Digital
Human Machine Interface

Pro-face®

# PSシリーズGタイプ
# アプリケーション開発キット
# （ADK）

## デベロッパーズマニュアル

# はじめに

このたびは、(株)デジタル製PSシリーズGタイプアプリケーション開発キットをお買い上げいただき、誠にありがとうございます。本書は、PSシリーズGタイプ（以下、PS-Gと称します）上で動作するアプリケーションを開発するために必要な、ライブラリや定義ファイルをパッケージした「PSシリーズGタイプアプリケーション開発キット（ADK）」のデベロッパーズマニュアルです。
この製品を正しくご使用いただくために、本書をよくお読みください。

## お断り

(1)「PSシリーズGタイプアプリケーション開発キット(ADK)」(以下本製品といいます)のプログラムおよびマニュアル類は、すべて(株)デジタルの著作物であり、(株)デジタルがユーザーに対し「ソフトウェア使用条件」に記載の使用権を許諾したものです。当該「ソフトウェア使用条件」に反する行為は、日本国内外の法令により禁止されています。
(2) 本書の内容については万全を期して作成しておりますが、万一お気づきの点がありましたら、(株)デジタル営業担当窓口までご連絡ください。
(3) 前項にかかわらず、本製品を使用したことによるお客様の損害、および逸失利益、または第三者からのいかなる請求につきましても、当社はその責任を負いかねますので、あらかじめご了承ください。
(4) 製品の改良のため、本書の記述と本製品のソフトウェアとの間に異なった部分が生じることがあります。最新の説明は、別冊ないし電子的な情報として提供していますので、あわせてご参照ください。
(5) 本書は、(株)デジタルから日本国内仕様として発売された製品専用です。
(6) 本製品が記録・表示する情報の中に、(株)デジタルまたは第三者が権利を有する無体財産権、知的所有権に関わる内容を含むことがありますが、これは(株)デジタルがこれらの権利の利用について、ユーザーまたはその他の第三者に、何らの保証や許諾を与えるものではありません。



商標・商号の権利については「商標権などについて」を参照してください。

# マニュアル表記について

このマニュアルでは、製品を正しく安全に使用するための重大な注意事項について説明しています。
製品使用前にこの注意事項を読み、製品を正しく使用してください。

## 絵表示について

このマニュアルでは、以下の絵表示を使用して、安全に関する重大な注意事項を説明しています。

| | |
|---|---|
| ⚠ 警告 | **この指示を無視して誤った取り扱いをすると、使用者が死亡または重傷を負う恐れがあります。** |
| ⚠ 注意 | **この指示を無視して誤った取り扱いをすると、使用者が軽傷を負うか物的な損害を受ける恐れがあります。** |
| ⃠ | **正しく使用するために、してはいけない（禁止）事項です。** |
| ❗ | **正しく使用するために、しなくてはならない（強制）事項です。** |

このマニュアルでは、安全上の注意事項のほかには、以下のマークを使用しています。

| | |
|---|---|
| MEMO | **関連する情報や補足説明です。** |

# 商標権などについて

本書に記載の会社名、商品名は、各社の商号、商標(登録商標を含む)またはサービスマークです。本製品の表示・記述の中では、これら権利に関する個別の表示は省略しております。

| 商　　標 | 権利者 |
|---|---|
| Microsoft, MS, Windows, Windows 95, Windows 98, Windows NT, Windows 2000, Windows CE, Windows エクスプローラ, eMbedded Visual C++, eMbedded Visual Basic | 米国マイクロソフト社 |
| Intel, Pentium | 米国インテル社 |
| Pro-face | (株)デジタル |
| IBM, VGA, PC/AT | 米国IBM社 |
| Adobe, Acrobat | アドビシステムズ社 |

なお、上記商号･商標類で、本書での表記が正式な表記と異なるものは以下の通りです。

| 本書での表記 | 正式な表記 |
|---|---|
| Windows 95 | Microsoft® Windows® 95 operating system |
| Windows 98 | Microsoft® Windows® 98 operating system |
| Windows NT | Microsoft® Windows NT® operating system |
| Windows 2000 | Microsoft® Windows® 2000 operating system |
| Windows CE | Microsoft® Windows® CE operating system |
| MS-DOS | Microsoft® MS-DOS® operating system |
| Pentium | Intel® Pentium® processors |
| Acrobat Reader 4.0 | Adobe® Acrobat® Reader 4.0 |
| eMbedded Visual Tools | Microsoft® eMbedded™ Visual Tools 3.0 |
| eMbedded Visual C++ | Microsoft® eMbedded™ Visual C++® 3.0 |
| eMbedded Visual Basic | Microsoft® eMbedded™ Visual Basic® 3.0 |
| ActiveSync 3.1 | Microsoft® ActiveSync® 3.1 |
| Microsoft Custom SDK | Microsoft® Custom Software Development Kit for Windows® CE, Version 3.0 |
| Windows CE Platform SDK (HPC Pro) | Microsoft® Software Development Kit for Windows® CE, Handheld P/C Professional Edition Version 3.01 |

# 製品型式について

PSシリーズGタイプには以下の機種があります。

| シリーズ名 | 形式 | 内容 | 海外規格対応 |
|---|---|---|---|
| PS-600G | PS600G-T11-J1 | OSにはWindows CE 3.0日本語版を搭載しています。<br>定格電圧はAC100Vです。 | 海外規格非対応 |
| | PS600G-T41-J124V | OSにはWindows CE 3.0日本語版を搭載しています。<br>定格電圧はDC24Vです。 | CEマーキング、<br>UL/ｃ-UL(CSA)規格対応 |
| | PS600G-T41-E124V | OSにはWindows CE 3.0英語版を搭載しています。<br>定格電圧はDC24Vです。 | CEマーキング、<br>UL/ｃ-UL(CSA)規格対応 |
| PS-400G | PS400G-T41-J124V | OSにはWindows CE 3.0日本語版を搭載しています。<br>定格電圧はDC24Vです。 | CEマーキング、<br>UL/ｃ-UL(CSA)規格対応 |
| | PS400G-T41-E124V | OSにはWindows CE 3.0英語版を搭載しています。<br>定格電圧はDC24Vです。 | CEマーキング、<br>UL/ｃ-UL(CSA)規格対応 |

ただし、このマニュアルでPS-Gとしているのはは PS600G-T11-J1、PS600G-T41-J124V、および PS400G-T41-J124Vです。

Windows CE 3.0日本語版を搭載した機種のアプリケーション開発には、アプリケーション開発キット日本語版を使用します。

Windows CE 3.0英語版を搭載した機種のアプリケーション開発には、アプリケーション開発キット英語版を使用します。

# CD-ROMの構成

本CD-ROMは以下の構成になっております。

| 品　名 | 備　考 | |
|---|---|---|
| PSシリーズGタイプアプリケーション開発キット(ADK)<br>デベロッパーズマニュアル<br>（本書） | PSシリーズGタイプアプリケーション開発キット（ADK）についてのPDFマニュアル | |
| Acrobat Reader 4.0 | PDFマニュアルを閲覧するために必要なソフトウェア（自己解凍型ファイル） | |
| PSシリーズGタイプ<br>アプリケーション開発キット<br>（ADK） | Microsoft Custom SDK | Windows CEの標準的なライブラリファイル |
| | PSシリーズGタイププラットフォームSDK（Microsoft Custom SDK用プラットフォーム デベロッパー コンポーネント） | RASなどPSシリーズGタイプ特有のハードウェアにアクセスするためのライブラリファイルやヘッダーファイル |
| ActiveSync 3.1 | アプリケーション開発用のパソコンとPS-Gを通信させるためのプログラム | |

# 使用上の注意

## ⚠ 警告

### 本製品の使用について

- タッチパネルスイッチは非常停止用スイッチとして使えません。産業用ロボットほか、労働大臣が指定する産業用機械設備の非常停止用スイッチとしては、必ず人間が直接操作するスイッチを設置することが関係法令で義務づけられています。また、これ以外の装置設備でも、安全確保のため、必ず同様のスイッチを設置してください。

## ⚠ 注意

### ディスクの取り扱いについて

- パソコン本体の電源のON/OFF は、ディスクを抜いてから行ってください。
- ディスクドライブのランプが点灯しているときは、CD-ROM を取り出さないでください。
- CD-ROM の記録面に手を触れないでください。
- 極端な高温や低温、湿気やホコリの多い場所にディスクを置かないでください。
- プログラム実行中に、パソコン本体の電源をOFFしないでください。

# 目次

はじめに ..... 1
マニュアル表記について ..... 2
商標権などについて ..... 3
製品型式について ..... 4
CD-ROMの構成 ..... 5
使用上の注意 ..... 6

第1章　開発環境
1 概要 ..... 1-2
2 ハードウェア環境 ..... 1-3
3 ソフトウェア環境 ..... 1-5
3.1 必要な開発用ソフトウェア ..... 1-5
3.2 ADK内のファイルについて ..... 1-6
4 アプリケーション開発ツールのインストール ..... 1-8
5 リモート接続手順 ..... 1-13
5.1 パソコン側の設定 ..... 1-13
5.2 PS-G側の設定 ..... 1-16
5.3 接続 ..... 1-17

第2章　アプリケーション開発手順
1 eMbedded Visual C++での開発 ..... 2-2
1.1 プロジェクトの作成 ..... 2-2
1.2 ビルドとダウンロード ..... 2-5
1.3 実行 ..... 2-6
2 eMbedded Visual Basicでの開発 ..... 2-7
2.1 プロジェクトの作成 ..... 2-7
2.2 ダウンロードと実行 ..... 2-9
3 エミュレーションモードでのデバッグ ..... 2-12
4 オートスタート ..... 2-14

第3章　ライブラリインターフェイスリファレンス
1 バックライトドライバ ..... 3-3
1.1 バックライトドライバAPI一覧 ..... 3-3
1.2 関数仕様詳細 ..... 3-4

2 SRAMドライバ ..... 3-9
2.1 SRAMドライバAPI一覧 ..... 3-9
2.2 関数仕様詳細 ..... 3-9
3 GMU-BUSドライバ ..... 3-14
3.1 GMU-BUSドライバAPI一覧 ..... 3-14
3.2 関数仕様詳細 ..... 3-14
4 RASドライバ ..... 3-20
4.1 RASドライバAPI一覧 ..... 3-20
4.2 関数仕様詳細 ..... 3-22
4.3 レジスタ詳細 ..... 3-42
5 タッチパネルドライバ ..... 3-46
5.1 タッチパネルドライバAPI一覧 ..... 3-46
5.2 関数仕様詳細 ..... 3-46

索引

# 1 開発環境

1 概要
2 ハードウェア環境
3 ソフトウェア環境
4 アプリケーション開発ツールのインストール
5 リモート接続手順

# 1　概要

アプリケーション開発キット（以下、ADKと称します）とは、PS-G上で動作するアプリケーションを開発するために必要な、ライブラリや定義ファイルをパッケージしたものです。

PS-Gは、(株)日立製作所製RiscCPU SH4(200MHz)を搭載した高性能なパネルコンピュータで、OSとしてWindows CE 3.0を使用しています。

PS-Gで動作するアプリケーションを開発するにはADKの他、eMbedded Visual Toolsが必要です。

eMbedded Visual Toolsは、通常のWin32アプリケーション統合開発環境（Visual Studio）と同じ操作性を持ち、PS-G用のアプリケーションを作成することができます。Win32アプリケーションのプログラミング知識を利用でき、デバッグなどのツールも同じように使用できますので、簡単にPS-Gで動作するWindows CEのアプリケーションを開発できます。Windows CEで使用するAPI（アプリケーションインターフェイス）はWin32 APIのサブセットとなっており、開発効率を高めています。

**MEMO**

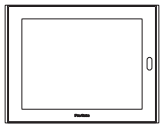
- **eMbedded Visual Toolsは本パッケージには含まれておりません。別途購入してください。**
- **Win32 APIの中には、Windows CEで使用できないAPIもあります。詳細は、MSDNを参照し、Windows CEのAPIが使用できるかどうかをご確認ください。**

## 2　ハードウェア環境

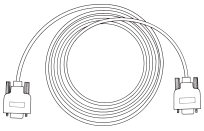
PS-G用のアプリケーションを開発し、動作させるために必要なハードウェア構成を以下に示します。

PS-G本体とアプリケーション開発用のパソコンは、ADKに付属のダウンロードケーブルで接続します。アプリケーション開発用のパソコンで開発したアプリケーションは、PS-Gにダウンロードし実行することができます。またCFカードが読み書きできるパソコンであれば、開発したアプリケーションをCFカードにコピーし、PS-Gにて実行することもできます。

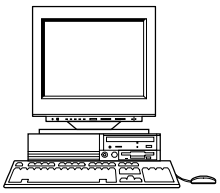
ダウンロードしたアプリケーションや、PS-Gのコントロールパネルで設定した内容などを保存するには、CFカードが必要です。



PS-G



ダウンロードケーブル
（ADKに付属）
PSG-CB01



アプリケーション開発用パソコン

CFカード
GP077-CF20 16Mバイト
GP077-CF30 32Mバイト

**MEMO**

- （株）デジタル製CFカードを使用してください。他社製のカードを使用した場合、PS-Gの仕様を満足しない場合があります。
- CFカードのデータは、必ずバックアップしてください。

