マニュアル

Touch Panel Computer Stand Type用

『Windows Embedded Standard 2009』

について

目　次

はじめに

第１章　概要

第２章　システムの操作

第３章　Touch Panel Computerについて

第４章　EWF API

第５章　組込みシステム機能ドライバ

第６章　システムリカバリ

付録

はじめに

この度は、アルゴシステム製品をお買い上げいただきありがとうございます。

弊社製品を安全かつ正しく使用していただく為に、お使いになる前に本書を十分に理解していただくようお願い申し上げます。

１） お願いと注意

本書では、Touch Panel Computer Stand Type（以降「AS シリーズ」と表記します）用 Windows Embedded Standard 2009 について説明します。

Windows Embedded Standard 2009 は Windows XP Embedded の後継製品で、Windows XP SP3 をベースとした組込み用 OS です。本書では、AS シリーズ用の Windows Embedded Standard 2009 に特有の仕様、操作について説明します。一般的な Windows の仕様、操作については省略させていただきます。

Windows Embedded Standard 2009 は Windows XP SP3 をベースとした OS ですが、組込み用 OS のため通常の PC 用 Windows とは動作が異なる可能性があります。詳しくは、「付録 マイクロソフト製品の組込み用 OS について」を参照してください。

本書は、アプリケーション開発、専用ドライバ仕様などの専門的な内容を含んでいます。これらの内容は、Windows アプリケーション開発、デバイス制御プログラミングに関する技術を必要とします。ご注意ください。

２） 対応機種について

本書では、AS シリーズについて説明しています。その他の機種については、それぞれの機種に対応するマニュアルを用意しております。機種に対応したマニュアルを参照してご使用ください。

第１章　概要

本章では、AS シリーズ用 Windows Embedded Standard 2009 の概要について説明します。

１－１　機能と特長

１－１－１　ASシリーズ用Windows Embedded Standard 2009 とは

Windows Embedded Standard 2009 は、Windows XP Embedded の後継製品です。Windows XP SP3 をベースとしており、Windows オペレーティングシステムのデスクトップ技術を組込みデバイスで使用することが可能となります。Windows Embedde Standard 2009 はコンポーネント形式で提供されます。必要な機能を選択し、組込みデバイスに合わせた Windows オペレーティングシステムを構築することができます。

AS シリーズ用 Windows Embedded Standard 2009 は、AS シリーズ用にカスタマイズされた Windows Embedded Standard 2009 です。AS シリーズ用に選定したコンポーネント、オンボード搭載デバイス用のドライバ及び設定ツールで構成されています。

１－１－２　機能と特長

AS シリーズ用 Windows Embedded Standard 2009 は、オンボードデバイスのサポートと Windows アプリケーション互換性のためのマクロコンポーネントを基本として構築されています。このため Windows XP で動作しているアプリケーションをほぼそのまま動作させることが可能となっています。また、Enhanced Write Filter のストレージ保護機能、IIS などのネットワークサーバー機能を追加することにより、組込みシステムとしてより堅牢で柔軟なシステムを構築できるようになっています。

● 互換性確保のためのマクロコンポーネント
Class Installers / Hardware Compatibility
Codepage Application Compatibility
Enterprise Features
Fonts Application Compatibility
Multimedia Application Compatibility
Networking Application Compatibility
Shell Application Compatibility
Test Application Compatibility
Windows Application Compatibility
Windows Management Instrumentation Technologies

表 1-1-2-1 に AS シリーズ用 Windows Embedded Standard 2009 に搭載されている主な機能を示します。

表 1-1-2-1. ASシリーズ用Windows Embedded Standard 2009 の主な機能

機能	内容
Windows XP Service Pack 3	Windwos XP Service Pack 3 がベースとなっています。 Windows XP Service Pack 3 には、以前にリリースされた修正プログラムやセキュリティ更新プログラム、および新機能が含まれています。
Internet Explorer 7	Internet Explorer 7 を搭載しています。 Web コンテンツの閲覧、Web アプリケーションへのアクセスが可能となります。
Windows Media Player 11	Windows Media Player 11 を搭載しています。 画像の閲覧、デジタルミュージック、動画の再生が可能です。

Microsoft .NET Framework 2.0	Microsoft .NET Framework 2.0 を搭載しています。 .NET Framework 2.0 の動作環境を提供します。.NET Framework 2.0 で動作するマネージドコードアプリケーションを動作させることができます。 **※　他のバージョンの Microsoft .NET Framework には対応していません。**
Internet Information Services (IIS)	Internet Information Services を搭載しています。 以下のサービスに対応しています。 ・ FTP サービス ・ Web サービス **※　出荷状態では、サービスは無効となっています。**
Telnet Service	Telnet クライアント、Telnet サーバを搭載しています。 **※　出荷状態では、サービスは無効となっています。**
Remote Desktop	リモートデスクトッププロトコルに対応しています。 ターミナルサービス、リモートデスクトップを使用することができます。
Wi-Fi Protection Access 2 (WPA2)	Wi-Fi Protection Access 2 に対応しています。 ワイヤレスセキュリティに WPA2 認証サポートが追加されます。 WPA2 を使用したワイヤレスネットワークに接続が可能となります。
Kernel Mode Driver Framework (KMDF)	Kernel Mode Driver Framework に対応しています。 KMDF を利用して開発されたカーネルモードデバイスドライバを動作させることができます。
Enhanced Write Filter (EWF)	Enhanced Write Filter を搭載しています。 EWF を用いるとドライブを書込み禁止状態にして OS を動作させることができます。EWF はドライブ毎に設定することができます。 書込み禁止状態で起動した OS は、シャットダウン処理なしで電源断することができます。(強制電源断)
Multi Language Suport	マルチ言語対応。日本語と英語に対応しています。 設定によって切換えることができます。
Windows 標準インターフェース対応機能	Windows 標準インターフェースを使用して使用できる機能、デバイスを搭載しています。 ・ グラフィック ・ タッチパネル ・ シリアルポート ・ 有線 LAN (HUB 機能内蔵) ・ 無線 LAN ・ サウンド ・ USB ポート ・ SD/SDHC カード ・ FeliCa リーダ・ライタ (オプション) ・ Bluetooth (オプション)
組込みシステム機能	組込みシステム向けの独自機能が搭載されています。 ・ タイマ割込み機能 ・ ブザー ・ 汎用入出力 ・ LCD バックライト制御 ・ RAS 機能 ・ ソフトウェア・ウォッチドッグタイマ ・ ASD Config (機能設定用コンパネアプリ)

１－２　　システム構成

１－２－１　　ドライブ構成

OS を格納するメインストレージは 2GByte の内蔵 SSD です。内蔵 SSD は、C ドライブ、D ドライブの 2 つのドライブに分割されています。C ドライブはシステムドライブとして OS 本体を格納しています。D ドライブは、ユーザードライブとしてユーザーが自由に使用できるドライブとなっています。ドライブの構成を表 1-2-1-1 に示します。

表 1-2-1-1. ASシリーズ 内蔵SSDドライブ構成

ドライブ	容量	空き容量	内容
C	1.52 GByte	約 540 MByte	システムドライブ オペレーティングシステム本体を格納しています。 **※　このドライブには、NTFS ドライブ圧縮が適用されています。**
D	348 MByte	約 344 MByte	ユーザードライブ ユーザーデータの格納など、ユーザーで自由に使用することができます。

１－２－２　　フォルダ/ファイル構成

Windows Embedded Standard 2009 は Windows XP SP3 がベースとなっています。システムドライブのフォルダ、ファイル構成は Windows XP SP3 に準拠したものです。

ドライブトップに存在するフォルダ、ファイルの構成を表 1-2-2-1 に示します。

表 1-2-2-1. ASシリーズ Windows Embedded Standard 2009 フォルダ/ファイル構成

ドライブ	フォルダ/ファイル *
C	<Documents and Settings> <inetpub> <Program Files> <Windows> fbwf.cfg HORM.DAT NTDETECT.COM ntldr WERUNTIME.INI
D	フォルダ/ファイルなし

※　フォルダは<>で表記しています。システム属性、隠し属性のフォルダ/ファイルは表記していません。

１－２－３　ユーザーアカウント

Windows Embedded Standard 2009 は、ログイン可能なユーザーが１つ必要です。初期状態ではログイン可能なユーザーアカウントは Administrator ユーザーのみとなっています。初期状態での Administrator ユーザーの状態を表 1-2-3-1 に示します。

パスワードの変更、別のユーザーアカウントが必要な場合は、OS 起動後に設定するようにしてください。

表 1-2-3-1. Administratorユーザー

ユーザー名	パスワード	グループ	説明
Administrator	Administrator	Administrators	完全な管理者権限を持つビルトインのユーザーアカウントです。

１－２－４　コンピュータ名

Windows Embedded Standard 2009 は、他の Windows システムと同様に「コンピュータ名」、「ドメイン」、「ワークグループ」の設定が必要となります。ネットワーク上の Windows システムは、これらの設定を用いて各々のシステム識別を行います。

初期状態での「コンピュータ名」、「ドメイン」、「ワークグループ」の設定を表 1-2-4-1 に示します。

※　同一ネットワーク上に AS シリーズまたは、他の弊社 Windows 製品を複数台接続する場合は、「コンピュータ名」が重複しないように変更してください。

表 1-2-4-1.コンピュータ名、ドメイン、ワークグループの初期設定

コンピュータ名	OEM-********** *の部分は本体ごとに異なった文字となります。
ワークグループ	WORKGROUP

１－３　　アプリケーション開発と実行

AS シリーズ用 Windows Embedded Standard 2009 では、アプリケーション開発、ドライバ開発に Microsoft Visual Studio など、普段使い慣れた Windows 用の開発環境を使用することができます。ただし、組込みシステムの制限として AS シリーズ本体での開発ができません。開発は Windows オペレーティングシステムが動作している PC で行います。作成したアプリケーションは、AS シリーズ本体にインストールして動作確認を行います。（クロス開発）

● アプリケーションの開発

Windows オペレーティングシステムが動作している PC を使用してアプリケーションの開発を行います。アプリケーションの開発には Microsoft Visual Studio などの一般的な Windows アプリケーション開発環境を使用します。AS シリーズ用 Windows Embedded Standard 2009 は互換性を重視して構築されていますので、開発 PC で動作したものをほぼそのまま動作させることができます。このため開発用 PC を使用してデバッグ、動作確認を行うことが可能です。

※　AS シリーズ特有のデバイスを使用している場合は、開発 PC で動作させることができませんので注意してください。

● アプリケーションの実行

AS シリーズ本体で最終動作の確認を行います。開発 PC で動作したものがほぼそのまま動作するように構築されていますが、組込み OS であるため動作が異なる可能性があります。アプリケーションを実際に利用する前に十分な動作検証を行ってください。

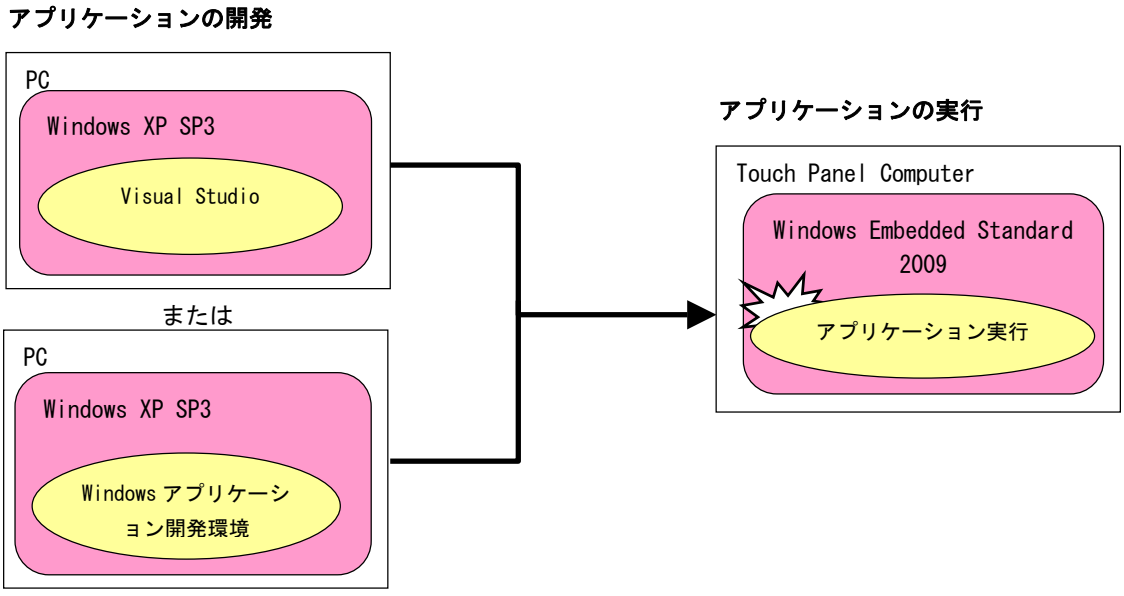

図 1-3-1. アプリケーション開発と実行

第２章　システムの操作

本章では、AS シリーズ用 Windows Embedded Standard 2009 の基本的な操作方法について説明します。

２－１　　OSの起動と終了

２－１－１　　OSの起動

AS シリーズ本体に電源を投入します。Windows ロゴの起動画面が表示され、Windows Embedded Standard 2009 が起動します。正常に起動すると、図 2-1-1 の様なデスクトップが表示されます。

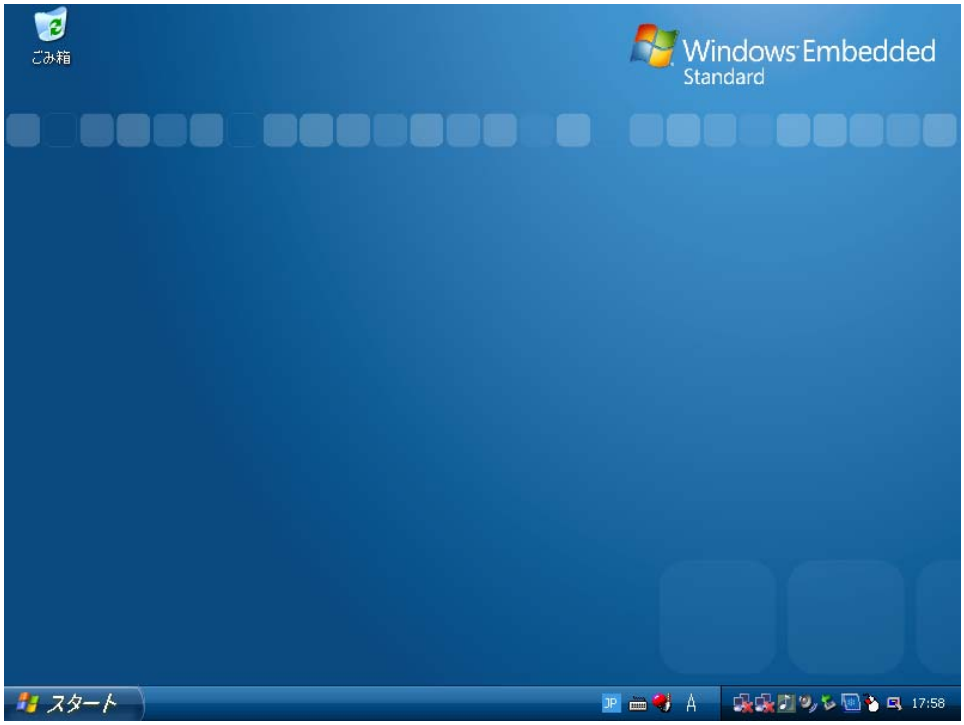

図 2-1-1-1.　デスクトップ

２－１－２　　OSの終了

スタートメニューから[終了オプション]→[電源を切る]を選択します。画面表示、POWER LED が消え、電源が待機状態になることを確認してください。

※　C ドライブ、D ドライブともに EWF が有効な場合は、終了処理を行わずに電源断を行うことができます（強制電源断）。C ドライブのみ、または D ドライブのみで EWF が有効な場合は、強制電源断を行うと EWF が無効となっているドライブのファイルシステムが破壊される可能性がありますので注意してください。
EWF についての詳細は「2-2 EWF について」を参照してください。

２－２　　EWF機能

２－２－１　EWFとは

EWF(Enhanced Write Filter)とは、Windows Embedded Standard 2009 機能で、書込みアクセスからドライブを保護する機能です。EWF を有効にするとドライブを書込み禁止にした状態で、システムを正常に動作させることが可能となります。

組込みデバイスでは、急なシャットダウン等の電源断に対しても機能が失われないシステム設計を要求されたり、書込み回数に制限のあるフラッシュメディアデバイスへの下書込みを抑止する必要があります。EWF は、組み込みデバイスにおけるこのようなニーズに対して提供されている機能です。システム運用中に誤って設定ファイルの変更がされた場合でも、再起動することによって EWF を有効にする直前の状態に戻すこともできます。

EWF では、書込み操作を実際のドライブとは別の記憶領域にリダイレクトすることによりドライブを保護します。ドライブ自体のデータは変更されないため、システム本体、ユーザーデータを保護することが可能となります。リダイレクトされる記憶領域のことをオーバーレイと呼びます。AS シリーズでは、オーバーレイに RAM を使用します。

● EWF の特徴

Enhanced Write Filter の略
ドライブの変更内容を RAM に保存
EWF で保護されたドライブの内容は変更されない
ドライブ保護の有効・無効が変更可能（Enable/Disable）
変更内容を保護されたボリュームに反映することも可能（Commit）

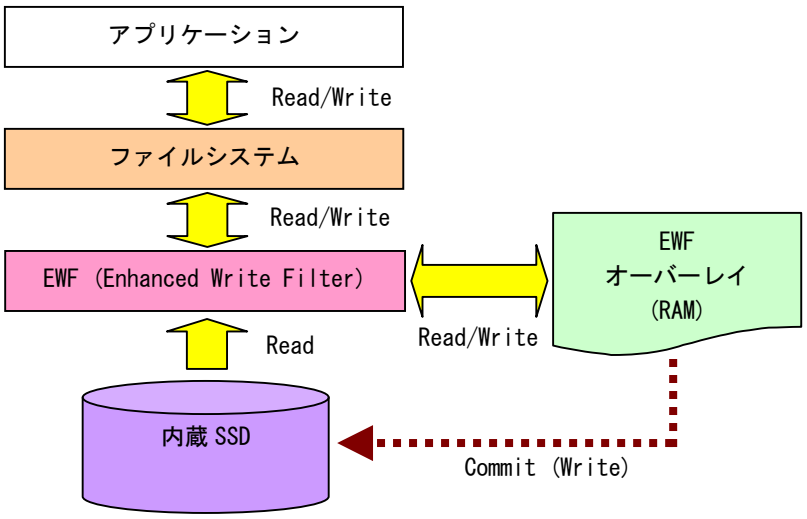

図 2-2-1-1. EWFの仕組み

２－２－２　ドライブとEWF設定

初期状態の EWF の状態は表 2-2-2-1 の様になっています。EWF の状態を変更する場合は「2-2-3 EWF の操作」を参照してください。

※　EWF が有効の場合、設定の変更、データの書き換えができません。変更を行う場合には、EWF を無効にしてください。

表 2-2-2-1. ASシリーズ 初期EWF設定

ドライブ	EWF	ドライブ内容
C	無効	システムドライブ オペレーティングシステム本体を格納します。
D	無効	ユーザードライブ ユーザーで自由に使用することができます。

２－２－３　EWFの設定方法

EWF Manager コマンドを使用して EWF を操作することができます。EWF Manger コマンドはコンソールアプリケーションです。スタートメニューから[すべてのプログラム]→[Accessories]→[Command Prompt]を選択しコマンドプロンプトを開き、コマンドを実行します。

● EWF 有効

C ドライブの EWF を有効にする場合は以下のとおりです。次回起動時に ENABLE コマンドが実行され EWF が有効になります。

```
> ewfmgr c: -enable          ←―― C ドライブを EWF 有効に

*** Enabling overlay

Protected Volume Configuration
Type             RAM
State            DISABLED
Boot Command     ENABLE      ←―― 次回起動時のコマンドに ENABLE が登録されます
Param1         0
Param2         0
         ¦
         ¦
```

同様に D ドライブの EWF を有効にする場合は以下のとおりです。

```
> ewfmgr d: -enable
```

● EWF 無効

C ドライブの EWF を無効にする場合は以下のとおりです。次回起動時に DISABLE コマンドが実行され EWF が無効になります。

```
> ewfmgr c: -disable          ←― C ドライブを EWF 無効に

*** Disabling overlay

Protected Volume Configuration
Type             RAM
State            ENABLED
Boot Command     DISABLE     ←― 次回起動時のコマンドに DISABLE が登録されます
Param1         0
Param2         0
```

D ドライブの EWF を無効にする場合は以下のとおりです。

```
> ewfmgr d: -disable
```

● コミット

EWF が有効の場合、ドライブへの書込みはオーバーレイにリダイレクトされます。そのまま終了させるとドライブへの書込みデータは消えてしまいます。コミットを行うとオーバーレイのデータをドライブに書込むことができます。C ドライブをコミットする場合は以下のとおりです。終了時にオーバーレイのデータがドライブに書込まれます。

```
> ewfmgr c: -commit           ←― C ドライブの変更分をコミット

*** Committing overlay to the protected volume.

Protected Volume Configuration
Type             RAM
State            ENABLED
Boot Command     COMMIT      ←― 終了時に COMMIT が実行されます
Param1         0
Param2         0
```

D ドライブをコミットする場合は以下のとおりです。

```
> ewfmgr d: -commit
```

※　変更分がドライブに書込まれるため終了処理に時間が掛かることがあります。終了処理中に電源を落とさないようにしてください。

● EWF の状態確認

C ドライブの EWF の状態を確認する場合は以下のとおりです。

```
> ewfmgr c:                    ←―― C ドライブの EWF 状態を確認

Protected Volume Configuration
Type             RAM
State            DISABLED
Boot Command     NO_CMD
Param1         0
Param2         0
         ¦
         ¦
```

D ドライブの EWF の状態を確認する場合は以下のとおりです。

```
> ewfmgr d:
```

● その他のコマンド

EWFMGR には、説明したコマンドの他にもコマンドが存在します。表 2-2-3-1 に AS シリーズで使用できるコマンド一覧を示します。

表 2-2-3-1. EWFMGRコマンド一覧

コマンド	内容
コマンドなし	ドライブを指定しない場合は、EWF ボリュームの表示と保護ドライブの一覧を表示します。 ドライブを指定する場合は、そのドライブの EWF 保護状態とオーバーレイの状態を確認することができます。 --- 例 1 --- > ewfmgr --- 例 2 --- > ewfmgr C:
-all	EWF の対象になっているすべてのドライブの EWF 保護状態を表示します。 --- 例 --- > ewfmgr -all
-enable	指定したドライブを EWF 有効にします。コマンド実行後、次回起動時に EWF が有効になります。 --- 例 --- > ewfmgr C: -enable
-disable	指定したドライブを EWF 無効にします。コマンド実行後、次回起動時に EWF が無効になります。 --- 例 --- > ewfmgr C: -disable

-commit	指定したドライブをコミットします。コマンド実行後の終了時にコミットされます。 --- 例 --- > ewfmgr C: -commit
-commitanddisable	指定したドライブをコミットし、EWF 無効にします。コマンド実行後の終了時にコミットが行われます。次回起動時に EWF は無効となります。 -live を指定するとその場でコミットと EWF 無効処理が行われます。この場合、コマンド実行後すぐに EWF が無効となります。 --- 例 --- > ewfmgr C: -commitanddisable --- 例 --- > ewfmgr C: -commitanddisable -live

● 設定ツールでの EWF 操作

スタートメニューから[すべてのプログラム]→[EWF]→[EWF_Config]に EWF 操作を行う設定ツールが登録されています。この設定ツールを使用しても基本的な操作(有効/無効)が可能です。詳細は「2-7　EWF Config Tool」を参照してください。

● アプリケーションからの EWF 操作

EWF API を使用することによってアプリケーションから EWF の操作が可能です。詳細は「第 3 章 EWF API」を参照してください。

２－２－４　EWFを使用するにあたっての注意事項

① EWF によるシステムメモリの消費

EWF はオーバーレイにシステムメモリを使用します。OS と EWF オーバーレイでシステムメモリを共有する構成となるため、EWF オーバーレイで消費された分だけ、OS が利用できるメモリは少なくなります。

OS が必要とするメモリと EWF オーバーレイで消費するメモリの合計が搭載メモリのサイズを超えた場合のシステムの動作は保障されません。

② EWF の消費メモリの開放

EWF で保護されたドライブに新たにファイルを作成、またはコピーした場合、EWF オーバーレイによってシステムメモリが消費されます。このとき消費されたメモリは、作成したファイルを削除しても開放されません。EWF オーバーレイは RAM Disk や Disk Cache と違い一度消費したメモリはシステムを再起動するまで解放しません。これは HDD のセクタをメモリ上にマッピングして（置き換えて）いるためだと考えられます。（現在、EWF の内部動作は公開されていませんので、詳細は確認できません。）

EWF を有効にした状態で長時間システムを使用する場合は、OS で使用するメモリと EWF オーバーレイで使用するメモリの合計が搭載メモリを超える前に再起動させる必要があります。

EWF オーバーレイのメモリ使用量は、「EWFMGR コマンド」で確認することができます。また、アプリケーションからは、「EWF API」を使用して確認することができます。

③ OS によるファイル作成

OS はレジストリ情報や、イベントログ、テンポラリファイルなどユーザが普段意識しないところでファイル作成、ファイル更新を行っています。システムドライブの EWF 保護を有効にする場合、これらのファイル作成、ファイル更新は EWF オーバーレイのメモリ消費を増加させてしまいます。設定を変更することで、ファイルの出力先を EWF 保護が無効なドライブへ変更することができます。表 2-2-4-1 に OS が作成するファイルと出力先の変更方法を示します。

表 2-2-4-1. OSが作成するファイルと出力先の変更方法

項目	内容
イベントログ	イベントログの場所を変更します。 レジストリを変更することによって、出力先を変更することができます。 表 2-2-4-2 に設定レジストリを示します。
インターネット一時ファイル	インターネット一時ファイルフォルダの場所の変更します。 インターネット一時ファイルはデフォルトでは「%USERPROFILE%¥Local Settings¥Temporary Internet Files」に設定されています。 レジストリ値を変更することによって、出力先を変更することができます。 表 2-2-4-3 に設定レジストリを示します。
TEMP、TMP フォルダ	TEMP、TMP フォルダの場所の変更します。 レジストリ値を変更することによって、出力先を変更することができます。 表 2-2-4-4 に設定レジストリを示します。

表 2-2-4-2. イベントログの出力先変更

キー	HKEY_LOCAL_MACHINE¥SYSTEM¥CurrentControlSet¥Services¥EventLog¥Application		
	名前	種類	データ
	File	REG_EXPAND_SZ	<ドライブ名とパス>¥AppEvent.evt
キー	HKEY_LOCAL_MACHINE¥SYSTEM¥CurrentControlSet¥Services¥EventLog¥Security		
	名前	種類	データ
	File	REG_EXPAND_SZ	<ドライブ名とパス>¥SecEvent.evt
キー	HKEY_LOCAL_MACHINE¥SYSTEM¥CurrentControlSet¥Services¥EventLog¥System		
	名前	種類	データ
	File	REG_EXPAND_SZ	<ドライブ名とパス>¥SysEvent.evt

表 2-2-4-3. インターネット一時ファイルの出力先変更

キー	HKEY_CURRENT_USER¥Software¥Microsoft¥Windows¥Current Version¥Explorer¥User Shell Folders		
	名前	種類	データ
	Cache	REG_EXPAND_SZ	<ドライブ名とパス>
キー	HKEY_CURRENT_USER¥Software¥Microsoft¥Windows¥CurrentVersion¥Explorer¥Shell Folders		
	名前	種類	データ
	Cache	REG_EXPAND_SZ	<ドライブ名とパス>

表 2-2-4-4. TEMP、TMPフォルダの変更

キー	HKEY_CURRENT_USER¥Environment		
	名前	種類	データ
	TEMP	REG_SZ	<ドライブ名とパス>
キー	HKEY_CURRENT_USER¥Environment		
	名前	種類	データ
	TMP	REG_SZ	<ドライブ名とパス>

④ アプリケーションでの注意

EWF が有効の場合、利用可能なメモリが減少します。アプリケーションでのメモリ、リソース確保時には注意が必要となります。また、中間ファイルを出力する可能性がある言語を使用する場合は、中間ファイルの出力先なども考慮する必要があります。表 2-2-4-2 にアプリケーションでの注意事項を示します。

表 2-2-4-2. アプリケーションでの注意事項

項目	内容
C++	malloc など、ヒープを確保する場合には、戻り値の確認を必ず行ってください。リソースについても同様です。エラーチェックを行ってください。ダイアログの作成・フォームの作成などについてもハンドルのチェックを行うようにしてください。
.NET Framework	CLR アセンブラは、ファイルとして実行時に作成されます。これらも EWF オーバーレイとして RAM を消費することになります。EWF を有効にする前に、使用する.NET Framework アプリケーションを一度実行する方が望ましいです。
ASP.NET	IE での履歴、テンポラリファイル出力で EWF オーバーレイとして RAM を消費します。 テンポラリファイルの出力先を変更する場合は、「③ OS のファイル書込み」を参考に変更してください。

２－２－５　ドメイン参加、ターミナルサーバ接続について

C ドライブの EWF を有効にした場合、レジストリは起動時に初期化され、レジストリに対する変更を保存することができません。Windows Embedded Standard 2009 の機能の中には、レジストリの変更が保存されなければ正しく動作しない機能が存在します。表 2-2-5-1 にレジストリ情報の保存が必要な機能を示します。

表 2-2-5-1. レジストリ情報の保存が必要な機能

機能・操作	レジストリ更新内容
ドメインへの参加	30 日ごとにドメインコントローラによりドメインシークレットキーが更新されます。
ターミナルサーバへの接続	アプリケーションサーバにリモートデスクトップで接続する際にライセンス情報が更新されます。

C ドライブの EWF を有効にした状態で、これらの機能を使用する場合は Registry Filter 機能を使用する必要があります。Registry Filter は出荷時の状態では有効になっていません。C ドライブの EWF を有効にする前に、以下の手順で Registry Filter を有効にしてください。

● 設定手順

① スタートメニューから[コントロール パネル]を選択します。
② [コントロール パネル]が開きます。[システム]を実行します。
③ [システムのプロパティ]ダイアログが開きます。
④ [ハードウェア]タブを選択します。[デバイス マネージャ]ボタンを押します。
⑤ [Device Manager]が開きます。(図 2-2-5-1)

