に記載の ROM/RAM 容量には、OS・ファイルシステム(ITF-FILE)・その他のライブラリ・ユーザアプリケーションの容量を含みません。

### 3.4 導入

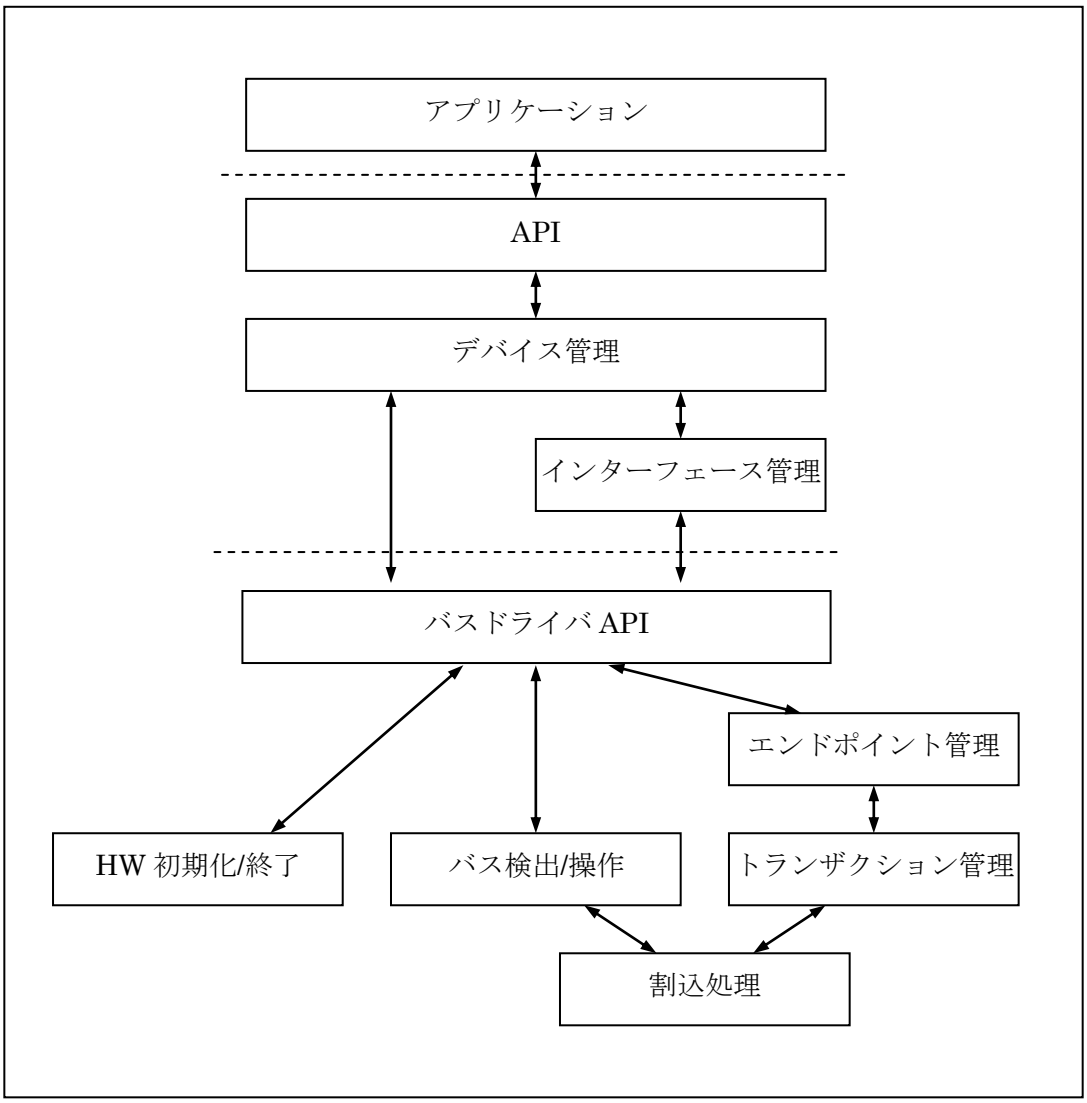
本ライブラリの導入方法については、「ITF-EasyHost Porting Manual」を参照して下さい。

# 4 ソフトウェア仕様

## 4.1 ブロック構成

ITF-EasyHost は機能により複数のブロックに分かれています。そのブロックが相互アクセスすることにより全ての機能が実現されています。

次は、システムに組み込んだ ITF-EasyHost の全体のブロック図です。




（図　ITF-EasyHost の全体ブロック構成）

### 4.2 API

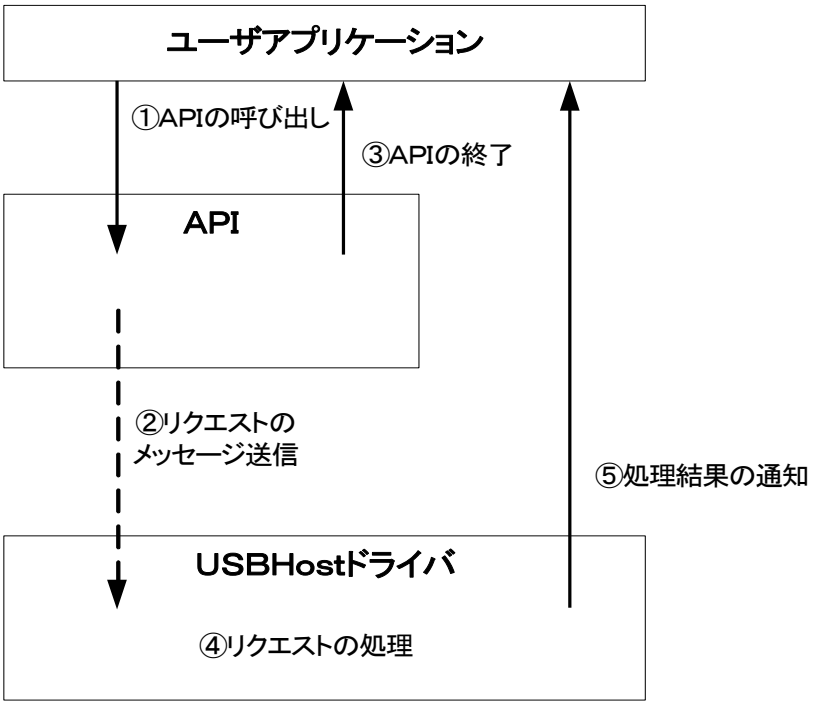
API は、ITF-EasyHost がユーザアプリケーションに公開するインターフェイスとなります。ユーザアプリケーションは API により、ITF-EasyHost の機能を使用することが可能となります。

API は即時に終了し、処理の結果は後から報告される非同期式で実現してあります。

#### 4.2.1 非同期式 API

ITF-EasyHost はユーザアプリケーションに非同期の API を提供します。ユーザは非同期式 API を使用することにより、アプリケーション設計の幅が広がります。非同期式 API は主に転送用の API をサポートします。

ユーザアプリケーションで非同期式 API を使用した時の処理の流れは次のようになります。




（図　非同期式 API 内部の流れ）

①API の呼び出し
　ユーザアプリケーションは ITF-EasyHost が提供する API を呼び出します。

②リクエストのメッセージ送信
　API は API の情報を格納したリクエストパケット（メモリブロック）を生成します。生成したリクエストパケットを USBHost ドライバへ通知します。

③API の終了
　API は USBHost ドライバへ通知し、API で設定した情報を解析します。解析した情報に従って処理を行い、終了します。

④処理結果の通知
　USBHost ドライバは、リクエストの処理が終了したら処理結果をユーザアプリケーションへ報告します。処理結果の報告は、コールバックルーチンを呼び出すことにより行われます。

#### 4.2.2 API 使用上の注意点

1．API は割り込みハンドラで呼び出されることを前提としておりませんので、割り込みハンドラでは使用しないで下さい。(但し、Usb_Host_Notify_1ms_Timer のみ割り込みハンドラで呼び出す必要があります。)

2．クラスの登録時にイベントのコールバックルーチンを登録します。また、対応しないクラスのイベントのコールバックルーチンは singlehostconfig.h で設定します。処理が終了したときに、対応するクラスの場合は登録したコールバックルーチンが、対応しない場合はデフォルトのコールバックルーチンが呼び出されます。コールバックルーチン内の処理は最低限の処理だけを行い即時に返る必要があります。

### 4.3 クラスドライバ

USB デバイスは機能によりクラスに分類されます。代表的はクラスとしては、ヒューマンインターフェイスデバイス（ＨＩＤ）クラス、マスストレージデバイスクラス等があります。それ以外に、ベンダ独自のクラスも存在します。

ITF-EasyHost にはＨＩＤキーボードクラス、ＨＩＤマウスクラス、マスストレージクラスが存在します。接続された USB デバイスの情報によって、これらのクラスドライバの中から適切なクラスドライバが選択されます。

#### 4.3.1 キーボードクラスドライバ

USB のクラスとしてＨＩＤクラスが規格されています。また、ＨＩＤクラスのサブクラスとしてキーボードが規定されています。キーボードクラスドライバはこの規格に沿った USB デバイスのためのドライバです。

キーボードクラスドライバは、キー入力（キーアップ、キーダウン）の通知、ＬＥＤの点灯／点滅のためのインターフェイスをユーザアプリケーションに提供します。

#### 4.3.2 マウスクラスドライバ

キーボードクラスドライバと同等にＨＩＤクラスにはサブクラスとしてマウスが規定されています。マウスクラスドライバはこの規格に沿った USB デバイスのためのドライバです。

マウスクラスドライバは、マウスの移動やボタン入力通知のためのインターフェイスをユーザアプリケーションに提供します。

#### 4.3.3 マスストレージクラスドライバ

USB のクラスとしてマスストレージクラスが規格されています。マスストレージクラスドライバはこの規格に沿った USB デバイスのためのドライバです。

マスストレージクラスドライバの詳細については別途参照して下さい。

### 4.4 ホストコントローラドライバ

ホストコントローラドライバは、USB デバイスと通信を行うための最下層のドライバです。ホストコントローラドライバでは、ホストコントローラを制御します。

### 4.4.1 インターフェイス

ホストコントローラドライバは次のインターフェイスを USB バスドライバに提供します。

（表　ホストコントローラのインターフェイス）

| | 機能 | メンバ名 |
|---|---|---|
| 1 | 初期化 | hc_init |
| 2 | 終了 | hc_release |
| 3 | 割込ポーリングルーチン | hc_check_interrupt |
| 4 | エンドポイントの確保 | hc_ep_alloc |
| 5 | エンドポイントの開放 | hc_ep_free |
| 6 | エンドポイントのリセット | hc_ep_reset |
| 7 | リクエストの送信 | urb_submit |
| 8 | リクエストのキャンセル | urb_cancel |
| 9 | フレーム番号取得 | get_frame_counter |
| 10 | ホストコントローラのリセット | hc_reset |
| 11 | ホストコントローラのサスペンド | hc_suspend |
| 12 | ホストコントローラのレジューム | hc_wakeup |
| 13 | メッセージ通知 | hc_message |
| 14 | タイマ供給 | hc_notify_timer |
| 15 | コントローラ割込許可 | hc_int_enable |
| 16 | コントローラ割込禁止 | hc_int_disable |
| 17 | 割込サービス関数 | 上位定義 |

1．初期化（hc_init）
ホストコントローラを初期化します。

2．終了（hc_release）
ホストコントローラを停止します。

3．割込ポーリングルーチン（hc_check_interrupt）
バスドライバから周期的に実行され, 割込の発生をチェックします。

4．エンドポイントの確保（hc_ep_alloc）
デバイスのエンドポイント情報を初期化し、エンドポイントにアクセス可能な状態にします。

5．エンドポイントの開放（hc_ep_free）
hc_ep_alloc で初期化したエンドポイント情報を開放します。

6．エンドポイントのリセット（hc_ep_reset）
エンドポイント情報をリセットします。上位で CLEAR FEATURE が送信された時に呼び出されま

す。

７．リクエストの送信（urb_submit）
　上位からの転送要求を処理します。上位からは転送要求の情報が格納された USB REQUEST BLOCK(URB)が渡されます。ホストコントローラドライバは URB の情報に従い転送を開始します。適切に処理された URB はリストに繋がれます。転送が終了したとき、またはキャンセルされたときに URB はリストから外されます。

８．リクエストのキャンセル（urb_cancel）
　urb_submit でリストに繋がれた URB の転送をキャンセルします。キャンセルされた URB はリクエストの完了が報告されます。

９．フレーム番号取得（get_frame_counter）
　フレーム番号を取得します。

１０．　バスリセット（hc_reset）
　バスをリセットします。

１１．　バスサスペンド（hc_suspend）
　バスをサスペンドさせます。

１２．　バスレジューム（hc_wakeup）
　サスペンドしたバスをレジュームします。

１３．　メッセージ通知　（hc_message）
　バスドライバ側からの各種メッセージを通知します。

１４．　タイマ供給（hc_notify_timer）
　設定した間隔でバスドライバからタイマを供給します。

１５．　コントローラ割込許可（hc_int_enable）
　コントローラの割込を許可します。

１６．　コントローラ割込禁止（hc_int_disable）
　コントローラの割込を禁止します。

１７．　割り込みサービス関数（上位定義）
　割り込み発生時に割り込みハンドラから呼び出す割り込み処理です。

これらのインターフェイスをホストコントローラドライバは用意します。これにより、USB バスドライバはホストコントローラの種類を意識することなく、どのホストコントローラも同じように制御することが可能となります。

# 5 機能

## 5.1 USB デバイス命名規則

接続された USB デバイスには全て名前が付けられます。

¥¥¥USB¥DDD¥0000
DDD　　：クラス名

※　全て半角大文字です。

現在登録済みクラス名は

KBD　　：キーボード
MOUSE　　：マウス
STOR　　：ストレージクラスデバイス

となります。
ストレージクラスデバイスを接続した場合、デバイス名は

ストレージ：**¥¥¥USB¥STOR¥0000**

となります。
キーボードとマウスの複合デバイスを接続した場合、デバイス名は

キーボード：**¥¥¥USB¥KBD¥0000**
マウス　　：**¥¥¥USB¥MOUSE¥0000**

となります。

このデバイス名はデバイスのオープン時に使用されます。API : Usb_Host_Open のパラメータにこのデバイス名を指定します。また、デバイス情報の取得でもこのデバイス名が取得できます。API : Usb_Host_Get_Device_Information で取得できるデバイス情報構造体のメンバ dev_name に、このデバイス名が設定されます。