アプリケーション開発用パソコンに必要なハードウェア環境を以下に示します。詳細は、各パッケージの必要なハードウェア環境をご確認ください。

**＜アプリケーション開発用パソコンのハードウェア環境＞**

| | |
|---|---|
| 基本ソフトウェア | Windows NT® Workstation 4.0(Service Pack 5以上)、Windows® 2000 ProfessionalまたはWindows® 98 Second Edition |
| コンピュータ本体 | Pentium CPUを搭載したパソコン<br>(Pentium 150MHz以上推奨) |
| メモリ | Windows 2000またはWindows NT 4.0の場合は32Mバイト<br>(48Mバイト以上推奨)<br>Windows 98の場合は24Mバイト（48Mバイト以上推奨） |
| ハードディスク<br>空き容量 | eMbedded Visual C++およびWindows CE Platform SDK (HPC Pro)、またはeMbedded Visual BasicおよびWindows CE Platform SDK (HPC Pro)で360Mバイト以上必要。<br>ADKで100Mバイト必要。 |
| ディスク装置 | CD-ROMドライブ必須 |

## ⚠注意

**Windows 98をアプリケーション開発用のパソコンに使用すると、デバッグ時エミュレーション機能やリモートツールなどが一部動作しません。ビルドだけでなくデバッグの必要がある開発を行う場合は、アプリケーション開発用のパソコンに、Windows NT 4.0またはWindows 2000をご使用ください。**

# 3　ソフトウェア環境

PS-Gで動作するアプリケーションを開発するにはeMbedded Visual ToolsとADKが必要です。eMbedded Visual ToolsにはeMbedded Visual C++およびeMbedded Visual Basicが含まれています。ADKはそのどちらにも対応しています。

## 3.1　必要な開発用ソフトウェア

以下のソフトウェアをインストールします。

| 品　名 | 備　考 | |
|---|---|---|
| eMbedded Visual Tools | eMbedded Visual C++またはeMbedded Visual Basic | |
| | Windows CE Platform SDK (HPC Pro) | 開発用パソコンでWindows CEのエミュレーション機能を使用する場合に必要 |
| PSシリーズGタイプ アプリケーション開発キット (ADK) | Microsoft Custom SDK | Windows CEの標準的なライブラリファイル |
| | PSシリーズGタイププラットフォームSDK（Microsoft Custom SDK用プラットフォーム デベロッパー コンポーネント） | RASなどPSシリーズGタイプ特有のハードウェアにアクセスするためのライブラリファイルやヘッダーファイル |
| ActiveSync 3.1 | アプリケーション開発用のパソコンとPS-Gを通信させるためのプログラム（ADKに付属） | |
| TCP/IPプロトコル | イーサネットを利用してダウンロードやデバッグを行う場合に必要（開発環境に応じてネットワークの設定をしてください） | |

eMbedded Visual Toolsには以下のリモートツールが含まれています。リモートツールを使用すると開発用のパソコンからActiveSync3.1を通してPS-G内部の情報を取得できるため、アプリケーションのデバッグや動作確認を効率よく行うことができます。

- リモートスパイ++（スパイ）
  ターゲットアプリケーションがどのようなメッセージを受け取ったかを確認したり、ウインドウハンドルやクラスなどの確認に使用します。
- リモートレジストリエディタ（レジストリエディタ）
  ターゲットのレジストリの編集を行います。
- リモートズームイン（ズーム）
  ターゲットの画面をキャプチャしてアプリケーション開発用のパソコンに取り込むことができます。
- リモートファイルビューア（ファイルビューア）
  ターゲットのファイル構成を確認することができます。
- リモートヒープウォーカ（ヒープウォーカ）
  ヒープIDやカレントヒープの確認を行います。
- リモートプロセスビューア（プロセスビューア）
  ターゲットで実行中のプロセス、スレッド、モジュールやメモリ領域の確認を行うことができます。

## 3.2 ADK内のファイルについて

ADKは「Microsoft Custom SDK」と「PSシリーズGタイププラットフォームSDK」の2つの開発環境を含んでいます。ADKインストール後は、各ファイルが以下のフォルダに展開されます。

### Microsoft Custom SDK

Windows CEの標準的なライブラリファイル

```
Windows CE Tools\Wce300\Bin
                        \Psgwce30\Atl
                                 \Help
                                 \Include
                                 \Lib
                                 \Mfc
                                 \Target
                                 \Template
                                 \Vbsdk
```

## PSシリーズGタイププラットフォームSDK

Windows CE Tools\Wce300\Psgwce30\Include\bldrvapi.h
\sramdrvapi.h
\gmudrvapi.h
\rasdrvapi.h
\tchdrvapi.h

Windows CE Tools\Wce300\Psgwce30\Lib\Sh4\bldrvif.lib
\sramdrvif.lib
\gmudrvif.lib
\rasdrvif.lib
\touchdrvif.lib
\bldrvif.exp
\sramdrvif.exp
\gmudrvif.exp
\rasdrvif.exp
\touchdrvif.exp

Windows CE Tools\Wce300\Psgwce30\Vbsdk\Sh4\bldrvif.lib
\sramdrvif.lib
\gmudrvif.lib
\rasdrvif.lib
\touchdrvif.lib
\bldrvif.exp
\sramdrvif.exp
\gmudrvif.exp
\rasdrvif.exp
\touchdrvif.exp
\bldrvdef.bas
\sramdrvdef.bas
\gmudrvdef.bas
\rasdrvdef.bas
\touchdrvdef.bas

# 4　アプリケーション開発ツールのインストール

アプリケーション開発ツールをインストールする前に「第1章 2 ハードウェア環境＜アプリケーション開発用パソコンのハードウェア環境＞」 を参照し、アプリケーション開発用のパソコンに必要な環境が整っているか確認してください。

## ⚠ 注意

**Windows 98をアプリケーション開発用のパソコンに使用すると、デバッグ時にエミュレーション機能やリモートツールなどが一部動作しません。ビルドだけでなくデバッグの必要がある開発を行う場合は、アプリケーション開発用のパソコンに、Windows NT 4.0またはWindows 2000をご使用ください。**

アプリケーション開発ツールのインストールは以下の手順で行ってください。順番を変更すると、誤動作の原因となりますのでご注意ください。

(1) eMbedded Visual Toolsのインストール
(2) eMbedded Visual C++用ADKまたはeMbedded Visual Basic用ADKのインストール
(3) ActiveSync 3.1のインストール

### eMbedded Visual Toolsのインストール

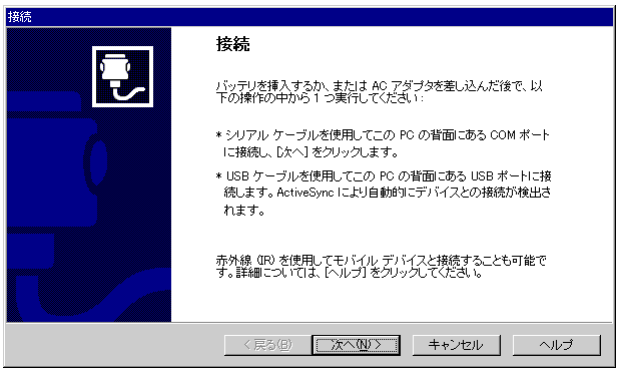
(1) eMbedded Visual Toolsをインストールします。
eMbedded Visual ToolsにはeMbedded Visual C++とeMbedded Visual Basicが含まれています。必要に応じてインストールしてください。

### ADKのインストール

(1) ADK CD-ROMをパソコンのCD-ROMドライブにセットし、eMbedded Visual C++用ADKまたはeMbedded Visual Basic用ADKを以下の手順でインストールします。
- eMbedded Visual C++用ADK
CD-ROMの[\jp\psg-adk\vc]フォルダにある[psgsdk.exe]を起動し、画面の指示に従ってインストールを行ってください。
- eMbedded Visual Basic用ADK
CD-ROMの[\jp\psg-adk\vb]フォルダにある[psgsdk.exe]を起動し、画面の指示に従ってインストールを行ってください。

## ActiveSync 3.1のインストールとPS-Gとの接続

(1) ActiveSync 3.1をインストールします。ADK CD-ROMの[\jp\async]フォルダにある[msasync.exe]を起動します。画面の指示に従ってインストールを行ってください。
インストールを進めるうちに、PS-Gとの接続待ち状態になります。ここではまだ 次へ (N)> ボタンをクリックせず、手順（11）までそのままの状態でお待ちください。



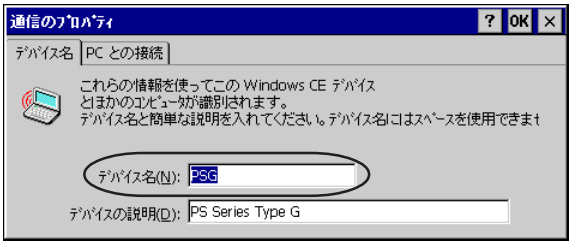

(2) PS-Gに電源を投入し、パソコンのCOMポートとPS-Gのシリアル I/F（COM1）をADK付属のダウンロードケーブルで接続します。

(3) PS-Gのコントロール パネルを開きます。
[スタート] ボタンをクリックし、[設定]をポイントします。次に、[コントロール パネル]をクリックします。

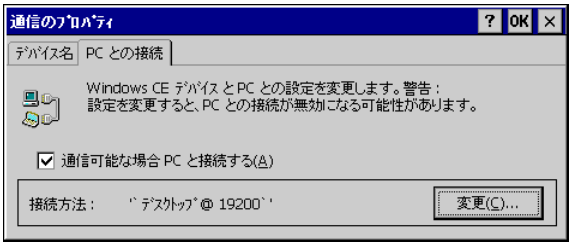
(4) 

をダブルクリックして[通信のプロパティ]を開きます。

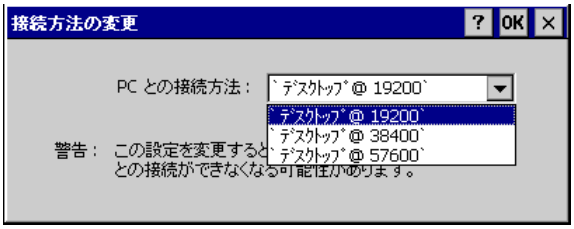
(5) [デバイス名]にPS-Gを識別する名前を入力します。




(6) [PCとの接続]タブで、 変更(C)... ボタンをクリックします。




(7) 必要に応じてパソコンとの接続方法を選択します。



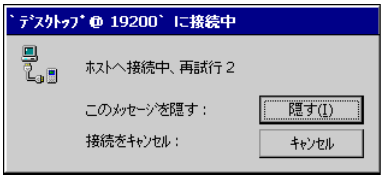

(8) OK ボタンで各設定を確定し、[通信のプロパティ]を終了します。

(9) コントロール パネルにある 

を起動し、設定内容をCFカードに保存します。その後、PS-Gを再起動してください。

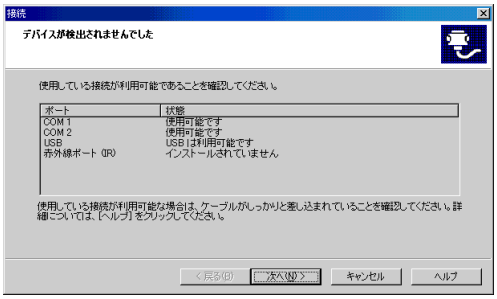
(10) PS-G再起動後、マイ コンピュータの[\Windows]フォルダを開きます。

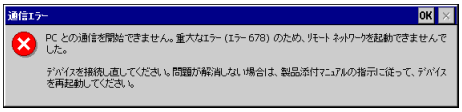
(11) 手順(1)で表示されたダイアログボックスの 次へ (N)> ボタンをクリックし、ほぼ同時に  （接続を実行するプログラムファイル）を起動します。


**パソコン側**　　**PS-G側**

PS-Gとの接続と、  起動のタイミングが合わなかった場合、接続が確立されません。パソコンおよびPS-Gにそれぞれ以下の画面が表示されます。再度手順(11)を実行してください。

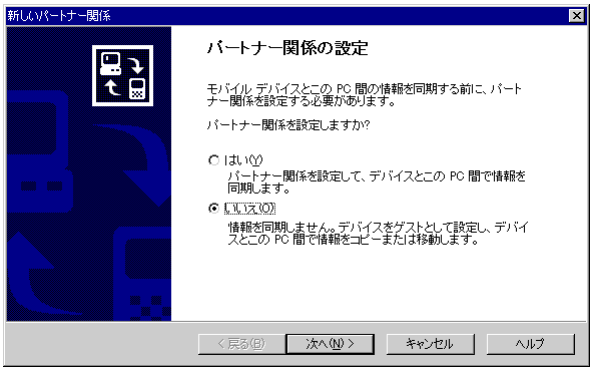


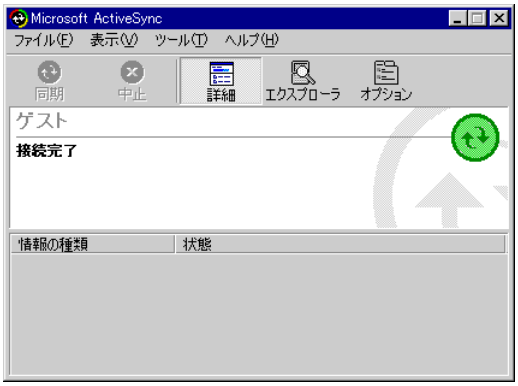

**パソコン側**　　**PS-G側**

(12) 接続が確立すると、パソコン側に以下の画面が表示されます。

「パートナー関係を設定しますか？」で「いいえ」を選択し、次へ (N)>ボタンをクリックします。




(13) 接続が完了すると、パソコン側ではActivesync 3.1の画面が表示されます。




PS-G側のタスクバーには接続を示すアイコンが表示されます。


**MEMO**

**一度接続するとActiveSync 3.1はパソコンに常駐し、接続の設定が保存されます。次回からはPS-Gで**  **を起動するだけで接続できます。**

repllog

# 5　リモート接続手順

アプリケーション開発用のパソコンからPS-Gに対してプログラムのダウンロードやデバッグをするために、ActyveSync 3.1でアプリケーション開発用のパソコンとPS-Gの接続を確立します。