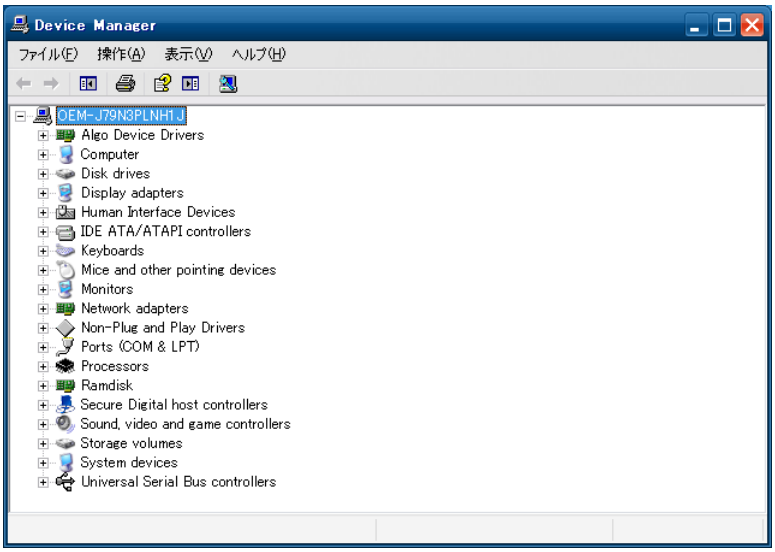

図 2-2-5-1. Device Manager

⑥ [表示]メニューから[非表示のデバイスの表示]を選択します。(図 2-2-5-2)

図 2-2-5-2. 非表示のデバイスの表示

⑦ デバイスツリーの[Non-Plug and Play Drivers]を選択し、ツリーを展開します。
⑧ [Registry Filter Driver]を右クリックしてメニューを開き、[有効]を選択します。(図 2-2-5-3)

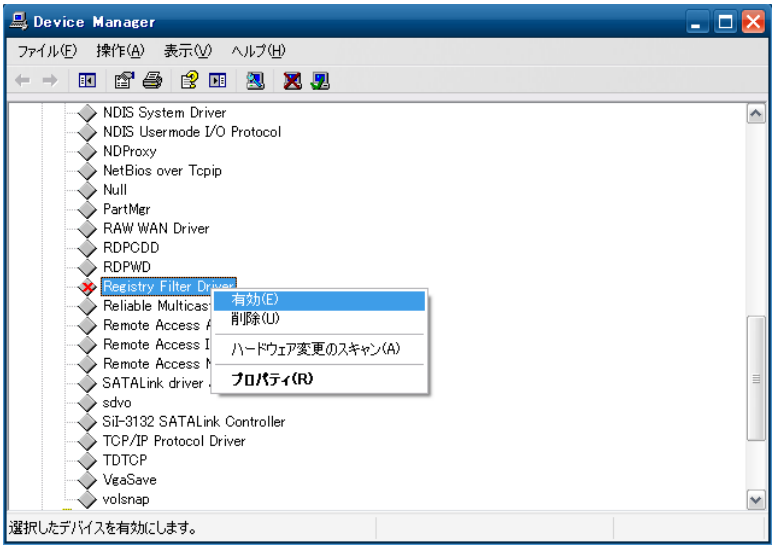

図 2-2-5-3. Registry Filter Driver有効

⑨ 再起動します。

２－３　　ログオン設定

２－３－１　自動ログオン設定

ログオンは、初期状態では Administrator アカウントで自動ログオンするように設定されています。ログオンアカウントを選択してログオンしたい場合は、設定を変更する必要があります。

● 設定手順

① スタートメニューから[ファイル名を選択して実行]を選択します。

② [ファイル名を選択して実行]ダイアログが開きます。「control userpasswords2」と入力して[OK]ボタンを押します。

③ [ユーザーアカウント]ダイアログが開きます。[ユーザーがこのコンピュータを使うには、ユーザー名とパスワードの入力が必要]チェックボックスにチェックを入れ、[OK]ボタンを押します。

２－４　　言語設定

２－４－１　マルチ言語機能

Windows Embedded Standard 2009 は、マルチ言語対応 OS となっています。AS シリーズでは、日本語と英語の 2 つの言語に対応しています。初期状態では日本語環境で起動します。

２－４－２　言語の変更方法

言語はコントロールパネルから変更可能となっています。英語環境にするには、以下の手順で設定を行ってください。

● 設定手順

⑩ スタートメニューから[コントロール パネル]を選択します。
⑪ [コントロール パネル]が開きます。[地域と言語のオプション]を実行します。
⑫ [地域と言語のオプション]ダイアログが開きます。
⑬ [地域オプション]タブを選択します。[標準と形式]のリストから英語を選択します。[場所]のリストから英語地域を選択します。
⑭ [言語]タブを選択します。[テキストサービスと入力言語]の[詳細]ボタンを押します。
⑮ [テキストサービスと入力言語]ダイアログが開きます。[規定の言語]でリストから英語の入力サービスを選択します。[OK]ボタンを押してダイアログを閉じます。
⑯ [言語]タブで[メニューとダイアログで使われる言語]のリストから[English]を選択します。
⑰ [詳細設定]タブを選択します。[Unicode 対応でないプログラムの言語]のリストから英語を選択します。[既定のユーザーアカウントの設定]の[全ての設定を現在のユーザーアカウントと既定のユーザープロファイルに適応する]にチェックを入れます。
⑱ [OK]ボタンを押します。ファイルコピーに関する確認のダイアログが表示されますので[はい]ボタンを押します。続いて、再起動確認のダイアログが表示されますので[はい]を押して再起動します。

２－５　　サービス設定

２－５－１　　サービス設定の変更

AS シリーズ用 Windows Embedded Standard 2009 には、様々なサービスが搭載されています。搭載されているサービスの中には､工場出荷状態では停止しているものもあります｡これらのサービスを利用するには、サービスの操作、起動設定の変更などを行う必要があります。

サービスの操作、起動設定の変更は以下の手順で行います。

● サービス操作、起動設定の変更

① スタートメニューから[コントロール パネル]を選択します。

② [コントロール パネル]から[管理ツール]、[Service]の順にウィンドウを開きます。

③ [Services]設定画面が開きます。(図 2-5-1-1)

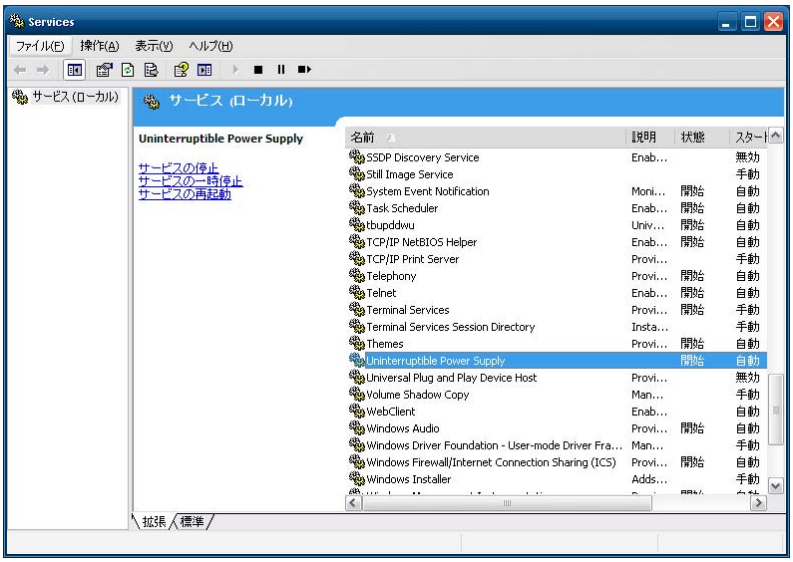

図 2-5-1-1 Services設定画面

④ 設定を変更するサービスをダブルクリックしてプロパティを開きます。(図 2-5-1-2)

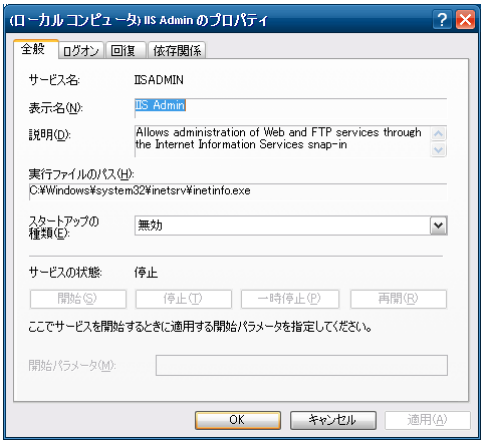

図 2-5-1-2 Services設定画面

⑤ 起動設定を変更する場合は、[スタートアップの種類]を目的の設定に変更します。

⑥ サービスの操作を行う場合は、[開始]、[停止]、[一時停止]、[再開]ボタンで行います。

２－５－２　Internet Information Service (IIS)

AS シリーズ用 Windows Embedded Standard 2009 では、ネットワークサービスとして Internet Information Service (IIS)を搭載しています。工場出荷状態では、IIS 関連のサービスは全て停止しています。使用する場合は、IIS 関連のサービスを起動させる必要があります。

IIS 関連サービスを起動させるには、サービスの起動設定を変更します。各サービスの起動設定を表 2-5-2-1 に示します。

表 2-5-2-1. IISサービス起動設定

サービス名	起動設定
IIS Admin	自動
World Wide Web Publishing	自動
FTP Publishing	自動
Simple Mail Transfer Protcol (SMTP)	自動

２－５－３　Telnet

AS シリーズ用 Windows Embedded Standard 2009 では、ネットワークサービスとして Telnet を搭載しています。工場出荷状態では、Telnet サービスは停止しています。使用する場合は、Telnet サービスを起動させる必要があります。

Telnet サービスを起動させるには、サービスの起動設定を変更します。起動設定を表 2-5-3-1 に示します。

表 2-5-3-1. Telnetサービス起動設定

サービス名	起動設定
Telnet	自動

２－６　ASD Config Tool

２－６－１　ASD Config Tool

「ASD Config Tool」は、AS シリーズ専用のコントロールパネルアプリケーションです。スタートメニューから[コントロール パネル]を開き、[ASD Config]から起動できます。設定内容を表 2-6-1-1 に示します。

表 2-6-1-1. ASD Config Tool 設定/表示内容

タブ	設定/表示内容
LCD Setting	LCD バックライトの輝度を調整することができます。
Buzzer Setting	タッチブザーの設定を行うことができます。
Board Information	ハードウェア、ソフトウェアのバージョンを確認することができます。

２－６－２　LCD Setting

LCD バックライトの輝度を調整できます。

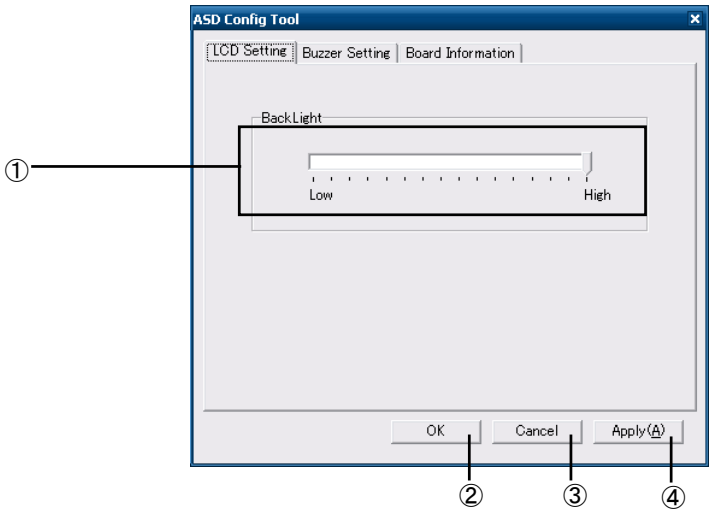

図 2-6-2-1. ASD Config – LCD Setting

① バックライトの輝度を 16 段階で調整できます。
② 設定を保存、適用して終了します。
③ 設定を破棄して終了します。
④ 設定を保存、適用します。

２－６－３　Buzzer Setting

タッ チブザーの設定を行うことができます。

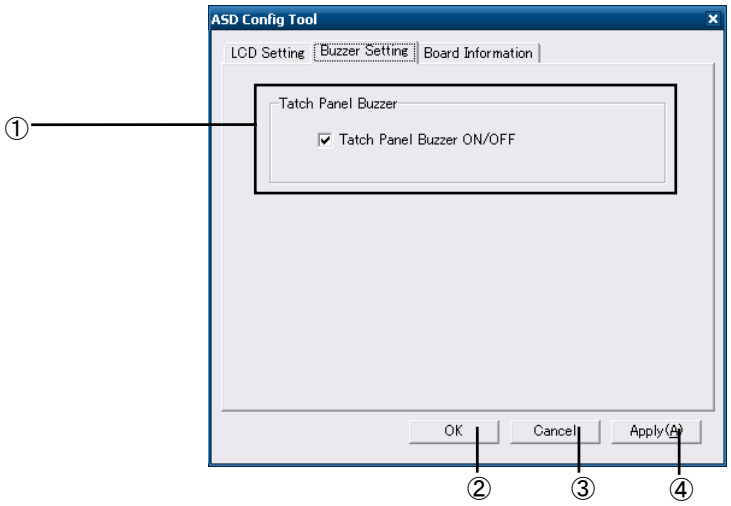

図 2-6-3-1. ASD Config – Buzzer Setting

① タッチパネルブザーの ON/OFF を設定できます。(チェックあり: ON、チェックなし: OFF)
② 設定を保存、適用して終了します。
③ 設定を破棄して終了します。
④ 設定を保存、適用します。

２－６－４　Board Information

ハードウェア、ソフトウェアのバージョンを確認することができます。

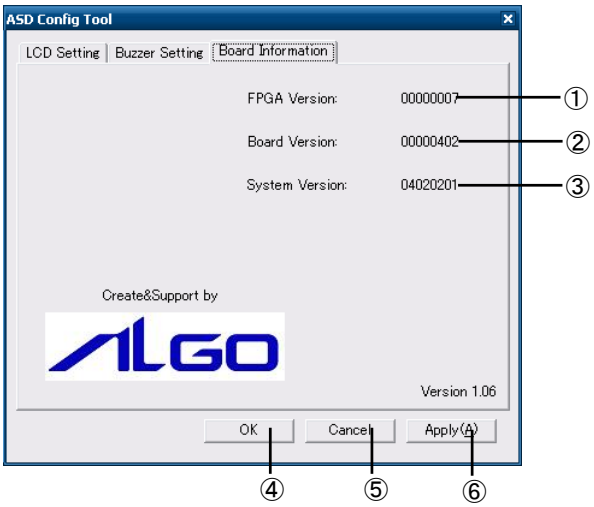

図 2-6-4-1. ASD Config – Board Information

① FPGA データのバージョンを表示します。
② メインボードのバージョンを表示します。
③ OS イメージのバージョンを表示します。
④ 設定を保存、適用して終了します。
⑤ 設定を破棄して終了します。
⑥ 設定を保存、適用します。

２－６－５　初期値

「ASD Config Tool」の設定初期値を表 2-6-5-1 に示します。

表 2-6-5-1. ASD Config Tool 設定初期値

タブ	設定項目	初期値
LCD Setting	Backlight	High 側最大値
Buzzer Setting	Touch Panel Buzzer	ON

２－７　　EWF Config Tool

２－７－１　EWF Config Tool

「EWF Config Tool」は、AS シリーズ専用の EWF 設定ツールです。スタートメニューから[すべてのプログラム]→[EWF]→[EWF_Config]で起動できます。設定/表示内容を表 2-7-1-1、表 2-7-1-2 に示します。

表 2-7-1-1. EWF Config Tool 表示内容

項目	表示内容
Device Name	保護されたボリュームのデバイス名を表示します。
Type	保護されたボリュームのオーバーレイの種類を表示します。 [DISK][RAM][RAM(REG)]のいずれかになります。
State	保護されたボリュームのオーバーレイの状態を表示します。 [ENABLED][DISABLED]のいずれかになります。
Boot Command	保護されたボリュームの再起動時に実行するコマンドを表示します。 [NO_CMD][ENABLE][DISABLE][COMMIT][SET_LEVEL]のいずれかになります。

表 2-7-1-2. EWF Config Tool 設定内容

コマンド	設定内容
Enable	EWF で保護されたボリュームで現在無効になっているオーバーレイを有効にします。
Disable	EWF で保護されたボリュームで現在有効になったオーバーレイを無効にします。
No Command	現在設定されているコマンドを取り消します。

２－７－２　Volume Infomation

ボリュームの状態を表示します。

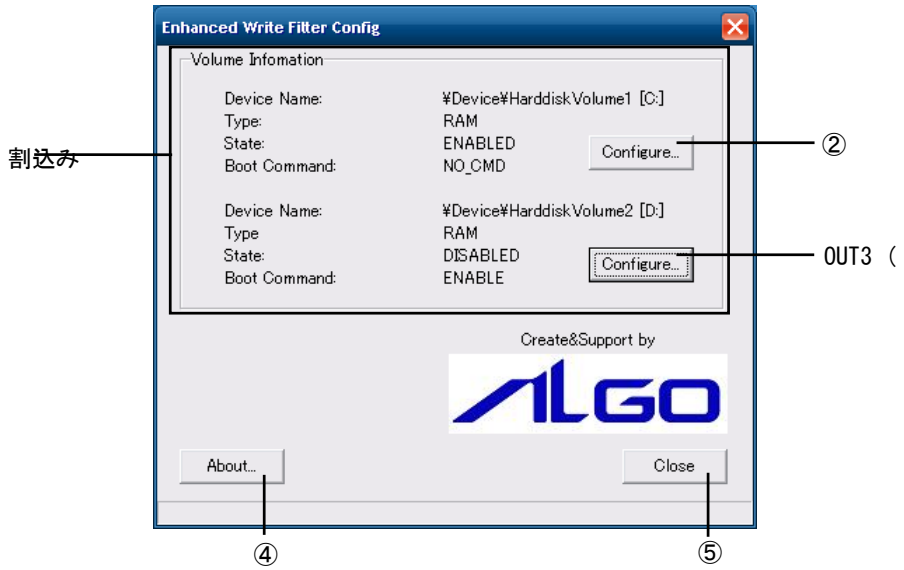

図 2-7-2-1. EWF Config – Volume Infomation

① 保護されたボリュームの現在の状況を表示します。
② 1 つ目の保護されたボリュームの設定画面に移項します。
③ 2 つ目の保護されたボリュームの設定画面に移項します。
④ 設定ツールのバージョン情報を読み出します。
⑤ 終了します。

２－７－３　EWF Configuration

再起動時に実行するコマンドの変更を行うことができます。

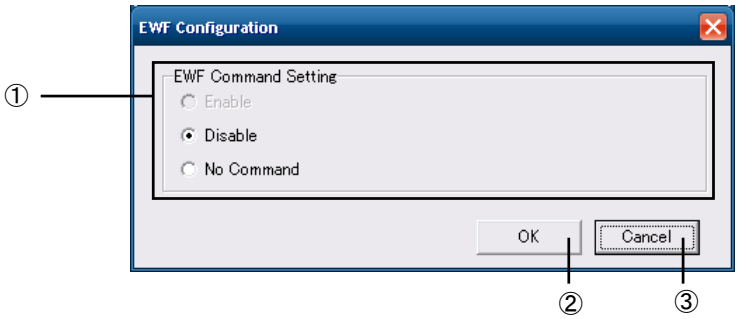

図 2-7-3-1. EWF Config – EWF Configuration

① 再起動時に実行するコマンドを選択します(※1)。
② ①で選択したコマンドを有効にして Volume Information 画面に戻ります。
③ 設定を変更しないで Volume Information 画面に戻ります。

※1　保護されたボリュームが現在有効の場合は、[Disable]と[No Command]が、無効の場合は[Enable]と[No Command]がそれぞれ設定可能です。

２－８ Software Watchdog Timer Config Tool

２－８－１ Software Watchdog Timer Config Tool

「Software Watchdog Timer Config Tool」は、ソフトウェア・ウォッチドッグタイマ機能の設定を行うためのコントロールパネルアプリケーションです。スタートメニューから[コントロール パネル]を開き、[Software Watchdog]から起動できます。

「Software Watchdog Timer Config Tool」では、ソフトウェア・ウォッチドッグタイマドライバの初期値を設定/表示することができます。ソフトウェア・ウォッチドッグタイマドライバを使用する場合、ドライバオープン時にドライバ設定が「Software Watchdog Timer Config Tool」で設定された値に初期化されます。設定内容を表 2-8-1-1 に示します。

表 2-8-1-1. Software Watchdog Timer Config Tool 設定/表示内容

項目	設定/表示内容
Action	ソフトウェア・ウォッチドッグタイマのタイムアウト時の動作を設定します。 ・Shutdown（シャットダウン） ・Reboot（再起動） ・Popup（ポップアップ通知） ・Event（ユーザーイベント通知）
Time	ソフトウェア・ウォッチドッグタイマのタイマ時間を設定します。 有効設定値: 1～65535 タイマ時間: 設定値 x100msec
Enable output message for Windows Event Log	ソフトウェア・ウォッチドッグタイマのタイムアウト時に、イベントログにメッセージを記録するかどうかを設定します。

２－８－２　Software Watchdog Timer Configuration

ソフトウェア・ウォッチドッグタイマ機能の設定を行うことができます。

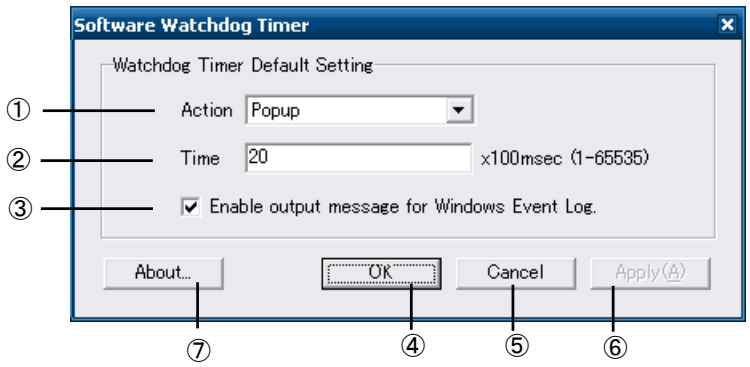

図 2-8-2-1. Software Watchdog Timer Configuration

① ソフトウェア・ウォッチドッグタイマ タイムアウト動作を設定します。
② ソフトウェア・ウォッチドッグタイマ タイマ時間を設定します。
③ ソフトウェア・ウォッチドッグタイマ イベントログへの記録を指定します。
④ 設定を保存、適用して終了します。
⑤ 設定を破棄して終了します。
⑥ 設定を保存、適用します。
⑦ バージョン情報を表示します。

２－８－３　初期値

「Software Watchdog Timer Config Tool」の設定初期値を表 2-8-3-1 に示します。

表 2-8-3-1. Software Watchdog Timer Config Tool 設定初期値

項目	初期値
Action	Popup
Time	20
Enable output message for Windows Event Log	ON

第３章　Touch Panel Computerについて

本章では、Touch Panel Computer Stand Type（AS シリーズ）に搭載されている機能について説明します。

３－１　Touch Panel Computerに搭載された機能について

AS シリーズにはグラフィック表示機能、通信機能、サウンド機能、USB 機能などが搭載されています。これらの機能は Windows の標準インターフェースを使用して操作することができます。また、AS シリーズでは組込みシステム向けに独自機能が追加されています。組込みシステム機能は、専用ドライバを使用して操作することが可能です。

AS シリーズ本体（AS450A）の外形図を図 3-1-1 に示します。各部名称を表 3-1-1 に示します。

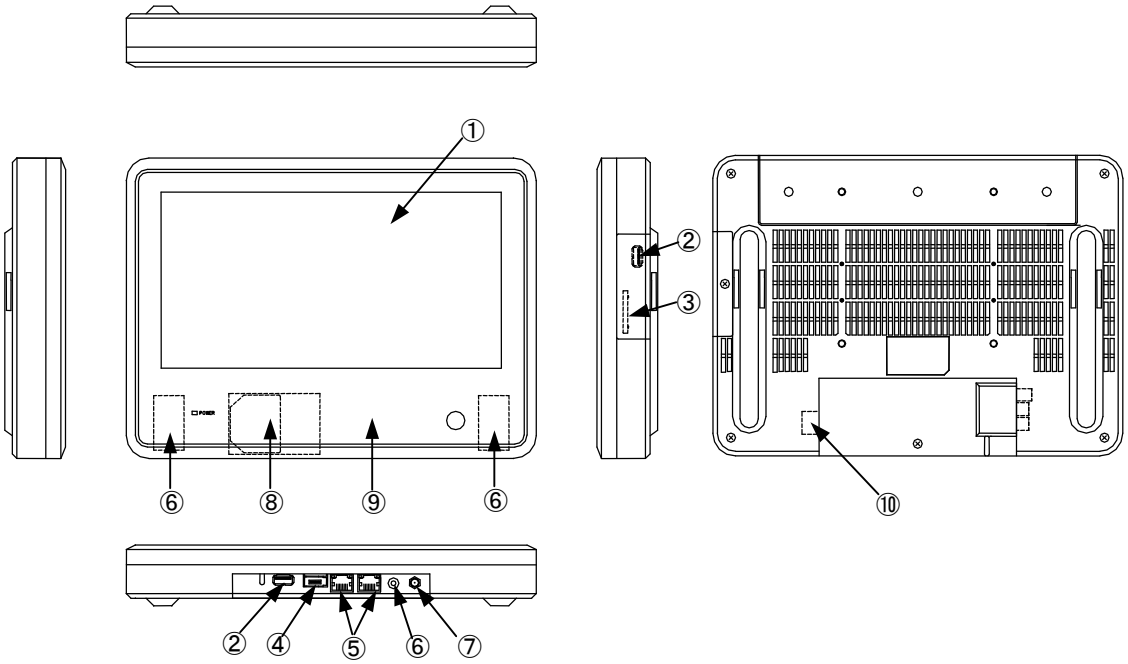

図 3-1-1. 外形図（AS450A）

表 3-1-1. 各部名称

No.	名称	機能	説明
①	液晶タッチパネル	グラフィック タッチパネル	画面表示を行います。 タッチパネルはポインティングデバイスとして使用可能です。
②	USB インターフェース	USB ポート	USB1.1/2.0 の機器を接続することができます。
③	SD カード インターフェース	SD/SDHC カード	記憶領域として SD/SDHC カードを使用することができます。
④	シリアル インタフェース	シリアルポート	シリアル通信が行えます。 SIO: COM1
⑤	ネットワーク インターフェース	有線 LAN （HUB 機能付き）	ネットワークポートとして使用できます。 HUB 機能付きです。 LAN1、LAN2: ローカルエリア接続
⑥	内蔵スピーカ 音声出力端子	サウンド	音声出力が使用できます。
⑦	無線 LAN アンテナコネクタ	無線 LAN	無線ネットワークデバイスとして使用できます。
⑧	FeliCa リーダ・ライタ （オプション）	FeliCa リーダ・ライタ	FeliCa リーダ・ライタモジュールです。 COM2 に接続されます。
⑨	Bluetooth （オプション）	Bluetooth	Bluetooth モジュールです。 Bluetooth デバイスと通信することができます。
⑩	DIO インターフェース	汎用入出力	汎用の入出力です。 入力 6 点、出力 4 点を制御できます。

３－２　　Windows標準インターフェース対応機能

本項では、AS シリーズに搭載されている Windows 標準インタフェース対応機能について説明します。

３－２－１　グラフィック

一般的な Windows と同様に、デスクトップ表示、アプリケーション表示を行います。AS シリーズでは LCD（図 3-1-1 ①）へデスクトップ表示を行います。

スタートメニューから[コントロールパネル]を選択し、[画面]を起動して設定を行います。

３－２－２　タッチパネル

タッチパネルをタッチすることにより､マウスなどのポインティングデバイス操作を行うことができます。本シリーズに搭載されたタッチパネルには以下の様な特徴があります。

- タッチパネルを操作することでマウスと同等な操作環境を実現することができます。
- マウスとの共存が可能なため、特別な設定を行うことなくタッチパネル、マウス双方を切替え使用することができます。
- マウス左右ボタン切替え、クリック操作に関係する詳細な設定、タッチ入力に対するイベントのカスタマイズ、精密なキャリブレーション機能などを提供します。

タッチパネルの設定、キャリブレーション機能などは[スタートメニュー]-[すべてのプログラム]-[UPDD]から行ってください。

タッチパネルドライバ､ツールのドキュメントは株式会社 DMC 様の WEB サイト(http://www.dmccoltd.com)より入手できます。ダウンロードページから「TSC-10/DD ユーザーガイド」をダウンロードしてください。

● キャリブレーションについて

AS シリーズでは、構造上タッチパネルの上端部、下端部でタッチが反応しにくいことがあります。ご使用に問題がある場合は、以下の手順で再度キャリブレーションを行ってください。上下方向の稼動範囲を広げることができます。

① [スタートメニュー]-[すべてのプログラム]-[UPDD]-[キャリブレーション]を選択し、キャリブレーションを開始します。
② 十字のターゲットが表示されます。上段のポイントと下段のポイントでは、十字の中心からずらして図 3-2-2-1 に示す赤いポイント付近をタッチします。
③ すべてのポイント入力が終わると確認画面が表示されます。
④ 問題がなければ[OK]ボタンを押してキャリブレーションを終了します。

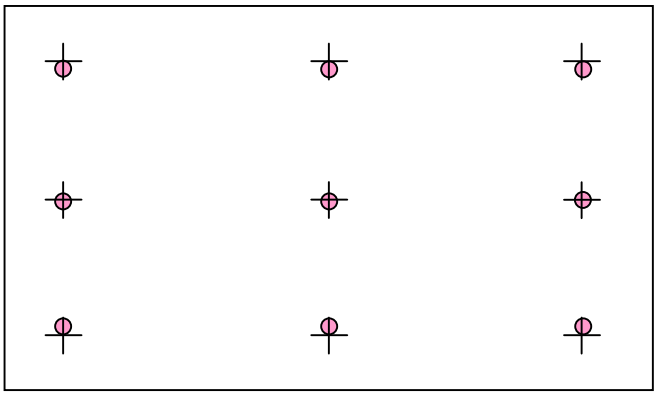

図 3-2-2-1. キャリブレーションタッチポイント

３－２－３　シリアルポート

一般的な Windows と同様に、COM ポートとしてシリアル通信に使用することができます。アプリケーションからは COM1 が使用可能です。搭載されている COM ポートの一覧を表 3-2-3-1 に示します。