### 5.2 イベント

ITF-EasyHost では、内部でイベントが発生した時にユーザアプリケーションにイベントを通知する機能が実装されています。イベントにはデバイスの検出やキーの検出等のイベントが用意されています。

#### 5.2.1 コールバックルーチン

イベントのユーザアプリケーションへの通知はコールバックルーチンによって行われます。ユーザアプリケーションはコールバックルーチンが呼び出されることによりイベントが発生したことを知ることができます。イベントにより処理が必要であるならばユーザアプリケーションは適切な処理を行います。

コールバックルーチンは singlehostconfig.h に設定します。 コールバックルーチンの形式は次の通りです。

## *イベントのコールバックルーチン*

**機　能**　イベント発生時のコールバックルーチン

**形　式**　**void Event_CallBack(ihandle dev, int32_t code, void *prm);**

| | |
|---|---|
| **dev** | **: デバイスハンドル** |
| **code** | **: イベント番号** |
| ***prm** | **: パラメータ** |

**解　説**　ITF-EasyHost がイベントを検出した時に呼び出します。
イベントの対象がデバイスの場合には dev にデバイスハンドルを、イベントコードを code に、イベントにより意味の異なるパラメータを prm に設定して呼び出します。
パラメータの詳細は後述します。

**戻　値**　なし

### 5.2.2 イベントの種類

ITF-EasyHost が通知するイベントには次の通りです。

（表　イベント一覧）

| イベント種別 | イベント番号(code)<br>(第２パラメータ) | デバイスハンドル(dev)<br>(第１パラメータ) | パラメータ(prm)<br>(第３パラメータ) |
|---|---|---|---|
| デバイス検出イベント | | | |
| デバイス接続 | NC_EVENT_CONNECT | 接続されたデバイスのハンドル | 未使用 |
| デバイス切断 | NC_EVENT_DISCONNECT | 切断されたデバイスのハンドル | 未使用 |
| ウェイクアップ要求 | NC_EVENT_WAKEUP_REQUEST | ウェイクアップを要求したデバイスのハンドル | 未使用 |
| リセット完了 | NC_EVENT_RESET_COMPLETE | リセット要求が完了したデバイスのハンドル | 未使用 |
| サスペンド完了 | NC_EVENT_SUSPEND_COMPLETE | サスペンド要求が完了したデバイスのハンドル | 未使用 |
| リクエスト完了 | NC_EVENT_REQUEST_COMPLETE | 非同期の要求が完了したデバイスのハンドル | io_overlapped_result_t * |
| キーボード入力イベント | | | |
| キーダウン | NC_EVENT_KEY_DOWN | キーダウンしたキーボードデバイスのハンドル | uint8_t * |
| キーアップ | NC_EVENT_KEY_UP | キーアップしたキーボードデバイスのハンドル | uint8_t * |
| コントロールキーダウン | NC_EVENT_CTL_KEY_DOWN | コントロールキーダウンしたキーボードデバイスのハンドル | uint8_t * |
| コントロールキーダウン | NC_EVENT_CTL_KEY_UP | コントロールキーアップしたキーボードデバイスのハンドル | uint8_t * |
| マウス入力イベント | | | |
| ボタンダウン | NC_EVENT_BTN_DOWN | ボタンダウンしたマウスデバイスのハンドル | uint8_t * |
| ボタンアップ | NC_EVENT_BTN_UP | ボタンアップしたマウスデバイスのハンドル | uint8_t * |
| Ｘ方向移動 | NC_EVENT_POS_X | Ｘ方向に移動したマウスデバイスのハンドル | int8_t * |
| Ｙ方向移動 | NC_EVENT_POS_Y | Ｙ方向に移動したマウスデバイスのハンドル | int8_t * |
| ホイール移動 | NC_EVENT_POS_WHEEL | ホイール移動したマウスデバイスのハンドル | int8_t * |
| キーボードＬＥＤイベント | | | |
| ＬＥＤ変更完了 | NC_EVENT_KEY_LED | ＬＥＤの変更が完了したキーボードデバイスのハンドル | uint8_t * |

| その他 | | | |
|---|---|---|---|
| 致命的なエラー | NC_EVENT_FATAL_ERROR | 未使用 | 未使用 |

#### 5.2.2.1 デバイス接続

デバイス接続イベントは、デバイスの接続処理が完了した時に通知されるイベントです。このイベントが通知された後にデバイスは使用可能な状態になります。

第１パラメータには接続されたデバイスのハンドルが渡されます。ユーザアプリケーションはこのハンドルを使用してデバイスの情報等の取得が可能になります。第３パラメータは使用しません。

尚、デバイスの接続処理でエラーが発生した場合には本イベントは通知されません。

#### 5.2.2.2 デバイス切断

デバイス切断イベントは、デバイスの切断処理が完了した時に通知されるイベントです。このイベントが通知された時にはデバイスが使用できない状態となっています。

第１パラメータには切断されたデバイスのハンドルが渡されます。第３パラメータは使用しません。

#### 5.2.2.3 ウェイクアップ要求

ウェイクアップ要求イベントは、サスペンド中のデバイスがウェイクアップし、バスがアイドル状態となった時に通知されるイベントです。このイベントは、API によるウェイクアップ要求およびデバイスによるリモートウェイクアップ要求により発生します。

第１パラメータにはウェイクアップしたデバイスのハンドルが渡されます。第３パラメータは使用しません。

本イベントは、接続中のすべてのデバイスハンドルに対して、登録されたコールバックにより通知され、標準のコールバックによりデバイスに無関係のイベントとしても通知されます。

#### 5.2.2.4 リセット完了

リセット完了イベントは、リセット要求が完了した時に通知されるイベントです。

第１パラメータにはリセットされたデバイスのハンドルが渡されます。第３パラメータは使用しません。

本イベントは、接続中のすべてのデバイスハンドルに対して、登録されたコールバックにより通知され、標準のコールバックによりデバイスに無関係のイベントとしても通知されます。

#### 5.2.2.5 サスペンド完了

サスペンド完了イベントは、サスペンド要求が完了した時に通知されるイベントです。
第１パラメータにはサスペンドされたデバイスのハンドルが渡されます。第３パラメータは使用しません。
本イベントは、接続中のすべてのデバイスハンドルに対して、登録されたコールバックにより通知され、標準のコールバックによりデバイスに無関係のイベントとしても通知されます。

#### 5.2.2.6 リクエスト完了

リクエスト完了イベントは､非同期の API による転送要求が完了した時に通知されるイベントです。
第１パラメータには完了したリクエストのデバイスのハンドルが渡されます。第３パラメータは io_overlapped_result_t 構造体へのポインタです。

#### 5.2.2.7 キーダウンイベント

キーダウンイベントは、キーボードデバイスでキーが押下された時に呼び出されるイベントです。
第１パラメータにはキーが押下されたデバイスのハンドルが渡されます。第３パラメータには押下されたキーコードが格納されたバッファのアドレスが符号なし８ビットデータで渡されます。
第３パラメータに渡されるキーコードは USB のデータから変換されていません。ユーザアプリケーションは適切なキーコードに変換する必要があります。
キーボードクラスを使用していない場合は、キーダウンイベントは通知されません。

#### 5.2.2.8 キーアップイベント

キーアップイベントは、キーボードデバイスでキーが離された時に呼び出されるイベントです。
第１パラメータにはキーが離されたデバイスのハンドルが渡されます。第３パラメータには離されたキーコードが格納されたバッファのアドレスが符号なし８ビットデータで渡されます。
第３パラメータに渡されるキーコードは USB のデータから変換されていません。ユーザアプリケーションは適切なキーコードに変換する必要があります。
キーボードクラスを使用していない場合は、キーアップイベントは通知されません。

#### 5.2.2.9 コントロールキーダウンイベント

コントロールキーダウンイベントは、キーボードデバイスでコントロールキーが押下された時に呼び出されるイベントです。

第１パラメータにはキーが押下されたデバイスのハンドルが渡されます。第３パラメータには押下されたキーコードが格納されたバッファアドレスが渡されます。渡されるバッファは符号なし８ビットデータです。

第３パラメータに渡されるキーコードは次の通りです。

（表　コントロールキー）

| キー種別 | キーコード | 値 |
|---|---|---|
| 左の Control キー | NCTL_KEY_LEFT_CONTROL | 0x01 |
| 左の Shift キー | NCTL_KEY_LEFT_SHIFT | 0x02 |
| 左の Alt キー | NCTL_KEY_LEFT_ALT | 0x04 |
| 左の Gui キー | NCTL_KEY_LEFT_GUI | 0x08 |
| 右の Control キー | NCTL_KEY_RIGHT_CONTROL | 0x10 |
| 右の Shift キー | NCTL_KEY_RIGHT_SHIFT | 0x20 |
| 右の Alt キー | NCTL_KEY_RIGHT_ALT | 0x40 |
| 右の Gui キー | NCTL_KEY_RIGHT_GUI | 0x80 |

コントロールキーダウンイベントで渡されるキーコードとキーダウン／キーアップイベントで渡されるキーコードは別の扱いとして下さい。

キーボードクラスを使用していない場合は、コントロールキーダウンイベントは通知されません。

#### 5.2.2.10 コントロールキーアップイベント

コントロールキーアップイベントは、キーボードデバイスでコントロールキーが離された時に呼び出されるイベントです。

第１パラメータにはキーが離されたデバイスのハンドルが渡されます。第３パラメータには離されたキーコードが格納されたバッファアドレスが渡されます。渡されるバッファは符号なし８ビットデータです。

第３パラメータに渡されるキーコードはコントロールキーダウンイベントと同等です。

キーボードクラスを使用していない場合は、コントロールキーアップイベントは通知されません。

#### 5.2.2.11 ボタンダウンイベント

ボタンダウンイベントは、マウスデバイスでボタンが押下された時に呼び出されるイベントです。
第１パラメータにはボタンが押下されたデバイスのハンドルが渡されます。第３パラメータには押下されたボタンの種別が格納されたバッファアドレスが渡されます。渡されるバッファは符号なし８ビットデータです。
第３パラメータに渡されるボタンの種別は次の通りです。

（表　ボタン種別）

| ボタン種別 | キーコード | 値 |
|---|---|---|
| Left ボタン | NBTN_LEFT | 0x01 |
| Rigth ボタン | NBTN_RIGTH | 0x02 |
| Middle ボタン | NBTN_MIDDLE | 0x04 |
| Side ボタン | NBTN_SIDE | 0x08 |
| Extra ボタン | NBTN_EXTRA | 0x10 |

マウスクラスを使用していない場合は、ボタンダウンイベントは通知されません。

#### 5.2.2.12 ボタンアップイベント

ボタンアップイベントは、マウスデバイスでボタンが離された時に呼び出されるイベントです。
第１パラメータにはボタンが離されたデバイスのハンドルが渡されます。第３パラメータには離されたボタンの種別が格納されたバッファアドレスが渡されます。渡されるバッファは符号なし８ビットデータです。
第３パラメータに渡されるボタンの種別はボタンダウンイベントと同等です。
マウスクラスを使用していない場合は、ボタンアップイベントは通知されません。

#### 5.2.2.13 Ｘ方向移動イベント

Ｘ方向移動イベントは、マウスデバイスでＸ軸方向の移動があった時に呼び出されるイベントです。
第１パラメータには移動があったデバイスのハンドルが渡されます。第３パラメータには移動量が格納されたバッファアドレスが渡されます。渡されるバッファは符号あり８ビットデータです。
マウスクラスを使用していない場合は、Ｘ方向移動イベントは通知されません。

#### 5.2.2.14 Ｙ方向移動イベント

Ｙ方向移動イベントは、マウスデバイスでＹ軸方向の移動があった時に呼び出されるイベントです。
第１パラメータには移動があったデバイスのハンドルが渡されます。第３パラメータには移動量が格納されたバッファアドレスが渡されます。渡されるバッファは符号あり８ビットデータです。
マウスクラスを使用していない場合は、Ｙ方向移動イベントは通知されません。

#### 5.2.2.15 ホイール移動イベント

ホイール移動イベントは、マウスデバイスでマウスホイールの移動（回転）があった時に呼び出されるイベントです。
第１パラメータには移動があったデバイスのハンドルが渡されます。第３パラメータには移動量が格納されたバッファアドレスが渡されます。渡されるバッファは符号あり８ビットデータです。
マウスクラスを使用していない場合は、ホイール移動イベントは通知されません。

#### 5.2.2.16 ＬＥＤ変更完了イベント

ＬＥＤ変更完了イベントは、キーボードデバイスでＬＥＤの変更があった時に呼び出されるイベントです。
第１パラメータにはＬＥＤの変更があったデバイスのハンドルが渡されます。第３パラメータには変更されたＬＥＤのデータが格納されたバッファアドレスが渡されます。渡されるバッファは符号なし８ビットデータです。
第３パラメータに渡されるＬＥＤデータは次の通りです。

（表　ＬＥＤデータ）

| ＬＥＤの種類 | ＬＥＤデータ | 値 |
|---|---|---|
| Num Lock | NLED_NUM_LOCK | 0x01 |
| Caps Lock | NLED_CAPS_LOCK | 0x02 |
| Scroll Lock | NLED_SCROLL_LOCK | 0x04 |
| Kana | NLED_KANA | 0x08 |
| Compose | NLED_COMPOSE | 0x10 |

ＬＥＤデータは対応するビットがセットされて渡されます。
ＬＥＤ変更完了イベントは、キーボードのＬＥＤ設定 API の Usb_Kbd_Set_Led が呼び出された後に通知されます。Usb_Kbd_Set_Led が正常に処理されれば変更後のＬＥＤデータが、処理に失敗した場合は以前のＬＥＤデータが渡されるはずです。
キーボードクラスを使用していない場合は、ＬＥＤ変更完了イベントは通知されません。

#### 5.2.2.17 致命的なエラーイベント

ITF-EasyHost のデバッグ機能が有効になっている場合に使用されるイベントです。
ITF-EasyHost で致命的なエラーを検出した場合に、致命的なエラーイベントが通知されます。このイベントが通知された場合、ITF-EasyHost は復帰が不可能な状態となっています。

### 5.2.3 注意

イベントのコールバックルーチンは、割り込みルーチン内で実行されています。ユーザアプリケーションは、コールバックルーチンの中では必要最低限の処理のみを行い、即時にコールバックルーチンを終了させる必要があります。

# 6 API 仕様

ITF-EasyHost の API について解説します。

## 6.1 初期化関数

### *Usb_Host_Init*

---

**機　能**　ITF-EasyHost の初期化

**形　式**　**istatus Usb_Host_Init(void);**

**解　説**　ITF-EasyHost の初期化及び資源の確保を行います。
本 API の呼び出し後に他の API が使用可能となります。
(但し、Usb_Host_Notify_Timer 呼び出しのためのタイマは本 API の呼び出し前に起動されている必要があります。このため、Usb_Host_Notify_Timer は本 API の呼び出し前及び呼び出し中も使用可能です。)

注１）プログラム起動後、他の API を使用する前に本 API を呼び出して下さい。
注２）本 API は Usb_Host_Exit を使用する前に再度呼び出すことはできません。

**戻　値**　**IE_OK**　　**：正常終了**
**エラーコード**　**：異常終了**

**例**　USB ホストの初期化を行います

```
void main( void )
{
    istatus r;

    r = Usb_Host_Init( );
    if (r != 0) {
        /* error */
    }
      :
}
```

## *Usb_Host_Exit*

**機　能**　ITF-EasyHost の終了

**形　式**　**istatus Usb_Host_Exit(void);**

**解　説**　Usb_Host_Init で開始した ITF-EasyHost の終了処理、使用した資源の解放を行います。
本 API は ITF-EasyHost を強制的に終了させますので、Usb_Host_Ep_Cancel や Usb_Host_Remove を使用して全ての API が未使用状態で呼び出して下さい。
Usb_Host_Exit の呼び出し後にイベントのコールバックルーチンは呼び出されません。

**戻　値**　**IE_OK**　：正常終了
**エラーコード**　：異常終了

**例**　USB ホストの終了

```
void main( void )
{
    istatus r;
     :
    r = Usb_Host_Exit( );
    if (r != 0) {
        /* error */
    }
     :
}
```

### 6.2 パイプ通信関数

## *Usb_Host_Open*

機　能　USB デバイスをオープンしハンドルを返します.