**MEMO**

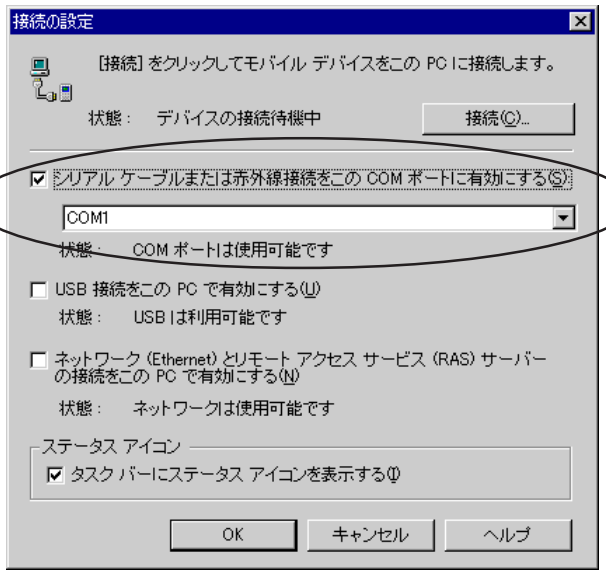
**イーサネット経由でダウンロードすると、ダウンロード時間を大幅に短縮できます。イーサネットを利用するには、開発環境に応じてネットワークの設定が必要です。**

PS-Gに電源を投入し、パソコンのCOMポートとPS-Gのシリアル I/F（COM1）をADK付属のダウンロードケーブルで接続します。イーサネットを利用する場合はイーサネットケーブルも接続します。

## 5.1　パソコン側の設定

### ActiveSync 3.1 - 接続の設定

ActiveSync 3.1を起動します。[ファイル]メニューから[接続の設定]を選択し、ADK付属のダウンロードケーブルを接続しているポートを指定します。



## eMbedded Visual Tools - プラットフォームマネージャの構成

プラットフォームマネージャの構成にPS-Gのデバイス名を登録します。これによって、ダウンロード先にPS-Gを指定できるようになります。また、通信方法にあわせてデバイスのプロパティを設定します。

(1) プラットフォームマネージャを以下の方法で表示します。

eMbedded Visual C++では、[ツール]から[Platform Managerを構成]を選択します。

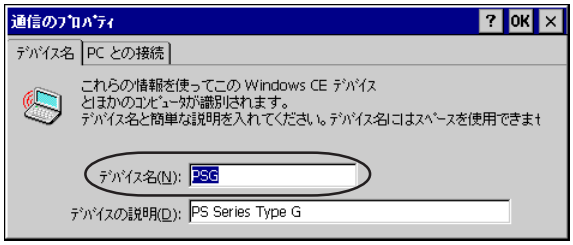
eMbedded Visual Basicでは、[プロジェクト]から[Project1のプロパティ]を選択し、

ターゲットの構成(C) ボタンをクリックします。

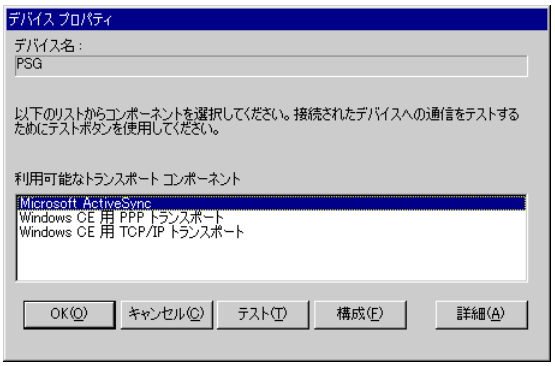

(2) [PSG]を選択し、 デバイスを追加(A) ボタンをクリックしてデバイス名を追加します。


**(図はデバイス名がPSGの場合)**

**MEMO** **このデバイス名は、PS-G側の設定とあわせる必要があります。[通信のプロパティ]でPS-Gに付けたデバイス名と同じ名前を指定してください。参照→第1章 5.2 PS-G側の設定**

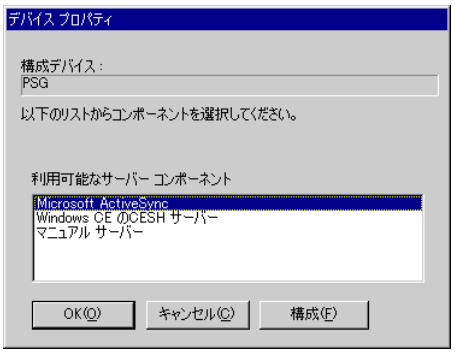


(3) プロパティ(P)... ボタンをクリックして[デバイスプロパティ]ダイアログボックスを表示します。「利用可能なトランスポートコンポーネント」で「Microsoft ActiveSync」を選択します。イーサネットを利用する場合は「Win CE用 TCP/IPトランスポート」を選択します。




**MEMO**　**イーサネット経由でダウンロードするとダウンロード時間を大幅に短縮できます。イーサネットを利用するには、開発環境に応じてネットワークの設定が必要です。**

(4) 詳細(A) ボタンをクリックすると、以下の画面が表示されます。「利用可能なサーバーコンポーネント」で「Microsoft ActiveSync」を選択します。




(5) OK ボタンをクリックして終了します。

## 5.2 PS-G側の設定

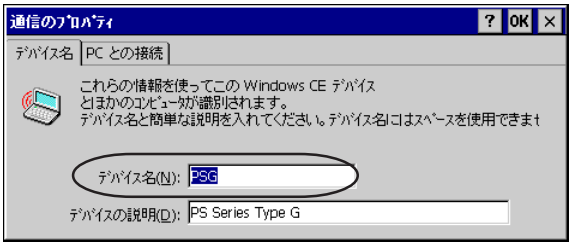
### コントロール パネル - 通信の設定

(1) PS-Gのコントロール パネルを開きます。
[スタート] ボタンをクリックし、[設定]をポイントします。次に、[コントロール パネル]をクリックします。

(2) 

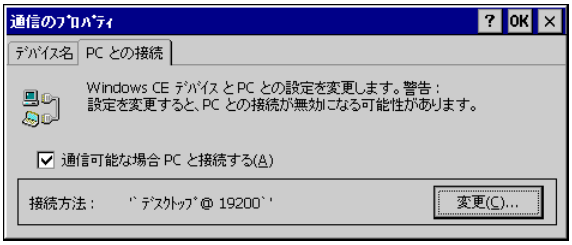
をダブルクリックして[通信のプロパティ]を開きます。


(3) [デバイス名]にPS-Gを識別する名前を入力します。

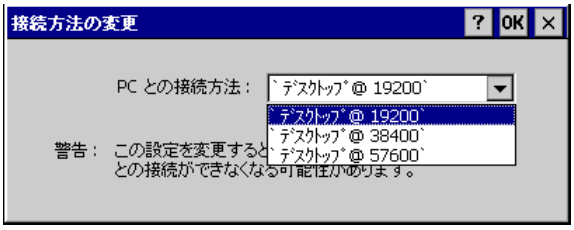



**MEMO** **デバイス名は、パソコン側の設定とあわせる必要があります。[プラットフォームマネージャの構成]で付けたPS-Gのデバイス名と同じ名前を指定してください。参照→「第1章 5.1 パソコン側の設定」**

(4) [PCとの接続]タブで、 変更(C)... ボタンをクリックします。




(5) 必要に応じてパソコンとの接続方法を選択します。




(6) OK ボタンで各設定を確定し、[通信のプロパティ]を終了します。

(7) コントロール パネルにある 

を起動し、設定内容をCFカードに保存します。その後、PS-Gを再起動してください。

## 5.3　接続

ActiveSync 3.1インストール時に一度接続を行った場合は 

を起動し、「第1章 4 アプリケーション開発ツールのインストール」の手順(12)以降に従って接続します。

接続を一度も行っていない場合、ActiveSync 3.1を起動し、[ファイル]から[接続]を選択すると、接続待ちの状態になりますので、「第1章 4 アプリケーション開発ツールのインストール」の手順に従って接続します。

# 2 アプリケーション開発手順

1 eMbedded Visual C++での開発
2 eMbedded Visual Basicでの開発
3 エミュレーションモードでのデバッグ
4 オートスタート

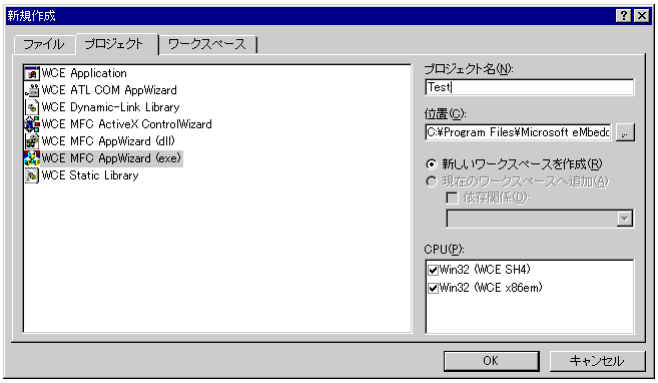
# 1 eMbedded Visual C++での開発

eMbedded Visual C++を使用して、画面にウィンドウを表示するための簡単なアプリケーションの作成手順を説明します。

## 1.1 プロジェクトの作成

(1) eMbedded Visual C++を起動し、[ファイル]メニューから[新規作成]を選択します。[プロジェクト]タブにて、作成したいプログラムの種類を選択し、プロジェクト名を入力します。また、CPUは「Win32(WCE SH4)」を指定します。パソコンのエミュレーションモードでデバックする場合は、「Win32(WCE x86em)」も指定します。

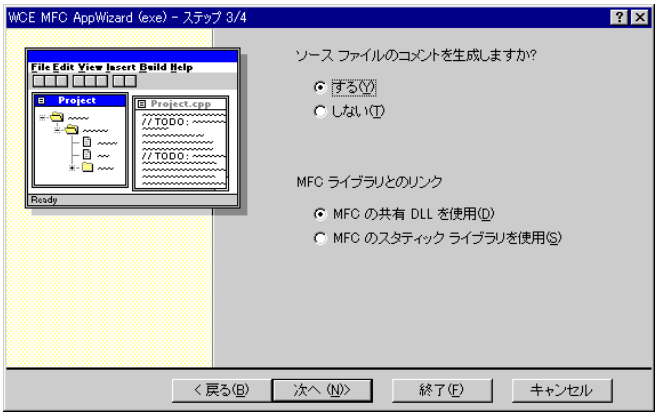
OK ボタンを押して次に進みます。



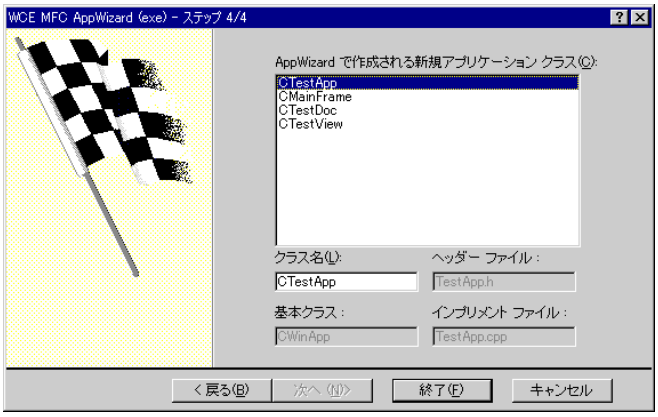

(2) 必要な項目を選択し、次へ (N)> ボタンで次のステップへ進みます。
ダイアログベースのプログラムの時は、「ダイアログベース」を選択します。

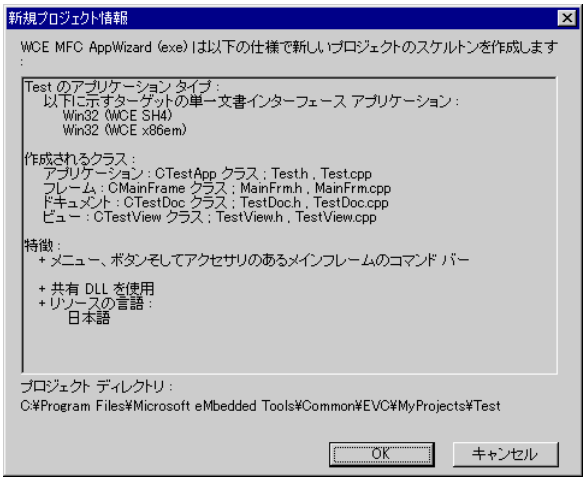
(3) 必要な項目を選択し、次へ (N)> ボタンで次のステップへ進みます。


(4) 必要な項目を選択し、次へ (N)> ボタンで次のステップへ進みます。




(5) 最後のステップで 終了(F) ボタンを押すと、[新規プロジェクト情報]ダイアログボックスが表示されます（次頁）。

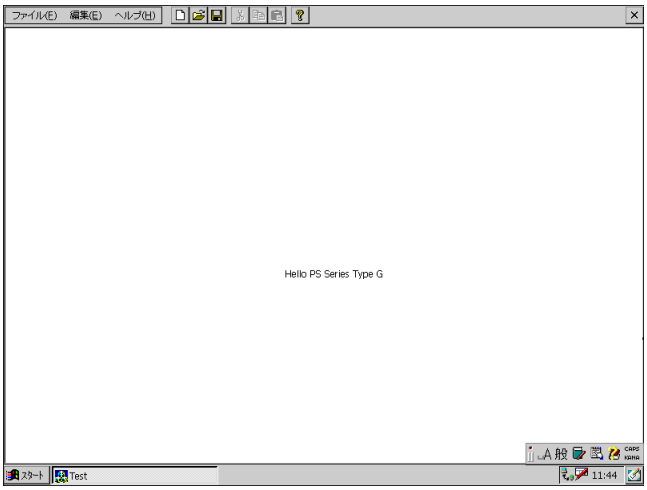





(6) OK ボタンを押すと、プロジェクトが作成されます。

## 1.2　ビルドとダウンロード

例としてウィンドウの中心に“Hello PS Series Type G”と表示させるプログラムを作成します。




ViewクラスのOnDrawに以下の一文を追加します。

```
pDC->ExtTextOut(350,300,NULL,NULL,TEXT(“Hello PS Series Type G”),NULL);
```