表 3-2-3-1. シリアルポート

COM ポート	説明
COM1	SIO（図 3-1-1 ④） アプリケーションでシリアル通信に使用できます。
COM2	FeliCa モジュールに接続されます。 （オプション）
COM3	リザーブ。 アプリケーションでは使用できません。
COM4	タッチパネルとの通信に使用しています。 アプリケーションでは使用できません。

３－２－４　有線LAN

AS シリーズには 100BASE イーサ対応の有線 LAN ポートが 1 ポート用意されています。この有線 LAN ポートは内蔵 HUB に接続されます。内蔵 HUB の 2 つのネットワークインタフェースが外部コネクタとして設置されます。図 3-2-4-1 に有線 LAN ポートと内蔵 HUB の接続を示します。

有線 LAN ポートは、一般的な Windows と同様にネットワークポートとして使用することができます。表 3-2-4-1 にネットワーク名称と外部コネクタとの対応を示します。

※　AS シリーズの有線 LAN ポートは内蔵 HUB に接続されています。このため、コネクタにケーブルが接続されていない状態でも、OS 上からは常に接続状態として認識されます。接続状態の変化をトリガとした処理（DHCP の IP 自動取得など）は、起動時の一度しか動作しませんので注意してください。

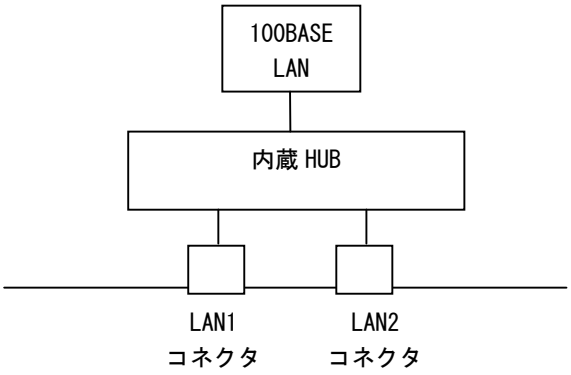

図 3-2-4-1. HUB機能

表 3-2-4-1. 有線LANポート

ネットワーク名称	説明
ローカルエリア接続	LAN1, LAN2（図 3-1-1 ⑤）

AS シリーズの有線 LAN ポートには、Wake On LAN 機能があります。Wake On LAN 機能を使用してスタンバイ状態から復帰させることができます。出荷状態ではこの機能は無効となっています。使用する場合は、以下の手順で機能を有効にしてください。

● 設定手順

① スタートメニューから[コントロール パネル]を選択します。
② [コントロール パネル]が開きます。[システム]を実行します。
③ [システムのプロパティ]ダイアログが開きます。
④ [ハードウェア]タブを選択します。[デバイス マネージャ]ボタンを押します。
⑤ [Device Manager]が開きます。デバイスツリーから[Network adapters]を選択しツリーを展開します。
⑥ [Realtek PCIe FE Famiry Controller]を右クリックしてメニューを開き、[プロパティ]を選択します。（図 3-2-4-2）

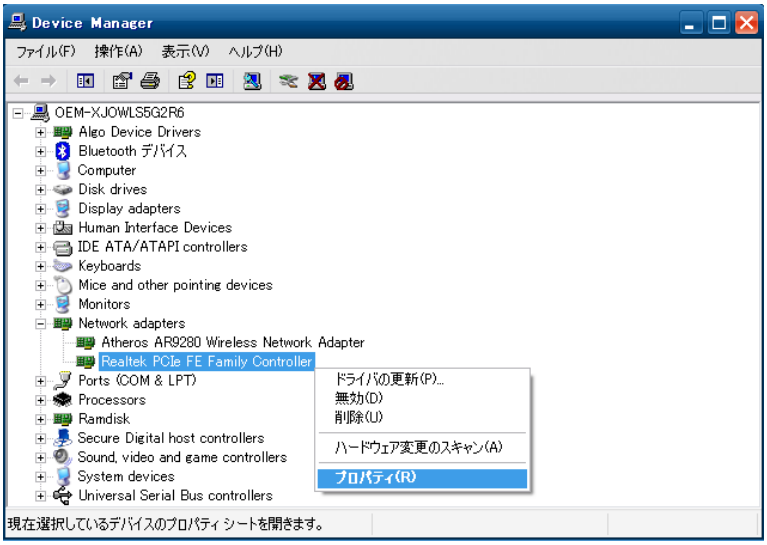

図 3-2-4-2. Device Manager

⑦ [電源の管理]タブを選択し、[このデバイスで、コンピュータのスタンバイ状態を解除できるようにする]にチェックを入れます。（図 3-2-4-3）

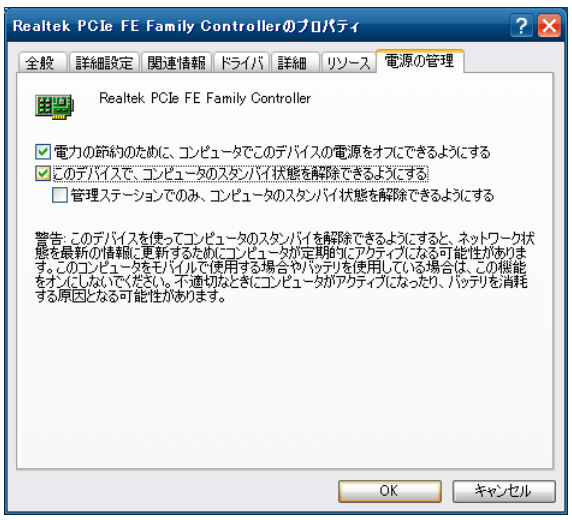

図 3-2-4-3. スタンバイ復帰の有効

⑧ [OK]ボタンを押してダイアログを閉じます。

３－２－５　無線LAN

AS シリーズには無線 LAN が搭載されています。スタートメニューの[プログラム]-[Atheros]-[Atheros Client Utility]から[Atheros クライアント ユーティリティ]を起動して、設定、操作を行ってください。使用方法は、[Atheros クライアント ユーティリティ]の[ヘルプ]-[Atheros クライアント ユーティリティ ヘルプ]から起動するマニュアルを参照してください。

３－２－６　サウンド

サウンドの機能として、音声出力を使用することができます。一般的な Windows と同様に使用することができます。サウンド設定は、スタートメニューから[コントロールパネル]を表示して、[サウンドとオーディオデバイス]及び[Realtek HD オーディオ設定]で行ってください。

３－２－７　USBポート

USB1.1/2.0 対応の USB ポートを外部コネクタとして 2 ポート用意しています。一般的な Windows と同様に USB 機器を接続して使用することができます。接続する USB 機器のドライバは、別途用意してください。

３－２－８　SD/SDHCカード

SD/SDHC に対応したカードスロットを搭載しています。データ記憶領域として SD/SDHC カードを使用することができます。

３－２－９　FeliCaリーダ・ライタ（オプション）

AS シリーズにはオプションで FeliCa リーダ・ライタモジュールを搭載することができます。FeliCa リーダ・ライタモジュールは、COM2 に接続されます。

※　FeliCa リーダ・ライタを使用するには、機密保持契約が必要です。ご使用の場合は、弊社営業にお問い合わせください。

３－２－１０ Bluetooth（オプション）

AS シリーズにはオプションで Bluetooth モジュールを搭載することができます。Bluetooth 搭載モデルは、Bluetooth マネージャがインストールされています。使用方法は、[コントロールパネル]-[Bluetooth 設定]-[ヘルプ]から「Bluetooth ワイヤレス ユーザーガイド」を参照してください。

３－３　　組込みシステム機能

ASシリーズには、組込みシステム向けに独自の機能が搭載されています。本項では、組込みシステム機能について説明します。組込みシステム機能の一覧を表3-3-1に示します。

ASシリーズ用Windows Embedded Standard 2009では、組込みシステム機能を使用するためにドライバを用意しています。ドライバの使用方法は「第5章 組込みシステム機能ドライバ」を参照してください。

表3-3-1. 組込みシステム機能

機能	説明
タイマ割込み機能	ハードウェアによるタイマ機能です。 完了時にイベントを発生させることができます。
ブザー	ブザーを鳴らすことができます。 タッチパネルのタッチ時のブザーも制御できます。
汎用入出力	汎用の入出力です。 入力6点、出力4点を制御できます。 （図3-1-1 ⑩）
LCDバックライト	LCDバックライトを制御できます。 バックライトのON/OFF、輝度調整ができます。
RAS機能	汎用入力のIN0、IN1にリセット機能、割込み機能があります。 IN0リセット、IN1割込みの制御ができます。
ソフトウェア ウォッチドッグタイマ機能	ソフトウェアによるウォッチドッグタイマを操作することができます。

３－３－１　　タイマ割込み機能

ハードウェアによるタイマ割込み機能が実装されています。この機能を使用すると指定した時間で周期的に割込みを発生させることができます。

アプリケーションでハードウェア割込みによる正確なタイマイベントを受けることができます。

３－３－２　　ブザー

ブザーのON/OFF、タッチパネルのタッチ音のON/OFFを行うことができます。

アプリケーションでブザーを鳴らすことができます。タッチパネルのタッチ音設定は、「ASD Config Tool」からも設定可能です。

３－３－３　　汎用入出力

汎用入出力が搭載されています。アプリケーションから入力6点、出力4点が制御可能です。

３－３－４　　LCDバックライト

LCDのバックライトを調整することができます。バックライトのON/OFFと輝度を変更することができます。

アプリケーションでLCDバックライトの調整が可能です。バックライトの輝度は、「ASD Config Tool」からも設定可能です。

３－３－５　　RAS機能

ハードウェアによるIN0リセット機能、IN1割込み機能が実装されています。

IN0入力時にハードウェアリセットを掛けることができます。アプリケーションでこの機能の有効/無効を制御できます。

IN1入力時に割込みを発生させることができます。アプリケーションでこの機能の有効/無効を制御できます。また、割込み発生時にイベントを受けることができます。

３－３－６　　ソフトウェア・ウォッチドッグタイマ機能

ソフトウェアによるウォッチドッグタイマが実装されています。アプリケーションのハングアップを検出することできます。

第４章　EWF API

本章では、Enhanced Write Filter (EWF) アプリケーション・プログラミング・インターフェース (API) の使用方法について説明します。

４－１　EWF APIについて

EWF API は、アプリケーションから EWF の操作を行うためのインターフェースを提供します。EWF API 関数を使用すると、アプリケーションから保護ドライブの EWF 設定の変更、オーバーレイデータのコミットなどを行うことができます。

AS シリーズで使用できる EWF API 関数の一覧を表 4-1-1 に示します。

表 4-1-1. EWF API関数一覧

関数	説明
EwfMgrGetDriveLetterFromVolumeName	指定したボリューム名のドライブ文字を取得します。
EwfMgrOpenProtected	EWF で保護されたボリュームを開きます。
EwfMgrClose	EWF で保護されたボリュームを閉じます。
EwfMgrClearCommand	次の再起動時に発生する可能性がある保留中のコマンドをクリアします。
EwfMgrDisable	EWF で保護されたボリュームで現在有効になったオーバーレイを無効にします。
EwfMgrEnable	EWF で保護されたボリュームで現在無効になっているオーバーレイを有効にします。
EwfMgrCommit	オーバーレイのデータを EWF で保護されたボリュームにすべてコミットします。
EwfMgrCommitAndDisableLive	オーバーレイのデータを EWF で保護されたボリュームに直ちにコミットした上で、EWF を無効にします。この関数は、再起動は不要です。
EwfMgrGetProtectedVolumeConfig	EWF で保護されたボリュームの構成情報を取得します。
EwfMgrGetProtectedVolumeList	すべての EWF で保護されたボリュームのリストを取得します。
EwfMgrVolumeNameListIsEmpty	EWF ボリューム名リストが空かどうかを判別します。
EwfMgrVolumeNameEntryPop	EWF ボリューム名リストから現在のエントリを削除し、現在のメモリエントリを解放します。
EwfMgrVolumeNameListDelete	EWF ボリューム名リストのすべてのエントリを削除します。

４－２　　EWF API関数リファレンス

EwfMgrGetDriveLetterFromVolumeName 関数

機能　　指定したボリューム名のドライブ文字を取得します。

書式

```
WCHAR EwfMgrGetDriveLetterFromVolumeName (
    LPCWSTR lpVolumeName
);
```

引数　　lpVolumeName
　　[in] 指定したボリュームへのロング ポインタ。

戻り値　　このメソッドが成功すると、ドライブ文字が WCHAR 型で返されます。関数が失敗すると、-1 が返されます。

説明　　この関数は、EWF で保護されたボリュームとして指定されたデバイス名のドライブ文字を取得します。
ボリューム名は、任意の有効な手段で表現できます。EwfMgrGetProtectedVolumeList によって返されるボリューム名のいずれか 1 つを使用することができます。

EwfMgrOpenProtected 関数

機能 EWF で保護されたボリュームを開きます。

書式
```
HANDLE EwfMgrOpenProtected (
    LPCWSTR lpVolume
);
```

引数 lpVolume
[in] 開くボリュームへのロング ポインタ。

戻り値 関数が成功した場合の戻り値は、デバイスへの HANDLE です。関数が失敗した場合の戻り値は INVALID_HANDLE_VALUE です。エラーの詳細情報については、GetLastError を呼び出します。

説明 この関数は、EWF で保護されているボリュームを開きます。EwfMgrClose 関数を使用して、EwfMgrOpenProtected によって返されたオブジェクトハンドルを閉じます。
ボリューム名は、任意の有効な手段で表現できます。EwfMgrGetProtectedVolumeList によって返されるボリューム名のいずれか 1 つを使用することができます。

EwfMgrClose 関数

機能 EWF で保護されたボリュームを閉じます。

書式
```
BOOL EwfMgrClose (
    HANDLE hDevice
);
```

引数 hDevice
[in] EWF で保護されたデバイスへのハンドル。

戻り値 この関数が正常に実行された場合、戻り値は TRUE です。 この関数が正常に実行されなかった場合、戻り値は FALSE です。エラーの詳細情報については、GetLastError を呼び出します。

説明 この関数は、EWF で保護されたボリュームを閉じます。

EwfMgrClearCommand 関数

機能 次の再起動時に発生する可能性がある保留中のコマンドをクリアします。

書式
```
BOOL EwfMgrClearCommand (
    HANDLE hDevice
);
```

引数 hDevice
[in] EWF で保護されたデバイスへのハンドル。

戻り値 この関数が正常に実行された場合、戻り値は TRUE です。この関数が正常に実行されなかった場合、戻り値は FALSE です。エラーの詳細情報については、GetLastError を呼び出します。

説明 この関数は、保留のコマンドをすべてクリアします。

EwfMgrDisable 関数

機能

EWF で保護されたボリュームで現在有効になったオーバーレイを無効にします。

書式

```
BOOL EwfMgrDisable (
    HANDLE hDevice,
    BOOL fCommit
);
```

引数

hDevice
[in] EWF で保護されたボリュームへのハンドル。
fCommit
[in] TRUE の場合、保護を無効にする前に現在のオーバーレイをコミットします。FALSE の場合、すべてのオーバーレイを破棄し、保護を無効にします。

戻り値

この関数が正常に実行された場合、戻り値は TRUE です。 この関数が正常に実行されなかった場合、戻り値は FALSE です。エラーの詳細情報については、GetLastError を呼び出します。

説明

この関数は、指定したボリュームで現在有効になっているオーバーレイを無効にし、fCommit が TRUE であれば現在のレベルをコミットし、fCommit が FALSE であればすべてのオーバーレイを破棄します。EWF で保護されたボリュームとして指定されたデバイス名のドライブ文字を取得します。
オーバーレイを次の再起動時に無効にします。

EwfMgrEnable 関数

機能　EWF で保護されたボリュームで現在無効になっているオーバーレイを有効にします。

書式
```
BOOL EwfMgrEnable (
    HANDLE hDevice
);
```

引数　hDevice
[in] EWF で保護されたボリュームへのハンドル。

戻り値　この関数が正常に実行された場合、戻り値は TRUE です。この関数が正常に実行されなかった場合、戻り値は FALSE です。エラーの詳細情報については、GetLastError を呼び出します。

説明　この関数は、特定のボリュームで現在無効になっているオーバーレイを有効にします。
オーバーレイを次の再起動時に無効にします。
ボリュームは EWF 保護用にすでに構成されており、無効状態である必要があります。

EwfMgrCommit 関数

機能 オーバーレイのデータを EWF で保護されたボリュームにすべてコミットします。

書式
```
BOOL EwfMgrCommit (
    HANDLE hDevice
);
```

引数 hDevice
[in] EWF で保護されたボリュームへのハンドル。

戻り値 この関数が正常に実行された場合、戻り値は TRUE です。この関数が正常に実行されなかった場合、戻り値は FALSE です。エラーの詳細情報については、GetLastError を呼び出します。

説明 この関数は、オーバーレイにある現在のデータをすべて保護されたボリュームにコミットします。
オーバーレイデータを終了時にコミットします。

EwfMgrCommitAndDisableLive 関数

機能

オーバーレイのデータを EWF で保護されたボリュームに直ちにコミットした上で、EWF を無効にします。

書式

```
BOOL EwfMgrCommitAndDisableLive (
    HANDLE hDevice
);
```

引数

hDevice
[in] EWF で保護されたボリュームへのハンドル。

戻り値

この関数が正常に実行された場合、戻り値は TRUE です。 この関数が正常に実行されなかった場合、戻り値は FALSE です。 エラー情報の詳細については、GetLastError を呼び出します。

説明

この関数は、オーバーレイの現在のすべてのデータを保護されているボリュームにコミットし、EWF を無効化します。
オーバーレイは保護されているボリュームに直ちにコミットされ、EWF は再起動を行わずに無効化されます。

EWF_VOLUME_CONFIG 構造体

機能　EWF で保護されているボリュームの構成情報を格納します。

書式

```
typedef struct _EWF_VOLUME_CONFIG
{
    EWF_TYPE Type;
    EWF_STATE State;
    EWF_COMMAND BootCommand;

    UCHAR PersistentData[EWF_MAX_PERSISTENT_DATA];
    USHORT MaxLevels;
    ULONG ClumpSize;
    USHORT CurrentLevel;

    union
    {
        struct
        {
            LONGLONG DiskMapSize;
            LONGLONG DiskDataSize;
        } DiskOverlay;

        struct
        {
            LONGLONG RamDataSize;
        } RamOverlay;
    };

    ULONG MemMapSize;
    EWF_VOLUME_DESC VolumeDesc;
    EWF_LEVEL_DESC LevelDescArray[1];
} EWF_VOLUME_CONFIG, * PEWF_VOLUME_CONFIG;
```

メンバー

Type
　このボリュームのオーバーレイの種類を指定します。RAM、レジストリで説明された RAM、ディスクのいずれかになります。

State
　このボリュームのオーバーレイの状態を指定します。状態は有効または無効のいずれかになります。

BootCommand
　再起動時に実行するコマンドを指定します。

PersistentData
　無効、有効、復元の操作を存続させるデータ。永続データのサイズは、EWF_MAX_PERSISTENT_DATA（既定では 32 バイト）です。

MaxLevels
　オーバーレイのチェックポイントレベルの最大数。

ClumpSize
　クランプのサイズをバイトで表示したもの。ClumpSize は 512 バイトに設定する必要があります。

CurrentLevel
　現在のチェックポイントレベル。

DiskMapSize
　保護されたボリュームのディスクに含まれたマッピング データのサイズをバイトで表したもの（ディスク オーバーレイのみ）。

DiskDataSize
　保護されたボリュームのディスクに保存されたデータのサイズをバイトで表したもの（ディスク オーバーレイのみ）。

RamDataSize
　保護されたボリュームの RAM に保存されたデータのサイズをバイトで表したもの（RAM オーバーレイのみ）。

MemMapSize
　保護されたボリュームのメモリに含まれたマッピング データのサイズをバイトで表したもの。

VolumeDesc
　ボリュームデバイス名とボリューム ID。

LevelDescArray
　レベルの説明と終了時間、およびレベルのデータサイズ。エントリ数は、MaxLevels で指定します。

説明

この読み取り専用の構造体には、EWF で保護されたボリュームに関する構成情報が含まれます。

EwfMgrGetProtectedVolumeConfig 関数

機能 EWF で保護されたボリュームの構成情報を取得します。

書式
```
PEWF_VOLUME_CONFIG EwfMgrGetProtectedVolumeConfig (
    HANDLE hDevice
);
```

引数 hDevice
[in] EWF で保護されたボリュームへのハンドル。

戻り値 関数が成功したときの戻り値は、EWF_VOLUME_CONFIG 構造体へのポインタです。関数が失敗した場合の戻り値は NULL です。エラーの詳細情報については、GetLastError を呼び出します。

説明 この関数は、EWF で保護されているボリュームの構成情報を取得します。
呼び出し元は、LocalFree 関数を使用して、不要になった構造に割り当てられたメモリを解放してください。EWF が無効となっている場合、EWF_VOLUME_CONFIG 構造体の CurrentLevel メンバーは 0 に設定されます。

EwfMgrGetProtectedVolumeList 関数

機能 すべての EWF で保護されたボリュームのリストを取得します。

書式 PEWF_VOLUME_NAME_ENTRY EwfMgrGetProtectedVolumeList (VOID);

引数 なし

戻り値 この関数が正常に実行された場合、戻り値は EWF_VOLUME_NAME_ENTRY 一覧へのポインタになります。関数が失敗した場合の戻り値は NULL です。エラーの詳細情報については、GetLastError を呼び出します。

説明 この関数は、すべての EWF で保護されたボリューム名の一覧を取得します。
呼び出し元は、一覧が不要になったら、EwfMgrVolumeNameListDelete 関数を使用して一覧を解放する必要があります。

EwfMgrVolumeNameListIsEmpty 関数

機能 EWF ボリューム名リストが空かどうかを判別します。

書式
```
BOOL EwfMgrVolumeNameListIsEmpty (
    PEWF_VOLUME_NAME_ENTRY pVolumeNameList
);
```

引数 pVolumeNameList
[in] EWF_VOLUME_NAME_LIST へのポインタ。

戻り値 リストが空だと、戻り値は TRUE になります。リストが空でないと、戻り値は FALSE になります。

説明 この関数は、EWF ボリューム名リスト(EWF_VOLUME_NAME_LIST)が空かどうかを判断します。

EwfMgrVolumeNameEntryPop 関数

機能　EWF ボリューム名リストから現在のエントリを削除し、現在のメモリエントリを解放します。

書式
```
VOID EwfMgrVolumeNameEntryPop (
    PEWF_VOLUME_NAME_ENTRY* ppVolumeNameList
);
```

引数　ppVolumeNameList
[in, out] EWF_VOLUME_NAME_LIST へのポインタのポインタ。

戻り値　この関数は、空の一覧と共に呼び出される場合があります。この関数により、ppVolumeNameList リンクリストのヘッドノードも変更されます。

説明　この関数により、EWF のボリューム名一覧(EWF_VOLUME_NAME_LIST)の現在のエントリを削除し、そのメモリが解放されます。

EwfMgrVolumeNameListDelete 関数

機能 EWF ボリューム名リストのすべてのエントリを削除します。

書式
```
VOID EwfMgrVolumeNameListDelete (
    PEWF_VOLUME_NAME_ENTRY pVolumeNameList
);
```

引数 pVolumeNameList
[in] EWF_VOLUME_NAME_LIST へのポインタ。

戻り値 なし

説明 この関数により、EWF ボリュームの名前一覧(EWF_VOLUME_NAME_LIST)のすべてのエントリが削除されます。
この関数は、空の一覧と共に呼び出される場合があります。

４－３ EWF API関数の使用について

「AS シリーズ用 Windows Embedded Standard 2009 リカバリ/SDK DVD」に EWF API 関数を使用するためのヘッダファイル、ライブラリファイル、EWF API 関数を使用したサンプルコードを用意しています。開発用ファイルは一般的な C/C++言語用です。Microsoft Visual Studio など Windows API を使用できる C/C++言語の開発環境で使用することが可能です。DVD に含まれる開発用ファイルの内容を表 4-3-1 に示します。

表 4-3-1. リカバリ/SDK DVD EWF API開発用ファイル

DVD-ROM のフォルダ	内容
¥SDK¥EWFAPI¥Develop	EWF API 関数を使用するためのヘッダファイル、ライブラリファイルを格納しています。
¥SDK¥EWFAPI¥Sample	EWF API 関数を使用したサンプルコードを格納しています。

４－４ サンプルコード

４－４－１ EWFの有効／無効

● EWF の有効

「¥SDK¥EWFAPI¥Sample¥EwfEnable.cpp」では、EwfMgrEnable 関数を使用して C ドライブの EWF を有効にしています。リスト 4-4-1-1 にサンプルコードを示します。

リスト 4-4-1-1. EWFの有効

```
#include <windows.h>
#include <stdio.h>
#include <stdlib.h>

#define EWFIMP
#include "ewfapi.h"

/*++
ルーチンの説明:
  このルーチンでは、EwfMgrEnable の用法を示す。
  保護されたボリュームへのハンドルを開き、次にコマンドを発行する。

  EwfMgrEnable:
    コマンド: EWF を有効にします。

    再起動: 必要。

引数:
  szProVolName
    EwfMgrGetProtectedVolumeList から取得するボリューム名または
    前に形式 ¥¥ が付いた EwfMgrGetOverlayStoreConfig から取得するデバイス名。
    ¥C: ここで C は、保護されたボリュームのドライブ名を表す。

戻り値:
  成功した場合は TRUE、失敗した場合は FALSE。
--*/

BOOL DoEwfEnable(LPWSTR szProVolName)
```

```
{
    HANDLE hProVol = INVALID_HANDLE_VALUE;
    BOOL bResult = FALSE;

    // ボリューム名を使ってこの保護されたボリュームへのハンドルを開く。
    hProVol = EwfMgrOpenProtected(szProVolName);
    if(hProVol == INVALID_HANDLE_VALUE){
        wprintf(L"EwfMgrOpenProtected failed LE = %u¥n", GetLastError());
        goto __exit;
    }

    // EWF を有効にする。
    bResult = EwfMgrEnable(hProVol);
    if(!bResult){
        wprintf(L"EwfMgrEnable failed LE = %u¥n", GetLastError());
        goto __exit;
    }
    wprintf(L"EwfMgrEnable succeeded¥n");

__exit:
    if(hProVol != INVALID_HANDLE_VALUE){
        EwfMgrClose(hProVol);
    }
    return bResult;
}

int main(void)
{
    BOOL bRet;

    // C ドライブを EWF 有効にします。
    bRet = DoEwfEnable(L"¥¥¥.¥¥C:");

    return (bRet ? 0 : -1);
}
```

● EWF の無効

「¥SDK¥EWFAPI¥Sample¥EwfDisable.cpp」では、EwfMgrDisable 関数を使用して C ドライブの EWF を無効にしています。リスト 4-4-1-2 にサンプルコードを示します。

リスト 4-4-1-2. EWFの無効

```
#include <windows.h>
#include <stdio.h>
#include <stdlib.h>

#define EWFIMP
#include "ewfapi.h"

/*++
ルーチンの説明:
```

```
  このルーチンでは、EwfMgrDisable の用法を示す。
  保護されたボリュームへのハンドルを開き、次にコマンドを発行する。

  EwfMgrDisable:
    コマンド: 指定したボリュームで現在有効になっているオーバーレイ
              を無効にする。
    再起動: 必要。

引数:
  szProVolName
    EwfMgrGetProtectedVolumeList から取得するボリューム名または
    前に形式 ¥¥ が付いた EwfMgrGetOverlayStoreConfig から取得するデバイス名。
    ¥C: ここで C は、保護されたボリュームのドライブ名を表す。

戻り値:
  成功した場合は TRUE、失敗した場合は FALSE。
--*/

BOOL DoEwfDisable(LPWSTR szProVolName)
{
    HANDLE hProVol = INVALID_HANDLE_VALUE;
    BOOL bResult = FALSE;

    // ボリューム名を使ってこの保護されたボリュームへのハンドルを開く。
    hProVol = EwfMgrOpenProtected(szProVolName);
    if(hProVol == INVALID_HANDLE_VALUE){
        wprintf(L"EwfMgrOpenProtected failed LE = %u¥n", GetLastError());
        goto __exit;
    }

    // EWF を無効にする。
    // 2 番目のパラメータはコミットするかどうか。
    bResult = EwfMgrDisable(hProVol, FALSE);
    if(!bResult){
        wprintf(L"EwfMgrDisable failed LE = %u¥n", GetLastError());
        goto __exit;
    }
    wprintf(L"EwfMgrDisable succeeded¥n");

__exit:
    if(hProVol != INVALID_HANDLE_VALUE){
        EwfMgrClose(hProVol);
    }
    return bResult;
}

int main(void)
{
    BOOL bRet;

    // C ドライブを EWF 無効にします。
    bRet = DoEwfDisable(L"¥¥¥.¥¥C:");
```

```
    return (bRet ? 0 : -1);
}
```

４－４－２　オーバーレイデータのコミット

● C ドライブのオーバーレイデータのコミット

「¥SDK¥EWFAPI¥Sample¥EwfCommit.cpp」では、EwfMgrCommit 関数を使用して C ドライブのオーバーレイデータをコミットしています。リスト 4-4-2-1 にサンプルコードを示します。

リスト 4-4-2-1. オーバーレイデータコミット

```
#include <windows.h>
#include <stdio.h>
#include <stdlib.h>

#define EWFIMP
#include "ewfapi.h"

/*++
ルーチンの説明:
  このルーチンでは、EwfMgrCommit の用法を示す。
  保護されたボリュームへのハンドルを開き、次にコマンドを発行する。

  EwfMgrCommit:
    コマンド: オーバーレイの情報をコミットします。

    再起動: 必要。

引数:
  szProVolName
    EwfMgrGetProtectedVolumeList から取得するボリューム名または
    前に形式 ¥¥ が付いた EwfMgrGetOverlayStoreConfig から取得するデバイス名。
    ¥C: ここで C は、保護されたボリュームのドライブ名を表す。

戻り値:
  成功した場合は TRUE、失敗した場合は FALSE。
--*/

BOOL DoEwfCommit(LPWSTR szProVolName)
{
    HANDLE hProVol = INVALID_HANDLE_VALUE;
    BOOL bResult = FALSE;

    // ボリューム名を使ってこの保護されたボリュームへのハンドルを開く。
    hProVol = EwfMgrOpenProtected(szProVolName);
    if(hProVol == INVALID_HANDLE_VALUE){
        wprintf(L"EwfMgrOpenProtected failed LE = %u¥n", GetLastError());
        goto __exit;
    }
```

```
    // オーバーレイの情報をコミットします。
    bResult = EwfMgrCommit(hProVol);
    if(!bResult){
        wprintf(L"EwfMgrCommit failed LE = %u¥n", GetLastError());
        goto __exit;
    }
    wprintf(L"EwfMgrCommit succeeded¥n");

__exit:
    if(hProVol != INVALID_HANDLE_VALUE){
        EwfMgrClose(hProVol);
    }
    return bResult;
}

int main(void)
{
    BOOL bRet;

    // C ドライブをコミットします。
    bRet = DoEwfCommit(L"¥¥¥¥.¥¥C:");

    return (bRet ? 0 : -1);
}
```