形　式　istatus Usb_Host_Open( const int8_t *name,
　　uint32_t flags
　　);
　　name　：USB デバイス名
　　flags　：フラグ（未使用：将来拡張用）

解　説　USB デバイスをオープンしハンドルを返します。
　　ユーザアプリケーションは、ここで取得したハンドルを使用して他 API を使用して下さい。

戻　値　ハンドル(正の数)：正常終了
　　エラーコード　：異常終了

例　デバイスをオープンします

```
#define DEV_NAME "¥¥¥¥¥¥USB¥¥DEV¥¥0000"
void main( void )
{
    ihandle handle;
     :
    handle = Usb_Host_Open( DEV_NAME, 0 );
    if(handle>0){
        /* 正常終了 */
    }
    else {
        /* オープンエラー */
    }
}
```

## *Usb_Host_Close*

機　能　USB デバイスのクローズ

形　式　**istatus Usb_Host_Close( ihandle handle);**
**handle　：デバイスハンドル**

解　説　Usb_Host_Close でオープンした USB デバイスをクローズします。

戻　値　**IE_OK　　：正常終了**
**エラーコード　　：異常終了**

例　　オープンしたデバイスをクローズします

```
#define DEV_NAME "¥¥¥¥¥USB¥¥DEV¥¥0000"
void main( void )
{
    ihandle handle;
    istatus r;
     :
    handle = Usb_Host_Open( DEV_NAME, 0 );
    if(handle>0){
         :
        r = Usb_Host_Close(handle);
        if (r < 0) {
            /* error */
        }
    }
     :
}
```

## *Usb_Host_Ep_Write*

**機　能**　エンドポイントを指定した書込み

**形　式**　**int32_t Usb_Host_Ep_Write( ihandle handle,**
**uint32_t epnum,**
**void *outbuf,**
**int32_t outsize,**
**struct io_overlapped_t *pov**
**);**

**handle**　：デバイスハンドル
**epnum**　：エンドポイントアドレス（0x01～0x0F, 0x81～0x8F）
***outbuf**　：書込み用データバッファポインタ
**outsize**　：書込みサイズ
***pov**　：非同期情報

**解　説**　エンドポイントを指定して非同期の USB 通信を行います。
handle で指定した USB デバイスの epnum で指定したエンドポイントへ outbuf のバッファから outsize バイト書き込みます。
epnum にはエンドポイントアドレスを設定します。エンドポイント番号が 1 の OUT 方向を設定する場合は 0x01 となります。epnum はエンドポイントディスクリプタの bEndpointAddress フィールドとなります。
pov が示す構造体には、非同期のための情報を設定します。非同期転送の詳細については、「非同期」を参照して下さい。

**戻　値**　**NE_PENDING**　：正常終了
**エラーコード**　：異常終了

例　　デバイスをオープンし、デバイスのエンドポイント（アドレス：0x02）へ100バイトの転送を行います

```
#define DEV_NAME "¥¥¥¥¥¥USB¥¥DEV¥¥0000"
#define FLAG 0
#define OUT_EPNUM   0x02

static int8_t outbuf[100];

#define CALLBACK_CODE   1
#define CALLBACK_PRM    outbuf
static void Event_Callback(ihandle dev, int32_t eventnum, void* param)
{
    if (eventnum == NC_EVENT_REQUEST_COMPLETE) {
        struct io_overlapped_result_t *presult = param;
        dev;                    /* デバイスハンドルが入ってくる */
        presult->status;        /* ステータスが入ってくる */
        presult->transferred; /* 処理バイト数が入ってくる */
        presult->code;          /* CALLBACK_CODE が入ってくる */
        presult->prm;           /* CALLBACK_PRM が入ってくる */
        if (presult->status != NE_URB_NORMAL_COMPLETION) ; /* エラー */
    }
}
void main( void )
{
    ihandle handle;
    istatus ret;
    struct io_overlapped_t ov;
      :
    handle = Usb_Host_Open( DEV_NAME, FLAG);
    if(handle>0){
        FILL_OVERLAPPED(&ov, CALLBACK_CODE, NCLASS_USB_DEVICE, CALLBACK_PRM);
        ret = Usb_Host_Ep_Write( handle,
                            OUT_EPNUM,
                            outbuf,
                            100,
                            &ov
                            );
        if (ret != NE_PENDING) ; /* エラー */
    }
}
```

## *Usb_Host_Ep_Read*

**機　能**　エンドポイントを指定した読込み

**形　式**　**int32_t Usb_Host_Ep_Read( ihandle hadnle,**
**uint32_t epnum,**
**void *inbuf,**
**int32_t insize,**
**struct io_overlapped_t *pov**
**);**

**handle**　：デバイスハンドル
**epnum**　：エンドポイントアドレス（0x01～0x0F, 0x81～0x8F）
***inbuf**　：読込み用データバッファポインタ
**insize**　：読込みサイズ
***pov**　：非同期情報

**解　説**　エンドポイントを指定して非同期の USB 通信を行います。
handle で指定した USB デバイスの epnum で指定したエンドポイントから inbuf のバッファへ insize バイト読み込みます。
epnum にはエンドポイントアドレスを設定します。エンドポイント番号が 1 の IN 方向を設定する場合は 0x81 となります。epnum はエンドポイントディスクリプタの bEndpointAddress フィールドとなります。
pov が示す構造体には、非同期のための情報を設定します。非同期転送の詳細については、「非同期」を参照して下さい。

デバイスから送られるデータはパケット単位で転送されます。パケットのサイズはエンドポイントごとに異なり、最大パケットサイズはエンドポイントディスクリプタの wMaxPacketSize によって決定されています。
inszie が小さく、転送されたデータサイズが insize より大きくなった場合、転送はエラーとなります。insize を最大パケットサイズ単位で指定することにより、このような転送のエラーは回避されます。デバイスから送られてくるデータサイズがわからない等の場合には、insize を最大パケットサイズ単位で指定して下さい。

**戻　値**　**NE_PENDING**　：正常終了
**エラーコード**　：異常終了

例　　デバイスをオープンし、デバイスのエンドポイント（アドレス：0x84）から 512 バイトの転送を行います

```
#define DEV_NAME "¥¥¥¥¥¥USB¥¥DEV¥¥0000"
#define FLAG 0
#define IN_EPNUM   0x84

static int8_t inbuf[512];

#define CALLBACK_CODE   2
#define CALLBACK_PRM    inbuf
static void Event_Callback(ihandle dev, int32_t eventnum, void* param)
{
    if (eventnum == NC_EVENT_REQUEST_COMPLETE) {
        struct io_overlapped_result_t *presult = param;
        dev;                    /* デバイスハンドルが入ってくる */
        presult->status;        /* ステータスが入ってくる */
        presult->transferred; /* 処理バイト数が入ってくる */
        presult->code;          /* CALLBACK_CODE が入ってくる */
        presult->prm;           /* CALLBACK_PRM が入ってくる */
        if (presult->status != NE_URB_NORMAL_COMPLETION) ;  /* エラー */
    }
}
void main( void )
{
    ihandle handle;
    istatus ret;
    struct io_overlapped_t ov;
      :
    handle = Usb_Host_Open( DEV_NAME,
                            FLAG);
    if(handle>0){
        FILL_OVERLAPPED(&ov, CALLBACK_CODE, NCLASS_USB_DEVICE CALLBACK_PRM);
        ret = Usb_Host_Ep_Read( handle,
                            IN_EPNUM,
                            inbuf,
                            512,
                            &ov
                            );
        if (ret != NE_PENDING) ; /* エラー */
    }
}
```

## *Usb_Host_Control*

**機　能**　コントロール転送

**形　式**　**int32_t Usb_Host_AControl( ihandle handle,**
**NUSB_CONTROL_T *usb_ctrl,**
**struct io_overlapped_t *pov**
**);**

**handle** : デバイスハンドル
***usb_ctrl** : コントロール通信
***pov** : 非同期情報

**解　説**　handle で指定した USB デバイスで非同期のコントロール転送を行います。
usb_ctrl には SETUP フィールドの情報を指定します。

| | |
|---|---|
| setup.req_type | : bmRequestType |
| setup.req | : bRequest |
| setup.value_h/_l | : wValue |
| setup.index_h/_l | : wIndex |
| setup.length_h/_l | : wLength |
| buf | : Data |

buf には、wLength で指定したサイズのバッファを指定します。buf は、setup.req_type のビット 7 が 1 の場合は読み込み用バッファ、0 の場合は書き込み用バッファとなります。
wLength が 0 の場合、buf は NULL となります。
pov が示す構造体には、非同期のための情報を設定します。非同期転送の詳細については、「非同期」を参照して下さい。

**戻　値**　**NE_PENDING** : 正常終了
**エラーコード** : 異常終了

例　　コントロール転送により、8 バイトのベンダリクエスト、100 バイトの OUT データの送出を行います。

```
#define DEV_NAME "¥¥¥¥¥¥USB¥¥DEV¥¥0000"
#define FLAG 0

static int8_t buffer[100];

#define CALLBACK_CODE   3
#define CALLBACK_PRM    buffer
static void Event_Callback(ihandle dev, int32_t eventnum, void* param)
{
    if (eventnum == NC_EVENT_REQUEST_COMPLETE) {
        struct io_overlapped_result_t *presult = param;
        dev;                    /* デバイスハンドルが入ってくる */
        presult->status;        /* ステータスが入ってくる */
        presult->transferred; /* 処理バイト数が入ってくる */
        presult->code;          /* CALLBACK_CODE が入ってくる */
        presult->prm;           /* CALLBACK_PRM が入ってくる */
        if (presult->status != NE_URB_NORMAL_COMPLETION) ;  /* エラー */
    }
}
void main( void )
{
    ihandle handle;
    istatus ret;
    NUSB_CONTROL_T usb_ctrl;
      :
    usb_ctrl.req_type = 0x40
    usb_ctrl.req      = 0;
    usb_ctrl.value_h  = 0; usb_ctrl.value_l  = 0;
    usb_ctrl.index_h  = 0; usb_ctrl.index_l  = 0;
    usb_ctrl.length_h = 0; usb_ctrl.length_l = 100;
    usb_ctrl.buf      = (void *)buffer;
    handle = Usb_Host_Open( DEV_NAME, FLAG);
    if(handle>0){
        FILL_OVERLAPPED(&ov, CALLBACK_CODE, NCLASS_USB_DEVICE, CALLBACK_PRM);
        ret = Usb_Host_Control( handle,
                        &usb_ctrl,
                        &ov
                        );
        if (ret != NE_PENDING) ; /* エラー */
    }
}
```

## *Usb_Host_Ep_Cancel*

**機　能**　指定エンドポイントの転送の中断

**形　式**　**istatus Usb_Host_Ep_Cancel( ihandle handle,**
**uint32_t epnum );**
**handle　: デバイスハンドル**
**epnum　: エンドポイントアドレス（0x00～0x0F, 0x80～0x8F）**

**解　説**　指定されたエンドポイントの転送を中断させます。
handle で指定した USB デバイスの epnum で指定したエンドポイントの転送をキャンセルします。
epnum にはエンドポイントアドレスを設定します。エンドポイント番号が 1 の IN 方向を設定する場合は 0x81 となります。epnum はエンドポイントディスクリプタの bEndpointAddress フィールドとなります。

**戻　値**　**IE_OK　　: 正常終了**
**エラーコード　: 異常終了**

例　　main で Read 転送中に転送を中断したい場合、転送キャンセル API を呼び転送の中断を行います

```
#define DEV_NAME "¥¥¥¥¥¥USB¥¥DEV¥¥0000"
#define FLAG 0
#define IN_EPNUM   0x84

static int8_t inbuf[512];
static volatile ibool complete;
static volatile ibool cancel;

#define CALLBACK_CODE   2
#define CALLBACK_PRM    inbuf
static void Event_Callback(ihandle dev, int32_t eventnum, void* param)
{
    if (eventnum == NC_EVENT_REQUEST_COMPLETE) {
        complete = TRUE;
    }
}
void main( void )
{
    ihandle handle;
    istatus ret;
    struct io_overlapped_t ov;
      :
    handle = Usb_Host_Open( DEV_NAME, FLAG );
    if(handle>0){
        complete = FALSE; cancel = FALSE;
        FILL_OVERLAPPED(&ov, CALLBACK_CODE, NCLASS_USB_DEVICE CALLBACK_PRM);
        ret = Usb_Host_Ep_Read( handle, IN_EPNUM, inbuf, 512, &ov );
        if (ret == NE_PENDING) {
            while (!complete) {
                if (cancel) Usb_Host_Ep_Cancel( handle, IN_EPNUM );
            }
        }
        else ; /* エラー */
    }
}

void Some_Interrupt(void)
{
    cancel = TRUE;
}
```