プログラムをビルドすると同時に指定されたデバイスへダウンロードします。ビルドする前に、[WCE構成]ツールバーの「アクティブWCE構成の選択」、「アクティブ構成の選択」および「規定のデバイスの選択」を設定します。ビルドするプログラムの用途によって[WCE構成]ツールバー設定は異なります。

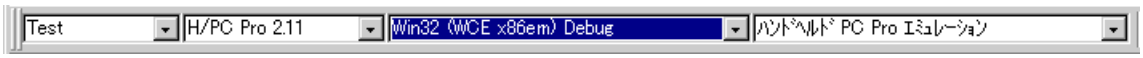
- x86エミュレーションモードでのデバッグ用プログラム
- PS-Gでのデバッグ用プログラム
- PS-Gでのリリース用プログラム

それぞれの場合のビルド方法を以下に示します。

### x86エミュレーションモードでのデバッグ用プログラム

x86エミュレーションモードでデバッグするには、Windows CE Platform SDK (HPC Pro)が必要です。参照→「第1章 3.1 必要な開発用ソフトウェア」

(1) [WCE構成]ツールバーを以下のように設定し、ビルドを実行します。



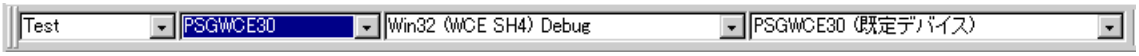

(2) ビルドが正常に完了すると、エミュレータへのダウンロードが開始されます。エミュレータが起動していない時は、エミュレータが自動的に起動します。

(3) プログラムは[\マイハンドヘルドPC]フォルダにダウンロードされます。プログラムのアイコンをダブルクリックして実行します。デバッグする場合はデバッガを起動して実行します。
参照→「第2章 3 エミュレーションモードでのデバッグ」

### PS-Gでのデバッグ用プログラム

(1) アプリケーション開発用パソコンとPS-GをActiveSync3.1で接続します。
参照→「第1章 5 リモート接続手順」

(2) [WCE構成]ツールバーを以下のように設定し、ビルドを実行します。




(3) ビルドが正常に完了すると、PS-Gへのダウンロードが開始されます。

(4) プログラムはPS-Gの[\マイ コンピュータ]フォルダにダウンロードされます。プログラムのアイコンをダブルクリックして実行します。デバッグする場合はパソコンからデバッガを起動して実行します。

### PS-Gでのリリース用プログラム

(1) アプリケーション開発用パソコンとPS-GをActiveSync3.1で接続します。
参照→「第1章 5 リモート接続手順」

(2) [WCE構成]ツールバーを以下のように設定し、ビルドを実行します。


(3) ビルドが正常に完了すると、PS-Gへのダウンロードが開始されます。

(4) プログラムはPS-Gの[\マイ コンピュータ]フォルダにダウンロードされます。プログラムのアイコンをダブルクリックして実行します。

## 1.3 実行

プログラムは指定したターゲットにダウンロードされます。ダウンロードされたプログラムのアイコンをダブルクリックして実行します。デバッグする場合は、パソコンからデバッガを起動して実行します。

また、CFカードにコピーしたプログラムをPS-Gで実行することもできます。その場合は、PS-GにCFカードを装着して、コントロールパネルにある 

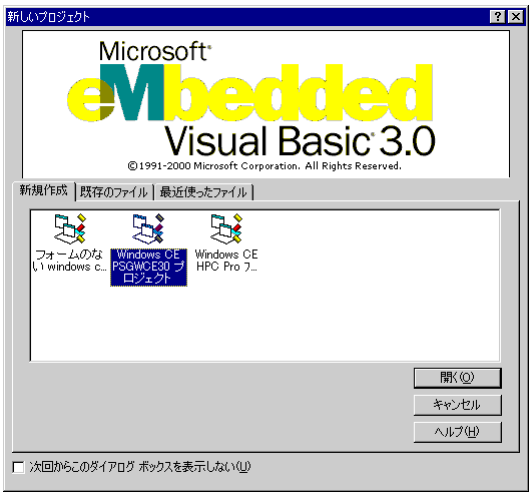
を起動し、アプリケーションが動作する環境もCFカードに保存しておきます。[\マイコンピュータ\Storage Card1]フォルダ内にあるプログラムのアイコンをダブルクリックして実行します。

# 2　eMbedded Visual Basicでの開発

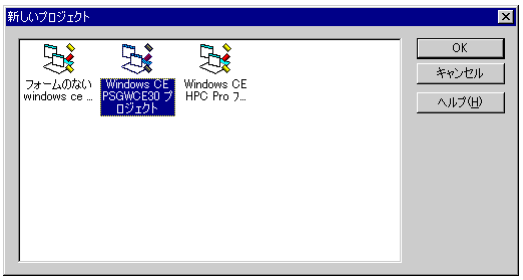
eMbedded Visual Basicを使用して、画面にウィンドウを表示するための簡単なアプリケーションの作成手順を説明します。

## 2.1　プロジェクトの作成

(1) eMbedded Visual Basicを起動すると、以下のようなダイアログボックスが表示されます。



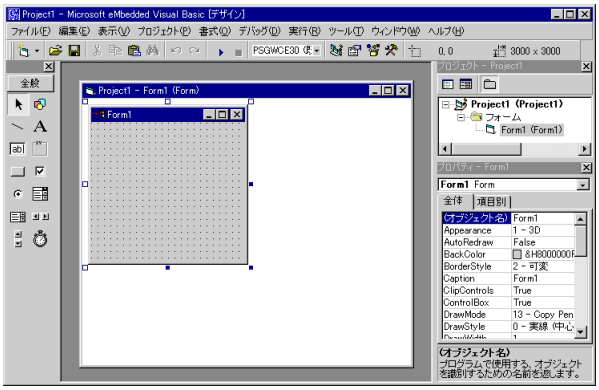

または、[ファイル]から[新しいプロジェクト]を選択し、[新しいプロジェクト]ダイアログボックスを表示します。



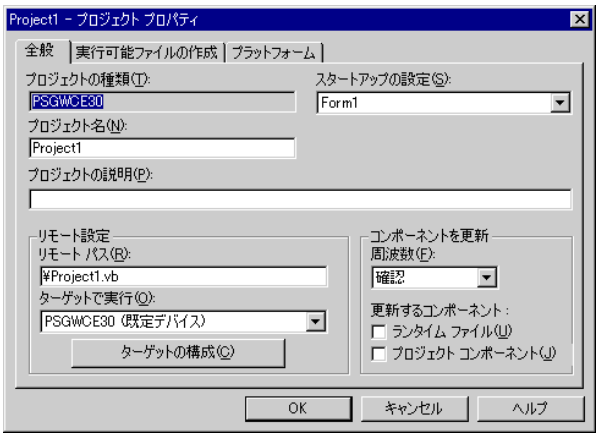

(2) PS-Gで動作するプログラムを作成する場合は、[Windows CE PSGWCE30 プロジェクト]を、エミュレーションモードで動作するプログラムを作成する場合は、[Windows CE HPC Pro プロジェクト]を選択します。

**MEMO**　**エミュレーションモードでデバッグしたプログラムをPS-Gで動作するプログラムとして使用するには、Windows CE HPC Pro プロジェクトで作成したフォームやモジュールをWindows CE PSGWCE30 プロジェクトへ追加してください。**

(3) プロジェクトを選択すると、以下のような画面が表示されます。

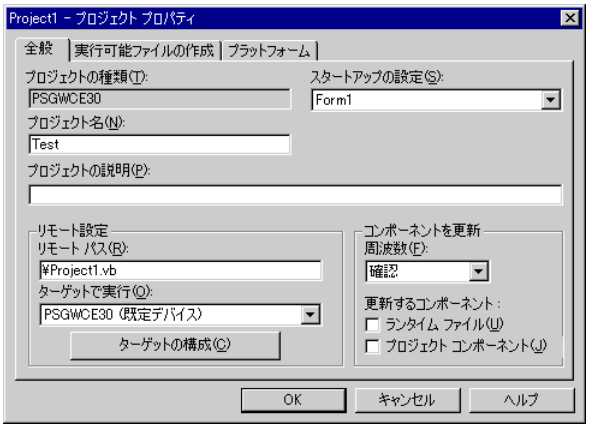



(4) [プロジェクト]から[Project1のプロパティ]を選択すると[プロジェクト プロパティ]ダイアログボックスが表示されます。



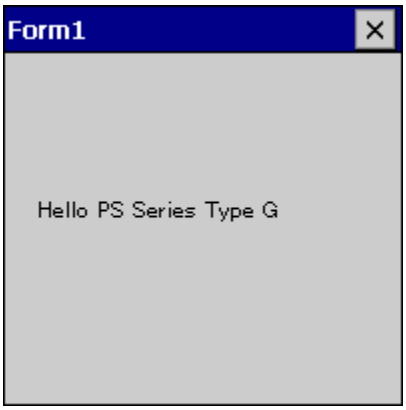

**(図はWindows CE PSGWCE30 プロジェクトの場合)**

(5) 必要に応じてプロジェクト名などを変更します。

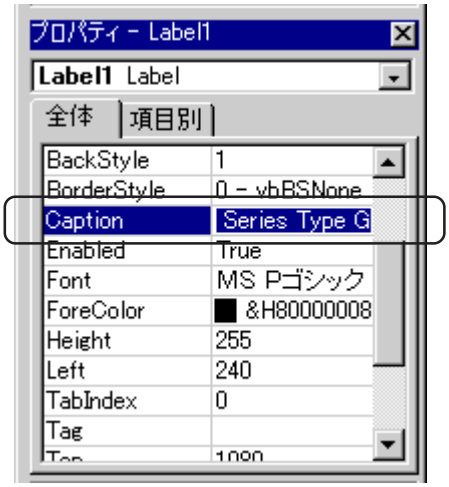


## 2.2　ダウンロードと実行

例としてウィンドウの中心に“Hello PS Series Type G”と表示させるプログラムを作成します。

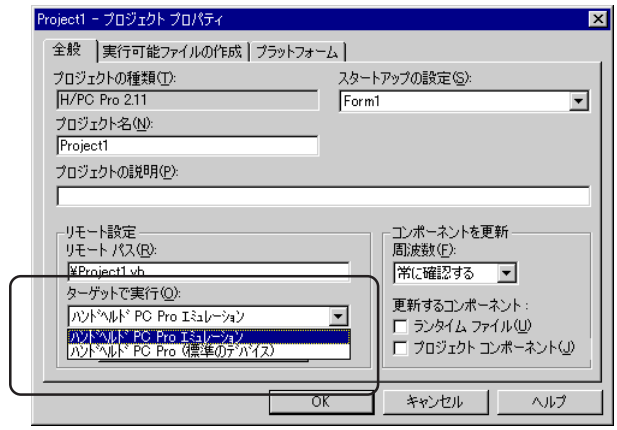



(1) ツールボックスの中から、Labelを使用して、フォームにLabelを貼り付けます。プロパティの「Caption」に“Hello PS Series Type G”と入力します。

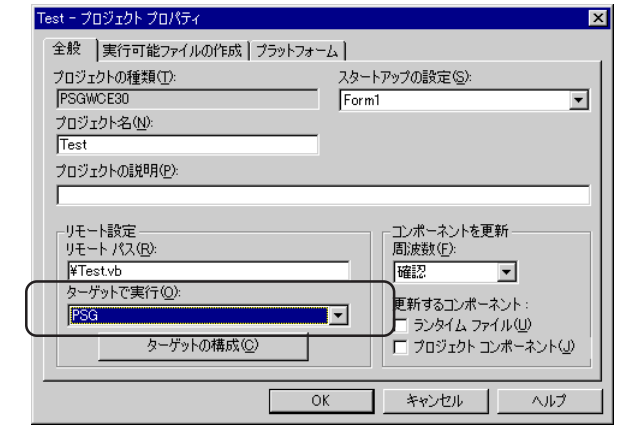


(2) [プロジェクト]から[Project1のプロパティ]を選択し、[プロジェクト プロパティ]でターゲットを指定します。「ターゲットで実行」のプルダウンメニューからターゲットを選択します。

エミュレーションモードでデバッグする場合（Windows CE HPC Pro プロジェクト）：
「ハンドヘルド PC Pro エミュレーション」を選択します。




PS-Gで動作するプログラムの場合（Windows CE PSGWCE30 プロジェクト）：
プラットフォームマネージャで登録したPS-Gの名前を選択します。参照→「第1章 4.1 パソコン側の設定」




(3) ターゲットがPS-Gの場合はダウンロードケーブルで開発用パソコンとPS-Gを接続します。

(4) [実行]から[デバッグの開始]、または[実行]を選択すると、指定したターゲットにプログラムがダウンロードされ、実行されます。

エミュレーションモードでデバッグする場合（Windows CE HPC Pro プロジェクト）、プログラムは[\マイハンドヘルドPC]フォルダにダウンロードされます。

PS-Gで動作するプログラムの場合（Windows CE PSGWCE30 プロジェクト）、プログラムはPS-Gの[\マイ コンピュータ]フォルダにダウンロードされます。

(5) ダウンロード後はプログラムのアイコンをダブルクリックして実行できます。デバッグする場合はパソコンからデバッガを起動して実行します。参照→「第2章 3 エミュレーションモードでのデバッグ」