● C ドライブのオーバーレイデータのコミットと EWF 無効（LIVE 処理）

「¥SDK¥EWFAPI¥Sample¥EwfCommitAndDisableLive.cpp」では、EwfMgrCommitAndDisableLive 関数を使用して C ドライブのオーバーレイデータをコミットと EWF 無効を行っています。処理は即座に行われます。リスト 4-4-2-2 にサンプルコードを示します。

リスト 4-4-2-2. コミット+EWF無効

```
#include <windows.h>
#include <stdio.h>
#include <stdlib.h>

#define EWFIMP
#include "ewfapi.h"

/*++
ルーチンの説明:
  このルーチンでは、EwfMgrCommitAndDisableLive の用法を示す。
  保護されたボリュームへのハンドルを開き、次にコマンドを発行する。

  EwfMgrCommitAndDisableLive:
    コマンド: オーバーレイの情報をコミットし、EWF を無効にします。
    処理はすぐに実行されます。

    再起動: 不要

引数:
```

```
  szProVolName
    EwfMgrGetProtectedVolumeList から取得するボリューム名または
    前に形式 ¥¥ が付いた EwfMgrGetOverlayStoreConfig から取得するデバイス名。
    ¥C: ここで C は、保護されたボリュームのドライブ名を表す。

戻り値:
  成功した場合は TRUE、失敗した場合は FALSE。
--*/

BOOL DoEwfCommitAndDisableLive(LPWSTR szProVolName)
{
    HANDLE hProVol = INVALID_HANDLE_VALUE;
    BOOL bResult = FALSE;

    // ボリューム名を使ってこの保護されたボリュームへのハンドルを開く。
    hProVol = EwfMgrOpenProtected(szProVolName);
    if(hProVol == INVALID_HANDLE_VALUE){
        wprintf(L"EwfMgrOpenProtected failed LE = %u¥n", GetLastError());
        goto __exit;
    }

    // コミットして EWF を無効にします。
    bResult = EwfMgrCommitAndDisableLive(hProVol);
    if(!bResult){
        wprintf(L"EwfMgrCommitAndDisableLive failed LE = %u¥n", GetLastError());
        goto __exit;
    }
    wprintf(L"EwfMgrCommitAndDisableLive succeeded¥n");

__exit:
    if(hProVol != INVALID_HANDLE_VALUE){
        EwfMgrClose(hProVol);
    }
    return bResult;
}

int main(void)
{
    BOOL bRet;

    // C ドライブをコミットして EWF 無効にします。
    bRet = DoEwfCommitAndDisableLive(L"¥¥¥.¥¥C:");

    return (bRet ? 0 : -1);
}
```

第５章　組込みシステム機能ドライバ

AS シリーズには、組込みシステム向けに独自の機能が搭載されています。AS シリーズ用 Windows Embedded Standard 2009 には、これら機能にアクセスするためのドライバを用意しています。このドライバを使用することでアプリケーションからこれらの機能を使用することができます。本章では、組込みシステム機能ドライバの使用方法について説明します。

５－１　　ドライバの使用について

５－１－１　開発用ファイル

「AS シリーズ用 Windows Embedded Standard 2009 リカバリ/SDK DVD」にドライバにアクセスするためのヘッダファイルとドライバを使用したサンプルコードを用意しています。開発用ファイルは一般的な C/C++言語用です。Microsoft Visual Studio など Windows API を使用できる C/C++言語の開発環境で使用することが可能です。DVD に含まれる開発用ファイルの内容を表 5-1-1-1 に示します。

表 5-1-1-1. リカバリ/SDK DVD 開発用ファイル

DVD-ROM のフォルダ	内容
¥SDK¥Algo¥Develop	ドライバアクセスに必要なヘッダファイルを格納しています。
¥SDK¥Algo¥Sample¥Sample_BackLight	LCD バックライト制御のサンプルコードです。
¥SDK¥Algo¥Sample¥Sample_Buzzer	ブザー制御のサンプルコードです。
¥SDK¥Algo¥Sample¥Sample_GenIO	汎用入出力制御のサンプルコードです。
¥SDK¥Algo¥Sample¥Sample_Interrupt	タイマ割込み機能 IN1 割込み機能サンプルコードです。
¥SDK¥Algo¥Sample¥Sample_Reset	IN0 リセット機能のサンプルコードです。
¥SDK¥Algo¥Sample¥Sample_SwWdt	ソフトウェア・ウォッチドッグのサンプルコードです。

５－１－２　DeviceIoControlについて

AS シリーズ専用機能のドライバは、ほとんどのものがドライバの機能にアクセスするために DeviceIoControl 関数を使用します。以下にその書式を示します。関数仕様の詳細は、Windows API の仕様を参照してください。

コントロールコード、コントロールコードに対応する動作及び引数は、ドライバ毎にリファレンスを用意していますので、各ドキュメントを参照してください。

関数書式

```
BOOL DeviceIoControl (
      HANDLE        hDevice,
      DWORD         dwIoControlCode,
      LPVOID        lpInBuf,
      DWORD         nInBufSize,
      LPVOID        lpOutBuf,
      DWORD         nOutBufSize,
      LPDWORD       lpBytesReturned,
      LPOVERLAPPED  lpOverlapped
);
```

パラメータ

hDevice	：デバイス、ファイル、ディレクトリいずれかのハンドル
dwIoControlCode	：実行する動作のコントロールコード
lpInBuf	：入力データを供給するバッファへのポインタ
nInBufSize	：入力バッファのバイト単位のサイズ
lpOutBuf	：出力データを受け取るバッファへのポインタ
nOutBufSize	：出力バッファのバイト単位のサイズ
lpBytesReturned	：lpOutBuf に格納されるバイト数を受け取る変数へのポインタ
lpOverlapped	：非同期動作を表す構造体へのポインタ

５－２　　タイマ割込み機能

５－２－１　　タイマ割込み機能について

AS シリーズには、ハードウェアによるタイマ割込み機能が実装されています。I/O ポート上にマッピングされた制御レジスタを操作することによって、指定した時間で周期的に割込みを発生させることができます。

５－２－２　　タイマドライバについて

タイマドライバはタイマ割込み機能を、ユーザーアプリケーションから利用できるようにします。ユーザーアプリケーションから、タイマの設定とイベントによるタイマ通知の機能を使用することができます。

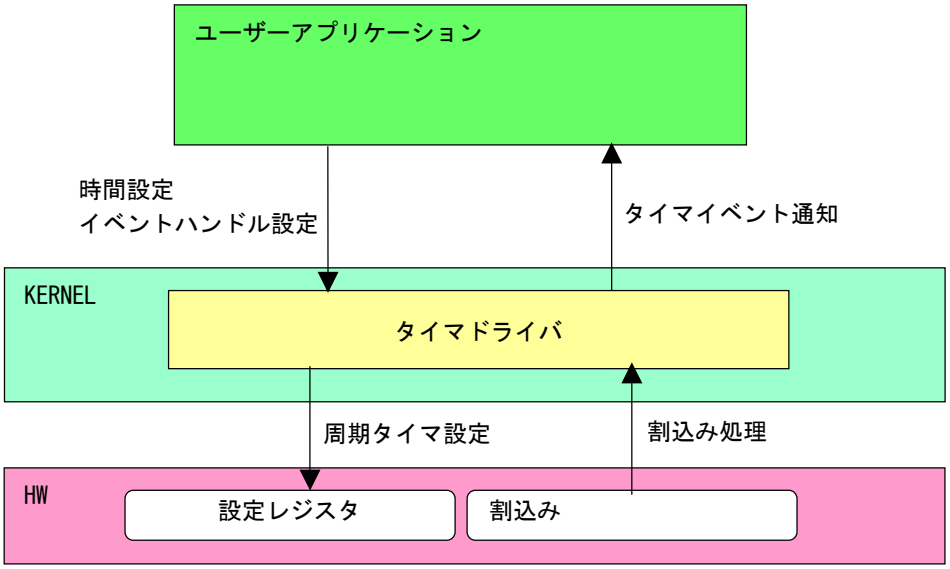

図 5-2-2-1. タイマドライバ

５－２－３　タイマデバイス

タイマドライバはタイマデバイスを生成します。ユーザーアプリケーションは、デバイスファイルにアクセスすることによってタイマ機能を操作します。

タイマデバイス

デバイスファイル

¥¥.¥FpgaTimer

説明

タイマ時間設定、タイマ開始、停止を行うことが出来ます。

レジストリ設定

[KEY]　　HKEY_LOCAL_MACHINE¥SYSTEM¥CurrentControlSet¥Services¥FpgaTimer
[VALUE:DWORD]　　TimerResolution
タイマ解像度をミリ秒単位で設定します。ドライバ起動時(OS起動時)にこの値を参照しタイマ解像度を設定します。(デフォルト値: 10)

CreateFile

デバイスファイル(¥¥.¥FpgaTimer)をオープンし、デバイスハンドルを取得します。

```
hTimer = CreateFile(
    "¥¥¥¥.¥¥FpgaTimer",
    GENERIC_READ | GENERIC_WRITE,
    FILE_SHARE_READ | FILE_SHARE_WRITE,
    NULL,
    OPEN_EXISTING,
    0,
    NULL
);
```

CloseHandle

デバイスハンドルをクローズします。

```
CloseHandle(hTimer);
```

ReadFile

使用しません。

WriteFile

使用しません。

DeviceIoControl

- **IOCTL_FPGATIMER_START**
  タイマを開始します。
- **IOCTL_FPGATIMER_STOP**
  タイマを停止します。
- **IOCTL_FPGATIMER_SETCONFIG**
  タイマを設定します。
- **IOCTL_FPGATIMER_GETCONFIG**
  現在のタイマ設定を取得します。

５－２－４　タイマドライバの動作

① 起動時に 10msec(レジストリ設定で変更可能)の周期割込み設定を行います。
② オープンされたデバイスハンドル毎に、タイマ情報を作成しタイマ情報テーブルへ追加します。オープンできるハンドルはシステム全体で 16 までとなります。タイマ情報テーブルへの追加はオープンした順番で追加されます。
③ ユーザーアプリケーションからの設定をタイマ情報テーブルへ反映させます。
④ 周期割込みが発生したらタイマ情報テーブルを参照し、各タイマ情報のカウント値を加算します。
⑤ カウント値が設定値に達したものは、イベントハンドルでタイマ通知を行います。カウント加算、イベント通知処理はタイマ情報テーブルの順番で処理されます。

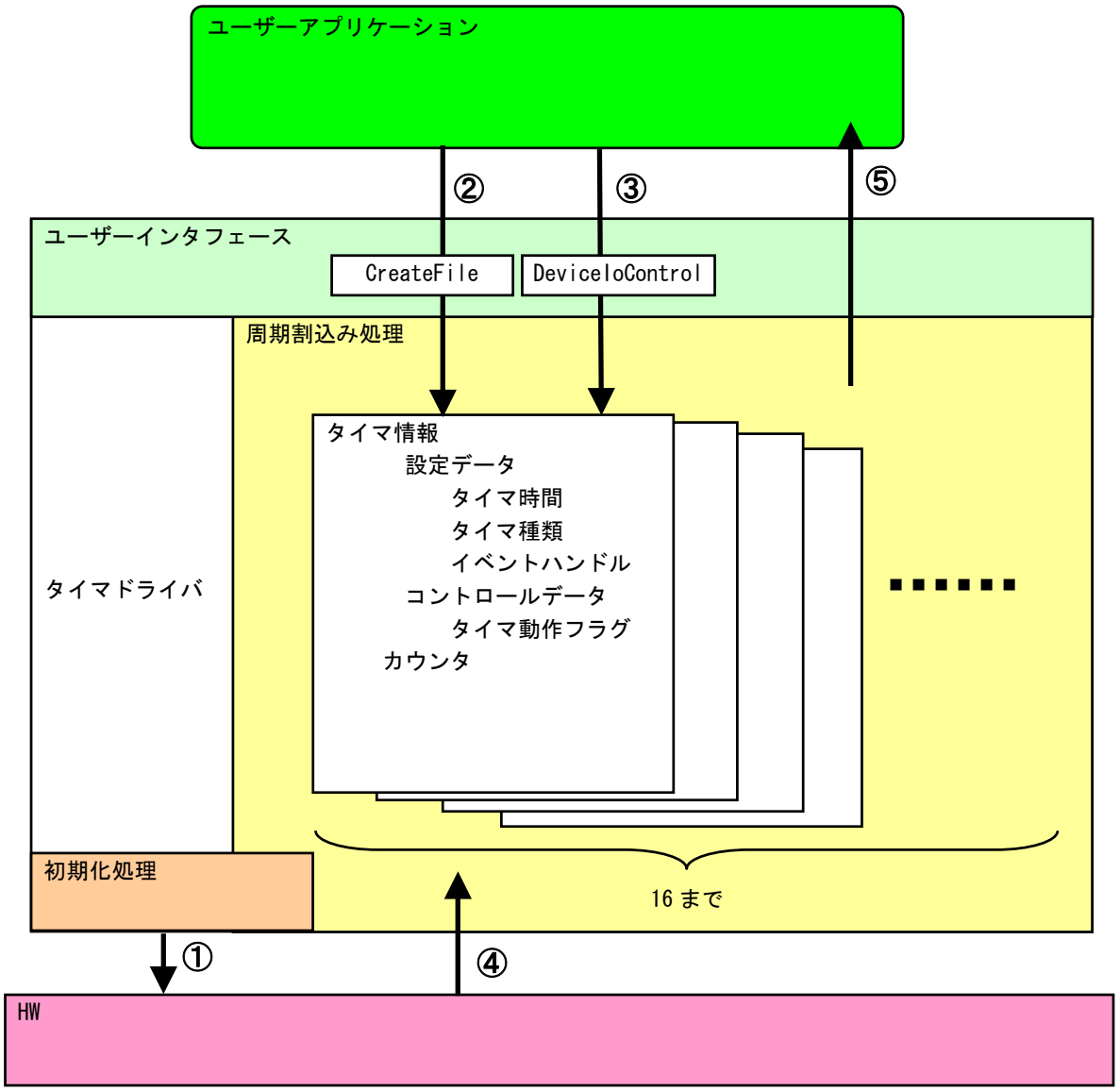

図 5-2-4-1. タイマドライバの動作

５－２－５　ドライバ使用手順

基本的な使用手順を以下に示します。タイマ通知用イベントハンドルを作成後、タイマデバイスにイベントハンドル、タイマ時間を設定します。タイマ通知用イベントハンドルでのイベント待ち準備が整ったところで、タイマをスタートさせます。

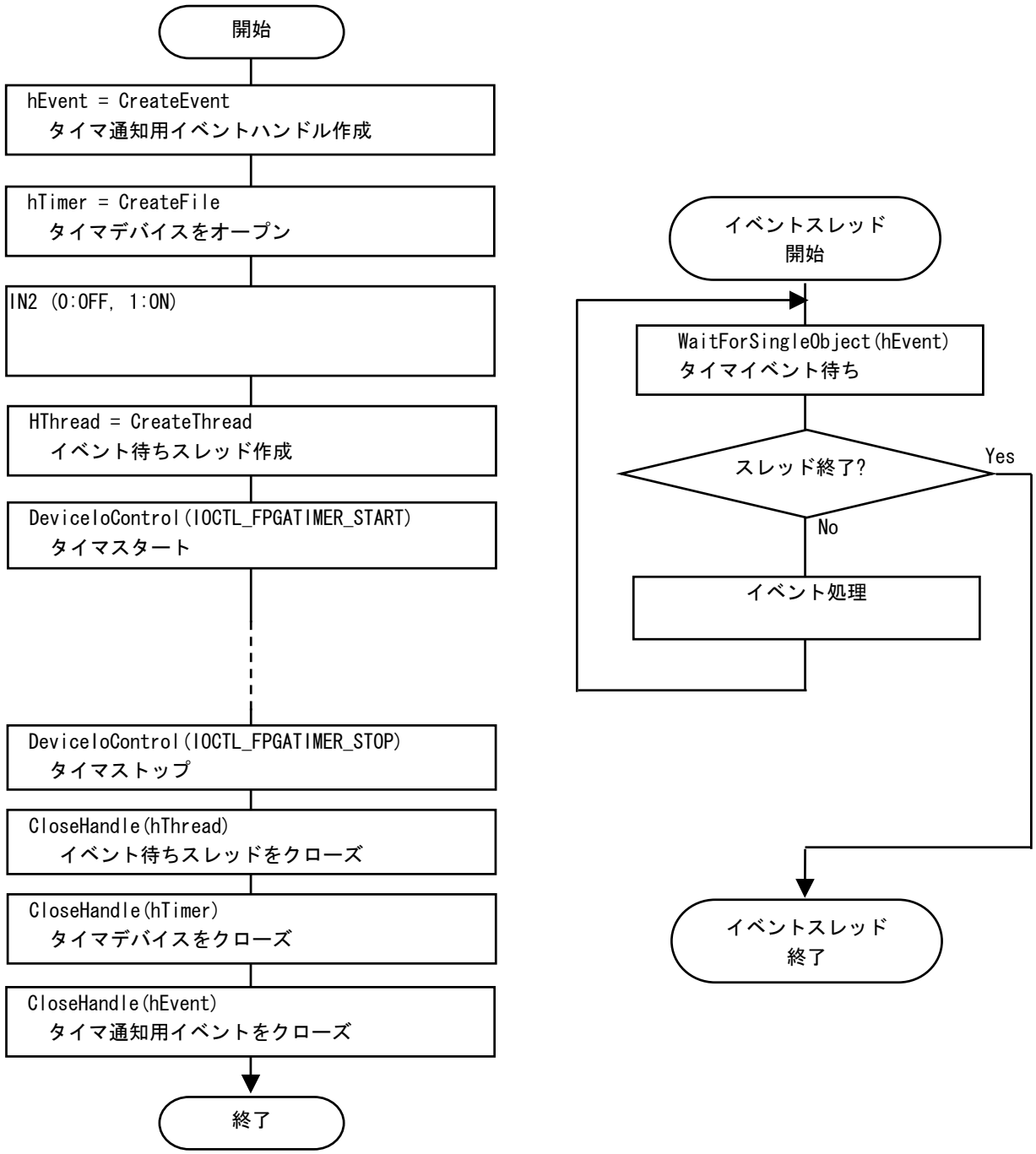

図 5-2-5-1. ドライバ使用手順

５－２－６　DeviceIoControlリファレンス

IOCTL_FPGATIMER_START

機能

タイマ処理を開始します。

パラメータ

lpInBuf　　：NULL を指定します。
NInBufSize　　：0 を指定します。
LpOutBuf　　：NULL を指定します。
NOutBufSize　　：0 を指定します。
LpBytesReturned　：実際の出力バイト数を受け取る変数へのポインタ。
LpOverlapped　　：NULL を指定します。

戻り値

処理が成功すると TRUE を返します。失敗の場合は FALSE を返します。

説明

IOCTL_FPGATIMER_SETCONFIG に設定した内容でタイマ処理を開始します。このコントロールを実行させる前に、必ず IOCTL_FPGATIMER_SETCONFIG を実行するようにしてください。

IOCTL_FPGATIMER_STOP

機能

タイマ処理を停止します。

パラメータ

lpInBuf	:	NULL を指定します。
NInBufSize	:	0 を指定します。
LpOutBuf	:	NULL を指定します。
NOutBufSize	:	0 を指定します。
LpBytesReturned	:	実際の出力バイト数を受け取る変数へのポインタ。
LpOverlapped	:	NULL を指定します。

戻り値

処理が成功すると TRUE を返します。失敗の場合は FALSE を返します。

説明

タイマ処理を停止します。タイマ通知イベントハンドルを破棄する前には、このコントロールを実行してタイマ通知を停止するようにしてください。

IOCTL_FPGATIMER_SETCONFIG

機能

タイマの設定を行います。

パラメータ

lpInBuf	:	FPGATIMER_CONFIG を格納するためのポインタ。
NInBufSize	:	FPGATIMER_CONFIG のサイズを指定します。
LpOutBuf	:	NULL を指定します。
NOutBufSize	:	0 を指定します。
LpBytesReturned	:	実際の出力バイト数を受け取る変数へのポインタ。
LpOverlapped	:	NULL を指定します。

FPGATIMER_CONFIG

```
typedef struct {
    HANDLE      hEvent;
    ULONG       Type;
    ULONG       DueTime;
} FPGATIMER_CONFIG, *P FPGATIMER_CONFIG;
```

hEvent	: タイマ通知用イベントハンドル
Type	: タイマ動作タイプ [0: 一回で終了, 1: 繰り返し]
DueTime	: タイマ時間

戻り値

処理が成功すると TRUE を返します。失敗の場合は FALSE を返します。

説明

タイマの設定を行います。IOCTL_FPGATIMER_START でタイマを開始する前に、このコントロールを実行してタイマの設定を行うようにしてください。

IOCTL_FPGATIMER_GETCONFIG

機能

タイマ設定を取得します。

パラメータ

lpInBuf	:	NULL を指定します。
NInBufSize	:	0 を指定します。
LpOutBuf	:	FPGATIMER_CONFIG を格納するためのポインタ。
NOutBufSize	:	FPGATIMER_CONFIG のサイズを指定します。
LpBytesReturned	:	実際の出力バイト数を受け取る変数へのポインタ。
LpOverlapped	:	NULL を指定します。

FPGATIMER_CONFIG

```
typedef struct {
    HANDLE      hEvent;
    ULONG       Type;
    ULONG       DueTime;
} FPGATIMER_CONFIG, *P FPGATIMER_CONFIG;
```

hEvent	: タイマ通知用イベントハンドル
Type	: タイマ動作タイプ［0: 一回で終了, 1: 繰り返し］
DueTime	: タイマ時間

戻り値

処理が成功すると TRUE を返します。失敗の場合は FALSE を返します。

説明

現在のタイマ設定値を取得します。

５－２－７　サンプルコード

「¥SDK¥Algo¥Sample¥Sample_Interrupt¥FpgaTimer」にタイマ割込み機能を使用したサンプルコードを用意しています。リスト 5-2-7-1 にサンプルコードを示します。サンプルコードでは、10 個の周期タイマを使用してタイマイベント通知を確認しています。

リスト 5-2-7-1. タイマ割込み機能

```
/**
    タイマ割込み制御サンプルソース
**/
#include <windows.h>
#include <winioctl.h>
#include <stdio.h>
#include <stdlib.h>
#include <mmsystem.h>
#include <conio.h>

#include "..¥Common¥FpgaTimerDD.h"

#define TIMERDRIVER_FILENAME    "¥¥¥¥.¥¥FpgaTimer"
#define MAX_TIMEREVENT          10

//---------------------------------------------------------------------------
typedef struct {
    int No;
    HANDLE  hEvent;
    HANDLE  hThread;
    volatile BOOL fStart;
    volatile BOOL fFinish;
    HANDLE  hTimer;
    FPGATIMER_CONFIG Config;
} TIMEREVENT_INFO, *PTIMEREVENT_INFO;

//---------------------------------------------------------------------------
/*
 * 割込みハンドラ
 */
DWORD WINAPI TimerEventProc(void *pData)
{
    PTIMEREVENT_INFO    info = (PTIMEREVENT_INFO)pData;
    DWORD ret;

    printf("TimerEventProc: Timer%02d: Start¥n", info->No);

    info->fFinish = FALSE;
    while(1){
        if(WaitForSingleObject(info->hEvent, INFINITE) != WAIT_OBJECT_0){
            break;
        }
        if(!info->fStart){
            break;
        }
        printf("TimerEventProc: Timer%02d: Tick(%d)¥n", info->No, timeGetTime());
```

```
    }
    info->fFinish = TRUE;

    printf("TimerEventProc: Timer%02d: Finish¥n", info->No);
    return 0;
}

//---------------------------------------------------------------------------
BOOL CreateTimerEventInfo(int No, PTIMEREVENT_INFO info)
{
    DWORD   thrd_id;
    ULONG   retlen;
    BOOL    ret;

    info->No = No;
    info->hEvent = NULL;
    info->hThread = NULL;
    info->fStart = FALSE;
    info->fFinish = FALSE;
    info->hTimer = INVALID_HANDLE_VALUE;

    /*
     * イベントオブジェクトの作成
     */
    info->hEvent = CreateEvent(NULL, FALSE, FALSE, NULL);
    if(info->hEvent == NULL){
        printf("CreateTimerEventInfo: CreateEvent: NG¥n");
        return FALSE;
    }

    /*
     * イベントスレッドを生成
     */
    info->hThread = CreateThread(
                        (LPSECURITY_ATTRIBUTES)NULL,
                        0,
                        (LPTHREAD_START_ROUTINE)TimerEventProc,
                        (LPVOID)info,
                        CREATE_SUSPENDED,
                        &thrd_id
                    );
    if(info->hThread == NULL){
        CloseHandle(info->hEvent);
        printf("CreateTimerEventInfo: CreateThread: NG¥n");
        return FALSE;
    }

    /*
     * ドライバオブジェクトの作成
     */
    info->hTimer = CreateFile(
                        TIMERDRIVER_FILENAME,
```

```
                        GENERIC_READ | GENERIC_WRITE,
                        FILE_SHARE_READ | FILE_SHARE_WRITE,
                        NULL,
                        OPEN_EXISTING,
                        0,
                        NULL
                    );
    if(info->hTimer == INVALID_HANDLE_VALUE){
        CloseHandle(info->hThread);
        CloseHandle(info->hEvent);
        printf("CreateTimerEventInfo: CreateFile: NG¥n");
        return FALSE;
    }

    /*
     * ドライバに初期値を設定
     */
    info->Config.hEvent = info->hEvent;
    info->Config.Type = 1;
    info->Config.DueTime = 200 * (No + 1);
    ret = DeviceIoControl(
                info->hTimer,
                IOCTL_FPGATIMER_SETCONFIG,
                &info->Config,
                sizeof(FPGATIMER_CONFIG),
                NULL,
                0,
                &retlen,
                NULL
          );
    if(!ret){
        CloseHandle(info->hTimer);
        CloseHandle(info->hThread);
        CloseHandle(info->hEvent);
        return FALSE;
    }

    return TRUE;
}

//---------------------------------------------------------------------------
void StartTimer(PTIMEREVENT_INFO info)
{
    ULONG   retlen;

    /*
     * イベントスレッドのリジューム
     */
    info->fStart = TRUE;
    ResumeThread(info->hThread);

    /*
```

```
     * タイマの開始
     */
    DeviceIoControl(
        info->hTimer,
        IOCTL_FPGATIMER_START,
        NULL,
        0,
        NULL,
        0,
        &retlen,
        NULL
    );
}

//---------------------------------------------------------------------------
void DeleteTimer(PTIMEREVENT_INFO info)
{
    ULONG   retlen;

    /*
     * イベントスレッドの Terminate
     */
    info->fStart = FALSE;
    SetEvent(info->hEvent);

    /*
     * タイマの停止
     */
    DeviceIoControl(
        info->hTimer,
        IOCTL_FPGATIMER_STOP,
        NULL,
        0,
        NULL,
        0,
        &retlen,
        NULL
    );

    while(!info->fFinish){
        Sleep(10);
    }

    /*
     * ハンドルのクローズ
     */
    CloseHandle(info->hThread);
    CloseHandle(info->hEvent);
    CloseHandle(info->hTimer);
}

//---------------------------------------------------------------------------
```

```
int main(void)
{
    int i;
    int c;
    TIMEREVENT_INFO info[MAX_TIMEREVENT];

    for(i = 0; i < MAX_TIMEREVENT; i++){
        if(!CreateTimerEventInfo(i, &info[i])){
            printf("CreatTimerEvent: NG: %d¥n", i);
            return -1;
        }
    }
    for(i = 0; i < MAX_TIMEREVENT; i++){
        StartTimer(&info[i]);
    }

    while(1){
        if(kbhit()){
            c = getch();
            if(c == 'q' || c == 'Q')
                break;
        }
    }

    for(i = 0; i < MAX_TIMEREVENT; i++){
        DeleteTimer(&info[i]);
    }

    return 0;
}

//---------------------------------------------------------------------------
```

５－３　　ブザー

５－３－１　　ブザーについて

ブザーの ON/OFF、タッチパネルのタッチ音の ON/OFF を行うことができます。

５－３－２　　ブザードライバについて

ブザードライバは I/O ポート上にマッピングされた制御レジスタを、ユーザーアプリケーションから操作できるようにします。ユーザーアプリケーションから、ブザーの ON/OFF 設定とタッチパネルのタッチ音の ON/OFF を変更することができます。