### 6.3 ディスクリプタ取得関数

## *Usb_Host_Get_Device_Descriptor*

**機　能**　デバイスディスクリプタの取得

**形　式**　**int32_t Usb_Host_Get_Device_Descriptor( ihandle handle,**
**DEVICE_DESC *devdesc,**
**struct io_overlapped_t *pov**
**);**

**handle**　：デバイスハンドル
***devdesc**　：デバイスディスクリプタ格納バッファ
***pov**　：非同期情報

**解　説**　handle で指定したデバイスのデバイスディスクリプタを devdesc に格納します。
戻値がNE_PENDINGの場合、リクエスト完了イベントにより取得の完了が通知されます。

**戻　値**　**取得できたディスクリプタ構造体のサイズ**　：正常終了
**NE_PENDING**　：未完了
**エラーコード**　：異常終了

例　　デバイスをオープンし、デバイスディスクリプタを取得します。

```
#define DEV_NAME "¥¥¥¥¥USB¥¥DEV¥¥0000"
#define FLAG 0
void main( void )
{
    ihandle handle;
    io_overlapped_t ov;
    DEVICE_DESC devdesc;
    int32_t cnt;
     :

    handle = Usb_Host_Open( DEV_NAME,  FLAG );
    if(handle>0){
        FILL_OVERLAPPED(&ov, CALLBACK_CODE, NCLASS_USB_DEVICE, CALLBACK_PARAM);
        cnt = Usb_Host_Get_Device_Descriptor( handle,
                                              &devdesc,
                                              &ov
                                             );
        if (cnt >= 0) {
            /* 成功 */
        }
        else if (cnt == NE_PENDING) {
            /* イベントコールバックで完了通知 */
        }
        else {
            /* error */
        }
    }
}
```

## *Usb_Host_Get_Configration_Descriptor*

**機　能**　コンフィグレーションディスクリプタ取得

**形　式**　**int32_t Usb_Host_Get_Configration_Descriptor( ihandle handle,**
**CONFIG_DESC *cfgdesc,**
**struct io_overlapped_t *pov**
**);**

**handle**　：デバイスハンドル
***cfgdesc**　：コンフィグレーションディスクリプタ格納バッファ
***pov**　：非同期情報

**解　説**　handle で指定したデバイスのコンフィグレーションディスクリプタを cfdesc に格納します。
戻値がNE_PENDINGの場合、リクエスト完了イベントにより取得の完了が通知されます。

**戻　値**　**取得できたディスクリプタ構造体のサイズ**　：正常終了
**NE_PENDING**　：未完了
**エラーコード**　：異常終了

例　　デバイスをオープンし、コンフィグレーションディスクリプタを取得します

```
#define DEV_NAME "¥¥¥¥¥USB¥¥DEV¥¥0000"
#define FLAG 0
void main( void )
{
    ihandle handle;
    io_overlapped_t ov;
    CONFIG_DESC cfgdesc;
    int32_t cnt;
     :

    handle = Usb_Host_Open( DEV_NAME,  FLAG );
    if(handle>0){
        FILL_OVERLAPPED(&ov, CALLBACK_CODE, NCLASS_USB_DEVICE, CALLBACK_PARAM);
        cnt = Usb_Host_Get_Configuration_Descriptor( handle,
                                                     &cfgdesc,
                                                     &ov
                                                    );
        if (cnt >= 0) {
            /* 成功 */
        }
        else if (cnt == NE_PENDING) {
            /* イベントコールバックで完了通知 */
        }
        else {
            /* error */
        }
    }
}
```

## *Usb_Host_Get_Interface_Descriptor*

**機　能**　インターフェイスディスクリプタの取得

**形　式**　**int32_t Usb_Host_Get_Interface_Descriptor( ihandle handle,**
**INTERFACE_DESC *ifdesc,**
**uint8_t altsetting,**
**io_overlapped_t *pov**
**);**

| | |
|---|---|
| **handle** | **：デバイスハンドル** |
| ***ifdesc** | **：インターフェイスディスクリプタ格納バッファ** |
| **altsetting** | **：代替設定値** |
| ***pov** | **：非同期情報** |

**解　説**　handle で指定したデバイスの altsetting で指定した代替設定値のインターフェイスディスクリプタを ifdesc に格納します。
代替設定値 altsetting はインターフェイスディスクリプタの bAlternateSetting フィールドとなります。
戻値がNE_PENDINGの場合、リクエスト完了イベントにより取得の完了が通知されます。

**戻　値**

| | |
|---|---|
| **取得できたディスクリプタ構造体のサイズ** | **：正常終了** |
| **NE_PENDING** | **：未完了** |
| **エラーコード** | **：異常終了** |

例　　デバイスをオープンし、代替設定値０のインターフェイスディスクリプタを取得します

```
#define DEV_NAME "¥¥¥¥¥USB¥¥DEV¥¥0000"
#define FLAG 0
void main( void )
{
    ihandle handle;
    io_overlapped_t ov;
    INTERFACE_DESC ifdesc;
    int32_t cnt;
    uint8_t altsetting = 0;
     :
    handle = Usb_Host_Open( DEV_NAME,
                            FLAG);
    if(handle>0){
        FILL_OVERLAPPED(&ov, CALLBACK_CODE, NCLASS_USB_DEVICE, CALLBACK_PARAM);
        cnt = Usb_Host_Get_Interface_Descriptor( handle,
                                                 &ifdesc,
                                                 altsetting,
                                                 &ov
                                                );
        if (cnt >= 0) {
            /* 成功 */
        }
        else if (cnt == NE_PENDING) {
            /* イベントコールバックで完了通知 */
        }
        else {
            /* error */
        }
    }
}
```

## *Usb_Host_Get_EndPoint_Descriptor*

**機　能**　エンドポイントディスクリプタの取得（エンドポイントアドレス指定）

**形　式**　**int32_t Usb_Host_Get_EndPoint_Descriptor( ihandle handle,**
**ENDPOINT_DESC *epdesc,**
**uint8_t altsetting,**
**uint8_t epnum,**
**struct io_overlapped_t *pov**
**);**

| | |
|---|---|
| **handle** | **：デバイスハンドル** |
| ***epdesc** | **：エンドポイントディスクリプタ格納バッファ** |
| **altsetting** | **：代替設定値** |
| **epnum** | **：エンドポイントアドレス（0x01～0x0F, 0x81～0x8F）** |
| ***pov** | **：非同期情報** |

**解　説**　handle で指定したデバイスの altsetting で指定した代替設定値、epnum で指定したエンドポイントのエンドポイントディスクリプタを epdesc に格納します。
epnum にはエンドポイントアドレスを設定します。エンドポイント番号が 1 の IN 方向を設定する場合は 0x81 となります。epnum はエンドポイントディスクリプタの bEndpointAddress フィールドとなります。

エンドポイントディスクリプタ構造体(ENDPOINT_DESC)のメンバ変数の bRefresh と bSynchAddress は Audio Class で使用されます。その他のクラスでは参照しないで下さい。

戻値が NE_PENDING の場合、リクエスト完了イベントにより取得の完了が通知されます。

**戻　値**

| | |
|---|---|
| **取得できたディスクリプタ構造体のサイズ** | **：正常終了** |
| **NE_PENDING** | **：未完了** |
| **エラーコード** | **：異常終了** |

例　　デバイスをオープンし、代替設定値 0 のエンドポイントアドレス 0x02 のエンドポイントディスクリプタを取得します。

```
#define DEV_NAME "¥¥¥¥¥¥USB¥¥DEV¥¥0000"
#define FLAG 0
void main( void )
{
    ihandle handle;
    struct io_overlapped_t ov;
    ENDPOINT_DESC epdesc;
    int32_t cnt;
    uint8_t altsetting = 0;
    uint8_t epnum = 0x02;
     :

    handle = Usb_Host_Open( DEV_NAME,
                            FLAG);
    if(handle>0){
        FILL_OVERLAPPED(&ov, CALLBACK_CODE, NCLASS_USB_DEVICE, CALLBACK_PARAM);
        cnt = Usb_Host_Get_Endpoint_Descriptor( handle,
                                                &epdesc,
                                                altsetting,
                                                epnum,
                                                &ov
                                               );
        if (cnt >= 0) {
            /* 成功 */
        }
        else if (cnt == NE_PENDING) {
            /* イベントコールバックで完了通知 */
        }
        else {
            /* error */
        }
    }
}
```

## *Usb_Host_Get_EndPoint_Descriptor_Count*

**機　能**　エンドポイントディスクリプタ取得（エンドポイント序数指定）

**形　式**　**int32_t Usb_Host_Get_EndPoint_Descriptor_Count( ihandle handle,**
**ENDPOINT_DESC *epdesc,**
**uint8_t altsetting,**
**uint8_t count,**
**io_overlapped_t *pov**
**);**

| | |
|---|---|
| **handle** | **：デバイスハンドル** |
| ***epdesc** | **：エンドポイントディスクリプタ格納バッファ** |
| **altsetting** | **：代替設定値** |
| **count** | **：カウント** |
| ***pov** | **：非同期情報** |

**解　説**　handle で指定したデバイスの altsetting で指定した代替設定値、count 番目に発見されたエンドポイントのエンドポイントディスクリプタを epdesc に格納します。
count は 1 以上の値となります。１番目に取得できたエンドポイントの場合でも、必ずしも次に取得できたエンドポイントより、エンドポイントアドレスが小さいということはありません。

エンドポイントディスクリプタ構造体(ENDPOINT_DESC)のメンバ変数の bRefresh と bSynchAddress は Audio Class で使用されます。その他のクラスでは参照しないで下さい。

戻値がNE_PENDINGの場合、リクエスト完了イベントにより取得の完了が通知されます。

**戻　値**

| | |
|---|---|
| **取得できたディスクリプタ構造体のサイズ** | **：正常終了** |
| **NE_PENDING** | **：未完了** |
| **エラーコード** | **：異常終了** |

例　　デバイスをオープンし、代替設定値 0 の３番目のエンドポイントのエンドポイントディスクリプタを取得します。

```
#define DEV_NAME "¥¥¥¥¥¥USB¥¥DEV¥¥0000"
#define FLAG 0
void main( void )
{
    ihandle handle;
    struct io_overlapped_t ov;
    ENDPOINT_DESC epdesc;
    int32_t cnt;
    uint8_t altsetting = 0;
    uint8_t count = 3;
     :

    handle = Usb_Host_Open( DEV_NAME,
                            FLAG);
    if(handle>0){
        FILL_OVERLAPPED(&ov, CALLBACK_CODE, NCLASS_USB_DEVICE, CALLBACK_PARAM);
        cnt = Usb_Host_Get_Endpoint_Descriptor_Count( handle,
                                                      &epdesc,
                                                      altsetting,
                                                      count,
                                                      &ov
                                                      );
        if (cnt >= 0) {
            /* 成功 */
        }
        else if (cnt == NE_PENDING) {
            /* イベントコールバックで完了通知 */
        }
        else {
            /* error */
        }
    }
}
```

## *Usb_Host_Get_String_Descriptor*

**機　能**　ストリングディスクリプタ取得

**形　式**　**int32_t Usb_Host_Get_String_Descriptor( ihandle handle,**
**uint16_t langid,**
**uint8_t index,**
**int8_t *buf,**
**uint16_t length,**
**struct io_overlapped_t *pov**
**);**

| | |
|---|---|
| **handle** | **: デバイスハンドル** |
| **langid** | **: 0 または言語 ID** |
| **index** | **: ディスクリプタへのインデックス** |
| ***buf** | **: ストリングディスクリプタ格納バッファ** |
| **length** | **: ディスクリプタサイズ** |
| ***pov** | **: 非同期情報** |

**解　説**　handle で指定したデバイスから langid の言語ＩＤ、index のインデックスのストリングディスクリプタを buf に length バイト取得します。
buf には UNICODE で文字列が格納されます。アプリケーションで使用する場合には、必要に応じてコードの変換が必要となります。
戻値が NE_PENDING の場合、リクエスト完了イベントにより取得の完了が通知されます。

**戻　値**

| | |
|---|---|
| **取得できたディスクリプタのサイズ** | **: 正常終了** |
| **NE_PENDING** | **: 未完了** |
| **エラーコード** | **: 異常終了** |

例　　デバイスをオープンし、インデックスが 2 のストリングディスクリプタを取得します。
buffer に格納される文字列は UNICODE 文字列ですのでご注意ください。

```
#define DEV_NAME "¥¥¥¥¥¥USB¥¥DEV¥¥0000"
void main( void )
{
    ihandle handle;
    struct io_overlapped_t ov;
    uint16_t langid = 0;
    uint8_t index = 2;
    int8_t buffer[256];
     :

    handle = Usb_Host_Open( DEV_NAME, 0 );
    if(handle>0){
        FILL_OVERLAPPED(&ov, CALLBACK_CODE, NCLASS_USB_DEVICE, CALLBACK_PARAM);
        cnt = Usb_Host_Get_String_Descriptor( handle,
                                              langid,
                                              index,
                                              buffer,
                                              256,
                                              &ov
                                              );
        if (cnt >= 0) {
            /* 成功 */
        }
        else if (cnt == NE_PENDING) {
            /* イベントコールバックで完了通知 */
        }
        else {
            /* error */
        }
    }
}
```

## *Usb_Host_Get_Others_Descriptor*

**機　能**　その他のディスクリプタ取得

**形　式**　int32_t Usb_Host_Get_String_Descriptor( ihandle handle,
int8_t *buf,
int32_t size,
int8_t altsetting,
int32_t count,
struct io_overlapped_t *pov
);

| | |
|---|---|
| handle | : デバイスハンドル |
| *buf | : ディスクリプタ格納バッファ |
| size | : ディスクリプタサイズ |
| altsetting | : 代替設定値 |
| count | : ディスクリプタインデックス |
| *pov | : 非同期情報 |

**解　説**　handle で指定したデバイスから count のインデックスの、インターフェース・エンドポイント以外のディスクリプタを buf に size バイト取得します。
戻値がNE_PENDINGの場合、リクエスト完了イベントにより取得の完了が通知されます。

**戻　値**

| | |
|---|---|
| 取得できたディスクリプタのサイズ | : 正常終了 |
| NE_PENDING | : 未完了 |
| エラーコード | : 異常終了 |