また、CFカードにコピーしたプログラムをPS-Gで実行することもできます。その場合は、PS-GにCFカードを装着して、コントロールパネルにある 

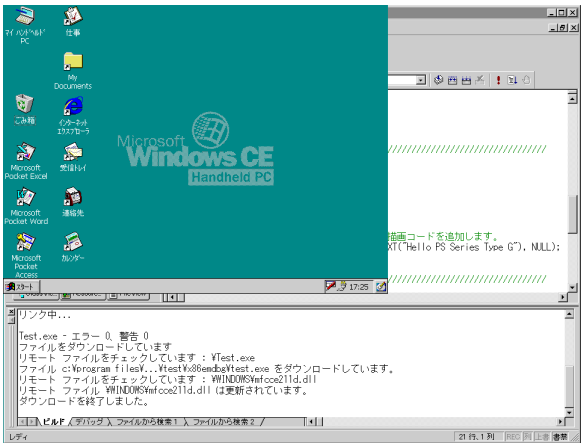
を起動し、アプリケーションが動作する環境もCFカードに保存しておきます。[\マイコンピュータ\Storage Card1]フォルダ内にあるプログラムのアイコンをダブルクリックして実行します。

# 3　エミュレーションモードでのデバッグ

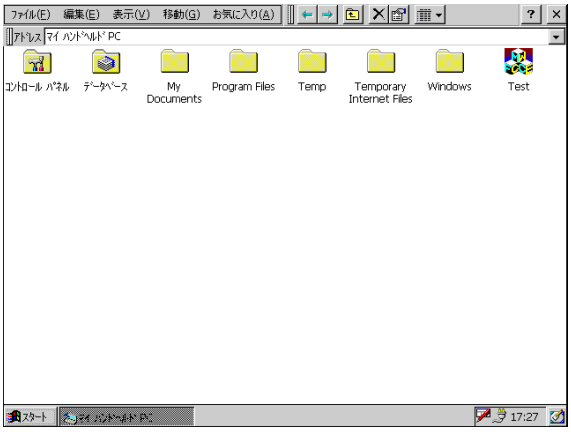
RASやタッチパネルなどのPS-G固有のハードウェアを使用しないアプリケーションのデバッグをする場合は、Windows CE Platform SDK (HPC Pro)を使用してエミュレーションモードで効率よくデバッグすることができます。

Windows CE Platform SDK (HPC Pro)はWindows NTとWindows 2000でのみ動作します。

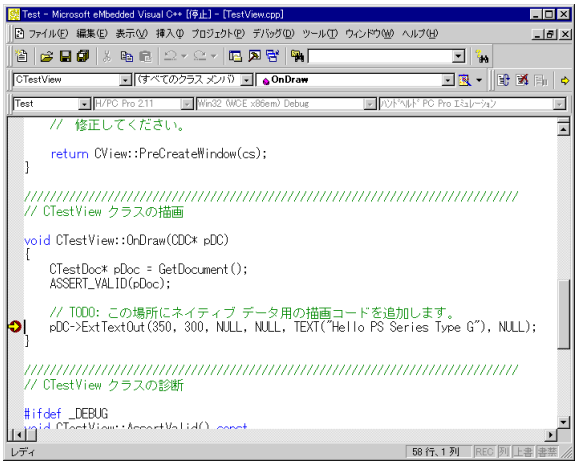
・ エミュレータが起動すると、開発用パソコンの画面に以下のようなエミュレート画面が表示されます。



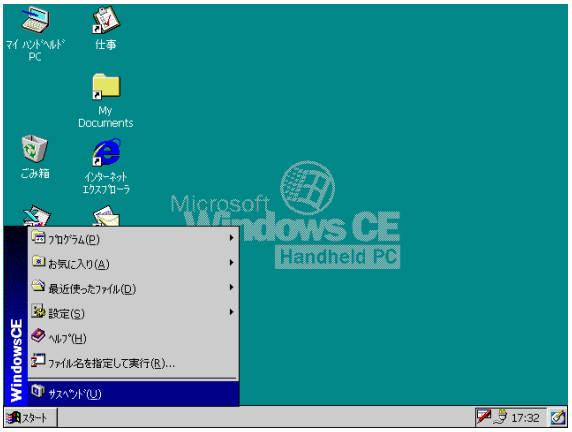

・ エミュレートされた環境にダウンロードされたプログラムは[\マイハンドヘルドPC]フォルダに保存されます。



・ 保存された実行ファイルを直接クリックして実行することもできます。
eMbedded Visual Toolsのデバッグ機能を使用してブレークポイントを設定し、ブレークすることもできます。




・ エミュレータを終了させる時は、エミュレート画面の[スタート]から[サスペンド]をクリックします。



# 4 オートスタート

開発したアプリケーションをPS-Gの起動時にオートスタート（自動起動）させることができます。以下に手順を示します。

## オブジェクトストアにあるアプリケーションを自動起動する場合

(1) PS-Gの[\Windows\スタートアップ]フォルダに、開発したアプリケーションへのショートカットを作成するか、またはアプリケーションそのものをコピーします。

(2) PS-GにCFカードを装着して、コントロールパネルにある 

を起動し、CFカードにアプリケーションが動作する環境を保存します。

(3) PS-GにCFカードを装着した状態で再起動します。

(4) 一度、Windows CEが起動した後、CFカードから環境を読み込み、スタートアップに登録されたアプリケーションが起動されます。

## CFカードにあるアプリケーションを自動起動する場合

(1) PS-Gの［\Windows\スタートアップ]フォルダに[\Windows]フォルダ内にある[AutoRun.exe]へのショートカットを作成します。

(2) (1)で作成したショートカットのプロパティを開きます。[ショートカット]で「リンク先」を以下のように変更し、自動起動するアプリケーションを指定します。

**リンク先(T) ｉ\Windows\AutoRun.exeｉ_ｉStorage_Card1\アプリケーション名ｉ**


**MEMO** **「リンク先」にｉ\Windows\AutoRun.exeｉ_/F_ｉStorage_Card1\アプリケーション名ｉのように /F を追加すると、画面下部のタスクバーを消した状態でアプリケーションを起動することができます。ユーザーアプリケーションからWindows CEのエクスプローラシェルを使用できないようにする場合に使用してください。**

(3) OKボタンを押します。

(4) PS-GにCFカードを装着して、コントロールパネルにある 

を起動し、CFカードにアプリケーションが動作する環境を保存します。

(5) PS-GにCFカードを装着した状態で再起動します。

(6) 一度、Windows CEが起動した後、CFカードから環境を読み込み、スタートアップに登録されたアプリケーションが起動されます。

# 3 ライブラリインターフェイスリファレンス

1 バックライトドライバ
2 SRAMドライバ
3 GMU-BUSドライバ
4 RASドライバ
5 タッチパネルドライバ

ADKには以下のドライバのインターフェイスが含まれています。
ユーザーアプリケーションからPS-Gのハードウェア特有の機能を使用するには、ユーザーアプリケーションのソースファイルに各インターフェイスライブラリ用のヘッダーファイルをインクルードし、オブジェクトファイルに各インターフェイスライブラリをリンクする必要があります。

- バックライトドライバ
- SRAMドライバ
- GMU-BUSドライバ
- RASドライバ
- タッチパネルドライバ

各ドライバのインターフェイスライブラリファイルとヘッダーファイルは以下の通りです。

| ドライバ | ライブラリファイル | eMbedded Visual C++ ヘッダーファイル | eMbedded Visual Basic ヘッダーファイル |
|---|---|---|---|
| バックライトドライバ | BlDrvIf.lib | BlDrvApi.h | BlDrvDef.bas |
| SRAMドライバ | SramDrvIf.lib | SramDrvApi.h | SramDrvDef.bas |
| GMU-BUSドライバ | GmuDrvIf.lib | GmuDrvApi.h | GmuDrvDef.bas |
| RASドライバ | RasDrvIf.lib | RasDrvApi.h | RasDrvDef.bas |
| タッチパネルドライバ | TouchDrvIf.lib | TchDrvApi.h | TchDrvDef.bas |

**MEMO** **各関数仕様詳細は、eMbedded Visual C++のインターフェイス仕様です。eMbedded Visual Basicで開発する場合、APIは共通ですが、引数の型が異なります。詳細は、eMbedded Visual Basic用の各ヘッダーファイルを参照してください。**

ADKに含まれる各APIを使用するには、使用前にドライバのOpen、使用後にCloseを行う必要があります。以下に示す例を参考にプログラムを開発してください。

例）

```
// ドライバの使用例
// SRAM以外のドライバも同様にドライバのOpen/Close処理が必要です。
// 変数宣言
BOOL Ret;
DWORD SramOffset;
BYTE SramData;
```

```
// アプリケーション初期化処理
// 各ドライバの使用前にはOpenする必用が有ります。
// 各ドライバのxxxDriverOpen()を呼び出します。
Ret = SramDriverOpen();                    // SRAMドライバOpen
      :
// SRAMへのアクセス
Ret = SramByteRead(SramOffset, &SramData); // SRAMからデータを読み込む
      :
Ret = SramByteWrite(SramOffset, SramData); // SRAMへデータを書き込む
      :
// アプリケーション終了処理
// 各ドライバの使用後にCloseが必用です。
// 各ドライバのxxxDriverClose()を呼び出します。
Ret = SramDriverClose();
```

# 1　バックライトドライバ

バックライトの状態設定と、コントロールを行います。コントロールパネルと同等の機能を提供しています。

## 1.1　バックライトドライバAPI一覧

| API名 | 内容 |
|---|---|
| GetBLDriverVersion | ドライババージョン取得 |
| BLDriverOpen | バックライトドライバのオープン |
| BLDriverClose | バックライトドライバのクローズ |
| GetBLStatus | バックライト状態の取得 |
| SetBLOnOff | バックライトのON/OFF |
| SetBLBright | バックライト輝度の設定 |
| SetBLAction | バックライト管切れ時の動作設定 |
| GetBLAction | バックライト管切れ時の動作設定取得 |
| SetBLTmOut | 一定時間無操作時のバックライトオフのタイマ値指定 |
| GetBLTmOut | 一定時間無操作時のバックライトオフの指定タイマ値取得 |

## 1.2 関数仕様詳細

### GetBLDriverVersion

**呼び出し形式**

BOOL WINAPI GetBLDriverVersion(WORD *pMajor, WORD *pMinor)

**戻り値**

TRUE:正常

FALSE:エラー

**引数**

WORD *pMajor　バージョン情報(Major, 0～99)へのポインタ

WORD *pMinor　バージョン情報(Minor, 0～99)へのポインタ

**処理概要**

ドライババージョン情報を取得します。

例）BOOL ret = GetBLDriverVersion(WORD &Major, WORD &Minor);

**補足**

バージョンが1.10の場合は、

Major:1　(10進数)

Minor:10　(10進数)

となります。

### BLDriverOpen

**呼び出し形式**

BOOL WINAPI BLDriverOpen(void)

**戻り値**

TRUE:正常

FALSE:エラー

**引数**

なし

**処理概要**

バックライトドライバをオープンし、設定動作可能とします。

例）BOOL ret = BLDriverOpen( );

## BLDriverClose

**呼び出し形式**

BOOL WINAPI BLDriverClose(void)

**戻り値**

TRUE:正常

FALSE:エラー

**引数**

なし

**処理概要**

バックライトドライバをクローズし、設定動作を無効とします。ただし、ドライバのThread動作は継続します。

例）BOOL ret = BLDriverClose( );

## GetBLStatus

**呼び出し形式**

BOOL WINAPI GetBLStatus(BYTE *pStatus)

**戻り値**

TRUE:正常

FALSE:エラー

**引数**

BYTE *pStatus　バックライトステータスの情報を取得するアドレスへのポインタ

バックライトの状態

D7 : エラー

D4 : バックライト管切れ　　1 : 切れている　　0 : 切れていない

D3 : バックライト状態　　1 : ON　　0 : OFF

D2～D0 : バックライト輝度

**処理概要**

バックライトのON/OFF状態とバックライトの輝度を取得します。

例）// バックライトの状態を取得する

```
BYTE Status;
BOOL ret = GetBLStatus(&Status);
```

## SetBLOnOff

**呼び出し形式**

BOOL WINAPI SetBLOnOff(BOOL bSwitch)

**戻り値**

TRUE:正常

FALSE:エラー

**引数**

BOOL bSwitch　TRUE：バックライトON

FALSE：バックライトOFF

**処理概要**

バックライトのコントロールを行います。

例）// バックライトをOFFする

```
BOOL ret = SetBLOnOff(FALSE);
```

## SetBLBright

**呼び出し形式**

BOOL WINAPI SetBLBright(BYTE Bright)

**戻り値**

TRUE:正常

FALSE:エラー

**引数**

BOOL Bright　0〜3：0 (暗) 〜 3 (明)

**処理概要**

バックライトの輝度コントロールを行います。

例）// 最高輝度に設定する

```
BOOL ret = SetBLBright(3);
```

## SetBLAction

**呼び出し形式**

BOOL WINAPI SetBLAction(BYTE Action)

**戻り値**

TRUE:正常

FALSE:エラー

**引数**

BYTE Action

D2 : タッチパネル入力動作　1 : 有効　0 : 無効

D1 : キーボード入力動作　1 : 有効　0 : 無効

D0 : マウス入力動作　1 : 有効　0 : 無効

**処理概要**

バックライト管切れ検出時のタッチパネル、キーボード、マウスの入力動作を有効、または無効(入力禁止)に設定します。

設定情報はレジストリに保存されます。

例）// バックライト管切れ検出時はタッチパネル、キーボード、マウスからの入力を無効にする

```
BOOL ret = SetBLAction(0);
```

## GetBLAction

**呼び出し形式**

BOOL WINAPI GetBLAction(BYTE *pAction)