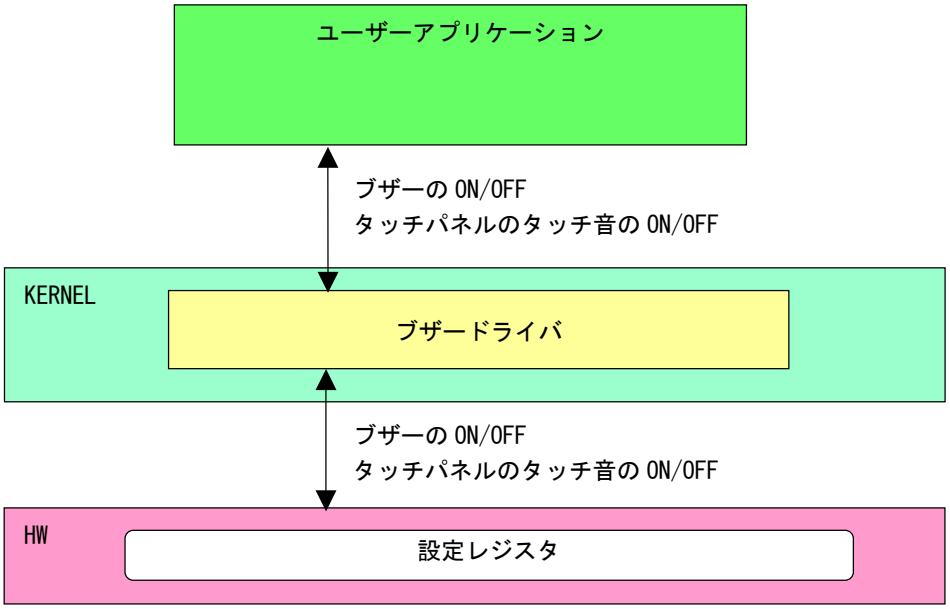

図 5-3-2-1. ブザードライバ

５－３－３　ブザーデバイス

ブザードライバはブザーデバイスを生成します。ユーザーアプリケーションは、デバイスファイルにアクセスすることによってブザー機能を操作します。

ブザーデバイス

デバイスファイル

¥¥.¥BuzzDrv

説明

ブザーのON/OFF、タッチパネルのタッチ音のON/OFFを行うことが出来ます。

レジストリ設定

[KEY]　　HKEY_LOCAL_MACHINE¥SYSTEM¥CurrentControlSet¥Services¥BuzzDrv
[VALUE:DWORD]　Auto　　有効:1 / 無効:0

タッチパネルのタッチ音の有効/無効を設定します。ドライバ起動時(OS起動時)にこの値を参照し、タッチ音を設定します。(デフォルト値: 有効)

CreateFile

デバイスファイル(¥¥.¥BuzzDrv)をオープンし、デバイスハンドルを取得します。

```
hBuzz = CreateFile(
    "¥¥¥¥.¥¥BuzzDrv",
    GENERIC_READ | GENERIC_WRITE,
    FILE_SHARE_READ | FILE_SHARE_WRITE,
    NULL,
    OPEN_EXISTING,
    0,
    NULL
);
```

CloseHandle

デバイスハンドルをクローズします。

```
CloseHandle(hBuzz);
```

ReadFile

使用しません。

WriteFile

使用しません。

DeviceIoControl

- **IOCTL_BUZZDRV_GETBUZZ**
  ブザーのON/OFFを取得します。
- **IOCTL_BUZZDRV_SETBUZZ**
  ブザーのON/OFFを設定します。
- **IOCTL_BUZZDRV_GETAUTO**
  タッチパネルのタッチ音を取得します。
- **IOCTL_BUZZDRV_SETAUTO**
  タッチパネルのタッチ音を設定します。

５－３－４　DeviceIoControlリファレンス

IOCTL_BUZZDRV_SETBUZZ

機能

ブザーの ON/OFF を設定します。

パラメータ

lpInBuf	:	ブザーON/OFF 情報を格納するポインタを指定します。
NInBufSize	:	ブザーON/OFF 情報を格納するポインタのサイズを指定します。
LpOutBuf	:	NULL を指定します。
NOutBufSize	:	0 を指定します。
LpBytesReturned	:	実際の出力バイト数を受け取る変数へのポインタを指定します。
LpOverlapped	:	NULL を指定します。

ブザーON/OFF 情報

データタイプ	:	ULONG
データサイズ	:	4 バイト
内容	:	1: ON, 0: OFF

戻り値

処理が成功すると TRUE を返します。失敗の場合は FALSE を返します。

説明

ブザー音の ON/OFF の設定を行います。

注意

アプリケーション毎にブザーの ON/OFF が可能ですので、アプリケーション 1 がブザーを OFF しても、アプリケーション 2 が ON している場合はブザーが鳴ったままになります。アプリケーションでブザーを ON した状態でアプリケーションを終了(ブザードライバをクローズ)した場合は、強制的に OFF になりますので注意してください。

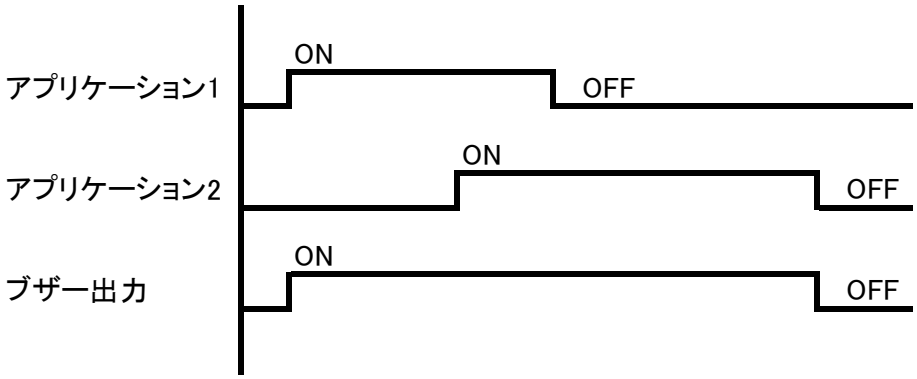

図 5-3-4-1. タイミングチャート

IOCTL_BUZZDRV_GETBUZZ

機能

ブザーの ON/OFF を取得します。

パラメータ

lpInBuf	:	NULL を指定します。
NInBufSize	:	0 を指定します。
LpOutBuf	:	ブザーON/OFF 情報を格納するポインタを指定します。
NOutBufSize	:	ブザーON/OFF 情報を格納するポインタのサイズを指定します。
LpBytesReturned	:	実際の出力バイト数を受け取る変数へのポインタを指定します。
LpOverlapped	:	NULL を指定します。

ブザーON/OFF 情報

データタイプ	:	ULONG
データサイズ	:	4 バイト
内容	:	1: ON, 0: OFF

戻り値

処理が成功すると TRUE を返します。失敗の場合は FALSE を返します。

説明

現状のブザーの ON/OFF の状態を取得できます。
ブザーが OFF にも関わらずブザーが鳴っている場合は、他のアプリケーションでブザーを ON しています。

IOCTL_BUZZDRV_SETAUTO

機能

タッチパネルのタッチ音の有効/無効の設定を行います。

パラメータ

lpInBuf	:	タッチパネル音情報を格納するポインタを指定します。
NInBufSize	:	タッチパネル音情報を格納するポインタのサイズを指定します。
LpOutBuf	:	NULL を指定します。
NOutBufSize	:	0 を指定します。
LpBytesReturned	:	実際の出力バイト数を受け取る変数へのポインタ。
LpOverlapped	:	NULL を指定します。

タッチパネル音情報

データタイプ	:	ULONG
データサイズ	:	4 バイト
内容	:	1: ON, 0: OFF

戻り値

処理が成功すると TRUE を返します。失敗の場合は FALSE を返します。

説明

タッチパネルのタッチ音の有効/無効の設定を行います。
タッチ音を ON にする場合は、タッチパネル音情報を格納するポインタに 1、OFF にする場合は 0 を設定してから DeviceIoControl を実行してください。

IOCTL_BUZZDRV_GETAUTO

機能

タッチパネルのタッチ音の有効/無効の取得を行います。

パラメータ

lpInBuf	:	NULL を指定します。
NInBufSize	:	0 を指定します。
LpOutBuf	:	タッチパネル音情報を格納するポインタを指定します。
NOutBufSize	:	タッチパネル音情報を格納するポインタのサイズを指定します。
LpBytesReturned	:	実際の出力バイト数を受け取る変数へのポインタを指定します。
LpOverlapped	:	NULL を指定します。

タッチパネル音情報

データタイプ	:	ULONG
データサイズ	:	4 バイト
内容	:	1: ON, 0: OFF

戻り値

処理が成功すると TRUE を返します。失敗の場合は FALSE を返します。

説明

タッチパネルのタッチ音の有効/無効の取得を行います。

５－３－５　サンプルコード

●ブザーON/OFF

「¥SDK¥Algo¥Sample¥Sample_Buzzer¥BuzzerOnOff」にブザーON/OFF 制御を行うサンプルコードを用意しています。リスト 5-3-5-1 にサンプルコードを示します。

リスト 5-3-5-1. ブザーON/OFF

```
/**
    ブザーの ON/OFF 制御サンプルソース
**/
#include <windows.h>
#include <winioctl.h>
#include <stdio.h>
#include <stdlib.h>
#include <mmsystem.h>
#include <conio.h>

#include "..¥Common¥BuzzDD.h"

#define DRIVER_FILENAME "¥¥¥¥.¥¥BuzzDrv"

int main(int argc, char **argv)
{
    HANDLE hBuzzer;
    ULONG retlen;
    BOOL ret;
    ULONG temp;

    /*
     * 起動引数からブザーの ON/OFF 変更値を取得します。
     *   1 : ブザーON
     *   0 : ブザーOFF
     */
    if(argc != 2){
        printf("invalid arg¥n");
        printf("BuzzerOnOff.exe [OnOff(0:OFF, 1:ON)]¥n");
        return -1;
    }
    sscanf(*(argv + 1), "%d", &temp);

    /*
     * ブザー調整用ファイルの Open
     */
    hBuzzer = CreateFile(
                    DRIVER_FILENAME,
                    GENERIC_READ | GENERIC_WRITE,
                    FILE_SHARE_READ | FILE_SHARE_WRITE,
                    NULL,
                    OPEN_EXISTING,
                    0,
                    NULL
                );
```

```
    if(hBuzzer == INVALID_HANDLE_VALUE){
        printf("CreateFile: NG¥n");
        return -1;
    }

    /*
     * ブザーON/OFF を書込み
     */
    ret = DeviceIoControl(
                hBuzzer,
                IOCTL_BUZZDRV_SETBUZZ,
                &temp,
                sizeof(ULONG),
                NULL,
                0,
                &retlen,
                NULL
          );
    if(!ret){
        printf("DeviceIoControl: IOCTL_BUZZDRV_SETBUZZ NG¥n");
        CloseHandle(hBuzzer);
        return -1;
    }

    /*
     * ブザーON/OFF を読出し
     */
    ret = DeviceIoControl(
                hBuzzer,
                IOCTL_BUZZDRV_GETBUZZ,
                NULL,
                0,
                &temp,
                sizeof(ULONG),
                &retlen,
                NULL
          );
    if(!ret){
        printf("DeviceIoControl: IOCTL_BUZZDRV_GETBUZZ NG¥n");
        CloseHandle(hBuzzer);
        return -1;
    }
    printf("Buzzer OnOff=%d¥n", temp);


    for(;;) {
        Sleep(10);
    }

    CloseHandle(hBuzzer);
    return 0;
}
```

●タッチブザー

「¥SDK¥Algo¥Sample¥Sample_Buzzer¥BuzzerAuto」にブザーの取得と設定を行うサンプルコードを用意しています。リスト 5-3-5-2 にサンプルコードを示します。

リスト 5-3-5-2. タッチブザー

```
/**
    タッチ音の ON/OFF 制御サンプルソース
**/
#include <windows.h>
#include <winioctl.h>
#include <stdio.h>
#include <stdlib.h>
#include <mmsystem.h>
#include <conio.h>

#include "..¥Common¥BuzzDD.h"

#define DRIVER_FILENAME "¥¥¥¥.¥¥BuzzDrv"

int main(int argc, char **argv)
{
    HANDLE hBuzzer;
    ULONG retlen;
    BOOL ret;
    ULONG temp;

    /*
     * 起動引数からタッチパネル音の ON/OFF 変更値を取得します。
     *   1 : タッチパネル音 ON
     *   0 : タッチパネル音 OFF
     */
    if(argc != 2){
        printf("invalid arg¥n");
        printf("BuzzerAuto.exe [auto(0:OFF, 1:ON)]¥n");
        return -1;
    }
    sscanf(*(argv + 1), "%d", &temp);

    /*
     * ブザー調整用ファイルの Open
     */
    hBuzzer = CreateFile(
                    DRIVER_FILENAME,
                    GENERIC_READ | GENERIC_WRITE,
                    FILE_SHARE_READ | FILE_SHARE_WRITE,
                    NULL,
                    OPEN_EXISTING,
                    0,
                    NULL
                );
    if(hBuzzer == INVALID_HANDLE_VALUE){
        printf("CreateFile: NG¥n");
```

```
        return -1;
    }

    /*
     * タッチパネル音 ON/OFF を書込み
     */
    ret = DeviceIoControl(
                hBuzzer,
                IOCTL_BUZZDRV_SETAUTO,
                &temp,
                sizeof(ULONG),
                NULL,
                0,
                &retlen,
                NULL
          );
    if(!ret){
        printf("DeviceIoControl: IOCTL_BUZZDRV_SETAUTO NG¥n");
        CloseHandle(hBuzzer);
        return -1;
    }

    /*
     * タッチパネル音 ON/OFF を読出し
     */
    ret = DeviceIoControl(
                hBuzzer,
                IOCTL_BUZZDRV_GETAUTO,
                NULL,
                0,
                &temp,
                sizeof(ULONG),
                &retlen,
                NULL
          );
    if(!ret){
        printf("DeviceIoControl: IOCTL_BUZZDRV_GETAUTO NG¥n");
        CloseHandle(hBuzzer);
        return -1;
    }
    printf("Auto OnOff=%d¥n", temp);

    CloseHandle(hBuzzer);
    return 0;
}
```

５－４　　汎用入出力

５－４－１　汎用入出力について

AS シリーズには、入力 6 点、出力 4 点の汎用入出力があります。

● 入力ポート

入力ポートのデータ形式を図 5-4-1-1 に示します。汎用入出力ドライバでは、データ型での操作とビット指定での操作を行うことができます。

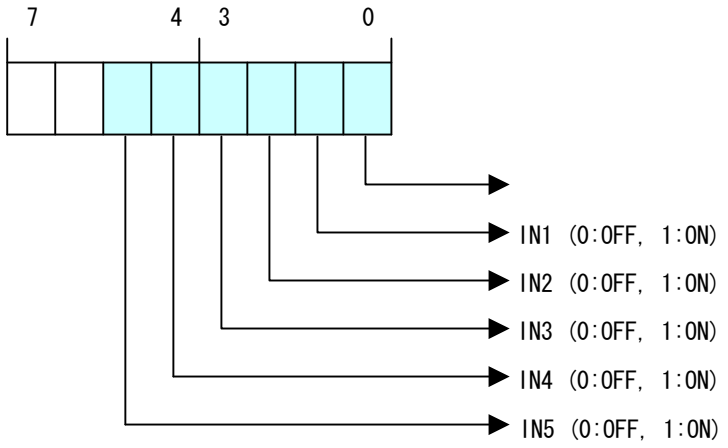

図 5-4-1-1. 入力データ

● 出力ポート

出力ポートのデータ形式を図 5-4-1-2 に示します。汎用入出力ドライバでは、データ型での操作とビット指定での操作を行うことができます。

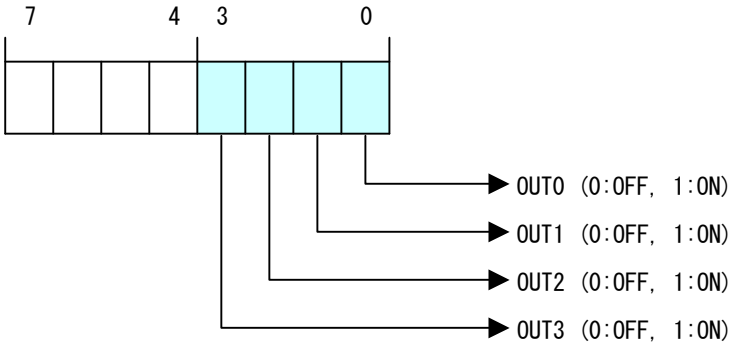

図 5-4-1-2. 出力データ

５－４－２　汎用入出力ドライバについて

汎用入出力ドライバは汎用入出力を、ユーザーアプリケーションから利用できるようにします。ユーザーアプリケーションからは、汎用入出力ドライバを直接制御することで汎用入出力を制御することが可能です。

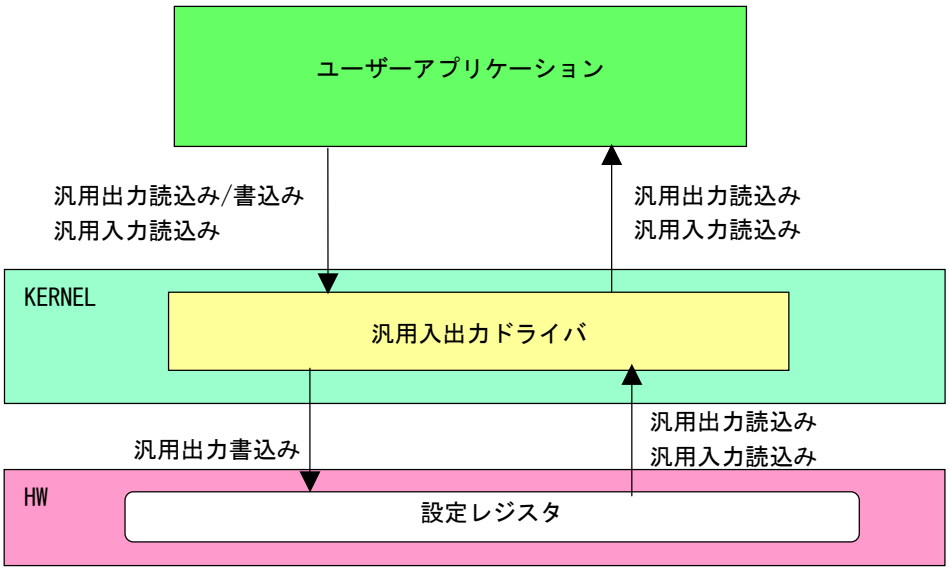

図 5-4-2-1. 汎用入出力ドライバ

５－４－３　汎用入出力デバイス

汎用入出力ドライバは汎用入出力デバイスを生成します。ユーザーアプリケーションは、デバイスファイルにアクセスすることによって汎用入出力を操作します。

汎用入出力デバイス

デバイスファイル

¥¥.¥GenIoDrv

説明

汎用入出力の制御を行うことが出来ます。

CreateFile

デバイスファイル(¥¥.¥GenIoDrv)をオープンし、デバイスハンドルを取得します。

```
hGenIo = CreateFile(
    "¥¥¥¥.¥¥GenIoDrv",
    GENERIC_READ | GENERIC_WRITE,
    FILE_SHARE_READ | FILE_SHARE_WRITE,
    NULL,
    OPEN_EXISTING,
    0,
    NULL
);
```

CloseHandle

デバイスハンドルをクローズします。

```
CloseHandle(hGenIo);
```

ReadFile

使用しません。

WriteFile

使用しません。

DeviceIoControl

● IOCTL_GENIODRV_RW

汎用入出力のリードライトを行います。

５－４－４　DeviceIoControlリファレンス

IOCTL_GENIODRV_RW

機能

汎用入出力のリードライトを行います。

パラメータ

lpInBuf　　：GENIODRV_RW_PAR を格納するためのポインタを指定します。
nInBufSize　　：GENIODRV_RW_PAR のサイズを指定します。
lpOutBuf　　：GENIODRV_RW_PAR を格納するためのポインタを指定します。
nOutBufSize　　：GENIODRV_RW_PAR のサイズを指定します。
lpBytesReturned　：実際の出力バイト数を受け取る変数へのポインタを指定します。
lpOverlapped　　：NULL を指定します。

GENIODRV_RW_PAR

```
typedef struct {
      ULONG RW;
      ULONG IoType;
      ULONG IoBit;
      ULONG Data;
} GENIODRV_RW_PAR, *P_GENIODRV_RW_PAR;
```

RW　　：リードライト [0: リード, 1: ライト, 2: リードビット, 3: ライトビット]
IoType　　：入出力ポート [0: 入力ポート, 1: 出力ポート]
IoBit　　：ビット番号(※1) [0Bit～5Bit]
Data　　：入出力データ

（※1）　RW が 2:リードビット、3:ライトビットの時のみ有効です。

戻り値

処理が成功すると TRUE を返します。失敗の場合は FALSE を返します。

説明

汎用入出力の制御を行います。
リードする場合は RW に「0:リード」か「2:リードビット」、ポート、ビット番号(リードビットの時のみ)を設定の上、lpInBuf と lpOutBuf に GENIODRV_RW_PAR 構造体を渡します。正常にリードできた場合は、入出力データに読み込んだポートの値が格納されます。

● データ型でのリード (RW = 0)
IoType：　出力ポート・入力ポートを指定します。
IoBit：　無視されます。
Data：　読込んだ値がデータ形式で格納されます。

● ビット指定でのリード (RW = 2)
　IoType： 出力ポート・入力ポートを指定します。
　IoBit： 読込むビットを指定します。
　Data： 読込んだビット状態（0、1）が格納されます。

ライトする場合はRWに「1:ライト」か「3:ライトビット」、ポート、ビット番号(ライトビットの時のみ)、ライトデータを入出力データに設定の上、lpInBufとlpOutBufにGENIODRV_RW_PAR構造体を渡します。

● データ型でのライト (RW = 1)
　IoType： 出力ポートを指定します。
　IoBit： 無視されます。
　Data： 書込む値をデータ形式で格納します。

● ビット指定でのライト (RW = 3)
　IoType： 出力ポートを指定します。
　IoBit： 書込むビットを指定します。
　Data： 書込むビット状態（0、1）を格納します。

５－４－５　サンプルコード

「¥SDK¥Algo¥Sample¥Sample_GenIO¥Genlo」に汎用入出力を使用したサンプルコードを用意しています。リスト 5-4-5-1 にサンプルコードを示します。

リスト 5-4-5-1. 汎用入出力

```
/**
    汎用入出力制御サンプルソース
**/

#include <windows.h>
#include <winioctl.h>
#include <stdio.h>
#include <stdlib.h>
#include <mmsystem.h>
#include <conio.h>

#include "..¥Common¥GenIoDD.h"

#define DRIVER_FILENAME "¥¥¥¥.¥¥GenIoDrv"

BOOL ReadIn(HANDLE hDevice, USHORT *pBuffer)
{
    BOOL    ret;
    ULONG   retlen;

    GENIODRV_RW_PAR rw_par;

    rw_par.RW = RW_READ;
    rw_par.IoType = PORT_INP;
    rw_par.IoBit = 0;

    ret = DeviceIoControl(hDevice,
                          IOCTL_GENIODRV_RW,
                          &rw_par,
                          sizeof(GENIODRV_RW_PAR),
                          &rw_par,
                          sizeof(GENIODRV_RW_PAR),
                          &retlen,
                          NULL);
    if(!ret){
        return FALSE;
    }
    if(retlen != sizeof(GENIODRV_RW_PAR)){
        return FALSE;
    }
    *pBuffer = (USHORT)rw_par.Data;
    return TRUE;
}
BOOL WriteOut(HANDLE hDevice, USHORT Data)
{
    BOOL    ret;
    ULONG   retlen;
```

```
    GENIODRV_RW_PAR rw_par;

    rw_par.RW = RW_WRITE;
    rw_par.IoType = PORT_OUT;
    rw_par.IoBit = 0;
    rw_par.Data = (ULONG)Data;

    ret = DeviceIoControl(hDevice,
                          IOCTL_GENIODRV_RW,
                          &rw_par,
                          sizeof(GENIODRV_RW_PAR),
                          &rw_par,
                          sizeof(GENIODRV_RW_PAR),
                          &retlen,
                          NULL);
    if(!ret){
        return FALSE;
    }
    if(retlen != sizeof(GENIODRV_RW_PAR)){
        return FALSE;
    }
    return TRUE;
}

BOOL ReadInBit(HANDLE hDevice, ULONG Bit, USHORT *pBuffer)
{
    BOOL    ret;
    ULONG   retlen;

    GENIODRV_RW_PAR rw_par;

    rw_par.RW = RW_READBIT;
    rw_par.IoType = PORT_INP;
    rw_par.IoBit = Bit;

    ret = DeviceIoControl(hDevice,
                          IOCTL_GENIODRV_RW,
                          &rw_par,
                          sizeof(GENIODRV_RW_PAR),
                          &rw_par,
                          sizeof(GENIODRV_RW_PAR),
                          &retlen,
                          NULL);
    if(!ret){
        return FALSE;
    }
    if(retlen != sizeof(GENIODRV_RW_PAR)){
        return FALSE;
    }
    *pBuffer = (USHORT)rw_par.Data;
    return TRUE;
```

```
}

BOOL WriteOutBit(HANDLE hDevice, ULONG Bit, USHORT Data)
{
    BOOL    ret;
    ULONG   retlen;

    GENIODRV_RW_PAR rw_par;

    rw_par.RW = RW_WRITEBIT;
    rw_par.ioType = PORT_OUT;
    rw_par.ioBit = Bit;
    rw_par.Data = (ULONG)Data;

    ret = DeviceIoControl(hDevice,
                          IOCTL_GENIODRV_RW,
                          &rw_par,
                          sizeof(GENIODRV_RW_PAR),
                          &rw_par,
                          sizeof(GENIODRV_RW_PAR),
                          &retlen,
                          NULL);
    if(!ret){
        return FALSE;
    }
    if(retlen != sizeof(GENIODRV_RW_PAR)){
        return FALSE;
    }
    return TRUE;
}

int main(int argc, char **argv)
{
    HANDLE  hGenio;
    BOOL    ret;
    ULONG   i;
    ULONG   retlen;
    ULONG   temp;
    USHORT  outdata=0x0001;
    USHORT  indata=0x0000;

    /* 汎用出力ﾃﾞﾊﾞｲｽのｵｰﾌﾟﾝ */
    hGenio = CreateFile(
                    DRIVER_FILENAME,
                    GENERIC_READ | GENERIC_WRITE,
                    FILE_SHARE_READ | FILE_SHARE_WRITE,
                    NULL,
                    OPEN_EXISTING,
                    0,
                    NULL
                );
    if(hGenio == INVALID_HANDLE_VALUE){
```

```
        printf("CreateFile: NG¥n");
        return -1;
    }

    /* 汎用出力
        汎用出力の４点を１点ずつ出力を行います。
    */
    for (i=0; i<4; i++){
        ret = WriteOut(hGenIo, outdata);
        if ( !ret ) {
            printf("DeviceIoControl: IOCTL_GENIODRV_RW NG¥n");
            CloseHandle(hGenIo);
            return -1;
        }
        outdata <<= 1;
        Sleep(500);
    }
    outdata = 0x0000;
    ret = WriteOut(hGenIo, outdata);
    if ( !ret ) {
        printf("DeviceIoControl: IOCTL_GENIODRV_RW NG¥n");
        CloseHandle(hGenIo);
        return -1;
    }
    for (i=0; i<4; i++){
        outdata = 0x0001;
        WriteOutBit(hGenIo, i, outdata);
        Sleep(500);
    }
    outdata = 0x0000;
    ret = WriteOut(hGenIo, outdata);
    if ( !ret ) {
        printf("DeviceIoControl: IOCTL_GENIODRV_RW NG¥n");
        CloseHandle(hGenIo);
        return -1;
    }

    /* 汎用入力
        汎用入力の６点ずつ出力を行います。
    */
    for(i=0; i<6; i++){
        ret = ReadInBit(hGenIo, i, &indata);
        if ( !ret ) {
            printf("DeviceIoControl: IOCTL_GENIODRV_RW NG¥n");
            CloseHandle(hGenIo);
            return -1;
        }
        else{
            /* IN状態 */
            if (indata & 1) printf("INBIT%d: ON¥n", i);
            else            printf("INBIT%d: OFF¥n", i);
        }
```

```
        Sleep(500);
    }

    ret = ReadIn(hGenIo, &indata);
    if ( !ret ) {
        printf("DeviceIoControl: IOCTL_GENIODRV_RW NG¥n");
        CloseHandle(hGenIo);
        return -1;
    }
    else{
        indata &= 0x3F;
        /* IN0 状態 */
        if (indata & 0x01) printf("IN0: ON¥n");
        else               printf("IN0: OFF¥n");
        /* IN1 状態 */
        if (indata & 0x02) printf("IN1: ON¥n");
        else               printf("IN1: OFF¥n");
        /* IN2 状態 */
        if (indata & 0x04) printf("IN2: ON¥n");
        else               printf("IN2: OFF¥n");
        /* IN3 状態 */
        if (indata & 0x08) printf("IN3: ON¥n");
        else               printf("IN3: OFF¥n");
        /* IN4 状態 */
        if (indata & 0x10) printf("IN4: ON¥n");
        else               printf("IN4: OFF¥n");
        /* IN5 状態 */
        if (indata & 0x20) printf("IN5: ON¥n");
        else               printf("IN5: OFF¥n");
    }

    /* 汎用入出力ﾃﾞﾊﾞｲｽのｸﾛｰｽﾞ */
    CloseHandle(hGenIo);

    return 0;
}
```

５－５　　LCDバックライト

５－５－１　LCDバックライトについて

AS シリーズには、バックライトの輝度を変更することができます。I/O ポート上にマッピングされた制御レジスタを操作することによって、バックライトの輝度を変更することができます。

５－５－２　LCDバックライトドライバについて

LCD バックライトドライバはバックライトの輝度を、ユーザーアプリケーションから変更できるようにします。

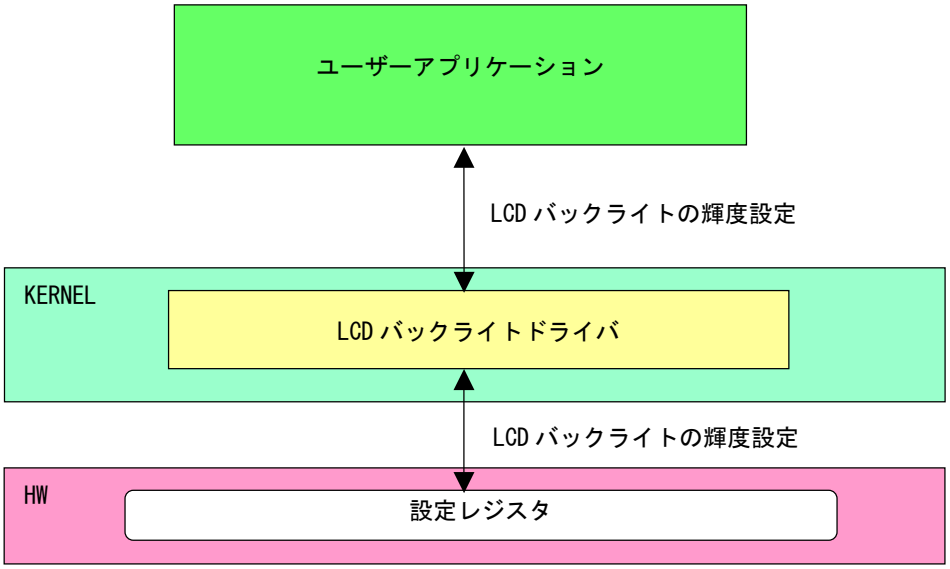

図 5-5-2-1. LCDバックライトドライバ

５－５－３　LCDバックライトデバイス

LCDバックライトドライバはLCDバックライトデバイスを生成します。ユーザーアプリケーションは、デバイスファイルにアクセスすることによってバックライトの輝度を操作します。