例　　デバイスをオープンし、代替設定値０の２番目のその他のディスクリプタを取得します。

```
#define DEV_NAME "¥¥¥¥¥USB¥¥DEV¥¥0000"
void main( void )
{
    ihandle handle;
    struct io_overlapped_t ov;
    int8_t buffer[256];
    uint8_t altsetting = 0;
    uint8_t index = 2;
    :

    handle = Usb_Host_Open( DEV_NAME, 0 );
    if(handle>0){
        FILL_OVERLAPPED(&ov, CALLBACK_CODE, NCLASS_USB_DEVICE, CALLBACK_PARAM);
        cnt = Usb_Host_Get_Others_Descriptor( handle,
                                              buffer,
                                              256,
                                              altsetting,
                                              index,
                                              &ov
                                              );
        if (cnt >= 0) {
            /* 成功 */
        }
        else if (cnt == NE_PENDING) {
            /* イベントコールバックで完了通知 */
        }
        else {
            /* error */
        }
    }
}
```

### 6.4 電源管理関数

## *Usb_Host_Suspend*

**機　能**　バスをサスペンドする

**形　式**　istatus Usb_Host_Suspend(void);

**解　説**　バスをサスペンドさせ、SOF の送出を停止します。
サスペンドは全デバイスの通信が行われていない状態で行って下さい。

バスをサスペンドさせた後、ウェイクアップする前にデバイスを切断しないで下さい。サスペンドした後は、ウェイクアップするまでその状態を維持して下さい。

戻値が NE_PENDING の場合、サスペンド完了イベントにより完了が通知されます。

本 API はデバイスに依存する要素が多いため、正式サポートではありません。(ただし、多くのデバイスで問題ないことは確認してあります。)

**戻　値**　IE_OK　　：正常終了
NE_PENDING　：未完了
エラーコード　：異常終了

**例**　デバイスをオープンし、サスペンドさせます

```
#define DEV_NAME "¥¥¥¥¥¥USB¥¥DEV¥¥0000"
void main( void )
{
    istatus r;
     :

    r = Usb_Host_Suspend( );
    if (r == NE_PENDING) {
        /* イベントコールバックで完了通知 */
    }
    else if (r < 0) {
        /* error */
    }
}
```

## *Usb_Host_Wakeup*

**機　能**　バスをウェイクアップさせる

**形　式**　istatus Usb_Host_Wakeup(void);

**解　説**　バスをウェイクアップさせ、SOF の送出を開始します。
戻値が NE_PENDING の場合、ウェイクアップ要求イベントにより完了が通知されます。

本 API はデバイスに依存する要素が多いため、正式サポートではありません。(ただし、多くのデバイスで問題ないことは確認してあります。)

**戻　値**　IE_OK　　：正常終了
NE_PENDING　：未完了
エラーコード　：異常終了

**例**　デバイスをオープンし、サスペンドさせた後、ウェイクアップします

```
#define DEV_NAME "¥¥¥¥¥USB¥¥DEV¥¥0000"
void main( void )
{
    istatus r;
     :
    r = Usb_Host_Suspend( );
    if (r == NE_PENDING) {
        /* イベントコールバックで完了通知 */
    }
    else if (r < 0) {
        /* error */
    }

    r = Usb_Host_Wakeup( );
    if (r == NE_PENDING) {
        /* イベントコールバックで完了通知 */
    }
    else if (r < 0) {
        /* error */
    }
}
```

## *Usb_Host_Remove*

**機　能**　デバイスの削除

**形　式**　**istatus Usb_Host_Remove(void);**
**handle　：デバイスハンドル**

**解　説**　デバイスを削除します。
ITF_EasyHost ではサポートされておりません。NE_NOT_CMD_SUPPORTED を返します。

**戻　値**　**IE_OK　　：正常終了**
**NE_PENDING　：未完了**
**エラーコード　　：異常終了**

## *Usb_Host_Reset*

**機　能**　デバイスのリセット

**形　式**　**istatus Usb_Host_Reset( void );**

**解　説**　バスをリセットします。
戻値が NE_PENDING の場合、リセット完了イベントにより完了が通知されます。

**戻　値**　**IE_OK**　：正常終了
**NE_PENDING**　：未完了
**エラーコード**　：異常終了

**例**　デバイスをオープンし、リセットを行います。

```
#define DEV_NAME "¥¥¥¥¥USB¥¥DEV¥¥0000"
void main( void )
{
    istatus r;
     :

    r = Usb_Host_Reset(handle);
    if (r == NE_PENDING) {
        /* イベントコールバックで完了通知 */
    }
    else if (r < 0) {
        /* error */
    }
}
```

## *Usb_Host_Set_Remote_Wakeup*

**機　能**　リモートウェイクアップ設定

**形　式**　istatus Usb_Host_Set_Remote_Wakeup(ihandle handle,
ibool bremote,
struct io_overlapped_t *pov
);

handle　：デバイスハンドル
bremote：リモートウェイクアップのフラグ
*pov　　：非同期情報

**解　説**　handle で指定したデバイスにリモートウェイクアップの設定を行います。
bremote が 0 ならリモートウェイクアップの無効を、1 ならリモートウェイクアップの有効を設定します。
戻値が NE_PENDING の場合、リクエスト完了イベントにより完了が通知されます。

**戻　値**　IE_OK　　：正常終了
NE_PENDING　：未完了
エラーコード　　：異常終了

**例**　デバイスをオープンし、リモートウェイクアップ(有効)を設定します。

```
#define DEV_NAME "¥¥¥¥¥USB¥¥DEV¥¥0000"
void main( void )
{
    ihandle handle;
    struct io_overlapped_t ov;
    istatus r;
     :

    handle = Usb_Host_Open( DEV_NAME, 0 );
    if(handle > 0){
        FILL_OVERLAPPED(&ov, CALLBACK_CODE, NCLASS_USB_DEVICE, CALLBACK_PARAM);
        r = Usb_Host_Set_Remote_Wakeup(handle, 1, &ov);
        if (r == NE_PENDING) {
            /* イベントコールバックで完了通知 */
        }
        else if (r < 0) {
            /* error */
        }
    }
}
```

## *Usb_Host_Get_Remote_Wakeup*

**機　能**　リモートウェイクアップ設定の取得

**形　式**　ibool Usb_Host_Get_Remote_Wakeup(ihandle handle);
handle　：デバイスハンドル

**解　説**　handle で指定したデバイスのリモートウェイクアップの設定を取得します。

**戻　値**　0 以外　　：リモートウェイクアップ有効
0　　：リモートウェイクアップ無効

**例**　デバイスをオープンし、リモートウェイクアップの設定を取得します。

```
#define DEV_NAME "¥¥¥¥¥USB¥¥DEV¥¥0000"
void main( void )
{
    ihandle handle;
    ibool bremote;
     :

    handle = Usb_Host_Open( DEV_NAME, 0 );
    if(handle > 0){
        bremote = Usb_Host_Get_Remote_Wakeup(handle);
        if (bremote) {
            /* RemoteWakeup Enable */
        }
        else {
            /* RemoteWakeup Disable */
        }
    }
}
```

### 6.5 デバイス情報取得関数

## *Usb_Host_Get_Device_Count*

**機　能**　接続されているデバイス数を取得する

**形　式**　**int32_t Usb_Host_Get_Device_Count ( void );**

**解　説**　接続されているデバイス数を取得します。
ここでカウントされるデバイス数は、インターフェイス数をカウントしています。例えば、ひとつのデバイスにキーボードとマウスのふたつの機能が実装されている場合には、それは２つのデバイスとしてカウントされます。

**戻　値**　**１以上**　　**：接続されているデバイス数**
**0**　　**：接続されているデバイスはない**

**例**　　接続されているデバイスの数を取得します

```
void main( void )
{
    int32_t dev_count;
     :
    dev_count = Usb_Host_Get_Device_Count();
     :
}
```

# *Usb_Host_Get_Device_Information*

**機　能**　デバイス情報の取得（デバイスハンドル指定）

**形　式**　**istatus Usb_Host_Get_Device_Information( ihandle handle,**
**NUSB_DEVICE_INFORMATION *info**
**);**

**handle　: デバイスハンドル**
***info　　: デバイス情報格納バッファ**

**解　説**　handle で指定したデバイスの情報を info に格納します。
info->dev_name[]にはデバイス名が設定されます。これは、Usb_Host_Open で指定する USB デバイス名となります。
info->class_type に次のクラス種別が設定されます。

| クラス種別コード | 値 | 説明 |
|---|---|---|
| NCLASS_USB_DEVICE | 0 | 下記以外のクラス |
| NCLASS_USB_HID_KEYBOARD | 3 | ＨＩＤ（キーボード）クラス |
| NCLASS_USB_HID_MOUSE | 4 | ＨＩＤ（マウス）クラス |
| NCLASS_USB_STROGE | 6 | マスストレージクラス |

**戻　値**　**IE_OK　　: 正常終了**
**エラーコード　: 異常終了**

例　　デバイスをオープンし、デバイス情報を取得します

```
#define DEV_NAME "¥¥¥¥¥USB¥¥DEV¥¥0000"
#define FLAG 0
void main( void )
{
    ihandle handle;
    istatus r;
    NUSB_DEVICE_INFORMATION info;
     :

    handle = Usb_Host_Open( DEV_NAME,
                            FLAG);
    if(handle>0){
        r = Usb_Host_Get_Device_Information( handle,
                                             &info
                                            );
        if (r < 0) {
            /* error */
        }
    }
}
```

## *Usb_Host_Search_Device*

**機　能**　デバイス情報の取得

**形　式**　istatus Usb_Host_Search_Device( NUSB_DEVICE_INFORMATION *info,
int32_t count
);
*info　：デバイス情報格納バッファ
count　：カウント数

**解　説**　count 番目に発見されたデバイスの情報を info に格納します。

**戻　値**　IE_OK　：正常終了
エラーコード　：異常終了

**例**　2 番目のデバイスのデバイス情報を取得します

```
void main( void )
{
    NUSB_DEVICE_INFORMATION info;
    istatus r;
    int32_t count=2;
     :

    r = Usb_Host_Get_Device_Information( &info,
                                         count
                                        );
    if (r < 0) {
        /* error */
    }
}
```

### 6.6 デバイスリクエスト発行関数

## *Usb_Host_Set_Interface*

**機　能**　代替設定値の設定

**形　式**　istatus Usb_Host_Set_Interface( ihandle handle,
uint8_t alternate,
struct io_overlapped_t *pov
);

handle　：デバイスハンドル
alternate　：代替設定値
*pov　：非同期情報

**解　説**　handle で指定したデバイスに alternate で指定した代替設定値を設定します。
本 API では標準リクエストの SET INTERFACE をデバイスに送信します。また、内部的に代替設定値の変更処理を行います。
戻値が NE_PENDING の場合、リクエスト完了イベントにより完了が通知されます。

**戻　値**　IE_OK　：正常終了
NE_PENDING　：未完了
エラーコード　：異常終了

例　　デバイスをオープンし代替設定値１の設定します

```
#define DEV_NAME "¥¥¥¥¥USB¥¥DEV¥¥0000"
#define FLAG 0
void main( void )
{
    ihandle handle;
    struct io_overlapped_t ov;
    istatus r;
    uint8_t alternate = 1;
     :

    handle = Usb_Host_Open( DEV_NAME, FLAG );
    if(handle>0){
        FILL_OVERLAPPED( &ov, CALLBACK_CODE, NCLASS_USB_DEVICE, CALLBACK_PARAM );
        r = Usb_Host_Set_Interface( handle,
                                    alternate,
                                    &ov
                                  );
        if (r == NE_PENDING) {
            /* イベントコールバックで完了通知 */
        }
        else if (r < 0) {
            /* error */
        }
    }
}
```

## *Usb_Host_Get_Interface*

**機　能**　代替設定値の取得

**形　式**　**istatus Usb_Host_Get_Interface( ihandle handle,**
**uint8_t *alternate,**
**struct io_overlapped_t *pov**
**);**

**handle**　：デバイスハンドル
***alternate**　：代替設定値格納バッファ
***pov**　：非同期情報

**解　説**　handle で指定したデバイスの代替設定値を alternate に格納します。
alternate は 1 バイトの領域が必要となります。
本 API では標準リクエストの GET INTERFACE をデバイスに送信します。
戻値が NE_PENDING の場合、リクエスト完了イベントにより完了が通知されます。

**戻　値**　**IE_OK**　：正常終了
**NE_PENDING**　：未完了
**エラーコード**　：異常終了

例　　デバイスをオープンし代替設定値を取得します

```
#define DEV_NAME "¥¥¥¥¥USB¥¥DEV¥¥0000"
#define FLAG 0
void main( void )
{
    ihandle handle;
    struct io_overlapped_t ov;
    istatus r;
    uint8_t alternate;
     :

    handle = Usb_Host_Open( DEV_NAME, FLAG );
    if(handle>0){
        FILL_OVERLAPPED( &ov, CALLBACK_CODE, NCLASS_USB_DEVICE, CALLBACK_PARAM );
        r = Usb_Host_Get_Interface( handle,
                                    &alternate,
                                    &ov
                                  );
        if (r == NE_PENDING) {
            /* イベントコールバックで完了通知 */
        }
        else if (r < 0) {
            /* error */
        }
    }
}
```

## *Usb_Host_Get_Status*

**機　能**　デバイス・インターフェイス・エンドポイントの状態の取得

**形　式**　**istatus Usb_Host_Get_Status( ihandle handle,**
**uint8_t type,**
**uint16_t target,**
**void *buf,**
**struct io_overlapped_t *pov**
**);**

**handle**　: デバイスハンドル
**type**　: デバイス(0) or インターフェイス(1) or エンドポイント(2)
**target**　: 0 or インターフェイス番号 or エンドポイント番号
***buf**　: ステータス格納バッファ
***pov**　: 非同期情報

**解　説**　handle で指定したデバイスの type 種別、target 番号のステータスを buf に格納します。
buf は２バイトの領域が必要となります。
本 API では標準リクエストの GET STATUS をデイバスに送信します。
戻値が NE_PENDING の場合、リクエスト完了イベントにより完了が通知されます。