**戻り値**

TRUE:正常

FALSE:エラー

**引数**

BYTE *pAction　バックライト管切れ検出時の動作の設定を取得するアドレスへのポインタ

BYTE Action

D2 : タッチパネル入力動作　1 : 有効　0 : 無効

D1 : キーボード入力動作　1 : 有効　0 : 無効

D0 : マウス入力動作　1 : 有効　0 : 無効

**処理概要**

バックライト管切れ検出時のタッチパネル、キーボード、マウスの入力動作の設定を取得します。

例）// バックライト管切れ検出時の設定を取得

```
BYTE Action;
BOOL ret = GetBLAction(&Action);
```

## SetBLTmOut

**呼び出し形式**

BOOL WINAPI SetBLTmOut(DWORD TmOut)

**戻り値**

TRUE:正常

FALSE:エラー

**引数**

**DWORD TmOut**　無操作時のバックライトOFFの指定タイマ値

0，15，30，60，120，300，600，900，1800のいずれかの値

**処理概要**

無操作時のバックライトOFFのタイマ値を設定します。

例）　DWORD TmOut = 120;

　　BOOL ret = SetBLTmOut(TmOut);

## GetBLTmOut

**呼び出し形式**

BOOL WINAPI GetBLTmOut(DWORD *pTmOut)

**戻り値**

TRUE:正常

FALSE:エラー

**引数**

DWORD *pTmOut　無操作時、バックライトOFF時の指定タイマ値のアドレスへのポインタ

**処理概要**

無操作時のバックライトOFFのタイマ値を取得します。

例）DWORD TmOut;

　BOOL ret = GetBLTmOut(&TmOut);

# 2　SRAMドライバ

SRAMへのR/Wファンクションです。

256Kバイトの領域が使用できます。各関数のリターン値は256Kバイトの領域を超えたアクセスでエラーとなります。

## 2.1　SRAMドライバAPI一覧

| API名 | 内容 |
|---|---|
| GetSramDriverVersion | ドライババージョン取得 |
| SramDriverOpen | SRAMドライバのオープン |
| SramDriverClose | SRAMドライバのクローズ |
| SramByteRead | SRAMデータの読み取り(Byte) |
| SramWordRead | SRAMデータの読み取り(Word) |
| SramLongRead | SRAMデータの読み取り(Long) |
| SramByteWrite | SRAMデータの書き込み(Byte) |
| SramWordWrite | SRAMデータの書き込み(Word) |
| SramLongWrite | SRAMデータの書き込み(Long) |

## 2.2　関数仕様詳細

### GetSramDriverVersion

**呼び出し形式**

BOOL WINAPI GetSramDriverVersion(WORD *pMajor, WORD *pMinor)

**戻り値**

TRUE:正常

FALSE:エラー

**引数**

WORD *pMajor　バージョン情報(Major, 0～99)へのポインタ

WORD *pMinor　バージョン情報(Minor, 0～99)へのポインタ

**処理概要**

ドライババージョン情報を取得します。

例) BOOL ret = GetSramDriverVersion(WORD &Major, WORD &Minor);

**補足**

バージョンが1.10の場合は、

Major:1　(10進数)

Minor:10　(10進数)

となります。

## SramDriverOpen

**呼び出し形式**

BOOL WINAPI SramDriverOpen(void)

**戻り値**

TRUE:正常

FALSE:エラー

**引数**

なし

**処理概要**

SRAMドライバをオープンし、動作可能とします。

例）BOOL ret = SramDriverOpen( );

## SramDriverClose

**呼び出し形式**

BOOL WINAPI SramDriverClose(void)

**戻り値**

TRUE:正常

FALSE:エラー

**引数**

なし

**処理概要**

SRAMドライバをクローズし、動作を無効とします。ただし、ドライバのThread動作は継続します。

例）BOOL ret = SramDriverClose( );

## SramByteRead

**呼び出し形式**

BOOL WINAPI SramByteRead(DWORD Offset, BYTE *pData)

**戻り値**

TRUE:正常

FALSE:エラー

**引数**

DWORD Offset　　読み出したいSRAMの先頭からのオフセット(Byte単位)
(0～3ffffh)

BYTE *pData　　読み出したデータを格納するアドレスへのポインタ

**処理概要**

指定したアドレスのSRAMデータを読み出します。

例）// SRAMの先頭から16バイト目のデータを読み出す

```
BYTE Data;
BOOL ret = SramByteRead(0x10, &Data);
```

## SramWordRead

**呼び出し形式**

BOOL WINAPI SramWordRead(DWORD Offset, WORD *pData)

**戻り値**

TRUE:正常

FALSE:エラー

**引数**

DWORD Offset　　読み出したいSRAMの先頭からのオフセット(Word単位)
(0～1ffffh)

WORD *pData　　読み出したデータを格納するアドレスへのポインタ

**処理概要**

指定したアドレスのSRAMデータを読み出します。

例）// SRAMの先頭から16ワード目のデータを読み出す

```
WORD Data;
BOOL ret = SramWordRead(0x10, &Data);
```

## SramLongRead

**呼び出し形式**

BOOL WINAPI SramLongRead(DWORD Offset, DWORD *pData)

**戻り値**

TRUE:正常

FALSE:エラー

**引数**

DWORD Offset　読み出したいSRAMの先頭からのオフセット(Dword単位)
(0～ffffh)

DWORD *pData　読み出したデータを格納するアドレスへのポインタ

**処理概要**

指定したアドレスのSRAMデータを読み出します。

例）// SRAMの先頭から16DWORD目のデータを読み出す

```
DWORD Data;
BOOL ret = SramLongRead(0x10, &Data);
```

## SramByteWrite

**呼び出し形式**

BOOL WINAPI SramByteWrite(DWORD Offset, BYTE Data)

**戻り値**

TRUE:正常

FALSE:エラー

**引数**

DWORD Offset　書き込みたいSRAMの先頭からのオフセット(Byte単位)
(0～3ffffh)

BYTE Data 書き込みするデータ

**処理概要**

指定したアドレスのSRAMにデータを書き込みます。

例）// SRAMの先頭から16バイト目にデータ(aah)を書き込む

```
BOOL ret = SramByteWrite(0x10, 0xaa);
```

## SramWordWrite

**呼び出し形式**

BOOL WINAPI SramWordWrite(DWORD Offset, WORD Data)

**戻り値**

TRUE:正常

FALSE:エラー

**引数**

DWORD Offset　書き込みたいSRAMの先頭からのオフセット(Word単位)
(0～1ffffh)

WORD Data 書き込みするデータ

**処理概要**

指定したアドレスのSRAMにデータを書き込みます。

例）// SRAMの先頭から16ワード目にデータ(55aah)を書き込む

```
BOOL ret = SramWordWrite(0x10, 0x55aa);
```

## SramLongWrite

**呼び出し形式**

BOOL WINAPI SramLongWrite(DWORD Offset, DWORD Data)

**戻り値**

TRUE:正常

FALSE:エラー

**引数**

DWORD Offset　書き込みたいSRAMの先頭からのオフセット(Dword単位)
(0～ffffh)

DWORD Data　書き込みするデータ

**処理概要**

指定したアドレスのSRAMにデータを書き込みます。

例）// SRAMの先頭から16DWORD目にデータ(55aa55aah)を書き込む

```
BOOL ret = SramLongWrite(0x10, 0x55aa55aa);
```

# 3 GMU-BUSドライバ

GMU-BUSへのコントロールを提供します。割り込みはユーザーコールバックルーチンの登録を行うことでサポートします。

## 3.1 GMU-BUSドライバAPI一覧

| API名 | 内容 |
|---|---|
| GetGmuDriverVersion | ドライババージョン取得 |
| GmuDriverOpen | GMU-BUSドライバのオープン |
| GmuDriverClose | GMU-BUSドライバのクローズ |
| GmuByteRead | GMUデータの読み取り(Byte) |
| GmuWordRead | GMUデータの読み取り(Word) |
| GmuLongRead | GMUデータの読み取り(Long) |
| GmuByteWrite | GMUデータの書き込み(Byte) |
| GmuWordWrite | GMUデータの書き込み(Word) |
| GmuLongWrite | GMUデータの書き込み(Long) |
| SetGmuCallbackIntA | GMU IntA Callback の登録 |
| SetGmuCallbackIntB | GMU IntB Callback の登録 |

## 3.2 関数仕様詳細

### GetGmuDriverVersion

**呼び出し形式**

BOOL WINAPI GetGmuDriverVersion(WORD *pMajor, WORD *pMinor)

**戻り値**

TRUE:正常

FALSE:エラー

**引数**

WORD *pMajor　バージョン情報(Major, 0～99)へのポインタ

WORD *pMinor　バージョン情報(Minor, 0～99)へのポインタ

**処理概要**

ドライババージョン情報を取得します。

例）BOOL ret = GetGmuDriverVersion(WORD &Major, WORD &Minor);

**補足**

バージョンが1.10の場合は、

Major:1　(10進数)

Minor:10　(10進数)

となります。

## GmuDriverOpen

**呼び出し形式**

BOOL WINAPI GmuDriverOpen(void)

**戻り値**

TRUE:正常

FALSE:エラー

**引数**

なし

**処理概要**

GMU-BUSドライバをオープンし、動作可能とします。

例）BOOL ret = GmuDriverOpen( );

## GmuDriverClose

**呼び出し形式**

BOOL WINAPI GmuDriverClose(void)

**戻り値**

TRUE:正常

FALSE:エラー

**引数**

なし

**処理概要**

GMU-BUSドライバをクローズし、動作を無効とします。ただし、ドライバのThread動作は継続します。

例）BOOL ret = GmuDriverClose( );

## GmuByteRead

**呼び出し形式**

BOOL WINAPI GmuByteRead(DWORD Offset, BYTE *pData)

**戻り値**

TRUE:正常

FALSE:エラー

**引数**

DWORD Offset　　読み出したいGMUの先頭からのオフセット(Byte単位)

BYTE *pData　　読み出したデータを格納するアドレスへのポインタ

**処理概要**

指定したアドレスのGMUデータを読み出します。

例）// GMUの先頭から16バイト目のデータを読み出す

```
BYTE Data;
BOOL ret = GmuByteRead(0x10, &Data);
```

## GmuWordRead

**呼び出し形式**

BOOL WINAPI GmuWordRead(DWORD Offset, WORD *pData)

**戻り値**

TRUE:正常

FALSE:エラー

**引数**

DWORD Offset　　読み出したいGMUの先頭からのオフセット(Word単位)

WORD *pData　　読み出したデータを格納するアドレスへのポインタ

**処理概要**

指定したアドレスのGMUデータを読み出します。

例）// GMUの先頭から16ワード目のデータを読み出す

```
    WORD Data;
    BOOL ret = GmuWordRead(0x10, &Data);
```

## GmuLongRead

**呼び出し形式**

BOOL WINAPI GmuLongRead(DWORD Offset, DWORD *pData)

**戻り値**

TRUE:正常

FALSE:エラー

**引数**

DWORD Offset　　読み出したいGMUの先頭からのオフセット(Dword単位)

DWORD *pData　　読み出したデータを格納するアドレスへのポインタ

**処理概要**

指定したアドレスのGMUデータを読み出します。

例）// GMUの先頭から16DWORD目のデータを読み出す

```
DWORD Data;
BOOL ret = GmuLongRead(0x10, &Data);
```

## GmuByteWrite

**呼び出し形式**

BOOL WINAPI GmuByteWrite(DWORD Offset, BYTE Data)

**戻り値**

TRUE:正常

FALSE:エラー

**引数**

DWORD Offset　　書き込みたいGMUの先頭からのオフセット(Byte単位)

BYTE Data 書き込みするデータ

**処理概要**

指定したアドレスのGMUにデータを書き込みます。

例）// GMUの先頭から16バイト目にデータ(aah)を書き込む

```
BOOL ret = GmuByteWrite(0x10, 0xaa);
```

## GmuWordWrite

**呼び出し形式**

BOOL WINAPI GmuWordWrite(DWORD Offset, WORD Data)

**戻り値**

TRUE:正常

FALSE:エラー

**引数**

DWORD Offset　書き込みたいGMUの先頭からのオフセット(Word単位)

WORD Data書き込みするデータ

**処理概要**

指定したアドレスのGMUにデータを書き込みます。

例）// GMUの先頭から16ワード目にデータ(55aah)を書き込む

```
BOOL ret = GmuWordWrite(0x10, 0x55aa);
```

## GmuLongWrite

**呼び出し形式**

BOOL WINAPI GmuLongWrite(DWORD Offset, DWORD Data)

**戻り値**

TRUE:正常

FALSE:エラー

**引数**

DWORD Offset　書き込みたいGMUの先頭からのオフセット(Dword単位)

DWORD Data　書き込みするデータ

**処理概要**

指定したアドレスのGMUにデータを書き込みます。

例）// GMUの先頭から16DWORD目にデータ(55aa55aah)を書き込む

```
BOOL ret = GmuLongWrite(0x10, 0x55aa55aa);
```

## SetGmuCallbackIntA

**呼び出し形式**

BOOL WINAPI SetGmuCallbackIntA(GMUPROC lpCallbackFunc)

**戻り値**

TRUE:正常

FALSE:エラー

**引数**

| | |
|---|---|
| GMUPROC lpCallbackFunc | 割り込み処理関数へのロングポインタ<br>NULLの場合には登録を削除します。 |

**処理概要**

GMU IntA発生時に呼び出される割り込み処理関数を登録します。

## SetGmuCallbackIntB

**呼び出し形式**

BOOL WINAPI SetGmuCallbackIntB(GMUPROC lpCallbackFunc)

**戻り値**

TRUE:正常

FALSE:エラー

**引数**

| | |
|---|---|
| GMUPROC lpCallbackFunc | 割り込み処理関数へのロングポインタ<br>NULLの場合には登録を削除します。 |