LCDバックライトデバイス

デバイスファイル

¥¥.¥LcdBacklight

説明

LCDバックライトの輝度を変更することができます。

レジストリ設定

[KEY]　　HKEY_LOCAL_MACHINE¥SYSTEM¥CurrentControlSet¥Services¥LcdBacklight
[VALUE:DWORD]　　Brightness
LCDバックライトの輝度を設定します。ドライバ起動時(OS起動時)にこの値を参照し
LCDバックライトの輝度を設定します。(デフォルト値: 0)

CreateFile

デバイスファイル(¥¥.¥LcdBacklight)をオープンし、デバイスハンドルを取得します。

```
hBacklight = CreateFile(
    "¥¥¥¥.¥¥LcdBacklight",
    GENERIC_READ | GENERIC_WRITE,
    FILE_SHARE_READ | FILE_SHARE_WRITE,
    NULL,
    OPEN_EXISTING,
    0,
    NULL
);
```

CloseHandle

デバイスハンドルをクローズします。

```
CloseHandle(hBacklight);
```

ReadFile

使用しません。

WriteFile

使用しません。

DeviceIoControl

- **IOCTL_LCDBACKLIGHT_SETBRIGHTNESS**
  LCDバックライトの輝度を設定します。
- **IOCTL_LCDBACKLIGHT_GETBRIGHTNESS**
  LCDバックライトの輝度を取得します。
- **IOCTL_LCDBACKLIGHT_SETBACKLIGHTPOWER**
  LCDバックライトのON/OFFを設定します。
- **IOCTL_LCDBACKLIGHT_GETBACKLIGHTPOWER**
  LCDバックライトのON/OFFを取得します。

５－５－４　DeviceIoControlリファレンス

IOCTL_LCDBACKLIGHT_SETBRIGHTNESS

機能

LCD バックライトの輝度を設定します。

パラメータ

lpInBuf	:	バックライト輝度を格納するポインタを指定します。
NInBufSize	:	バックライト輝度を格納するポインタのサイズを指定します。
LpOutBuf	:	NULL を指定します。
NOutBufSize	:	0 を指定します。
LpBytesReturned	:	実際の出力バイト数を受け取る変数へのポインタ。
LpOverlapped	:	NULL を指定します。

バックライト輝度

データタイプ	:	ULONG
データサイズ	:	4 バイト
内容	:	0: 明るい　～　255: 暗い

戻り値

処理が成功すると TRUE を返します。失敗の場合は FALSE を返します。

説明

LCD バックライトの輝度の設定を行います。
バックライトの輝度は 0(明るい)～255(暗い)の 256 段階で設定できます。バックライトの輝度を設定の上、DeviceIoControl を実行してください。

IOCTL_LCDBACKLIGHT_GETBRIGHTNESS

機能

LCD バックライトの輝度を取得します。

パラメータ

lpInBuf	:	NULL を指定します。
NInBufSize	:	0 を指定します。
LpOutBuf	:	バックライト輝度を格納するポインタを指定します。
NOutBufSize	:	バックライト輝度を格納するポインタのサイズを指定します。
LpBytesReturned	:	実際の出力バイト数を受け取る変数へのポインタ。
LpOverlapped	:	NULL を指定します。

バックライト輝度

データタイプ	:	ULONG
データサイズ	:	4 バイト
内容	:	0: 明るい　～　255: 暗い

戻り値

処理が成功すると TRUE を返します。失敗の場合は FALSE を返します。

説明

LCD バックライトの輝度の取得を行います。

IOCTL_LCDBACKLIGHT_SETBACKLIGHTPOWER

機能

LCD バックライトの ON/OFF を設定します。

パラメータ

lpInBuf	:	バックライト ON/OFF 情報を格納するポインタを指定します(32 ビットデータ)。
NInBufSize	:	バックライト ON/OFF 情報を格納するポインタのサイズを指定します。
LpOutBuf	:	NULL を指定します。
NOutBufSize	:	0 を指定します。
LpBytesReturned	:	実際の出力バイト数を受け取る変数へのポインタ。
LpOverlapped	:	NULL を指定します。

バックライト ON/OFF 情報

データタイプ	:	ULONG
データサイズ	:	4 バイト
内容	:	0: ON, 1: OFF

戻り値

処理が成功すると TRUE を返します。失敗の場合は FALSE を返します。

説明

LCD バックライトの ON/OFF の設定を行います。
バックライトを ON する場合は、バックライト ON/OFF 情報を格納するポインタに 1、OFF にする場合は 0 を設定の上、DeviceIoControl を実行してください。

IOCTL_LCDBACKLIGHT_GETBACKLIGHTPOWER

機能

LCD バックライトの ON/OFF を取得します。

パラメータ

lpInBuf	:	NULL を指定します。
NInBufSize	:	0 を指定します。
LpOutBuf	:	バックライト ON/OFF 情報を格納するポインタを指定します(32 ビットデータ)。
NOutBufSize	:	バックライト ON/OFF 情報を格納するポインタのサイズを指定します。
LpBytesReturned	:	実際の出力バイト数を受け取る変数へのポインタ。
LpOverlapped	:	NULL を指定します。

バックライト ON/OFF 情報

データタイプ	:	ULONG
データサイズ	:	4 バイト
内容	:	0: ON, 1: OFF

戻り値

処理が成功すると TRUE を返します。失敗の場合は FALSE を返します。

説明

LCD バックライトの輝度の取得を行います。

５－５－５　サンプルコード

●LCD バックライト輝度

「¥SDK¥Algo¥Sample¥Sample_BackLight¥BacklightBrightnessCtrl」に LCD バックライト輝度の取得と設定のサンプルコードを用意しています。リスト 5-5-5-1 にサンプルコードを示します。

リスト 5-5-5-1. LCDバックライト輝度

```
/**
    バックライトの輝度変更制御サンプルソース
**/
#include <windows.h>
#include <winioctl.h>
#include <stdio.h>
#include <stdlib.h>
#include <mmsystem.h>
#include <conio.h>

#include "..¥Common¥LcdBacklightDD.h"

#define DRIVER_FILENAME "¥¥¥¥.¥¥LcdBacklight"

int main(int argc, char **argv)
{
    ULONG set_data;
    ULONG get_data;
    HANDLE hLcdBacklight;
    ULONG retlen;
    BOOL ret;

    /*
     * 起動引数からバックライト光量変更値を取得
     *   0～255 の範囲で設定します
     *   0：明るい　～　255：暗い
     */
    if(argc != 2){
        printf("invalid arg¥n");
        return -1;
    }
    sscanf(*(argv + 1), "%x", &set_data);

    /*
     * バックライト調整用ファイルの Open
     */
    hLcdBacklight = CreateFile(
                        DRIVER_FILENAME,
                        GENERIC_READ | GENERIC_WRITE,
                        FILE_SHARE_READ | FILE_SHARE_WRITE,
                        NULL,
                        OPEN_EXISTING,
                        0,
                        NULL
```

```
                    );
    if(hLcdBacklight == INVALID_HANDLE_VALUE){
        printf("CreateFile: NG¥n");
        return -1;
    }

    /*
     * バックライト光量変更値を書込み
     */
    ret = DeviceIoControl(
                hLcdBacklight,
                IOCTL_LCDBACKLIGHT_SETBRIGHTNESS,
                &set_data,
                sizeof(ULONG),
                NULL,
                0,
                &retlen,
                NULL
          );
    if(!ret){
        printf("DeviceIoControl: IOCTL_LCDBACKLIGHT_SETBRIGHTNESS NG¥n");
        CloseHandle(hLcdBacklight);
        return -1;
    }

    /*
     * バックライト光量変更値を読出し
     */
    ret = DeviceIoControl(
                hLcdBacklight,
                IOCTL_LCDBACKLIGHT_GETBRIGHTNESS,
                NULL,
                0,
                &get_data,
                sizeof(ULONG),
                &retlen,
                NULL
          );
    if(!ret){
        printf("DeviceIoControl: IOCTL_LCDBACKLIGHT_GETBRIGHTNESS NG¥n");
        CloseHandle(hLcdBacklight);
        return -1;
    }
    printf("Get LCD Backlight Brightness: %x¥n", get_data);

    CloseHandle(hLcdBacklight);
    return 0;
}
```

●LCD バックライト ON/OFF

「¥SDK¥Algo¥Sample¥Sample_BackLight¥BacklightOnOff」に LCD バックライト ON/OFF 制御のサンプルコードを用意しています。リスト 5-5-5-2 にサンプルコードを示します。

リスト 5-5-5-2. LCDバックライトON/OFF

```
/**
    バックライトの ON/OFF 制御サンプルソース
**/
#include <windows.h>
#include <winioctl.h>
#include <stdio.h>
#include <stdlib.h>
#include <mmsystem.h>
#include <conio.h>

#include "..¥Common¥LcdBacklightDD.h"

#define DRIVER_FILENAME "¥¥¥¥.¥¥LcdBacklight"

int main(int argc, char **argv)
{
    ULONG set_data;
    ULONG get_data;
    HANDLE hLcdBacklight;
    ULONG retlen;
    BOOL ret;

    /*
     * 起動引数からバックライトの ON/OFF 変更値を取得します。
     *   0 : バックライト ON
     *   1 : バックライト OFF
     */
    if(argc != 2){
        printf("invalid arg¥n");
        return -1;
    }
    sscanf(*(argv + 1), "%x", &set_data);

    /*
     * バックライト調整用ファイルの Open
     */
    hLcdBacklight = CreateFile(
                        DRIVER_FILENAME,
                        GENERIC_READ | GENERIC_WRITE,
                        FILE_SHARE_READ | FILE_SHARE_WRITE,
                        NULL,
                        OPEN_EXISTING,
                        0,
                        NULL
                    );
    if(hLcdBacklight == INVALID_HANDLE_VALUE){
        printf("CreateFile: NG¥n");
```

```
        return -1;
    }

    /*
     * バックライト ON/OFF を書込み
     */
    ret = DeviceIoControl(
                hLcdBacklight,
                IOCTL_LCDBACKLIGHT_SETBACKLIGHTPOWER,
                &set_data,
                sizeof(ULONG),
                NULL,
                0,
                &retlen,
                NULL
          );
    if(!ret){
        printf("DeviceIoControl: IOCTL_LCDBACKLIGHT_SETBACKLIGHTPOWER NG¥n");
        CloseHandle(hLcdBacklight);
        return -1;
    }

    /*
     * バックライト ON/OFF を読出し
     */
    ret = DeviceIoControl(
                hLcdBacklight,
                IOCTL_LCDBACKLIGHT_GETBACKLIGHTPOWER,
                NULL,
                0,
                &get_data,
                sizeof(ULONG),
                &retlen,
                NULL
          );
    if(!ret){
        printf("DeviceIoControl: IOCTL_LCDBACKLIGHT_GETBACKLIGHTPOWER NG¥n");
        CloseHandle(hLcdBacklight);
        return -1;
    }
    printf("Get LCD BackLight Power: %x¥n", get_data);

    CloseHandle(hLcdBacklight);
    return 0;
}
```

５－６　　RAS機能

５－６－１　RAS機能について

AS シリーズには、ハードウェアによる IN0 リセット機能、IN1 割込み機能が実装されています。I/O ポート上にマッピングされた制御レジスタを操作することによって、IN0 入力時にハードウェアリセットを掛けることができます。また、IN1 入力時に割込みを発生させることができます。

５－６－２　RAS-INドライバについて

RAS-IN ドライバは IN0 リセット機能、IN1 割込み機能を、ユーザーアプリケーションから利用できるようにします。ユーザーアプリケーションから、IN0 リセット機能と IN1 割込み機能の ON/OFF 設定とイベントによる IN1 割込み通知の機能を使用することができます。

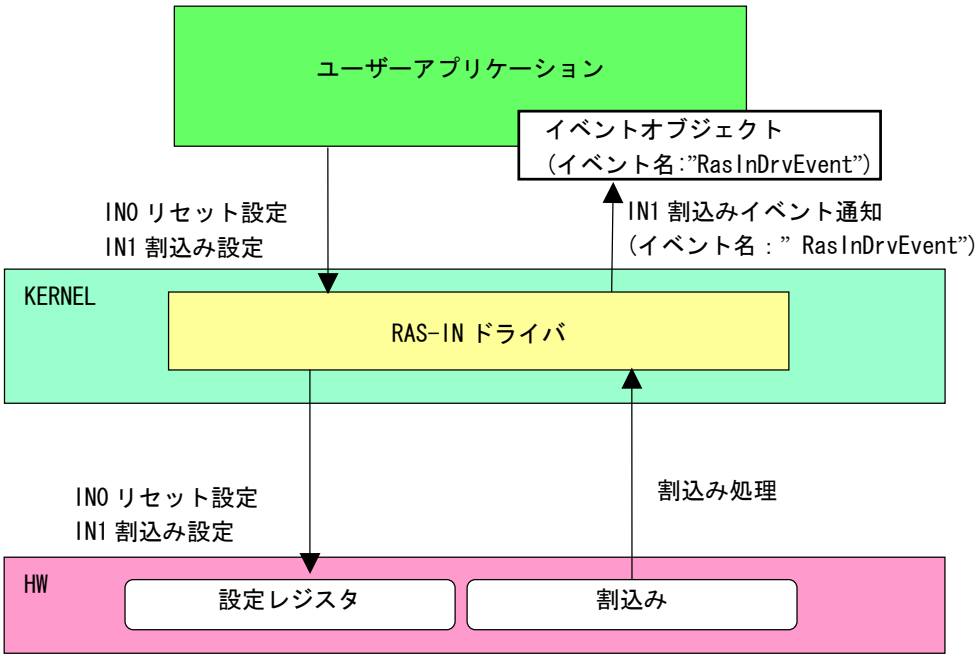

図 5-6-2-1. RAS-INドライバ

５－６－３　RAS-INデバイス

RAS-IN ドライバは RAS-IN デバイスを生成します。ユーザーアプリケーションは、デバイスファイルにアクセスすることによって RAS-IN 機能を操作します。

RAS-INデバイス

デバイスファイル

¥¥.¥RasInDrv

説明

IN0リセット設定、IN1割り込み設定を行うことが出来ます。デバイスの使用は
１アプリケーションのみです。複数のアプリケーションから使用することはできません。

CreateFile

デバイスファイル(¥¥.¥RasInDrv)をオープンし、デバイスハンドルを取得します。

```
hRasIn = CreateFile(
    "¥¥¥¥.¥¥RasInDrv",
    GENERIC_READ | GENERIC_WRITE,
    FILE_SHARE_READ | FILE_SHARE_WRITE,
    NULL,
    OPEN_EXISTING,
    0,
    NULL
);
```

CloseHandle

デバイスハンドルをクローズします。

```
CloseHandle(hRasIn);
```

ReadFile

使用しません。

WriteFile

使用しません。

DeviceIoControl

- **IOCTL_RASINDRV_SINORST**
  IN0リセットを設定します。
- **IOCTL_RASINDRV_GINORST**
  現在のIN0リセット設定を取得します。
- **IOCTL_RASINDRV_SIN1INT**
  IN1割り込みを設定します。
- **IOCTL_RASINDRV_GIN1INT**
  現在のIN1割り込み設定を取得します。

５－６－４　IN1 割込みの使用手順

基本的な使用手順を以下に示します。

IN1 割込み通知用イベントハンドルを作成後、IN1 割込み通知用イベントハンドルでのイベント待ち準備が整ったところで、RAS-IN デバイスに IN1 割込み有効を設定します。

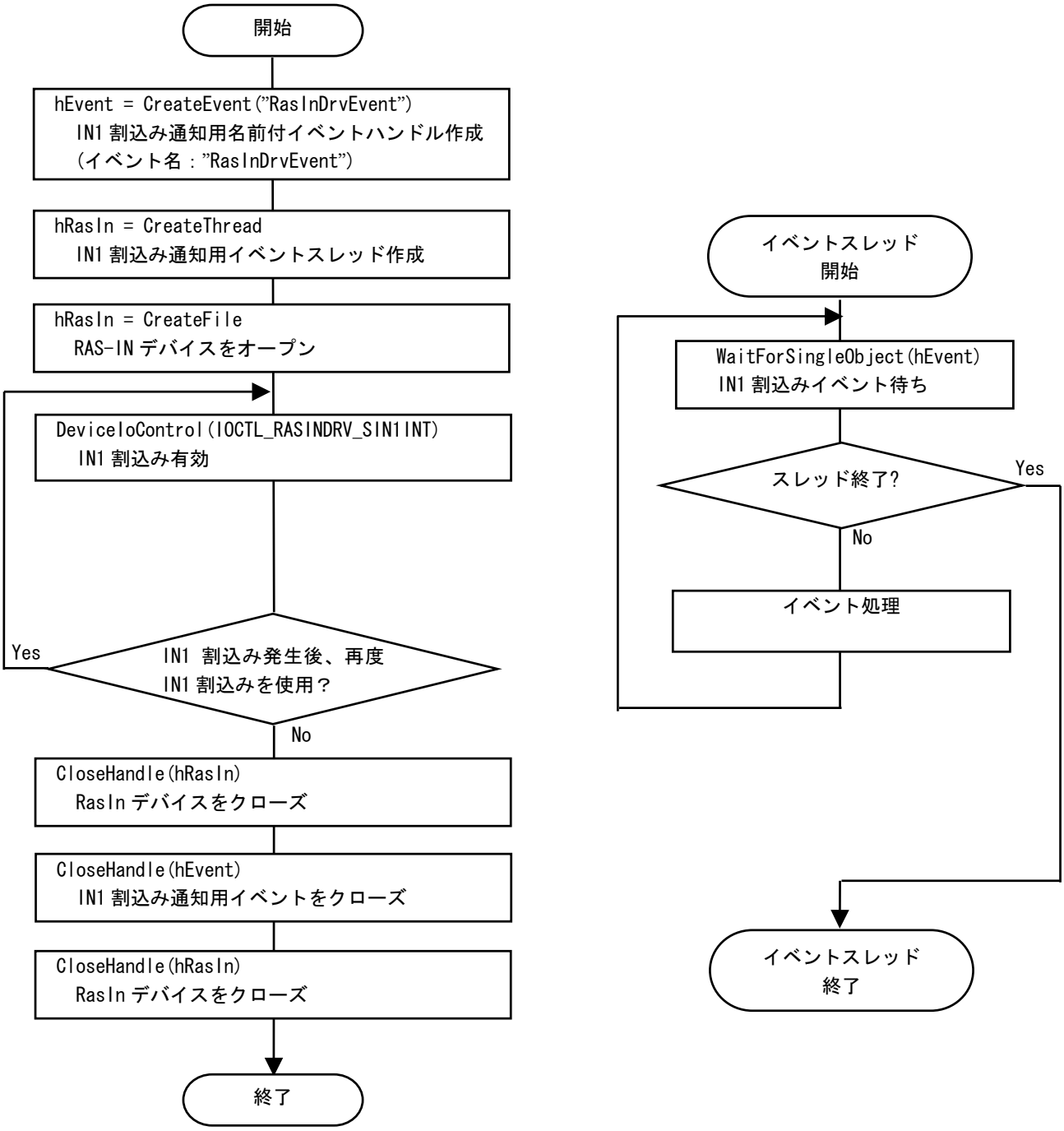

図 5-6-4-1. IN1 割込み仕様手順

５－６－５　複数アプリケーションでIN1 割込み発生時に同時に使用する場合

複数アプリケーションで IN1 割込み発生時に同時にイベント処理する場合の使用手順を以下に示します。

メインアプリケーションで IN1 割込み通知用名前付き手動リセットのイベントハンドルを作成後、IN1 割込み通知用イベントハンドルでのイベント待ち準備が整ったところで、RAS-IN デバイスに IN1 割込み有効を設定します。サブアプリケーションは、手動リセットの IN1 割込み通知用名前付きイベントハンドルを作成後、IN1 割込み通知用イベントハンドルでのイベント待ち準備を行います。IN1 割込みが発生すれば、メイン、サブの両アプリケーションに同時にイベントが発生します。

※　IN1 割込み有効/無効は RAS-IN デバイスをオープンしたアプリケーションしか行えません。複数のアプリケーションからは IN1 割込み有効/無効できませんので注意してください。

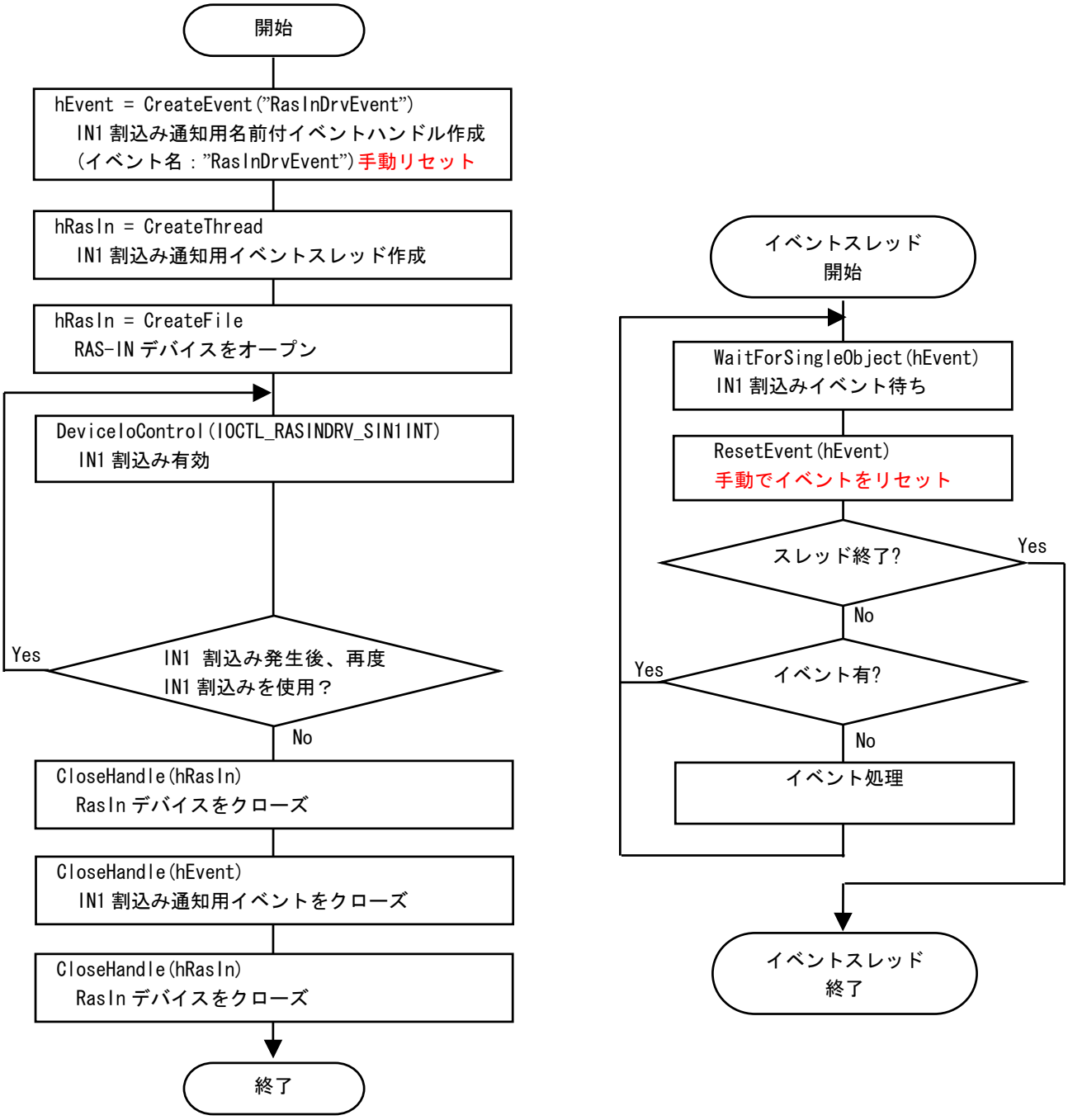

図 5-6-5-1. 複数アプリケーションでIN1 割込みを使用する手順

５－６－６　DeviceIoControlリファレンス

IOCTL_RASINDRV_SINORST

機能

INO リセットを設定します。

パラメータ

lpInBuf	:	INO リセット情報を格納するポインタを指定します。
nInBufSize	:	INO リセット情報を格納するポインタのサイズを指定します。
lpOutBuf	:	NULL を指定します。
nOutBufSize	:	0 を指定します。
lpBytesReturned	:	実際の出力バイト数を受け取る変数へのポインタ。
lpOverlapped	:	NULL を指定します。

INO リセット情報

データタイプ	:	ULONG
データサイズ	:	4 バイト
内容	:	1: INO リセット有効, 0: INO リセット無効

戻り値

処理が成功すると TRUE を返します。失敗の場合は FALSE を返します。

説明

INO リセットを設定します。
INO リセットを有効にする場合は、INO リセットの設定を格納するポインタに 1、無効にする場合は 0 を設定の上、DeviceIoControl を実行してください。

IOCTL_RASINDRV_GINORST

機能

INO リセット設定を取得します。

パラメータ

lpInBuf	:	NULL を指定します。
NInBufSize	:	0 を指定します。
LpOutBuf	:	INO リセット情報を格納するポインタを指定します。
NOutBufSize	:	INO リセット情報を格納するポインタのサイズを指定します。
LpBytesReturned	:	実際の出力バイト数を受け取る変数へのポインタ。
LpOverlapped	:	NULL を指定します。

INO リセット情報

データタイプ	:	ULONG
データサイズ	:	4 バイト
内容	:	1: INO リセット有効, 0: INO リセット無効

戻り値

処理が成功すると TRUE を返します。失敗の場合は FALSE を返します。

説明

現在の INO リセット設定値を取得します。

IOCTL_RASINDRV_SIN1INT

機能

IN1 割込みを設定します。

パラメータ

lpInBuf	:	IN1 割込み情報を格納するポインタを指定します。
NInBufSize	:	IN1 割込み情報を格納するポインタのサイズを指定します。
LpOutBuf	:	NULL を指定します。
NOutBufSize	:	0 を指定します。
LpBytesReturned	:	実際の出力バイト数を受け取る変数へのポインタを指定します。
LpOverlapped	:	NULL を指定します。

IN1 割込み情報

データタイプ	:	ULONG
データサイズ	:	4 バイト
内容	:	1: IN1 割込み有効, 0: IN1 割込み無効

戻り値

処理が成功すると TRUE を返します。失敗の場合は FALSE を返します。

説明

IN1 割込みの設定を行います。
IN1 割込みを有効にする場合は、IN1 割込みの設定を格納するポインタに 1、無効にする場合は 0 を設定の上、DeviceIoControl を実行してください。

IOCTL_RASINDRV_GIN1INT

機能

IN1 割込み設定を取得します。

パラメータ

lpInBuf	:	NULL を指定します。
NInBufSize	:	0 を指定します。
LpOutBuf	:	IN1 割込み情報を格納するためのポインタを指定します。
NOutBufSize	:	IN1 割込み情報のサイズを指定します。
LpBytesReturned	:	実際の出力バイト数を受け取る変数へのポインタを指定します。
LpOverlapped	:	NULL を指定します。

IN1 割込み情報

データタイプ	:	ULONG
データサイズ	:	4 バイト
内容	:	1: IN1 割込み有効, 0: IN1 割込み無効

戻り値

処理が成功すると TRUE を返します。失敗の場合は FALSE を返します。

説明

現在の IN1 割込み設定値を取得します。

５－６－７　サンプルコード

●IN0 リセット

「¥SDK¥Algo¥Sample¥Sample_Reset¥In0Reset」に IN0 リセット機能のサンプルコードを用意しています。リスト 5-6-7-1 にサンプルコードを示します。

リスト 5-6-7-1. IN0 リセット

```
/**
    汎用入力 IN0 リセット制御方法サンプルソース
**/
#include <windows.h>
#include <winioctl.h>
#include <stdio.h>
#include <stdlib.h>
#include <conio.h>

#include "..¥Common¥RasinDrvDD.h"

#define RASINDRIVER_FILENAME    "¥¥¥¥.¥¥RasInDrv"

int main(void)
{
    HANDLE hRasin;
    ULONG retlen;
    ULONG reset;
    ULONG errno;
    BOOL ret;

    /*
     * ドライバオブジェクトの作成
     */
    hRasin = CreateFile(
                    RASINDRIVER_FILENAME,
                    GENERIC_READ | GENERIC_WRITE,
                    FILE_SHARE_READ | FILE_SHARE_WRITE,
                    NULL,
                    OPEN_EXISTING,
                    0,
                    NULL
                );
    if(hRasin == INVALID_HANDLE_VALUE){
        printf("CreateFile: NG¥n");
        return -1;
    }

    /*
     * IN0 リセットを有効にする
     *
     * reset: 1    有効
     *      : 0    無効
     */
    reset = 1;
```

```
    ret = DeviceIoControl(
                hRasin,
                IOCTL_RASINDRV_SINORST,
                &reset,
                sizeof(ULONG),
                NULL,
                0,
                &retlen,
                NULL
                );
    if(!ret){
        errno = GetLastError();
        fprintf(stderr, "ioctl set INORST error: %d¥n", errno);
        CloseHandle(hRasin);
        return -1;
    }
    else {
        fprintf(stdout, "ioctl set INORST success: %d¥n", reset);
    }

    /*
     * INO リセットを確認する
     */
    ret = DeviceIoControl(
                hRasin,
                IOCTL_RASINDRV_GINORST,
                NULL,
                0,
                &reset,
                sizeof(ULONG),
                &retlen,
                NULL
                );
    if(!ret){
        errno = GetLastError();
        fprintf(stderr, "ioctl get INORST error: %d¥n", errno);
        CloseHandle(hRasin);
        return -1;
    }
    else {
        fprintf(stdout, "ioctl get INORST success: %d¥n", reset);
    }

    CloseHandle(hRasin);
    return 0;
}
```

●IN1 割込み

「¥SDK¥Algo¥Sample¥Sample_Interrupt¥In1Interrupt」に IN1 割込み機能を使用したサンプルコードを用意しています。リスト 5-6-7-2 にサンプルコードを示します。