**戻　値**　**IE_OK**　: 正常終了
**NE_PENDING**　: 未完了
**エラーコード**　: 異常終了

例　　デバイスをオープンし、デバイスに対してステータスの取得を行います

```
#define DEV_NAME "¥¥¥¥¥USB¥¥DEV¥¥0000"
#define FLAG 0
void main( void )
{
    ihandle handle;
    struct io_overlapped_t ov;
    istatus r;
    uint8_t type   =0;
    uint16_t target=0;
    int8_t buffer[2];
     :

    handle = Usb_Host_Open( DEV_NAME, FLAG );
    if(handle>0){
        FILL_OVERLAPPED( &ov, CALLBACK_CODE, NCLASS_USB_DEVICE, CALLBACK_PARAM );
        r = Usb_Host_Get_Status( handle,
                                 type,
                                 target,
                                 buffer,
                                 &ov
                                );
        if (r == NE_PENDING) {
            /* イベントコールバックで完了通知 */
        }
        else if (r < 0) {
            /* error */
        }
    }
}
```

# *Usb_Host_Ep_Reset*

**機　能**　パイプのリセット

**形　式**　istatus Usb_Host_Ep_Reset( ihandle handle,
uint32_t epnum,
struct io_overlapped_t *pov
);

| | |
|---|---|
| handle | : デバイスハンドル |
| epnum | : エンドポイントアドレス（0x00～0x0F、0x81～0x8F） |
| *pov | : 非同期情報 |

**解　説**　handle で指定したデイバスの epnum で指定したアドレスのエンドポイントをリセットします。
本 API では標準リクエストの CLEAR FEATURE をデバイスに送信します。また、内部的にパイプのリセット処理を行います。

本 API は通常、エンドポイントがストールした場合に使用します。また、Usb_Host_Ep_Abort によって停止したエンドポイントを再び使用する場合にも使用します。
戻値が NE_PENDING の場合、リクエスト完了イベントにより完了が通知されます。

**戻　値**

| | |
|---|---|
| IE_OK | : 正常終了 |
| NE_PENDING | : 未完了 |
| エラーコード | : 異常終了 |

例　　デバイスをオープンしエンドポイント（アドレス：0x02）をリセットします

```
#define DEV_NAME "¥¥¥¥¥USB¥¥DEV¥¥0000"
#define FLAG 0
void main( void )
{
    ihandle handle;
    struct io_overlapped_t ov;
    istatus r;
    uint32_t epnum = 0x02;
     :

    handle = Usb_Host_Open( DEV_NAME, FLAG );
    if(handle>0){
        FILL_OVERLAPPED( &ov, CALLBACK_CODE, NCLASS_USB_DEVICE, CALLBACK_PARAM );
        r = Usb_Host_Ep_Reset( handle,
                               epnum,
                               &ov
                             );
        if (r == NE_PENDING) {
            /* イベントコールバックで完了通知 */
        }
        else if (r < 0) {
            /* error */
        }
    }
}
```

## *Usb_Host_Ep_Abort*

**機　能**　パイプの停止

**形　式**　**istatus Usb_Host_Ep_Abort( ihandle handle,**
**uint32_t epnum,**
**struct io_overlapped_t *pov**
**);**

**handle**　：デバイスハンドル
**epnum**　：エンドポイントアドレス（0x00～0x0F、0x81～0x8F）
***pov**　：非同期情報

**解　説**　handle で指定したデイバスの epnum で指定したアドレスのエンドポイントを停止します。
本 API では標準リクエストの SET FEATURE をデバイスに送信します。また、内部的にパイプの停止処理を行います。
戻値が NE_PENDING の場合、リクエスト完了イベントにより完了が通知されます。

**戻　値**　**IE_OK**　：正常終了
**NE_PENDING**　：未完了
エラーコード　：異常終了

例　　デバイスをオープンしエンドポイント（アドレス：0x81）を停止します

```
#define DEV_NAME "¥¥¥¥¥USB¥¥DEV¥¥0000"
#define FLAG 0
void main( void )
{
    ihandle handle;
    struct io_overlapped_t ov;
    istatus r;
    uint32_t epnum = 0x81;
     :

    handle = Usb_Host_Open( DEV_NAME,  FLAG );
    if(handle>0){
        FILL_OVERLAPPED( &ov, CALLBACK_CODE, NCLASS_USB_DEVICE, CALLBACK_PARAM );
        r = Usb_Host_Ep_Abort( handle,
                               epnum,
                               &ov
                             );
        if (r == NE_PENDING) {
            /* イベントコールバックで完了通知 */
        }
        else if (r < 0) {
            /* error */
        }
    }
}
```

# *Usb_Host_Poll_Event*

**機　能**　イベントのポーリング

**形　式**　**void Usb_Host_Poll_Event( void);**

**解　説**　非 OS 環境では、イベントの通知をポーリングに設定した場合、ITF-EasyHost を使用している間、十分短い間隔で本 API を呼び続ける必要があります。本 API を呼び出すことで、イベントの発生がチェックされ、イベントが発生した場合イベントコールバックが実行されます。
OS 対応の場合、本 API はイベントポーリングルーチンにより内部的に呼び出されますので、ユーザ側で本 API を呼び出す必要はありません。

**戻　値**　なし

**例**　イベントのポーリングを行います。

```
void main( void )
{
    Usb_Host_Init();
    while(1) {
        :
        Usb_Host_Poll_Event();
    }
}
```

## *Usb_Host_Notify_Timer*

**機　能**　タイマの通知

**形　式**　**void Usb_Host_Notify_Timer( void);**

**解　説**　非 OS 環境では、ユーザ側で NUSB_TIMER_INTERVAL で指定した周期のタイマを用意し、指定した間隔で本 API を呼び出す必要があります。
本 API は他の API と異なり、割り込みから呼び出す必要があります。また、本 API は Usb_Host_Init 及び Usb_Host_Exit の実行中に呼び出す必要があるため、本 API の呼び出しを行うタイマは Usb_Host_Init の呼び出し前に起動され、Usb_Host_Exit の完了後に停止する必要があります。
OS 対応の場合、本 API は周期ハンドラにより内部的に呼び出されますので、ユーザ側で本 API を呼び出す必要はありません。

**戻　値**　なし

**例**　NUSB_TIMER_INTERVAL 周期のタイマを ITF-EasyHost に通知します。

```
void timer_interrupt( void )
{
    :
    Usb_Host_Notify_Timer();
}
```

# *Usb_Kbd_Set_Led*

**機　能**　キーボードの LED 設定

**形　式**　**istatus Usb_Kbd_Set_Led( ihandle handle,**
**int32_t led**
**);**

**handle**　　**: デバイスハンドル**
**led**　　**: LED 設定値**

**解　説**　handle で指定したキーボードデバイスの LED の点灯、消灯を行います。
led には、点灯、または消灯させる LED を設定します。

| **LED 種類** | **データ** |
|---|---|
| Num Lock | NLED_NUM_LOCK(0x01) |
| Caps Lock | NLED_CAPS_LOCK(0x02) |
| Scroll Lock | NLED_SCROLL_LOCK(0x04) |
| Kana | NLED_KANA(0x08) |
| Compose | NLED_COMPOSE(0x10) |

点灯させる LED はビットに”1”をセットし、消灯させる LED のビットには”0”をセットします。本 API は 1 回の呼び出しで全ての LED を設定します。LED の状態はユーザアプリケーションで管理する必要があります。
戻値が NE_PENDING の場合、LED 変更完了イベントにより完了が通知されます。

**戻　値**　**IE_OK**　　**: 正常終了**
**NE_PENDING**　**: 未完了**
**エラーコード**　**: 異常終了**

例　　イベントコールバック内で LED の点灯･消灯を行います

```
void main(void)
{
    Usb_Host_Init(event_callback);
    while(1);
}
void event_callback(ihandle dev, int32_t event, void *param)
{
    istatus r;
    int32_t new_led;
    static int32_t old_led = 0;
    if(event == NC_EVENT_KEY_DOWN){
            switch (*(uint8_t *)param) {
                case 0x53:      /* Num Lock */
                    new_led = (old_led ^ NLED_NUM_LOCK);
                    r = Usb_Kbd_Set_Led(dev, new_led);
                    break;
                case 0x39:      /* Caps Lock */
                    new_led = (old_led ^ NLED_CAPS_LOCK);
                    r = Usb_Kbd_Set_Led(dev, new_led);
                    break;
                case 0x47:      /* Scroll Lock */
                    new_led = (old_led ^ NLED_SCROLL_LOCK);
                    r = Usb_Kbd_Set_Led(dev, new_led);
                    break;
                default:
                    r = 0;
                    break;
            }
            if (r < 0 && r != NE_PENDING ) {
                /* error */
            }
    }
}
```

### 6.10 マスストレージクラス API

## *Scd_Driver*

**機　能**　USB マスストレージデバイス API

**形　式**　**istatus Scd_Driver( SRHEAD *sr);**
**sr　　：ドライバリクエスト**

**解　説**　ファイルシステムを実装している上位モジュールは、この API を通して、マスストレージデバイスと入出力処理を行います。

**戻　値**　**0　　：正常終了**
**エラーコード　　：異常終了**

### 6.11 情報取得関数

## *ITFEasyHost_Version*

**機　能**　製品バージョン番号読み出し

**形　式**　istatus ITFEasyHost_Version( int16_t* major,
int16_t* minor
);

major　：メジャーバージョンを格納する領域
minor　：マイナーバージョンを格納する領域

**解　説**　本ライブラリの製品バージョン番号を読み出します。

**戻　値**　0　　：正常終了
!0　　：異常終了

**例**　　ITF-EasyHost の製品バージョン番号を読み出します。

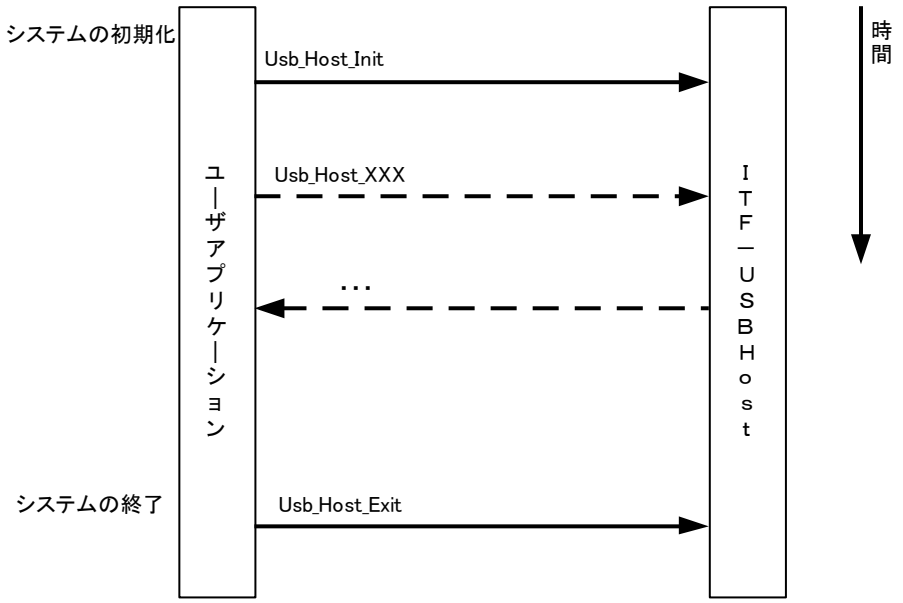
```
void task( void )
{
    istatus r;
    int16_t major, minor;
    :
    r = ITFEasyHost_Version(&major, &minor);
    if (r == 0) {
        /* 読み出し成功 */
    }
    :
}
```

# 7 動作

ITF-EasyHost の主な動作について解説します。

## 7.1 初期化

ITF-EasyHost の初期化／終了時の流れは次の通りです。




（図　初期化／終了時の流れ）

１．システムの初期化(Usb_Host_Init)

ITF-EasyHost を使用するためには初期化を行います。Usb_Host_Init は ITF-EasyHost が提供する API です。初期化が正常に行われることにより ITF-EasyHost の機能を使用できるようになります。ITF-EasyHost は初期化時に全てのリソースを確保します。

２．システムの終了(Usb_Host_Exit)

初期化された ITF-EasyHost を停止するためには終了処理を行います。Usb_Host_Exit は ITF-EasyHost が提供する API です。ITF-EasyHost は終了時に全てのリソースを開放します。終了処理後は ITF-EasyHost が使用できなくなります。

ITF-EasyHost の終了処理では、全ての処理を強制的に停止させています。保留中の処理がある場合でも、終了処理後はユーザアプリケーションに何も通知されません。ユーザアプリケーションは、ITF-EasyHost の終了と同時に全ての ITF-EasyHost に関連する処理をリセット、または初期化する必要があります。

## 7.2 デバイスの検出

ITF-EasyHost はデバイスのホットプラグに対応しています。ITF-EasyHost の起動後はデバイスを任意の時間に接続、または切断が可能になります。ITF-EasyHost は常にデバイスの接続／切断を監視しています。

### 7.2.1 デバイスの接続検出

ITF-EasyHost は、デバイスの接続を検出するとデバイスを使用するために必要な処理を行い、その後ユーザアプリケーションへデバイスの接続を通知します。ここでいう接続とは次の状態のことを指します。

- バス電源のデバイスが USB ポートに接続された。
- 自己電源のデバイスの電源が投入されている状態で USB ポートに接続された。
- 自己電源のデバイスが USB ポートに挿されている状態で、電源が投入された。

※ バス電源のデバイスとは、電源が USB バスから供給されるデバイスのことを指します。
※ 自己電源のデバイスとは、電源を自身で確保しているデバイスのことを指します。

### 7.2.2 デバイスの切断検出

ITF-EasyHost は、デバイスの切断を検出するとデバイスを削除し、ユーザアプリケーションへデバイスの切断を通知します。ここでいう切断とは次の状態のことを指します。

- デバイスが USB ポートから抜かれた。
- 自己電源のデバイスの電源が切断された。

上記はデバイスが接続されている状態からの動作です。

## 7.3 デバイスと通信

ITF-EasyHost は、ユーザアプリケーションに USB デバイスと通信を行うための機能を提供します。USB には複数の転送方式がありますが、ITF-EasyHost では次の転送方式をサポートしています。

（表　サポートする転送方式）

| 転送方式 | 説明 |
|---|---|
| コントロール転送 | 送受信可能な制御用転送。 |
| バルク転送 | データが保証された比較的高速な転送。 |
| インタラプト転送 | 比較的小データ量の定周期の転送。 |
| アイソクロナス転送 | 転送時間が保証された転送。 |

（表　コントローラ別の転送方式の対応状況）

| | コントロール | バルク | インタラプト | アイソクロナス |
|---|---|---|---|---|
| MB90330<br>MB91660<br>MB9BF506 | ○ | ○ | ○ | × |
| M66596 | ○ | ○ | ○ | ○ |