**処理概要**

GMU IntB発生時に呼び出される割り込み処理関数を登録します。

# 4 RASドライバ

RASへのコントロールを提供します。ウォッチドッグ機能、リモートリセット入力、汎用入力2点、汎用出力1点、アラーム出力1点をサポートします。

それぞれコントロールパネルによっても、入出力イベント発生時の動作を指定可能としています。コントロールパネルと同等の機能を提供しています。

## 4.1 RASドライバAPI一覧

### ＜共通ファンクション＞

| | 設定/表示ダイアログコントロール | API名 | 機能 | H/Wレジスタ/レジストリ |
|---|---|---|---|---|
| 共通 | | GetRasDriverVersion | ドライババージョン取得 | |
| | | RasDriverOpen | RASドライバのオープン | |
| | | RasDriverClose | RASドライバのクローズ | |
| | | BuzzerControl | BUZZER ON/OFF制御 | |
| | | AlarmControl | ALARMOUT ON/OFF制御 | RAS IN/OUT |
| | | DoutControl | DOUT ON/OFF制御 | RAS IN/OUT |
| | | SetRasRegister | RASレジスタへのライト | |
| | | GetRasRegister | RASレジスタのリード | |

### ＜ウォッチドッグタイマファンクション＞

| | 設定/表示ダイアログコントロール | API名 | 機能 | H/Wレジスタ/レジストリ |
|---|---|---|---|---|
| WDT | Enable | SetWdtEnable | イネーブル設定 | レジストリW |
| | | GetWdtEnable | イネーブル設定取得 | レジストリR |
| | Counter | SetWdtCounter | カウンタ値設定 | レジストリW |
| | | GetWdtCounter | カウンタ設定値取得 | レジストリR |
| | Alarm | SetWdtMask | アラームマスク設定 | レジストリW |
| | | GetWdtMask | アラームマスク設定取得 | レジストリR |
| | Buzzer | SetWdtBuzzer | ブザー設定 | レジストリW |
| | | GetWdtBuzzer | ブザー設定取得 | レジストリR |
| | Reset | SetWdtReboot | リセット設定 | レジストリW |
| | | GetWdtReboot | リセット設定取得 | レジストリR |
| | Popup Enable | SetWdtPopup | ポップアップメッセージイネーブル設定 | レジストリW |
| | | GetWdtPopup | ポップアップメッセージイネーブル設定取得 | レジストリR |
| | Popup Message | SetWdtMessage | ポップアップメッセージ設定 | レジストリW |
| | | GetWdtMssage | ポップアップメッセージ設定取得 | レジストリR |

## ＜ウォッチドッグタイマファンクション＞

| | 設定/表示ダイアログコントロール | API名 | 機能 | H/Wレジスタ/レジストリ |
|---|---|---|---|---|
| WDT | User application | SetWdtControl | WDTタイマスタートストップ設定 | WDT_CNTL |
| | | GetWdtControl | WDTタイマスタートストップ設定取得 | WDT_CNTL |
| | | GetWdtTimeout | WDTタイマタイムアウト取得 | WDT_STT |
| | | ClearWdtTimeout | WDTタイマタイムアウトクリア | WDT_STT |
| | | RefleshWdtTime | WDTタイマリフレッシュ | WDT_COUNT |

## ＜DINファンクション＞

| | 設定/表示ダイアログコントロール | API名 | 機能 | H/Wレジスタ/レジストリ |
|---|---|---|---|---|
| DIN | Enable | SetDinEnable | イネーブル設定 | レジストリW |
| | | GetDinEnable | イネーブル設定取得 | レジストリR |
| | Alarm | SetDinAlarmOut | アラーム出力設定 | レジストリW |
| | | GetDinAlarmOut | アラーム出力設定取得 | レジストリR |
| | Buzzer | SetDinBuzzer | ブザー設定 | レジストリW |
| | | GetDinBuzzer | ブザー設定取得 | レジストリR |
| | RAS Output | SetDinDout | RAS OUT設定 | レジストリW |
| | | GetDinDout | RAS OUT設定取得 | レジストリR |
| | Popup Enable | SetDinPopup | ポップアップメッセージイネーブル設定 | レジストリW |
| | | GetDinPopup | ポップアップメッセージイネーブル設定取得 | レジストリR |
| | Popup Message | SetDinMessage | ポップアップメッセージ設定 | レジストリW |
| | | GetDinMessage | ポップアップメッセージ設定取得 | レジストリR |

## ＜リモートリセットファンクション＞

| | 設定/表示ダイアログコントロール | API名 | 機能 | H/Wレジスタ/レジストリ |
|---|---|---|---|---|
| Remote Reset | Enable | SetRstEnable | イネーブル設定 | レジストリW |
| | | GetRstEnable | イネーブル設定取得 | レジストリR |

## 4.2 関数仕様詳細

### ＜共通ファンクション＞

#### GetRasDriverVersion

**呼び出し形式**

BOOL WINAPI GetRasDriverVersion(WORD *pMajor, WORD *pMinor)

**戻り値**

TRUE:正常

FALSE:エラー

**引数**

WORD *pMajor　バージョン情報(Major, 0～99)へのポインタ

WORD *pMinor　バージョン情報(Minor, 0～99)へのポインタ

**処理概要**

ドライババージョン情報を取得します。

例）BOOL ret = GetRasDriverVersion(WORD &Major, WORD &Minor);

**補足**

バージョンが1.10の場合は、

Major:1　(10進数)

Minor:10　(10進数)

となります。

#### RasDriverOpen

**呼び出し形式**

BOOL WINAPI RasDriverOpen(void)

**戻り値**

TRUE:正常

FALSE:エラー

**引数**

なし

**処理概要**

RASドライバをオープンし、設定動作可能とします。

例）BOOL ret = RasDriverOpen( );

## RasDriverClose

**呼び出し形式**

BOOL WINAPI RasDriverClose(void)

**戻り値**

TRUE:正常

FALSE:エラー

**引数**

なし

**処理概要**

RASドライバをクローズし、設定動作を無効とします。ただし、ドライバのThread動作は継続します。

例）BOOL ret = RasDriverClose( );

## BuzzerControl

**呼び出し形式**

BOOL WINAPI BuzzerControl(short length)

**戻り値**

TRUE:正常

FALSE:エラー

**引数**

short length　ON(-1) ブザーON
　　OFF(0) ブザーOFF
　　1～32767 ブザーの長さ（単位ms）

**処理概要**

ブザーのON／OFFコントロールを行います。

例）//ブザー100ms鳴らす
　　BOOL ret = BuzzerControl(100);

## AlarmControl

**呼び出し形式**

BOOL WINAPI AlarmControl(BOOL bSwitch)

**戻り値**

TRUE:正常

FALSE:エラー

**引数**

BOOL bSwitch　TRUE：ALARMOUT出力

FALSE：ALARMOUT出力を停止

**処理概要**

ALARMOUTポートのコントロールを行います。

例）//ALARMOUTをOFFする

```
BOOL ret = AlarmControl(FALSE);
```

## DoutControl

**呼び出し形式**

BOOL WINAPI DoutControl(BOOL bSwitch)

**戻り値**

TRUE:正常

FALSE:エラー

**引数**

BOOL bSwitch　TRUE：DOUT出力

FALSE：DOUT出力を停止

**処理概要**

DOUTポートのコントロールを行います。

例）//DOUTをONする

```
BOOL ret = DoutControl(TRUE);
```

## SetRasRegister

**呼び出し形式**

BOOL WINAPI SetRasRegister(WORD RegNo, BYTE Data)

**戻り値**

TRUE:正常

FALSE:エラー

**引数**

WORD RegNo　　レジスタの指定 ※1

BYTE Data レジスタへセットするデータ

**レジスタの指定**

```
#define  RAS_IN_MASK  0
#define  RAS_IN_OUT   1
#define  WDT_CR       2
#define  WDT_COUNT    3
```

**処理概要**

アプリケーションより指定されたデータを指定されたポートにセットします。

例）//RAS_IN_MASKレジスタに0をセットする

```
BOOL ret = SetRasRegister(RAS_IN_MASK,0);
```

## GetRasRegister

**呼び出し形式**

BOOL WINAPI GetRasRegister(WORD RegNo, BYTE *pData)

**戻り値**

TRUE:正常

FALSE:エラー

**引数**

WORD RegNo　　リードしたいレジスタ番号

BYTE *pData　　リードするデータへのポインタ

**処理概要**

アプリケーションより指定されたポートをリードします。

例）//RAS_IN_MASKレジスタをリードする

```
BYTE Data;
BYTE ret = GetRasRegister(RAS_IN_MASK,&Data);
```

※1 レジスタに関する詳細は、第3章 4.3 レジスタ詳細 を参照してください。

## ＜ウォッチドッグタイマファンクション＞

### SetWdtEnable

**呼び出し形式**

BOOL WINAPI SetWdtEnable(BOOL bEnable)

**戻り値**

TRUE:正常

FALSE:エラー

**引数**

BOOL bEnable　TRUE : ウォッチドッグタイマ有効
　　FALSE: ウォッチドッグタイマ無効

**処理概要**

ウォッチドッグタイマの有効／無効を設定します。

例）BOOL ret = SetWdtEnable(TRUE);

### GetWdtEnable

**呼び出し形式**

BOOL WINAPI GetWdtEnable(BOOL *pEnable)

**戻り値**

TRUE:正常

FALSE:エラー

**引数**

BOOL *pEnable　ウォッチドッグタイマ設定状態へのポインタ

**処理概要**

ウォッチドッグタイマの設定状態を取得します。

例）BOOL Enable;
　　BOOL ret = GetWdtEnable(&Enable);

## SetWdtCounter

**呼び出し形式**

BOOL WINAPI SetWdtCounter(BYTE Counter)

**戻り値**

TRUE:正常

FALSE:エラー

**引数**

BYTE Counter　ウォッチドッグタイマ初期カウンタ値(5～255)　(単位:秒)

**処理概要**

ウォッチドッグタイマの初期カウンタ値を設定します。（WDT COUNT）

例）// ウォッチドッグタイマ初期カウンタ値を10秒に設定

```
BOOL ret = SetWdtCounter(10);
```

## GetWdtCounter

**呼び出し形式**

BOOL WINAPI GetWdtCounter(BYTE *pCounter)

**戻り値**

TRUE:正常

FALSE:エラー

**引数**

BYTE *pCounter　ウォッチドッグタイマの初期カウンタ値(単位:秒)へのポインタ

**処理概要**

現在のウォッチドッグタイマの初期カウンタ値を取得します。(WDT COUNT)

例）BYTE Counter;

```
BOOL ret = GetWdtCounter(&Counter);
```

## SetWdtMask

**呼び出し形式**

BOOL WINAPI SetWdtMask(BOOL bMask)

**戻り値**

TRUE:正常

FALSE:エラー

**引数**

BOOL bMask　　TRUE :ウォッチドッグタイマアラームマスク設定

　　FALSE:ウォッチドッグタイマアラームマスク解除

**処理概要**

ウォッチドッグタイマタイムアウト時に出力するアラーム警告とフロントLEDコントロールのマスクをします。

例）// ALARM出力のマスクを解除する

```
BOOL ret = SetWdtMask(FALSE);
```

## GetWdtMask

**呼び出し形式**

BOOL WINAPI GetWdtMask(BOOL *pMask)

**戻り値**

TRUE:正常

FALSE:エラー

**引数**

BOOL *pMask　　ウォッチドッグタイママスク設定状態へのポインタ

**処理概要**

ウォッチドッグタイマタイムアウト時の警告出力とフロントLEDコントロールのマスク情報を取得します。

例）// ALARMのマスク情報取得

```
BOOL Mask;
BOOL ret = GetWdtMask(&Mask);
```

## SetWdtBuzzer

**呼び出し形式**

BOOL WINAPI SetWdtBuzzer(BOOL bBuzzer)

**戻り値**

TRUE:正常

FALSE:エラー

**引数**

BOOL bBuzzer　　TRUE :ブザー有効設定

　　FALSE:ブザー無効設定

**処理概要**

ウォッチドッグタイマタイムアウト時のブザーの動作状態を設定します。

例）BOOL　ret = SetWdtBuzzer(TRUE);

## GetWdtBuzzer

**呼び出し形式**

BOOL WINAPI GetWdtBuzzer(BOOL *pBuzzer)

**戻り値**

TRUE:正常

FALSE:エラー

**引数**

BOOL *pBuzzer　　ブザー有効設定状態へのポインタ

**処理概要**

ウォッチドッグタイマタイムアウト時のブザーの動作状態を取得します。

例）BOOL Buzzer;

　　BOOL　ret = GetWdtBuzzer(&Buzzer);

### SetWdtReboot

**呼び出し形式**

BOOL WINAPI SetWdtReboot(BOOL bReboot)

**戻り値**

TRUE:正常

FALSE:エラー

**引数**

BOOL bReboot　TRUE :リブート有効設定
　　FALSE:リブート無効設定

**処理概要**

ウォッチドッグタイマタイムアウト時のリブートの動作状態を設定します。

例）BOOL　ret = SetWdtReboot(TRUE);

### GetWdtReboot

**呼び出し形式**

BOOL WINAPI GetWdtReboot(BOOL *pReboot)

**戻り値**

TRUE:正常

FALSE:エラー

**引数**

BOOL *pReboot　リブート設定状態へのポインタ

**処理概要**

ウォッチドッグタイマタイムアウト時のリブートの設定状態を取得します。

例）BOOL Reboot;
　　BOOL　ret = SetWdtReboot(&Reboot);

### SetWdtPopup

**呼び出し形式**

BOOL WINAPI SetWdtPopup(BOOL bPopup)