リスト 5-6-7-2. IN1 割込み

```
/**
    汎用入力 IN1 割込み制御サンプルソース
**/
#include <windows.h>
#include <winioctl.h>
#include <stdio.h>
#include <stdlib.h>
#include <mmsystem.h>
#include <conio.h>

#include "..¥Common¥RasinDrvDD.h"

#define RASINDRIVER_FILENAME    "¥¥¥¥.¥¥RasInDrv"

//---------------------------------------------------------------------------
typedef struct {
    int No;
    HANDLE  hEvent;
    HANDLE  hThread;
    HANDLE  hRasin;
    volatile BOOL fIn1Int;
    volatile BOOL fStart;
    volatile BOOL fFinish;
} RASINEVENT_INFO, *PRASINEVENT_INFO;

//---------------------------------------------------------------------------
/*
 * 割込みハンドラ
 */
DWORD WINAPI In1IntEventProc(void *pData)
{
    PRASINEVENT_INFO    info = (PRASINEVENT_INFO)pData;
    DWORD ret;

    printf("In1IntEventProc: Start¥n");

    info->fFinish = FALSE;
    while(1){
        if(WaitForSingleObject(info->hEvent, INFINITE) != WAIT_OBJECT_0){
            break;
        }
        ResetEvent(info->hEvent);       // イベントオブジェクト生成時に
                                        // 手動リセットを設定した場合は
                                        // 手動でイベントオブジェクトを
                                        // 非シグナル状態にする必要があります。
        if(!info->fStart){
            break;
```

```
        }
        printf("In1IntEventProc: Interrupt¥n");
    }
    info->fFinish = TRUE;

    printf("In1IntEventProc: Finish¥n");
    return 0;
}

//---------------------------------------------------------------------------
BOOL CreateIn1IntEventInfo(PRASINEVENT_INFO info)
{
    DWORD   thrd_id;
    ULONG   retlen;
    BOOL    ret;

    info->hEvent = NULL;
    info->hThread = NULL;
    info->hRasin = INVALID_HANDLE_VALUE;

    info->fStart = FALSE;
    info->fFinish = FALSE;
    info->fIn1Int = FALSE;

    /* イベントオブジェクトの作成
     *   複数アプリケーションでイベントを共有する場合は、
     *   CreateFile でドライバオブジェクトを作成するより前に
     *   CreateEvent で手動リセットを有効にした名前付き
     *   イベントを作成する必要があります。
     *   単アプリケーションの場合、自動リセットで問題ありません。
     */
    info->hEvent = CreateEvent(
                        NULL,
                        TRUE,                   // 手動リセットを指定します。
                        FALSE,
                        RASINDRV_EVENT_NAME     // イベント名を指定します。
                        );
    if(info->hEvent == NULL){
        printf("CreateIn1IntEventInfo: CreateEvent: NG¥n");
        return FALSE;
    }

    /*
     * イベントスレッドを生成
     */
    info->hThread = CreateThread(
                        (LPSECURITY_ATTRIBUTES)NULL,
                        0,
                        (LPTHREAD_START_ROUTINE)In1IntEventProc,
                        (LPVOID)info,
                        CREATE_SUSPENDED,
                        &thrd_id
```

```
                        );
    if(info->hThread == NULL){
        CloseHandle(info->hEvent);
        printf("CreateIn1IntEventInfo: CreateThread: NG¥n");
        return FALSE;
    }

    /*
     * ドライバオブジェクトの作成
     */
    info->hRasin = CreateFile(
                        RASINDRIVER_FILENAME,
                        GENERIC_READ | GENERIC_WRITE,
                        FILE_SHARE_READ | FILE_SHARE_WRITE,
                        NULL,
                        OPEN_EXISTING,
                        0,
                        NULL
                        );
    if(info->hRasin == INVALID_HANDLE_VALUE){
        CloseHandle(info->hThread);
        CloseHandle(info->hEvent);
        printf("CreateIn1IntEventInfo: CreateFile: NG¥n");
        return FALSE;
    }

    return TRUE;
}

//---------------------------------------------------------------------------
void DeleteIn1IntEvent(PRASINEVENT_INFO info)
{
    ULONG   retlen;

    // Stop Thread
    info->fStart = FALSE;
    SetEvent(info->hEvent);

    // Wait Thread Stop, Close Thread
    while(!info->fFinish){
        Sleep(10);
    }

    // Close Handle
    CloseHandle(info->hThread);
    CloseHandle(info->hEvent);
    CloseHandle(info->hRasin);
}

//---------------------------------------------------------------------------
/*
 * IN1 Interruptの取得
```

```
 */
BOOL Get_IN1Int(HANDLE hdriver, ULONG *value)
{
    BOOL    ret;
    ULONG   retlen;

    // Get IN1 Interrupt
    ret = DeviceIoControl(
                hdriver,
                IOCTL_RASINDRV_GIN1INT,
                NULL,
                0,
                &value,
                sizeof(ULONG),
                &retlen,
                NULL
                );

    return ret;
}

//---------------------------------------------------------------------------
/*
 * IN1 Interruptの設定
 */
BOOL Set_IN1Int(HANDLE hdriver, ULONG value)
{
    BOOL    ret;
    ULONG   retlen;

    // Set IN1 Interrupt
    ret = DeviceIoControl(
                hdriver,
                IOCTL_RASINDRV_SIN1INT,
                &value,
                sizeof(ULONG),
                NULL,
                0,
                &retlen,
                NULL
                );

    return ret;
}
//---------------------------------------------------------------------------

int main(void)
{
    int     c;
    BOOL    ret;
    DWORD   errno;
    DWORD   onoff;
```

```
    RASINEVENT_INFO info;

    /*
     * イベントオブジェクト、
     * イベントスレッド、
     * ドライバオブジェクトの作成
     */
    if( !CreateIn1IntEventInfo(&info) ){
        printf("CreateIn1IntEvent: NG¥n");
        return -1;
    }

    /*
     * 現在の設定を取得
     *
     * onoff: 1    有効
     *      : 0    無効
     */
    ret = Get_IN1Int(info.hRasin, &onoff);
    if( !ret ){
        errno = GetLastError();
        fprintf(stderr, "ioctl get IN1INT error: %d¥n", errno);
        DeleteIn1IntEvent(&info);
        return -1;
    }
    else {
        fprintf(stdout, "ioctl get IN1INT success: %d¥n", onoff);
    }

    /*
     * IN1 割込みを有効にする
     *  有効の場合向こうに変更し終了
     * onoff: 1    有効
     *      : 0    無効
     */
    if (onoff != 1)
        onoff = 1;
    else
        onoff = 0;

    ret = Set_IN1Int(info.hRasin, onoff);
    if( !ret ){
        errno = GetLastError();
        fprintf(stderr, "ioctl set IN1INT error: %d¥n", errno);
        DeleteIn1IntEvent(&info);
        return -1;
    }
    else {
        fprintf(stdout, "ioctl set IN1INT success: %d¥n", onoff);
        // Resume Thread
        info.fStart = TRUE;
        ResumeThread(info.hThread);
```

```
        if (onoff == 0){
            fprintf(stdout, "IN1 Interrupt off¥n");
            return 1;
        }
    }

    while(1){
        if( kbhit() ){
            c = getch();
            if(c == 'q' || c == 'Q')
                break;
        }
    }

    /*
     * イベントオブジェクト、
     * イベントスレッド、
     * ドライバオブジェクトの破棄
     */
    DeleteIn1IntEvent(&info);

    return 0;
}

//---------------------------------------------------------------------------
```

５－７　　ソフトウェア・ウォッチドッグタイマ機能

５－７－１　　ソフトウェア・ウォッチドッグタイマ機能について

AS シリーズには、ソフトウェアによるウォッチドッグタイマ機能が実装されています。ドライバでタイマ機能を構築し、アプリケーションからウォッチドッグタイマ機能を利用できるようにします。

ソフトウェア・ウォッチドッグタイマは、タイムアウト処理がソフトウェアとなります。OS がハングアップした場合などは、タイムアウト処理が実行されない場合があります。

５－７－２　　ソフトウェア・ウォッチドッグタイマドライバについて

ソフトウェア・ウォッチドッグタイマドライバはウォッチドッグタイマ機能を、ユーザーアプリケーションから利用できるようにします。

ユーザーアプリケーションから動作設定、開始/停止、タイマクリアを行うことができます。

タイムアウト処理をシャットダウン、再起動、ポップアップ通知に設定した場合、タイムアウトはソフトウェア・ウォッチドッグタイマ監視サービスに通知されます。ソフトウェア・ウォッチドッグタイマ監視サービスは設定に従い、シャットダウン、再起動、ポップアップ通知の処理を行います。

タイムアウト処理をイベント通知とした場合、タイムアウト通知をユーザーアプリケーションでイベントとして取得することができます。ユーザーアプリケーションで独自のタイムアウト処理を行うことができます。

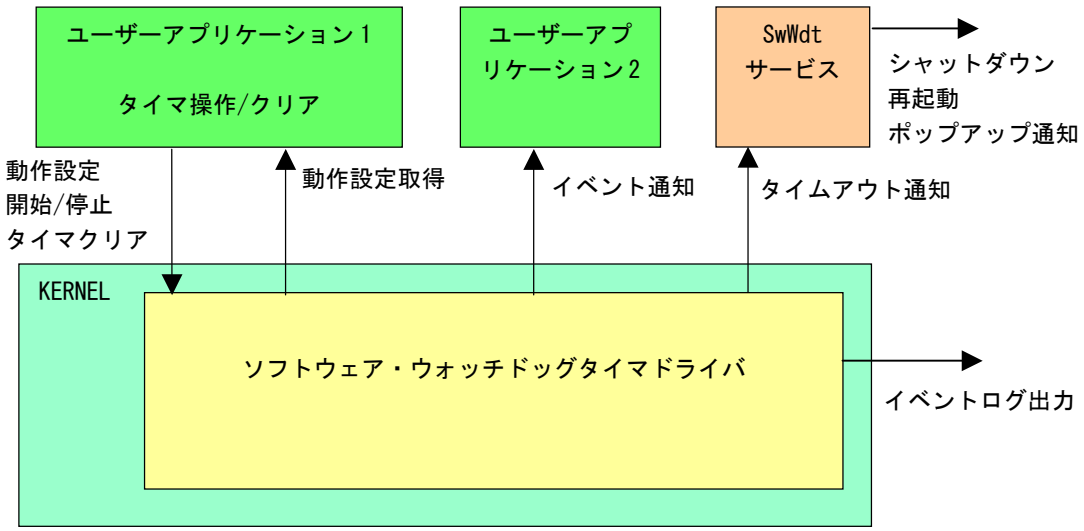

図 5-7-2-1. ソフトウェア・ウォッチドッグタイマドライバ

ソフトウェア・ウォッチドッグタイマは、タイムアウト発生を Windows イベントログに記録することができます。タイムアウト発生直後にイベントログ出力が行われます。

５－７－３　ソフトウェア・ウォッチドッグタイマデバイス

ソフトウェア・ウォッチドッグタイマドライバはソフトウェア・ウォッチドッグタイマデバイスを生成します。ユーザーアプリケーションは、デバイスファイルにアクセスすることによってウォッチドッグタイマ機能を操作します。

ソフトウェア・ウォッチドッグタイマデバイス

デバイスファイル

¥¥.¥SwWdtDrv

説明

ソフトウェア・ウォッチドッグタイマの動作設定、開始/停止、タイマクリアを行うことができます。

レジストリ設定

[KEY]　HKEY_LOCAL_MACHINE¥SYSTEM¥CurrentControlSet¥Services¥SwWdtDrv¥Parameters
[VALUE:DWORD]　Action
タイムアウト時の動作を設定します。タイムアウト時の動作は、デバイスオープン時に、このレジスタ値に初期化されます。(デフォルト値: 2)
「Software Watchdog Timer Config Tool」で設定可能。
0: シャットダウン
1: 再起動
2: ポップアップ通知
3: イベント通知

[KEY]　HKEY_LOCAL_MACHINE¥SYSTEM¥CurrentControlSet¥Services¥SwWdtDrv¥Parameters
[VALUE:DWORD]　Time
タイムアウト時間を設定します。タイムアウト時間は、デバイスオープン時に、このレジスタ値に初期化されます。タイムアウト時間は[Time x 100msec]となります。(デフォルト値: 20)
「Software Watchdog Timer Config Tool」で設定可能。
1～65535

[KEY]　HKEY_LOCAL_MACHINE¥SYSTEM¥CurrentControlSet¥Services¥SwWdtDrv¥Parameters
[VALUE:DWORD]　EventLog
タイムアウト時のイベントログ出力を設定します。イベントログ出力の設定は、デバイスオープン時にこのレジスタ値に初期化されます。(デフォルト値: 1)
「Software Watchdog Timer Config Tool」で設定可能。
0: 無効
1: 有効

CreateFile

デバイスファイル(¥¥.¥SwWdtDrv)をオープンし、デバイスハンドルを取得します。

```
hWdog = CreateFile(
    "¥¥¥¥.¥¥SwWdtDrv",
    GENERIC_READ | GENERIC_WRITE,
    FILE_SHARE_READ | FILE_SHARE_WRITE,
    NULL,
    OPEN_EXISTING,
    0,
    NULL
);
```

ReadFile
使用しません。
WriteFile
使用しません。
DeviceIoControl
● IOCTL_SWWDT_SETCONFIG ソフトウェア・ウォッチドッグタイマの動作設定を行います。 ● IOCTL_SWWDT_GETCONFIG ソフトウェア・ウォッチドッグタイマの動作設定を取得します。 ● IOCTL_SWWDT_CONTROL ソフトウェア・ウォッチドッグタイマの開始/停止を行います。 ● IOCTL_SWWDT_STATUS ソフトウェア・ウォッチドッグタイマの動作状態を取得します。 ● IOCTL_SWWDT_CLEAR ソフトウェア・ウォッチドッグタイマのタイマクリアを行います。

５－７－４　DeviceIoControlリファレンス

IOCTL_SWWDT_SETCONFIG

機能

ソフトウェア・ウォッチドッグタイマの動作設定を行います。

パラメータ

lpInBuf	:	SWWDT_CONFIG を格納するポインタを指定します。
nInBufSize	:	SWWDT_CONFIG を格納するポインタのサイズを指定します。
lpOutBuf	:	NULL を指定します。
nOutBufSize	:	0 を指定します。
lpBytesReturned	:	実際の出力バイト数を受け取る変数へのポインタを指定します。
lpOverlapped	:	NULL を指定します。

SWWDT_CONFIG

```
typedef struct {
      ULONG Action;
      ULONG Time;
      ULONG EventLog;
} SWWDT_CONFIG, *PSWWDT_CONFIG;
```

Action　：タイムアウト時の動作 (0～3)
　　[0: シャットダウン, 1: 再起動, 2: ポップアップ, 3: イベント通知]
Time　：タイムアウト時間 (1～65535)
　　[Time x 100msec]
EventLog　：イベントログ出力 (0, 1)
　　[0: 無効, 1: 有効]

戻り値

処理が成功すると TRUE を返します。失敗の場合は FALSE を返します。

説明

ソフトウェア・ウォッチドッグタイマの動作設定を行います。
動作設定はデバイスオープン時にレジスタ設定値に初期化されます。オープン後に動作を変更したい場合は、この IOCTL コードを実行します。

IOCTL_SWWDT_GETCONFIG

機能

ソフトウェア・ウォッチドッグタイマの動作設定を取得します。

パラメータ

lpInBuf	:	NULL を指定します。
nInBufSize	:	0 を指定します。
lpOutBuf	:	SWWDT_CONFIG を格納するポインタを指定します。
nOutBufSize	:	SWWDT_CONFIG を格納するポインタのサイズを指定します。
lpBytesReturned	:	実際の出力バイト数を受け取る変数へのポインタを指定します。
lpOverlapped	:	NULL を指定します。

戻り値

処理が成功すると TRUE を返します。失敗の場合は FALSE を返します。

説明

ソフトウェア・ウォッチドッグタイマの動作設定を取得します。

IOCTL_SWWDT_CONTROL

機能

ソフトウェア・ウォッチドッグタイマの開始/停止を行います。

パラメータ

lpInBuf	:	タイマ制御情報を格納するポインタを指定します。
nInBufSize	:	タイマ制御情報を格納するポインタのサイズを指定します。
lpOutBuf	:	NULL を指定します。
nOutBufSize	:	0 を指定します。
lpBytesReturned	:	実際の出力バイト数を受け取る変数へのポインタを指定します。
lpOverlapped	:	NULL を指定します。

タイマ制御情報

データタイプ	:	ULONG
データサイズ	:	4 バイト
内容	:	0: タイマ停止 1: タイマ開始（デバイスクローズ時続行） 2: タイマ開始（デバイスクローズ時停止）

戻り値

処理が成功すると TRUE を返します。失敗の場合は FALSE を返します。

説明

ソフトウェア・ウォッチドッグタイマの開始/停止の制御を行います。
タイマ動作を開始する場合、デバイスクローズ時にタイマ動作を停止させるか、続行させるかを指定することができます。

IOCTL_SWWDT_STATUS

機能

ソフトウェア・ウォッチドッグタイマの動作状態を取得します。

パラメータ

lpInBuf	:	NULL を指定します。
nInBufSize	:	0 を指定します。
lpOutBuf	:	タイマ制御情報を格納するポインタを指定します。
nOutBufSize	:	タイマ制御情報を格納するポインタのサイズを指定します。
lpBytesReturned	:	実際の出力バイト数を受け取る変数へのポインタを指定します。
lpOverlapped	:	NULL を指定します。

戻り値

処理が成功すると TRUE を返します。失敗の場合は FALSE を返します。

説明

ソフトウェア・ウォッチドッグタイマの動作状態を取得します。
IOCTL_SWWDT_CONTROL でのタイマ制御状態を取得することができます。

IOCTL_SWWDT_CLEAR

機能

ソフトウェア・ウォッチドッグタイマのタイマクリアを行います。

パラメータ

lpInBuf　　：NULL を指定します。
nInBufSize　　：0 を指定します。
lpOutBuf　　：NULL を指定します。
nOutBufSize　　：0 を指定します。
lpBytesReturned　：実際の出力バイト数を受け取る変数へのポインタを指定します。
lpOverlapped　　：NULL を指定します。

戻り値

処理が成功すると TRUE を返します。失敗の場合は FALSE を返します。

説明

ソフトウェア・ウォッチドッグタイマのタイマクリアを行います。
タイマクリアすると、タイマが初期化されタイマカウントが再開されます。

５－７－５　サンプルコード

●タイマ操作

「¥SDK¥Algo¥Sample¥Sample_SwWdt¥SwWdt」にソフトウェア・ウォッチドッグタイマのタイマ操作のサンプルコードを用意しています。リスト 5-7-5-1 にサンプルコードを示します。

リスト 5-7-5-1. ソフトウェエア・ウォッチドッグタイマ タイマ操作

```
/**
    ソフトウェアウォッチドッグタイマ
    タイマ操作サンプルソース
**/
#include <windows.h>
#include <winioctl.h>
#include <stdio.h>
#include <stdlib.h>
#include <conio.h>

#include "..¥Common¥SwWdtDD.h"

#define DRIVER_FILENAME "¥¥¥¥.¥¥SwWdtDrv"

int main(int argc, char **argv)
{
    HANDLE h_swwdt;
    ULONG retlen;
    BOOL ret;

    int wdt_action;
    int wdt_time;
    int wdt_eventlog;
    SWWDT_CONFIG wdt_config;

    ULONG startval;

    int keych;

    /*
     * 引数から動作を取得します。
     * 引数なしの場合は、動作設定を変更しません。
     */
    if(argc == 4){
        sscanf(*(argv + 1), "%d", &wdt_action);
        sscanf(*(argv + 2), "%d", &wdt_time);
        sscanf(*(argv + 3), "%d", &wdt_eventlog);
    }
    else if(argc != 1){
        printf("invalid arg¥n");
        printf("SwWdtClear.exe [<WDT ACTION> <WDT TIME> <WDT EVENTLOG>]¥n");
        return -1;
    }

    /*
```

```
    * ソフトウェアウォッチドッグの OPEN
    */
    h_swwdt = CreateFile(
                    DRIVER_FILENAME,
                    GENERIC_READ | GENERIC_WRITE,
                    FILE_SHARE_READ | FILE_SHARE_WRITE,
                    NULL,
                    OPEN_EXISTING,
                    0,
                    NULL
              );
    if(h_swwdt == INVALID_HANDLE_VALUE){
        printf("CreateFile: NG¥n");
        return -1;
    }

    /*
    * OPEN 直後は、動作設定がデフォルト値となります。
    * 引数でタイムアウト動作、タイムアウト時間を
    * 指定した場合は、動作設定を変更します。
    */
    if(argc != 1){
        wdt_config.Action = (ULONG)wdt_action;
        wdt_config.Time = (ULONG)wdt_time;
        wdt_config.EventLog = (ULONG)wdt_eventlog;
        ret = DeviceIoControl(
                    h_swwdt,
                    IOCTL_SWWDT_SETCONFIG,
                    &wdt_config,
                    sizeof(SWWDT_CONFIG),
                    NULL,
                    0,
                    &retlen,
                    NULL
              );
        if(!ret){
            printf("DeviceIoControl: IOCTL_SWWDT_SETCONFIG NG¥n");
            CloseHandle(h_swwdt);
            return -1;
        }
    }

    /*
    * 動作設定の表示
    */
    memset(&wdt_config, 0x00, sizeof(SWWDT_CONFIG));
    ret = DeviceIoControl(
                h_swwdt,
                IOCTL_SWWDT_GETCONFIG,
                NULL,
                0,
                &wdt_config,
```

```
                sizeof(SWWDT_CONFIG),
                &retlen,
                NULL
        );
    if(!ret){
        printf("DeviceIoControl: IOCTL_SWWDT_GETCONFIG NG¥n");
        CloseHandle(h_swwdt);
        return -1;
    }
    printf("SwWdt Action = %d¥n", wdt_config.Action);
    printf("SwWdt Time = %d¥n", wdt_config.Time);
    printf("SwWdt EventLog = %d¥n", wdt_config.EventLog);

    /*
     * ウォッチドッグタイマスタート
     */
    startval = SWWDT_CONTROL_START;     // クローズしても停止しません
    ret = DeviceIoControl(
                h_swwdt,
                IOCTL_SWWDT_CONTROL,
                &startval,
                sizeof(ULONG),
                NULL,
                0,
                &retlen,
                NULL
        );
    if(!ret){
        printf("DeviceIoControl: IOCTL_SWWDT_CONTROL NG¥n");
        CloseHandle(h_swwdt);
        return -1;
    }

    /*
     * タイマクリア処理('Q'または'q'キーで終了します)
     */
    while(1){
        if(kbhit()){
            keych = getch();
            if(keych == 'Q' || keych == 'q'){
                break;
            }
        }

        /*
         * クリア
         */
        DeviceIoControl(
            h_swwdt,
            IOCTL_SWWDT_CLEAR,
            NULL,
            0,
```

```
            NULL,
            0,
            &retlen,
            NULL
        );
        Sleep(100);
    }

    /*
     * ウォッチドッグタイマ停止
     */
    startval = SWWDT_CONTROL_STOP;
    ret = DeviceIoControl(
                h_swwdt,
                IOCTL_SWWDT_CONTROL,
                &startval,
                sizeof(ULONG),
                NULL,
                0,
                &retlen,
                NULL
          );
    if(!ret){
        printf("DeviceIoControl: IOCTL_SWWDT_CONTROL NG¥n");
        CloseHandle(h_swwdt);
        return -1;
    }

    CloseHandle(h_swwdt);
    return 0;
}
```

●イベント通知取得

タイムアウト時の動作をイベント通知に設定した場合、ユーザーアプリケーションでタイムアウト通知をイベントとして取得することができます。

「¥SDK¥Algo¥Sample¥Sample_SwWdt¥SwWdt」にソフトウェア・ウォッチドッグタイマのイベント取得処理のサンプルコードを用意しています。リスト 5-7-5-2 にサンプルコードを示します。

リスト 5-7-5-2. ソフトウェエア・ウォッチドッグタイマ イベント通知取得

```
/**
    ソフトウェアウォッチドッグタイマ
    イベント取得サンプルソース
**/
#include <windows.h>
#include <winioctl.h>
#include <stdio.h>
#include <stdlib.h>
#include <conio.h>

#include "..¥Common¥SwWdtDD.h"

#define THREADSTATE_STOP        0
#define THREADSTATE_RUN         1
#define THREADSTATE_QUERY_STOP  2

#define MAX_EVENT   2

enum {
    EVENT_FIN = 0,
    EVENT_USER
};

HANDLE hEvent[MAX_EVENT];
HANDLE hThread;
ULONG ThreadState;

DWORD WINAPI EventThread(void *pData)
{
    DWORD ret;

    printf("EventThread: Start¥n");
    ThreadState = THREADSTATE_RUN;

    /*
     * ウォッチドッグ ユーザーイベントを待ちます
     */
    while(1){
        ret = WaitForMultipleObjects(MAX_EVENT, &hEvent[0], FALSE, INFINITE);
        if(ret == WAIT_FAILED){
            break;
        }
        if(ThreadState == THREADSTATE_QUERY_STOP){
            break;
        }
```

```
        if(ret == WAIT_OBJECT_0 + EVENT_USER){
            printf("EventThread: UserEvent¥n");
        }
    }
    ThreadState = THREADSTATE_STOP;

    printf("EventThread: Finish¥n");
    return 0;
}


int main(int argc, char **argv)
{
    DWORD thid;
    int keych;
    int i;

    /*
     * スレッド終了用イベント
     */
    hEvent[EVENT_FIN] = CreateEvent(NULL, FALSE, FALSE, NULL);

    /*
     * ウォッチドッグ ユーザーイベント ハンドル取得
     */
    hEvent[EVENT_USER] = OpenEvent(SYNCHRONIZE, FALSE, SWWDT_USER_EVENT_NAME);
    if(hEvent[EVENT_USER] == NULL){
        printf("CreateEvent: NG¥n");
        return -1;
    }

    /*
     * ウォッチドッグ ユーザーイベント取得スレッド ハンドル取得
     */
    ThreadState = THREADSTATE_STOP;
    hThread = CreateThread(
                        (LPSECURITY_ATTRIBUTES)NULL,
                        0,
                        (LPTHREAD_START_ROUTINE)EventThread,
                        NULL,
                        0,
                        &thid
                    );
    if(hThread == NULL){
        CloseHandle(hEvent);
        printf("CreateThread: NG¥n");
        return -1;
    }

    /*
     * 'Q'または'q'キーで終了します。
```

```
     */
    while(1){
        if(kbhit()){
            keych = getch();
            if(keych == 'Q' || keych == 'q'){
                break;
            }
        }
    }

    /*
     * スレッドを終了
     */
    ThreadState = THREADSTATE_QUERY_STOP;
    SetEvent(hEvent[EVENT_FIN]);
    while(ThreadState != THREADSTATE_STOP){
        Sleep(10);
    }
    CloseHandle(hThread);
    for(i = 0; i < MAX_EVENT; i++){
        CloseHandle(hEvent[i]);
    }

    return 0;
}
```

第６章　システムリカバリ

本章では、「AS シリーズ用 Windows Embedded Standard 2009 リカバリ/SDK DVD」を使用したシステムのリカバリとバックアップについて説明します。

６－１　　リカバリDVDの起動

AS シリーズ本体を「AS シリーズ用 Windows Embedded Standard 2009 リカバリ/SDK DVD」（以降、「リカバリ DVD」と表記します。）で起動させると、システムのリカバリを行うことができます。リカバリで行える処理は以下のとおりです。

- システムの復旧（工場出荷状態）
- システムのバックアップ
- システムの復旧（バックアップデータ）

リカバリ DVD を起動させる前に、本体に接続されている LAN ケーブル、ストレージ（USB メモリ、SD カードなど）を取り外してください。

リカバリ DVD で起動させるには、USB 接続の DVD ドライブが必要です。DVD ドライブにリカバリ DVD を入れて DVD ドライブを USB で AS シリーズ本体に接続します。AS シリーズ本体の電源を入れ DVD で起動させます。正常に起動すると図 6-1-1 のリカバリメイン画面が表示されます。

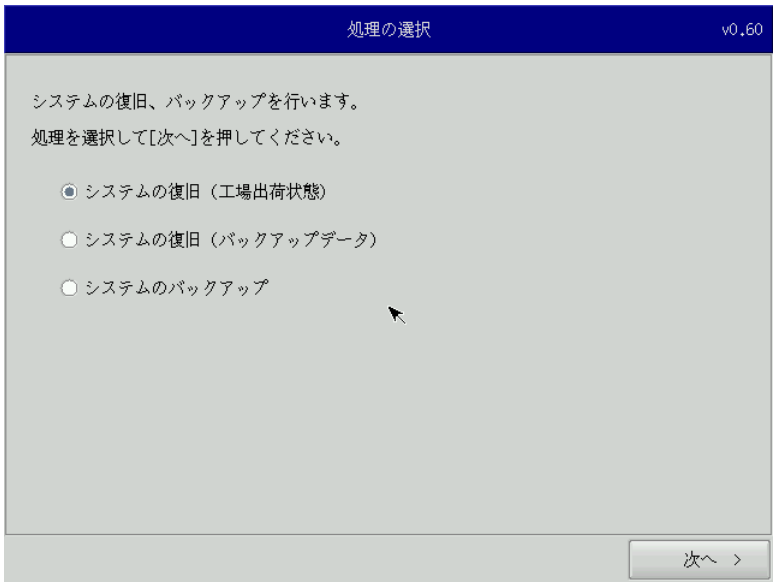

図 6-1-1. リカバリメイン画面

６－２　　システムの復旧（工場出荷状態）

工場出荷イメージを内蔵 SSD に書込むことで、システムを工場出荷状態に復旧することができます。

※　システムを工場出荷状態へ復旧すると内蔵 SSD にあるデータはすべて消えてしまいます。必要なデータがある場合は、復旧作業を行う前に保存してください。

※　工場出荷状態へのシステム復旧には、ソフトウェア使用許諾契約に同意していただく必要があります。ソフトウェア使用許諾契約は、製品に同梱されている「Microsoft Software License Term for: Windows XP Embedded and Windwos Embedded Standard Runtime」に記載されています。システム復旧を行う場合は、内容を確認するようにしてください。