また、次のスピードのデバイスをサポートしています。

- Full-Speed（最大 12Mbps）
- Low-Speed（最大 1.5Mbps）
- High-Speed (最大 480Mbps)

※（）内は理論上の最大転送レートです。その速度を保証するものではありません。

（表　コントローラ別の転送方式の対応状況）

| | Low-Speed | Full-Speed | High-Speed |
|---|---|---|---|
| MB90330 | ○ | ○ | × |
| MB91660 | × | ○ | × |
| MB9BF506 | ○ | ○ | × |
| M66596 | × | ○ | ○ |

### 7.3.1 非同期転送の使用方法

非同期転送の API にはパラメータに非同期情報（struct io_overlapped_t）を設定します。ユーザアプリケーションは API の呼び出しの前にこの非同期情報パラメータを初期化する必要があります。非同期情報の初期化には初期化用のマクロを使用します。

非同期情報の初期化用マクロの形式は次の通りです。

# FILL_OVERLAPPED

**機　能**　非同期情報構造体を初期化する

**形　式**　**void FILL_OVERLAPPED(struct io_overlapped_t *pov,**
**int32_t code,**
**int16_t classtype,**
**void *prm**
**);**

| | |
|---|---|
| ***pov** | **：初期化する非同期情報構造体のアドレス** |
| **code** | **：コールバックルーチンに渡すコード** |
| **classtype** | **：コールバックルーチンのクラス種別** |
| ***prm** | **：コールバックルーチンに渡すパラメータ** |

**解　説**　pov が示す非同期情報構造体をパラメータに従って初期化します。
code はコールバックルーチンに渡されるコードです。ユーザが独自に使用できます。
classtype はコールバックルーチンのクラスを指定します。ユーザがクラスドライバを介さずに API を呼び出す場合、0（NCLASS_USB_DEVICE）を設定します。
prm はコールバック時の void *型のパラメータになります。このパラメータはユーザが独自に使用できます。

**戻　値**　なし

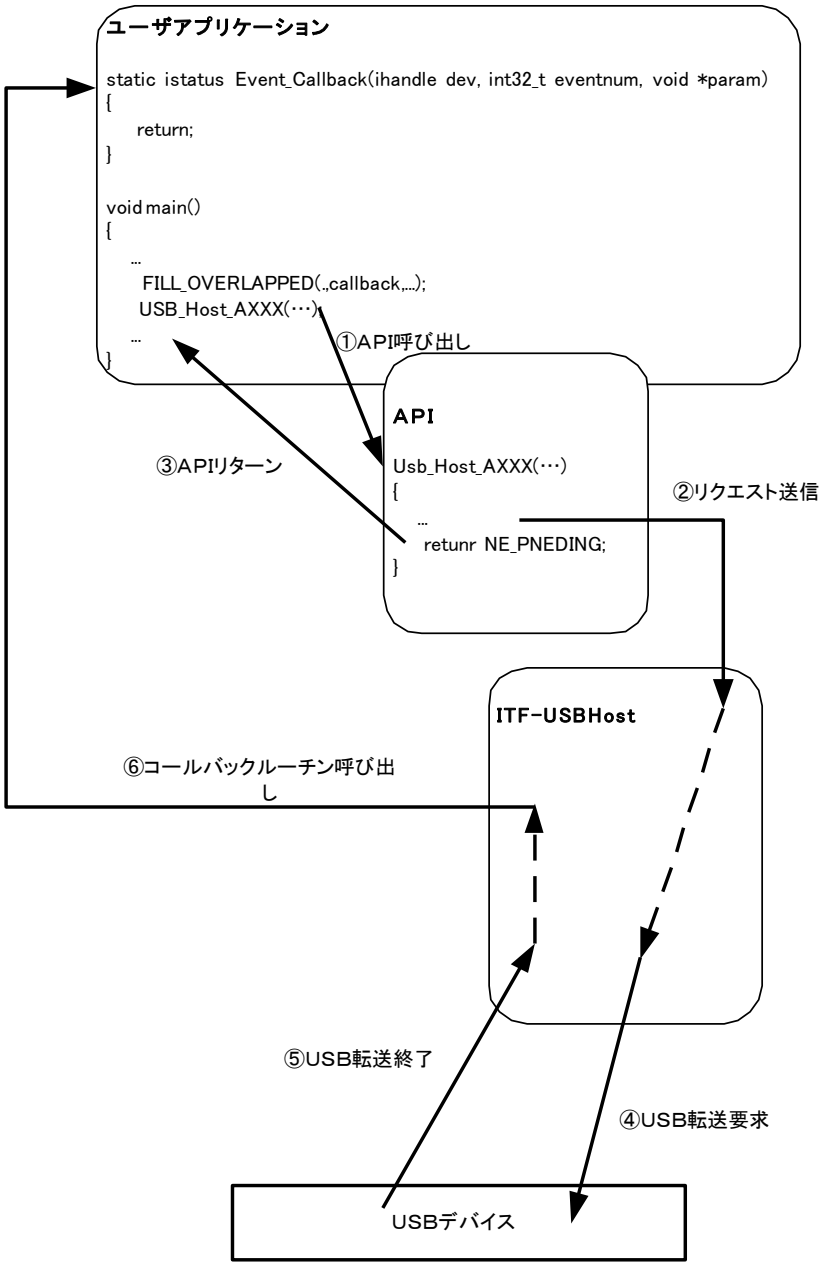
初期化した非同期情報構造体をパラメータに指定して API を呼び出すことにより非同期転送が開始されます。

非同期転送 API は即時に呼び出し側に返ってきます。非同期転送 API の正常終了時の戻り値は NE_PENDING です。異常終了時の戻り値はエラーコードとなります。NE_PENDING は処理が保留されたことを表します。

保留された処理が終了したときは io_overlapped_t 構造体の classtype で指定したクラスのイベントコールバックルーチンが呼び出されます。

### 7.3.2 非同期転送の処理の流れ

非同期転送の API を使用した時の流れは次の通りです。




（図　非同期転送の処理の流れ）

API の使用例については、各 API の使用例を参照して下さい。

### 7.4 デバイスの電源制御

ITF-EasyHost はデバイスの電源制御を行っています。ユーザが任意に次の電源制御が行えます。

（表　電源制御一覧）

| 対象 | 機能 |
|---|---|
| デバイス | サスペンド |
| | レジューム |
| | リセット |

#### 7.4.1 サスペンド

サスペンドではデバイスをサスペンドさせます。サスペンドしたデバイスは低消費電力で動作します。

#### 7.4.2 レジューム

サスペンドしたデバイスをレジュームし、デバイスが使用できる状態にします。処理の流れはサスペンドの場合と同等で、ユーザアプリケーションからレジュームの要求が ITF-EasyHost へ渡され、ポートのサスペンドが解除されます。

#### 7.4.3 リセット

リセットではデバイスをリセットします。リセットされたデバイスは初期状態に戻ります。ユーザアプリケーションからは、デバイスがどうしても復帰できない場合に使用すべきで、むやみにリセットするものではありません。

# 8 注意･制限事項

## 8.1 コンフィグレーション

複数の構成（コンフィグレーション）を持つデバイスが接続された場合、ITF-EasyHost は初めに発見されたコンフィグレーションを設定します。コンフィグレーションを選択する機能は実装されていません。

## 8.2 サスペンド

デバイスをサスペンドした後に、そのデバイスをポートから切断しないで下さい。サスペンド後から、ウェイクアップするまでの間は、サスペンドさせたときの状態を維持するようにして下さい。

## 8.3 ファイルシステムのポーティング

ITF-EasyHost(ストレージオプション付き)をご購入頂いたお客様で、ファイルシステム(当社ファイルファイルシステム ITF-FILE 以外)のポーティングを行うご予定の場合、パッケージをそのままコンパイルしてもコンパイルエラーが発生します。

具体的には、当社ファイルシステム ITF-FILE で用意されている、「DriverIF.h」、「itffileerror.h」が同封されていないためです。

これは、ファイルシステムに依存した箇所となり、お客様自身でこのファイルの内容に相当する箇所を修正して頂く必要があり、それを明示的にすることを目的として、このようなパッケージ形態を取っております。

もし、このファイルを参考にされたいと言うのであれば、サポート窓口までご連絡頂ければ、サンプルとしてご提供させて頂きます。

## 8.4 ITF-EasyHost 付属の ITF-Lib について

ITF-EasyHost(ストレージオプション付き)及び ITF-FILE をご購入いただいた場合、ITF-Lib は ITF-EasyHost 及び ITF-FILE のそれぞれに付属しますが、ITF-EasyHost 用のプロジェクトファイルは ITF-EasyHost に付属する ITF-Lib のみに含まれます。また、ITF-EasyHost 付属の ITF-Lib では、itfconfig.h の一部が ITF-EasyHost 向けに修正されています。

## 8.5 SOFTUNE 使用時の注意

MB90330 向けに SOFTUNE を用いてコンパイルする場合、ITF-Lib において itferror.c のコンパイル時に“W1352C”のワーニングが発生しますが、動作上の問題はございません。

MB91660 向けに SOFTUNE を用いてコンパイルする場合、ITF-Lib において itfos.c のコンパイル時に“W1174B”のワーニングが発生しますが、動作上の問題はございません。

# 9 一覧

## 9.1 エラーコード

| シンボル名 | エラー番号 | 内容 |
|---|---|---|
| **ITF-EasyHost の標準エラー** | | |
| NE_XXX | -20001 | その他エラー |
| NE_PENDING | -20002 | ペンディング |
| NE_CANCELED | -20003 | キャンセル |
| NE_INVALIED_HANDLE | -20004 | ハンドル異常 |
| NE_TIMEOUT | -20005 | タイムアウト |
| NE_INVALIED_PARAMETER | -20006 | パラメータ異常 |
| NE_NO_RESOURCE | -20007 | リソースがない |
| NE_NO_SECURE_RESOURCE | -20008 | そのリソースは未登録 |
| NE_NO_DEVICE_HANDLE | -20009 | デバイスハンドルがない |
| NE_INVALIED_IO | -20010 | 不正Ｉ／Ｏ |
| **ITF-EasyHost のシステムエラー** | | |
| NE_INVALIED_MAJOR_CODE | -20101 | メジャーコード異常 |
| NE_INVALIED_MINOR_CODE | -20102 | マイナーコード異常 |
| NE_INVALIED_DEVICE | -20103 | デバイス異常 |
| NE_INVALIED_DEVICE_HANDLE | -20104 | デバイスハンドル異常 |
| NE_INVALIED_DEVICE_NAME | -20105 | デバイス名が異常 |
| NE_INVALIED_DEVICE_INDEX | -20106 | デバイスインデックス異常 |
| NE_NOT_ALLOCATED_DEVICE_MEMORY | -20107 | デバイスメモリ確保失敗 |
| NE_NOT_DEVICE_ENABLE | -20108 | デバイスは有効でない |
| NE_NOT_DEVICE_DISABLE | -20109 | デバイスは無効でない |
| NE_NOT_BUS_SUPPORTED | -20110 | バスをサポートしていない |
| NE_NOT_SUPPORT_OPERATION | -20111 | オペレーション未対応 |
| NE_IRP_NO_QUEUE | -20112 | ＩＲＰがキューにない |
| NE_IRP_NOT_COMPLETE | -20113 | ＩＲＰが完了できない |
| NE_IRP_NOT_CANCELL | -20114 | ＩＲＰがキャンセルできない |
| NE_IRP_ALREADY_COMPLETE | -20115 | ＩＲＰは既に完了されています |
| NE_NO_IRP_RESOURCE | -20116 | ＩＲＰ領域がない |
| NE_NO_BUS | -20117 | バスがありません |
| NE_NO_BUS_DEVICE | -20118 | バスデバイスがありません |
| NE_NO_DRIVER | -20119 | ドライバがない |
| NE_SYSTEM_BUSY | -20120 | システムビジー |
| NE_SYSTEM_ENABLE | -20121 | システムビジー |
| NE_SYSTEM_DISABLE | -20122 | システムビジー |
| NE_SHORT_LENGTH_RETUEN | -20123 | 要求サイズより小さいサイズが返りました |

| | | |
|---|---|---|
| NE_FAILED_NMB_INIT | -20124 | nmb の初期化に失敗 |
| **ITF-EasyHost のデバイスエラー** | | |
| NE_DEVICE_BUSY | -20201 | デバイスはビジーです |
| NE_DEVICE_ABORT | -20202 | デバイスは停止しています |
| NE_DEVICE_SUSPEND | -20203 | デバイスはサスペンドしています |
| NE_DEVICE_ALREADY_SUSPENDED | -20204 | デバイスは既にサスペンドしています |
| NE_DEVICE_ALREADY_EXECTION | -20205 | デバイスは既に起動しています |
| NE_DEVICE_NO_BUS | -20206 | デバイスに対応するバスがありません |
| NE_BUS_SUSPENDED | -20207 | バスがサスペンドされています |
| NE_DEVICE_STATUS | -20208 | デバイスステータス異常 |
| **ITF-EasyHost のドライバエラー** | | |
| NE_NOT_OPENED | -20301 | オープンできません |
| NE_NO_DATA | -20302 | データがありません |
| NE_NOT_SUPPORTED | -20303 | サポートしていません |
| NE_FIFO_OVER | -20305 | ＦＩＦＯオーバー |
| NE_NOT_CMD_SUPPORTED | -20306 | コマンド未対応 |
| NE_INVALIED_OPEN_NAME | -20307 | オープン名異常 |
| NE_NO_MEDIUM | -20308 | メディアがありません |
| **ITF-EasyHost のＨｏｓｔドライバエラー** | | |
| NE_INVALIED_ENDPOINT | -20401 | 異常エンドポイント番号 |
| NE_INVALIED_CONFIGURATION_VALUE | -20402 | コンフィグレーション値が異常 |
| NE_INVLAIED_ALTERNATE_SETTING | -20403 | 代替設定値が異常 |
| NE_PIPE_NOT_OPENED | -20404 | パイプがオープンできません |
| NE_PIPE_BUSY | -20405 | パイプがビジー |
| NE_PIPE_ALREADY_TRANSFER | -20406 | パイプで既に転送されています |
| NE_PIPE_INVALIED_TRANSFER_TYPE | -20407 | 転送種別が異常 |
| NE_NO_DESCRIPTOR | -20408 | ディスクリプタがない |
| NE_NO_CONFIGURATION | -20409 | コンフィグレーションがない |
| NE_NO_ENDPOINT | -20410 | エンドポイントがない |
| NE_NO_URB_RESOURCE | -20411 | ＵＲＢ領域がない |
| NE_NO_MEMORY_RESOURCE | -20412 | 転送用メモリ領域がない |
| NE_NO_PIPE_RESOURCE | -20413 | パイプ領域がない |
| NE_NO_DEVICE_RESOURCE | -20414 | デバイス領域がない |
| NE_NO_DRIVER_RESOURCE | -20415 | ドライバ領域がない |
| NE_NO_EXTENSION_RESOURCE | -20416 | デバイス拡張領域がない |
| NE_NO_HCI | -20417 | コントローラがありません |
| NE_NO_DEVICE_ADDRESS | -20418 | デバイスアドレスがありません |
| NE_NO_INTERFACE | -20419 | インターフェイスがありません |
| NE_NO_PORT | -20420 | ポートがない |
| NE_NO_PIPE | -20421 | パイプがない |

| | | |
|---|---|---|
| NE_NO_TX_PIPE | -20422 | 送信用パイプがない |
| NE_NO_RX_PIPE | -20423 | 受信用パイプがない |
| NE_NO_BULK_PIPE | -20424 | バルクパイプがない |
| NE_NO_BULK_TX_PIPE | -20425 | 送信用バルクパイプがない |
| NE_NO_BULK_RX_PIPE | -20426 | 受信用バルクパイプがない |
| NE_NO_INTERRUPT_PIPE | -20427 | インタラプトパイプがない |
| NE_NO_INTERRUPT_TX_PIPE | -20428 | 送信用インタラプトパイプがない |
| NE_NO_INTERRUPT_RX_PIPE | -20429 | 受信用インタラプトパイプがない |
| NE_NOT_COTROL_PIPE | -20430 | コントロールパイプでない |
| NE_NOT_BULK_PIPE | -20431 | バルクパイプでない |
| NE_NOT_INTRRUPT_PIPE | -20432 | インタラプトパイプでない |
| NE_MPS_ZERO | -20433 | 最大パケットサイズが０ |
| NE_ACTUAL_LENGTH_SHORT | -20434 | 転送サイズが小さい |
| NE_CONFIGURATION_BUFFER_OVER | -20435 | 実際のサイズがコンフィグレーションバッファより大きい |
| NE_FAILED_ENUMERATION | -20436 | エニュメレーション失敗 |
| NE_NOT_ENOUGH_BANDWIDTH | -20437 | 大域幅が不足 |
| | | |
| NE_URB | -20500 | ＵＲＢエラー（-0 ～ -14） |
| NE_URB_NOT_DONE | -20500 | ＵＲＢは完了していない |
| NE_URB_NORMAL_COMPLETION | -20501 | ＵＲＢは通常完了した |
| NE_URB_CANCELING | -20502 | ＵＲＢはキャンセル中 |
| NE_URB_CANCELED | -20503 | ＵＲＢはキャンセルされた |
| NE_URB_ERROR_STALL | -20504 | エンドポイントがストールした |
| NE_URB_ERROR_DEVICE_NOT_RESPONDING | -20505 | デバイスから応答がない |
| NE_URB_ERROR_UNNKOWN | -20506 | 不明なエラーが発生した |
| NE_URB_ERROR_CRC | -20507 | ＣＲＣエラー |
| NE_URB_ERROR_BIT_STUFFING | -20508 | ビットスタッフィング違反 |
| NE_URB_ERROR_DATA_TOGGLE_MISMATCH | -20509 | データトグルが不一致 |
| NE_URB_ERROR_PID_CHECK_FAILURE | -20510 | ＰＩＤチェックに失敗 |
| NE_URB_ERROR_UNEXPECTED_PID | -20511 | 予期されないＰＩＤ |
| NE_URB_ERROR_DATA_RUN | -20512 | データがバッファを越えた |
| NE_URB_ERROR_BUFFER_OVERRUN | -20513 | コントローラがデータを書き込むより速くデータを受信した |
| NE_URB_ERROR_BUFFER_UNDERRUN | -20514 | コントローラがデータを引き出すのが間に合わない |

### 9.2 API

| 関数名 | 内容 |
|---|---|
| **初期化関数** | |
| Usb_Host_Init | ITF-EasyHost の初期化、資源の確保 |
| Usb_Host_Exit | ITF-EasyHost の終了、資源の解放 |
| **タイマ通知関数** | |
| Usb_Host_Notify_Timer | タイマの通知（非 OS 環境のみ） |
| **イベント取得関数** | |
| Usb_Host_Poll_Event | イベントのポーリング（非 OS 環境のみ） |
| **パイプ通信関数** | |
| Usb_Host_Open | 通信パイプをオープンする |
| Usb_Host_Close | 通信パイプをクローズする |
| Usb_Host_Ep_Write | OUT パイプ転送 |
| Usb_Host_Ep_Read | IN パイプ転送 |
| Usb_Host_Ep_Cancel | パイプ転送の中断 |
| Usb_Host_Control | コントロール転送 |
| Usb_Host_Ep_AWrite | OUT パイプ非同期転送 |
| Usb_Host_Ep_ARead | IN パイプ非同期転送 |
| Usb_Host_AControl | コントロール非同期転送 |
| **ディスクリプタ取得関数** | |
| Usb_Host_Get_Device_Descriptor | デバイスディスクリプタ取得 |
| Usb_Host_Get_Configration_Descriptor | コンフィグレーションディスクリプタ取得 |
| Usb_Host_Get_Interface_Descriptor | インターフェイスディスクリプタ取得 |
| Usb_Host_Get_EndPoint_Descriptor | エンドポイントディスクリプタ取得(エンドポイントアドレス指定) |
| Usb_Host_Get_EndPoint_Descriptor_Count | エンドポイントディスクリプタ取得（序数指定） |
| Usb_Host_Get_String_Descriptor | ストリングディスクリプタ取得 |
| **電源管理関数** | |
| Usb_Host_Suspend | バスをサスペンドする |
| Usb_Host_Wakeup | バスをウェイクアップする |
| Usb_Host_Remove | デバイスの切断 |
| Usb_Host_Reset | バスリセット |
| **デバイス情報取得関数** | |
| Usb_Host_Get_Device_Count | デバイス数の取得 |
| Usb_Host_Get_Device_Information | デバイス情報の取得（ハンドル指定） |
| Usb_Host_Search_Device | デバイス情報の取得（序数指定） |
| **デバイスリクエスト発行関数** | |
| Usb_Host_Set_Interface | 代替設定値の設定 |
| Usb_Host_Get_Interface | 代替設定値の取得 |
| Usb_Host_Get_Status | デバイス・インターフェイス・エンドポイントの状態の取得 |

| | |
|---|---|
| **パイプ制御関数** | |
| Usb_Host_Ep_Reset | パイプのリセット |
| Usb_Host_Ep_Abort | パイプの停止 |
| **HID クラス API** | |
| Usb_Kbd_Set_Led | キーボードの LED 制御処理 |
| **マスストレージクラス API** | |
| Scd_Driver | USB マスストレージデバイス API |
| **情報取得関数** | |
| ITFEasyHost_Version | バージョン情報読み出し |

## 9.3 構造体

公開構造体一覧

```
Ｉ／Ｏコントロール（コントロール転送）用構造体
typedef struct _NUSB_CONTROL_T
{
    struct {
            uint8_t req_type;   bmRequestType
            uint8_t req;        bRequest
            uint8_t value_l;    wValue 下位
            uint8_t value_h;    wValue 上位
            uint8_t index_l;    wIndex 下位
            uint8_t index_h;    wIndex 上位
            uint8_t length_l;   転送バイト数 下位
            uint8_t length_h;   転送バイト数 上位
    } setup;
    void *buf;          転送バッファ
} NUSB_CONTROL_T, *PNUSB_CONTROL_T;


デバイスディスクリプタ
typedef struct DEVICE_DESCRIPTOR_T {
    uint8_t bLength;              ディスクリプタのサイズ
    uint8_t bDescriptorType;      ディスクリプタのタイプ
    uint8_t bcdUSB_l;             BCD 表現による USB リリース番号 下位
    uint8_t bcdUSB_h;             BCD 表現による USB リリース番号 上位
    uint8_t bDeviceClass;         クラスコード
    uint8_t bDeviceSubClass;      サブクラスコード
    uint8_t bDeviceProtocol;      プロトコルコード
    uint8_t bMaxPacketSize0;      ＥＰ０の最大パケットサイズ
    uint8_t idVender_l;           ベンダＩＤ(USB IF が割り当てる) 下位
    uint8_t idVender_h;           ベンダＩＤ(USB IF が割り当てる) 上位
    uint8_t idProduct_l;          ProductＩＤ(Vender が割り当てる) 下位
    uint8_t idProduct_h;          ProductＩＤ(Vender が割り当てる) 上位
    uint8_t bcdDevice_l;          BCD 表現による DEVICE リリース番号 下位
    uint8_t bcdDevice_h;          BCD 表現による DEVICE リリース番号 上位
    uint8_t iManufacturer;        String_Descript 用 Index(製造者)
    uint8_t iProduct;             String_Descript 用 Index(製品)
    uint8_t iSerialNumber;        String_Descript 用 Index(番号)
    uint8_t bNumConfigurations;   構成可能な数
} DEVICE_DESC, *PDEVICE_DESC ;
```

エンドポイントディスクリプタ

```
typedef struct ENDPOINT_DESCRIPTOR_T {
    uint8_t bLength;            ディスクリプタのサイズ
    uint8_t bDescriptorType;    ディスクリプタのタイプ
    uint8_t bEndpointAddress;   エンドポイントアドレス
    uint8_t bmAttributes;       属性(BIT1-0:転送タイプ)
    uint8_t wMaxPacketSize_l;   最大パケットサイズ 下位
    uint8_t wMaxPacketSize_h;   最大パケットサイズ 上位
    uint8_t bInterval;          ポーリング間隔
} ENDPOINT_DESC, *PENDPOINT_DESC;
```

インターフェースディスクリプタ

```
typedef struct INTERFACE_DESCRIPTOR_T {
    uint8_t bLength;              ディスクリプタのサイズ
    uint8_t bDescriptorType;      ディスクリプタのタイプ
    uint8_t bInterfaceNumber;     この INTERFACE を表す Index 番号
    uint8_t bAlternateSetting;    代替設定用の引数値
    uint8_t bNumEndpoints;        INTERFACE のＥＰ数(０は除く)
    uint8_t bInterfaceClass;      クラスコード
    uint8_t bInterfaceSubClass;   サブクラスコード
    uint8_t bInterfaceProtocol;   プロトコルコード
    uint8_t iInterface;           String_Descript 用の Index
} INTERFACE_DESC, *PINTERFACE_DESC;
```

構成ディスクリプタ

```
typedef struct CONFIGURATION_DESCRIPTOR_T {
    uint8_t bLength;               ディスクリプタのサイズ
    uint8_t bDescriptorType;       ディスクリプタのタイプ
    uint8_t wTotalLength_l;        構成全体(自身含む)の長さ 下位
    uint8_t wTotalLength_h;        構成全体(自身含む)の長さ 上位
    uint8_t bNumInterface;         構成の持つインターフェース数
    uint8_t bConfigurationValue;   この構成を選択するための引数
    uint8_t iConfiguration;        String_Descript 用の Index
    uint8_t bmAttributes;          構成の特性
    uint8_t MaxPower;              最大バス消費量(２ｍＡ単位)
} CONFIG_DESC, *PCONFIG_DESC;
```

デバイス情報構造体

```
typedef struct _NUSB_DEVICE_INFORMATION {
    uint16_t idVender;      ベンダＩＤ
    uint16_t idProduct;     ProductＩＤ
    uint16_t bcdDevice;     デバイスリリース番号
    uint8_t dev_addr;       デバイスアドレス
    uint8_t ifnum;          インターフェイス番号
    uint8_t altnum;         代替設定の数
    uint8_t altact;         現在の代替設定値
    uint8_t epnum;          現在の代替設定値のＥＰ数
    uint8_t epmax;          エンドポイントの数
    uint16_t dev_idx;       デバイスインデックス
    uint16_t class_idx;     クラスインデックス
    uint16_t class_type;    クラスタイプ
    int8_t dev_name[32];    デバイス名
} NUSB_DEVICE_INFORMATION, *PNUSB_DEVICE_INFORMATION;
```

ドライバ情報構造体

```
typedef struct _NDRIVER_INFORMATION_T {
    int8_t *name;           名称
    int8_t *version;        バージョン
    int8_t *revision;       リビジョン
} NDRIVER_INFORMATION_T, *PNDRIVER_INFORMATION_T;
```

非同期情報構造体

```
struct io_overlapped_t {
    int32_t code;                パラメータ１
    int16_t classtype            クラスコード
    void *prm;                   パラメータ２
};
```

非同期結果構造体

```
struct io_overlapped_result_t {
    istatus status;              ステータス
    uint32_t transferred;        処理したバイト数
    int32_t code;                パラメータ１
    void *prm;                   パラメータ２
};
```

### 9.4 イベント

| イベントコード | 説明 |
|---|---|
| NC_EVENT_CONNECT | デバイス接続 |
| NC_EVENT_DISCONNECT | デバイス切断 |
| NC_EVENT_WAKEUP_REQUEST | ウェイクアップ要求 |
| NC_EVENT_RESET_COMPLETE | リセット完了 |
| NC_EVENT_SUSPEND_COMPLETE | サスペンド完了 |
| NC_EVENT_REQUEST_COMPLETE | リクエスト完了 |
| NC_EVENT_KEY_DOWN | キーダウン |
| NC_EVENT_KEY_UP | キーアップ |
| NC_EVENT_CLT_KEY_DOWN | コントロールキーダウン |
| NC_EVENT_CLT_KEY_UP | コントロールキーダウン |
| NC_EVENT_BTN_DOWN | ボタンダウン |
| NC_EVENT_BTN_UP | ボタンアップ |
| NC_EVENT_POS_X | Ｘ方向移動 |
| NC_EVENT_POS_Y | Ｙ方向移動 |
| NC_EVENT_POS_WHEEL | ホイール移動 |
| NC_EVENT_KEY_LED | ＬＥＤ変更完了 |
| NC_EVENT_FATAL_ERROR | 致命的なエラー |

## ご注意



■お問い合わせ

**インターフェイス株式会社**

〒190-0022　東京都立川市錦町２－２－１１

TEL　042-528-8651　　FAX　042-528-8678

E-mail　sales@itf.co.jp