● システム復旧（工場出荷状態）の手順

① LAN ケーブルが接続されている場合は、LAN ケーブルを取り外してください。
② USB メモリ、SD カードなどのストレージメディアが接続されている場合は、ストレージメディアを取り外してください。
③ 「6-1 リカバリ DVD の起動」を参考にリカバリ DVD を起動させます。
④ リカバリメイン画面（図 6-1-1）で[システムの復旧（工場出荷状態）]を選択し、[次へ]ボタンを押します。
⑤ リカバリデータ選択画面（図 6-2-1）が表示されます。製品に合わせてリカバリデータをを選択し、[次へ]ボタンを押します。表 6-2-1 に AS シリーズで選択できるリカバリデータを示します。

表 6-2-1. ASシリーズ リカバリデータ

#	リカバリデータ名	内容
1	WindowsEmbeddedStandard AS シリーズ	AS シリーズ用 Windows Embedded Standard 2009 リカバリデータ

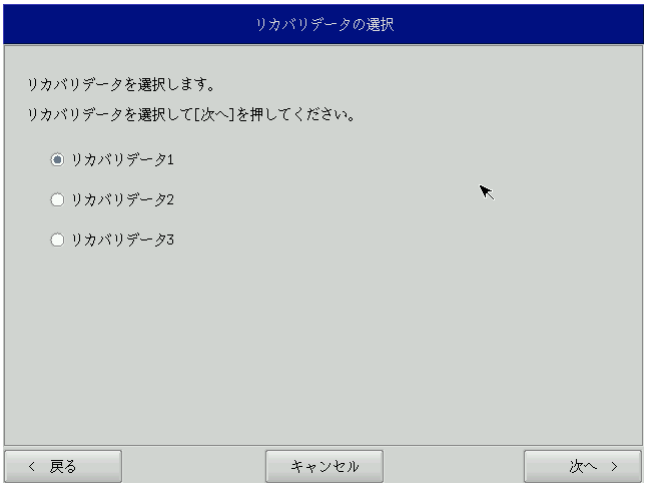

図 6-2-1. リカバリデータ選択画面

⑥ ソフトウェア使用許諾契約確認画面（図 6-2-2）が表示されます。使用許諾契約を確認し、使用許諾契約の諸条件に同意できる場合は[次へ]ボタンを押します。

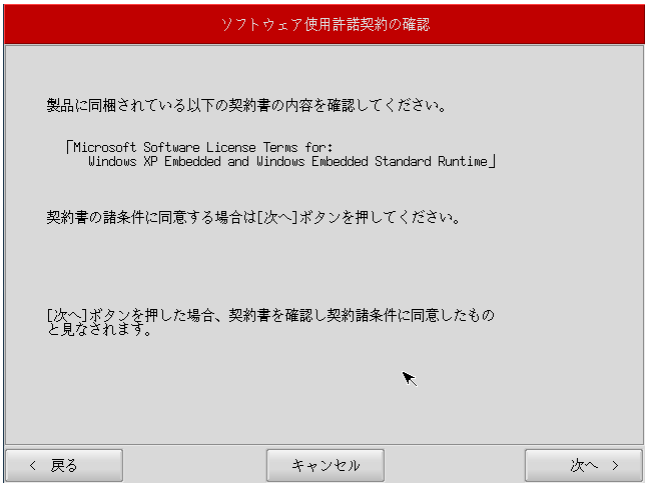

図 6-2-2. ソフトウェア使用許諾契約確認画面

⑦ 確認画面（図 6-2-3）が表示されます。リカバリデータを確認し、 [次へ]ボタンを押して処理を開始します。

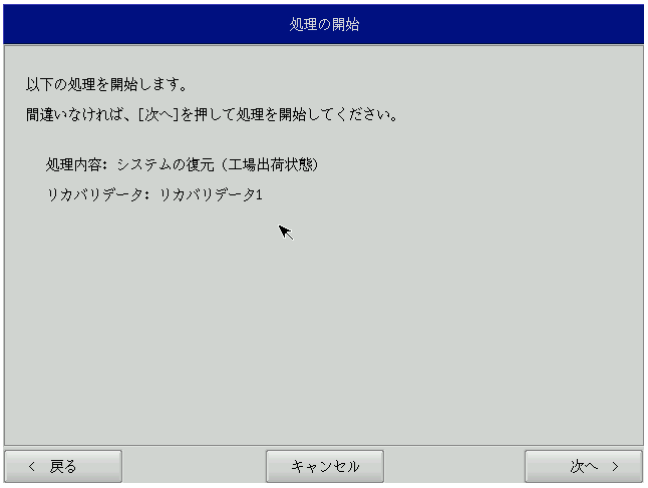

図 6-2-3. 確認画面

⑧　処理が始まります（図 6-2-4）。実行中は DVD ドライブを外したり、電源を落としたりしないでください。

図 6-2-4. 実行中画面

⑨　終了画面（図 6-2-5）が表示されると工場出荷イメージの書込みは完了です。電源を落として DVD ドライブを外します。

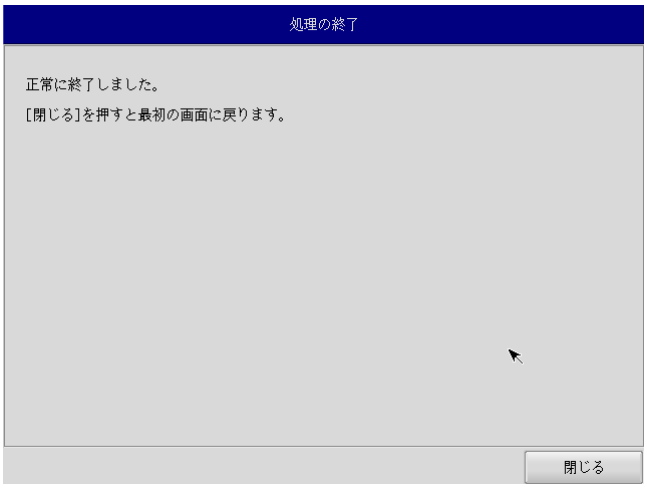

図 6-2-5. 終了画面

⑩　電源を入れます。

※　システムを工場出荷状態へ復旧する場合、一度目の起動時にシステムの初期化を行います。工場出荷イメージ書込み後は、かならず電源を入れて再起動を行ってください。

⑪ Windows Embedded Standard 2009 が起動すると初期化処理が開始されます。

※　デスクトップが表示されますが、初期化処理は引き続き行われています。初期化処理中に行われるダイアログでの操作以外は、システムの操作を行わないでください。

⑫ 初期化処理が完了すると再起動ダイアログが表示されます。(図 6-2-6)
[OK]ボタンを押して再起動を行います。

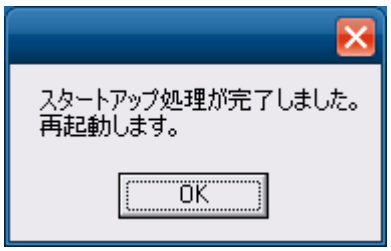

図 6-2-6. 再起動ダイアログ

⑬ 起動を確認します。正常に起動すればシステム復旧作業は完了です。

※　工場出荷状態に復旧した場合、一度目の起動時にシステムから再起動を求められる場合があります。この場合は指示にしたがい再起動してください。

６－３　　システムのバックアップ

現在のシステム状態をファイルに保存します。バックアップファイル保存のために外部メモリとして USB メモリ、または SD カードが必要となります。外部メモリの空き容量は、バックアップファイルの保存に十分な空き容量が必要となります。安全のため 2GByte の空き容量がある外部メモリを用意してください。

※　バックアップファイルのサイズは、システムの状態によって変化しますので注意してください。

※　作成されたバックアップファイルは、バックアップ作業を行った本体でのみ動作します。同じ型の本体であっても、他の本体では動作しませんので注意してください。

●システムのバックアップの手順

①　リカバリメイン画面（図 6-1-1）で[システムのバックアップ]を選択し、[次へ]ボタンを押します。

②　メディア選択画面（図 6-3-1）が表示されます。バックアップファイルを保存するメディアを選択します。USB メモリまたは SD カードを選択し、本体にメディアを接続します。[次へ]ボタンを押します。

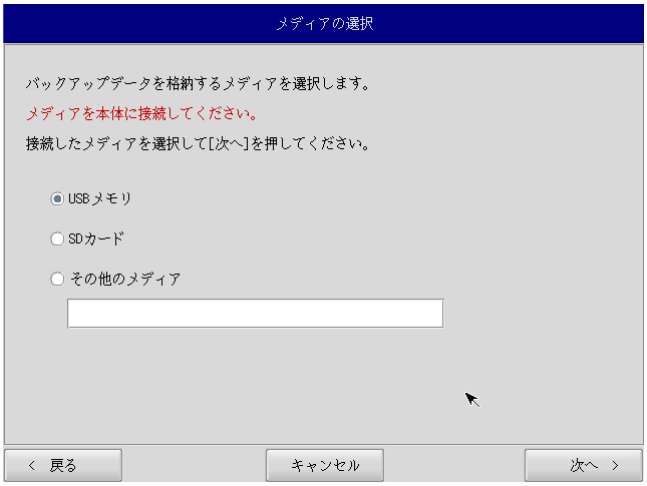

図 6-3-1. メディア選択画面

③　フォルダ選択画面（図 6-3-2）が表示されます。 [参照]ボタンを押します。

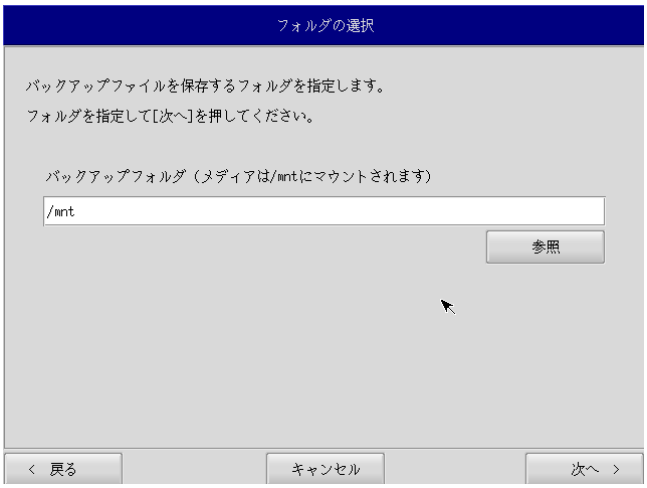

図 6-3-2. フォルダ選択画面

④　フォルダ参照画面（図 6-3-3）が表示されます。②で接続したメディアは、/mnt にマウントされていますので、/mnt 以下のフォルダを選択してください。[OK]を押すとフォルダ選択画面にもどります。

※　USB メモリに backup というフォルダがあり、このフォルダに保存する場合
/mnt/backup
を指定します。

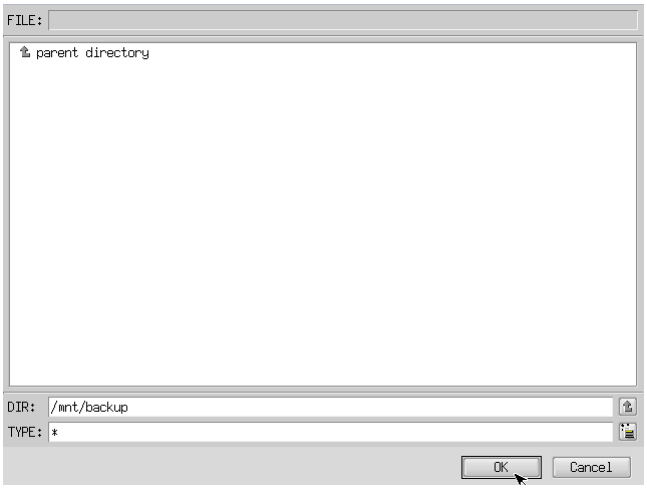

図 6-3-3. フォルダ参照画面

⑤　フォルダ選択画面（図 6-3-2）で指定したバックアップフォルダが入力されていることを確認します。[次へ]ボタンを押します。

⑥　確認画面（図 6-3-4）が表示されます。メディア、保存ファイルを確認します。[次へ]ボタンを押します。

※　保存ファイル名は、現在時刻から自動生成されます。

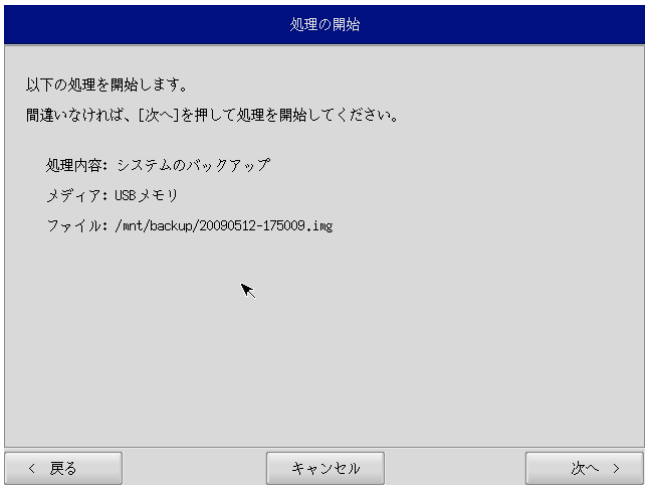

図 6-3-4. 確認画面

⑦　実行中画面（図 6-3-5）が表示され、処理が開始されます。実行中は DVD ドライブ、保存メディアを外さないでください。また、電源を落とさないようにしてください。

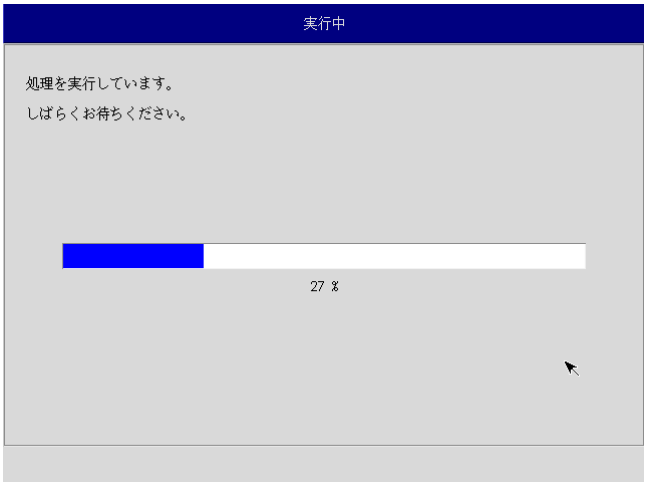

図 6-3-5. 実行中画面

⑧　終了画面（図 6-3-6）が表示されるとバックアップ作業は完了です。電源を落としてください。

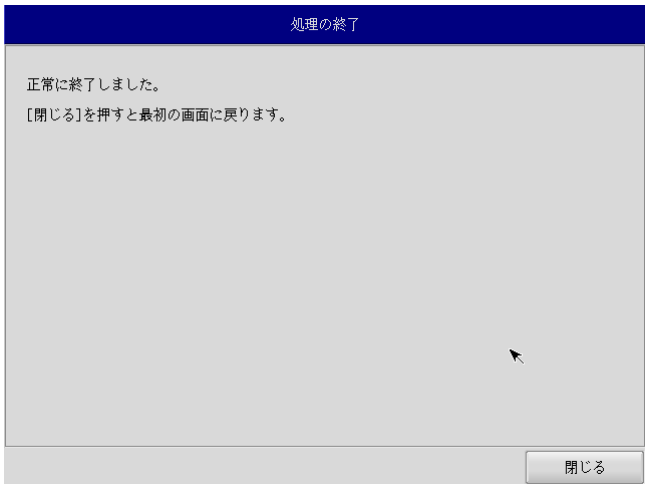

図 6-3-6. 終了画面

６－４　　システムの復旧（バックアップデータ）

「システムのバックアップ」で作成したバックアップファイルを使用して、バックアップファイルの状態に復旧させることができます。

※　バックアップファイルは、必ず対象となる本体で作成されたものを使用してください。他の本体のバックアップファイルでは動作しないので注意してください。

※　バックアップデータで復旧を行うと内蔵 SSD のデータは、バックアップファイルの状態に戻ります。必要なデータがある場合は、復旧作業を行う前に保存してください。

●システムの復旧（バックアップデータ）の手順

① リカバリメイン画面（図 6-1-1）で[システムの復旧（バックアップデータ）]を選択し、[次へ]ボタンを押します。

② メディア選択画面（図 6-4-1）が表示されます。バックアップファイルが保存されているメディアを選択します。USB メモリまたは SD カードを選択し、本体にメディアを接続します。[次へ]ボタンを押します。

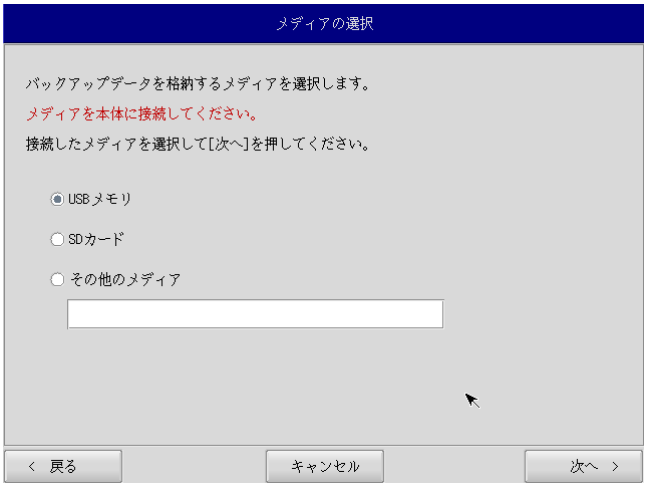

図 6-4-1. メディア選択画面

③ ファイル選択画面（図 6-4-2）が表示されます。[参照]ボタンを押します。

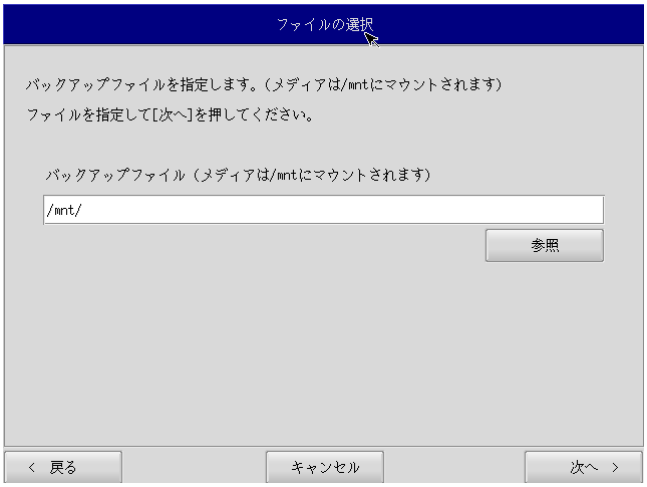

図 6-4-2. ファイル選択画面

④　ファイル参照画面（図 6-4-3）が表示されます。②で接続したメディアは、/mnt にマウントされていますので、/mnt 以下から目的のファイルを探してください。[OK]を押すとファイル選択画面にもどります。

※　USB メモリの¥backup¥20090512-175009.img というバックアップファイルを指定する場合
/mnt/backup/20090512-175009.img
を指定します。

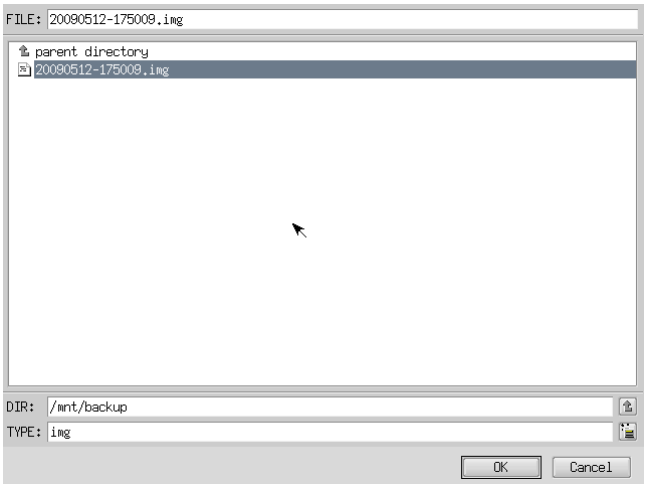

図 6-4-3.　ファイル参照画面

⑤　ファイル選択画面(図 6-4-2)でに指定したバックアップファイルが入力されていることを確認します。[次へ]ボタンを押します。

⑥　確認画面（図 6-4-4）が表示されます。メディア、バックアップファイルを確認します。[次へ]ボタンを押します。

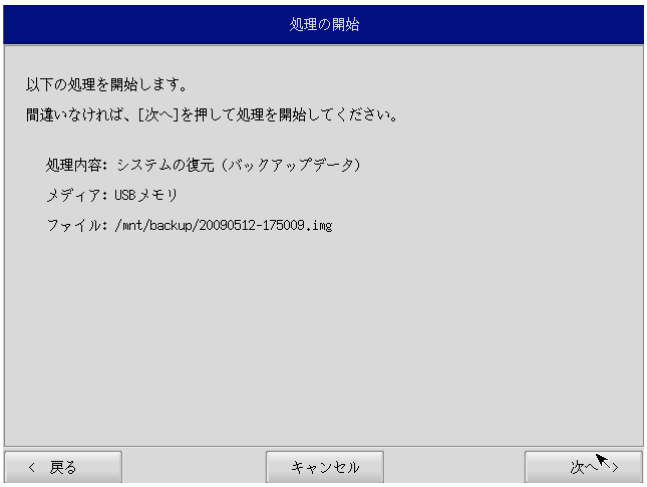

図 6-4-4.　確認画面

⑦　実行中画面（図 6-4-5）が表示され、処理が開始されます。実行中は DVD ドライブ、保存メディアを外さないでください。また、電源を落とさないようにしてください。

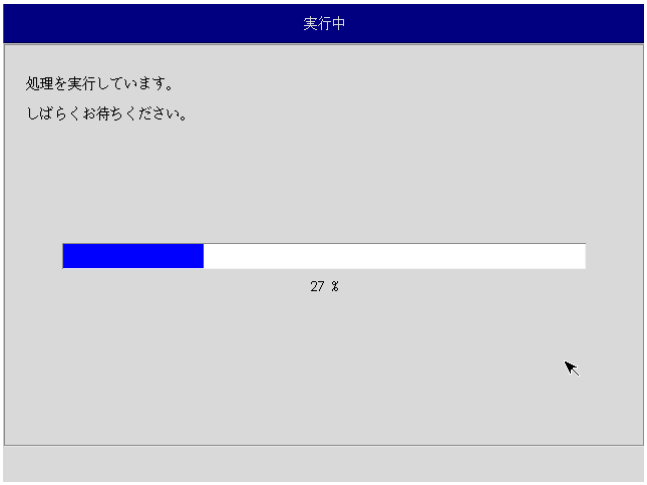

図 6-4-5. 実行中画面

⑧　終了画面（図 6-4-6）が表示されるとシステム復旧作業は完了です。電源を落として DVD ドライブ、保存メディアを外します。

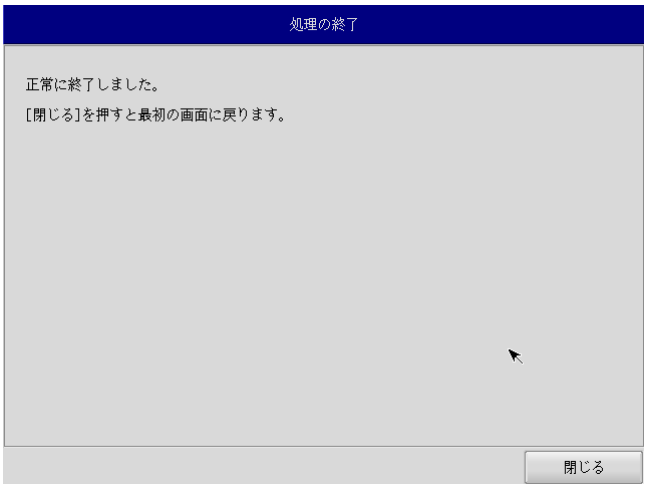

図 6-4-6. 終了画面

⑨　電源を入れ起動を確認します。

付録

A－1　マイクロソフト製品の組込み用OS（Embedded）について

【OS の注意事項】

本製品に搭載している OS は組込み用（Embedded）であり、一般的なパソコンで使用される OS とは異なります。そのためソフトウェアや接続デバイスの動作が異なる（または動作しない、インストールできない）等の可能性がありますので、お客様にて十分な動作確認を実施して頂きますようお願いいたします。

【OS 種別】

本製品には以下のマイクロソフト製品の OS が存在します。各々下記の意味で使用しておりますので、ご認識ください。

○　Windows Embedded Standard 2009

：マイクロソフト社が Windows XP Professional をベースに組込み開発で使用できるようにコンポーネント化した OS

【OS 用のアプリケーション開発】

・Windows Embedded Standard 2009 は多言語 OS の為、日本語 OS と動作、表示が異なる場合があります。アプリケーション開発時にはご注意ください。

・クロス環境によるアプリケーション開発をお願いします。実機での開発はできません。

【I/O 機器の接続について】

本製品のインターフェースを介して周辺デバイスを接続する場合、組込み用（Embedded）OS では通常 OS とは機能が異なります。十分な動作確認を実施してください。

各種 I/O 機器の動作において動作不良が発生した場合には、機器提供メーカーへお問い合わせをお願いいたします。

【提案に際して】

・お客様への提案に際して、事前に装置の寿命年数と条件、保守条件（有寿命部品）等の諸条件の説明をお願いいたします。

・本装置は、医療機器･原子力設備や機器、航空宇宙機器・輸送設備など、人命に関わる設備や機器および高度な信頼性を必要とする設備や機器などへの組込み、また、これらの機器の制御などを目的とした使用は意図されておりません。これらの設備や機器、制御システムなどに本装置を使用した結果、人身事故、財産損害などが生じてもいかなる責任も負いかねます。

・本装置（ソフトウェアを含む）が、外国為替および外国貿易法の規定により、輸出規制品に該当する場合は、海外輸出の際に日本国政府の輸出許可申請等必要な手続きをお取り下さい。また、米国再輸出規制等外国政府の規制を受ける場合には、所定の手続きを行ってください。

＜制約事項＞

- Embedded ライセンスは、組込機器や特定業務用機器の OS としてのみご使用いただけます。汎用目的（「市販のアプリケーションをエンドユーザがインストールして使用する」など）ではご使用いただけません。
- OS はお客様で開発した専用アプリケーションとあわせて利用しなければなりません。必ず専用アプリケーションをインストールしてご使用ください。
- 医療機器･原子力設備や機器、航空宇宙機器・輸送設備など、誤動作により被害が想定される装置への使用はできません。
- 製品構成に関する制限
  —Embedded ライセンス契約であることを示す「Certificate of Authenticity」ステッカーを装置本体に貼り付けて出荷します。
  —OS のインストール媒体は添付できません。復旧媒体（アプリケーション込み）の添付は可能です。
- 専用アプリケーションに関する制限
  —専用アプリケーションのユーザインタフェースからのみ、他のアプリケーションやファンクションにアクセスできるようにしなければなりません。
  —Microsoft のユーザインタフェース（ロゴ、ブートアップ、デスクトップ画面、フォルダ、ツールバーなど）を一切表示してはいけません。
- 組込みシステムの再販売･再頒布に際しマイクロソフト社の OS 製品の COA と APM を各システムに必ず貼付・添付しなければなりません。
- 組込み型システムとは別にマイクロソフト社の OS 製品またはその製品の一部を宣伝したり､価格提示したり、あるいは販売したり再頒布したりしてはなりません。
- ここに定めた条件を守っていないことが判明した場合、株式会社アルゴシステムはマイクロソフト株式会社との契約に従って状況報告をマイクロソフト株式会社に行い、出荷停止・状況改善依頼・調査を行うことができます。

＜用語の説明＞

- 「APM」とは「関連製品資料」のことで、使用許諾製品の再配布可能コンポーネントとしてマイクロソフト社が適宜指定する、使用許諾製品に関連するドキュメント、ソフトウェアや他の有形資料を含んだ外部メデイアを意味します。なお、APM に COA は含まれません。
- 「COA」すなわち「Certificate of Authenticity」は、マイクロソフト社が使用許諾製品のみに作成した削除不可のステッカーを意味します。
- 「組込み型アプリケーション」とは、業界または業務固有のソフトウェアプログラムを意味し、以下の属性をすべて備えているものです。
  （１） 組込み型システムの主要な機能がある。
  （２） 組込み型システムが販売される業界に特有な機能要求に合わせて設計されている。
  （３） 使用許諾製品ソフトウェアに加えて重要な機能がある。
- 「組込み型システム」とは、組込み型アプリケーション用に設計され、組込み型アプリケーションと共に配布され且つ､汎用的なパーソナルコンピューテイングデバイス（パーソナルコンピュータなど）または多機能サーバとして販売もしくは使用されません。また、これらシステムの代替品として商業的に実現不可能な、お客様のイメージ付きコンピューテイングシステムまたはデバイスを意味します。

このマニュアルについて

(1)本書の内容の一部又は全部を当社からの事前の承諾を得ることなく、無断で複写、複製、掲載することは固くお断りします。
(2)本書の内容に関しては、製品改良のためお断りなく、仕様などを変更することがありますのでご了承下さい。
(3)本書の内容に関しては万全を期しておりますが、万一ご不審な点や誤りなどお気付きのことがございましたらお手数ですが巻末記載の弊社までご連絡下さい。その際、巻末記載の書籍番号も併せてお知らせ下さい。

７７Ｗ０１０００６Ｅ　　２０１０年　１０月　第４版
７７Ｗ０１０００６Ｂ　　２００９年　１０月　初版

株式会社アルゴシステム

本社
〒５８７－００２１　大阪府堺市美原区小平尾６５６番地
ＴＥＬ（０７２）３６２－５０６７
ＦＡＸ（０７２）３６２－４８５６

東京支社
〒１０４－００６１　東京都中央区銀座７－１５－８　銀座堀ビル２Ｆ
ＴＥＬ（０３）３５４１－７１７０
ＦＡＸ（０３）３５４１－７１７５

大阪支社
〒５４２－００８１　大阪府大阪市中央区南船場１－１２－３　船場グランドビル３Ｆ
ＴＥＬ（０６）６２６３－９５７５
ＦＡＸ（０６）６２６３－９５７６

名古屋営業所
〒４６１－０００４　愛知県名古屋市東区葵２－３－１５　ふぁみーゆ葵ビル５０３
ＴＥＬ（０５２）９３９－５３３３
ＦＡＸ（０５２）９３９－５３３０

ホームページ　http://www.algosystem.co.jp