**戻り値**

TRUE:正常

FALSE:エラー

**引数**

BOOL bPopup　TRUE :ポップアップメッセージ有効設定
　　FALSE:ポップアップメッセージ無効設定

**処理概要**

ウォッチドッグタイマタイムアウト時ポップアップメッセージの有無を設定します。

例）BOOL　ret = SetWdtPopup(TRUE);

## GetWdtPopup

**呼び出し形式**

BOOL WINAPI GetWdtPopup(BOOL *pPopup)

**戻り値**

TRUE:正常

FALSE:エラー

**引数**

BOOL *pPopup　　ポップアップ設定へのポインタ

**処理概要**

ウォッチドッグタイマタイムアウト時のポップアップメッセージの設定状態を取得します。

例）BOOL Popup;

　　BOOL　ret = GetWdtPopup(&Popup);

## SetWdtMessage

**呼び出し形式**

BOOL WINAPI SetWdtMessage(wchar_t *pMessage)

**戻り値**

TRUE:正常

FALSE:エラー

**引数**

wchar_t *pMessage　　ポップアップメッセージへのポインタ

**処理概要**

ウォッチドッグタイマタイムアウト時のポップアップメッセージを設定します。

例）BOOL　ret = SetWdtMessage(L”Watch dog timer timeout”);

## GetWdtMessage

**呼び出し形式**

BOOL WINAPI GetWdtMessage(wchar_t *pMessage)

**戻り値**

TRUE:正常

FALSE:エラー

**引数**

wchar_t *pMessage　　ポップアップメッセージ格納領域へのポインタ

**処理概要**

ウォッチドッグタイマタイムアウト時のポップアップメッセージを取得します。

例）wchar_t msg[32];

　　BOOL　ret = GetWdtMessage(msg);

## SetWdtControl

**呼び出し形式**

BOOL WINAPI SetWdtControl(BOOL bCont)

**戻り値**

TRUE:正常

FALSE:エラー

**引数**

BOOL bCont　　TRUE :ウォッチドッグタイマをスタートさせます。

　　FALSE: ウォッチドッグタイマをストップさせます。

**処理概要**

ウォッチドッグタイマのスタート／ストップを行います。ウォッチドッグタイマはドライバ内でスタート／ストップ／リフレッシュのトリガはとらず、アプリケーションからのドライバファンクションコールのタイミングで行います。ドライバはタイムアウトを検出して、設定された内容に従ってアクションを起こします。

例）BOOL ret = SetWdtControl(TRUE);

## GetWdtControl

**呼び出し形式**

BOOL WINAPI GetWdtControl (BOOL *bCont)

**戻り値**

TRUE:正常

FALSE:エラー

**引数**

BOOL *bCont　　ウォッチドッグタイマのスタート／ストップの設定へのポインタ
　　TRUE:ウォッチドッグタイマスタート
　　FALSE:ウォッチドッグタイマストップ

**処理概要**

ウォッチドッグタイマのスタート／ストップの設定を取得します。

例）BOOL Cont;
　　BOOL ret = GetWdtControl(&Cont);

## GetWdtTimeout

**呼び出し形式**

BOOL WINAPI GetWdtTimeout(BOOL *pTOut)

**戻り値**

TRUE:正常

FALSE:エラー

**引数**

BOOL *pTOut　　タイマタイムアウト情報へのポインタ
　　TURE :タイムアウト
　　FALSE:タイムアウトしていない

**処理概要**

WDT_STTを読み、ウォッチドッグタイマがタイムアウトしているかどうかを返します。

例）BOOL TOut;
　　BOOL ret = GetWdtTimeOut(&TOut);

## ClearWdtTimeout

**呼び出し形式**

BOOL WINAPI ClearWdtTimeout(void);

**戻り値**

TRUE:正常

FALSE:エラー

**引数**

なし

**処理概要**

ウォッチドッグタイムアウトの状態をクリアします。(WDT_STT)

例）// ウォッチドッグのタイムアウト状態クリア

```
BOOL ret = ClearWdtTimeout( );
```

## RefleshWdtTime

**呼び出し形式**

BOOL WINAPI RefleshWdtTime(void);

**戻り値**

TRUE:正常

FALSE:エラー

**引数**

なし

**処理概要**

ウォッチドッグタイマカウンタ(WDT_STT)にSetWdtCountで設定されたカウンタ値をセットします。

例）// ウォッチドッグタイマのリフレッシュ

```
BOOL ret = RefleshWdtTime( );
```

## ＜DINファンクション＞

### SetDinEnable

**呼び出し形式**

BOOL WINAPI SetDinEnable(WORD inp, BOOL bEnable)

**戻り値**

TRUE:正常

FALSE:エラー

**引数**

| | | |
|---|---|---|
| WORD inp | 0または1 | DINポート番号 |
| BOOL bEnable | TRUE :DINポート入力有効 | |
| | FALSE:DINポート入力無効 | |

**処理概要**

DINポート0または1の入力の有効／無効を設定します。

例）BOOL ret = SetDinEnable(0, TRUE);

### GetDinEnable

**呼び出し形式**

BOOL WINAPI GetDinEnable(WORD inp, BOOL *pEnable)

**戻り値**

TRUE:正常

FALSE:エラー

**引数**

| | | |
|---|---|---|
| WORD inp | 0または1 | DINポート番号 |
| BOOL *pEnable | DINポートの設定状態を示すポインタ | |

**処理概要**

DINポート0または1の設定状態を取得します。

例）// DINポートの設定取得

```
BOOL Enable;
BOOL ret = GetDinEnable(0, &Enable);.
```

## SetDinAlarmOut

**呼び出し形式**

BOOL WINAPI SetDinAlarmOut(WORD inp, BOOL bAlm)

**戻り値**

TRUE:正常

FALSE:エラー

**引数**

| | | |
|---|---|---|
| WORD inp | 0または1 | DINポート番号 |
| BOOL bAlm | TRUE :アラーム出力設定有効 | |
| | FALSE:アラーム出力設定無効 | |

**処理概要**

アラーム出力の有効／無効を設定します。

例）BOOL ret = SetDinAlarmOut(0, TRUE);

## GetDinAlarmOut

**呼び出し形式**

BOOL WINAPI GetDinAlarmOut(WORD inp, BOOL *pAlm)

**戻り値**

TRUE:正常

FALSE:エラー

**引数**

| | | |
|---|---|---|
| WORD inp | 0または1 | DINポート番号 |
| BOOL *pAlm | DINポート0または1アクティブ時のアラーム出力が有効か無効かを示すポインタ | |

**処理概要**

DINポート0または1アクティブ時のアラーム出力の設定状態を取得します。

例）// DINポートの設定取得

```
BOOL Alm;
BOOL ret = GetDinAlarmOut(0, &Alm);
```

## SetDinBuzzer

**呼び出し形式**

```
BOOL WINAPI SetDinBuzzer(WORD inp, BOOL bBuzzer);
```

**戻り値**

TRUE:正常

FALSE:エラー

**引数**

WORD inp　　0または1　　DINポート番号

BOOL bBuzzer　　ブザー出力の有効／無効を設定します。

**処理概要**

DINポート0または1アクティブ時のブザー出力の有効／無効設定を行います。

例）// DINポートのBUZZERの出力状態をOFFに設定

```
BOOL ret = SetDinBuzzer(0, FALSE);
```

## GetDinBuzzer

**呼び出し形式**

```
BOOL WINAPI GetDinBuzzer(WORD inp, BOOL *pBuzzer);
```

**戻り値**

TRUE:正常

FALSE:エラー

**引数**

WORD inp　　0または1　　DINポート番号

BOOL *pBuzzer　　ブザー出力の有効／無効の設定を取得する領域へのポインタ

**処理概要**

DINポート0または1アクティブ時のブザー出力の設定状態を取得します。

例）// DINポートのBUZZERの出力状態をOFFに設定

```
BOOL Buzzer;
BOOL ret = GetDinBuzzer(0, &Buzzer);
```

## SetDinDout

**呼び出し形式**

```
BOOL WINAPI SetDinDout(WORD inp, BOOL bRasout);
```

**戻り値**

TRUE:正常

FALSE:エラー

**引数**

| | | |
|---|---|---|
| WORD inp | 0または1 | DINポート番号 |
| BOOL bRasout | | DOUT出力の有効／無効を設定します。 |

**処理概要**

DINポート0または1アクティブ時のDOUT出力を設定します。

例）// DOUTの出力状態をONに設定

```
BOOL ret = SetDinDout(0, TRUE);
```

## GetDinDout

**呼び出し形式**

```
BOOL WINAPI GetDinDout(WORD inp, BOOL *pRasOut);
```

**戻り値**

TRUE:正常

FALSE:エラー

**引数**

| | | |
|---|---|---|
| WORD inp | 0または1 | DINポート番号 |
| BOOL *pRasOut | | DOUT出力の有効／無効の設定を取得する領域へのポインタ |

**処理概要**

DINポート0または1アクティブ時のDOUT出力の設定を取得します。

例）// DOUTの出力状態を取得

```
BOOL RasOut;
BOOL ret = GetDinDout(0, &RasOut);
```

## SetDinPopup

**呼び出し形式**

```
BOOL WINAPI SetDinPopup(WORD inp, BOOL bPopup);
```

**戻り値**

TRUE:正常

FALSE:エラー

**引数**

WORD inp　　0または1　　DINポート番号

BOOL bPopup　　ポップアップメッセージ出力の有効／無効を設定します。

**処理概要**

DINポート0または1アクティブ時のポップアップメッセージ出力を設定します。

例）// ポップアップメッセージの出力状態をONに設定

```
BOOL ret = SetDinPopup(0, TRUE);
```

## GetDinPopup

**呼び出し形式**

```
BOOL WINAPI GetDinPopup(WORD inp, BOOL *pPopup);
```

**戻り値**

TRUE:正常

FALSE:エラー

**引数**

WORD inp　　0または1　　DINポート番号

BOOL *pPopup　　ポップアップメッセージ出力の有効／無効の設定を取得する領域へのポインタ

**処理概要**

DINポート0または1アクティブ時ポップアップメッセージ出力の設定を取得します。

例）// ポップアップメッセージの出力状態を取得

```
BOOL Popup;
BOOL ret = GetDinPopup(0, &Popup);
```

## SetDinMessage

**呼び出し形式**

BOOL WINAPI SetDinMessage(WORD inp, wchar_t *pMessage)

**戻り値**

TRUE:正常

FALSE:エラー

**引数**

WORD inp　　0または1　　DINポート番号

wchar_t *pMessage　　ポップアップメッセージへのポインタ

**処理概要**

DIN1ポート0または1アクティブ時のポップアップメッセージを設定します。

例）BOOL　ret = SetDinMessage(1, L”Universal Input Active”);

## GetDinMessage

**呼び出し形式**

BOOL WINAPI GetDinMessage(WORD inp, wchar_t *pMessage)

**戻り値**

TRUE:正常

FALSE:エラー

**引数**

WORD inp　　0または1　　Dinポート番号

wchar_t *pMessage　　ポップアップメッセージへのポインタ

**処理概要**

DIN1ポート0または1アクティブ時のポップアップメッセージを取得します。

例）wchar_t msg[32];

　　BOOL　ret = GetDinMessage(1, msg);

## ＜リモートリセットファンクション＞

### SetRstEnable

**呼び出し形式**

BOOL WINAPI SetRstEnable(BOOL bEnable)

**戻り値**

TRUE:正常

FALSE:エラー

**引数**

BOOL bEnable　　TRUE : リモートリセット入力有効

　　FALSE: リモートリセット入力無効

**処理概要**

リモートリセット入力の有効／無効を設定します。

例）BOOL ret = SetRstEnable(TRUE);

### GetRstEnable

**呼び出し形式**

BOOL WINAPI GetUniEnable(BOOL *pEnable)

**戻り値**

TRUE:正常

FALSE:エラー

**引数**

BOOL *pEnable　リモートリセット入力が有効か無効かを示すポインタ

**処理概要**

リモートリセット入力が有効か無効かの設定状態を取得します。

例）// リモートリセット入力の設定取得

```
BOOL Enable;
BOOL ret = GetRstEnable(&Enable);
```

## 4.3 レジスタ詳細

### RAS_IN_MASK

**RAS IN MASKレジスタ**

**Read／Write**

| レジスタ | B7 | B6 | B5 | B4 | B3 | B2 | B1 | B0 | 備考 |
|---|---|---|---|---|---|---|---|---|---|
| RAS IN MASK | -- | -- | RMT_RST_MASK | -- | -- | -- | IN1_MASK | IN0_MASK | 15～8ビット無効読み取り“0” |

**RMT_RST_MASK　リモートリセットマスク**

1：外部リモートリセット入力有効（デフォルト）

0：外部リモートリセット入力無効

**IN1_MASK　DIN1入力マスク**

1：DIN1入力有効

0：DIN1入力無効（デフォルト）

**IN0_MASK　DIN0入力マスク**

1：DIN0入力有効

0：DIN0入力無効（デフォルト）

### RAS_IN_OUT

**RAS　IN/OUTレジスタ**

**Read／Write**

| レジスタ | B7 | B6 | B5 | B4 | B3 | B2 | B1 | B0 | 備考 |
|---|---|---|---|---|---|---|---|---|---|
| RAS IN/OUT | LEDR | LEDG | -- | | RAS_ALM | RAS_OUT0 | RAS_IN1 | RAS_IN0 | 15～8ビット無効読み取り“0” |

**LEDR　フロントLED赤コントロール（Alarm Output出力とは非連動）**

RAS_ALMレジスタが“０”で、かつWDT_ALM_MSKレジスタが“１”の場合（この時、WDT_STTレジスタの状態には非依存）に、コントロールが有効となります。

RAS_ALMレジスタが“１”、またはWDT_ALM_MSKレジスタが“０”で、かつWDT_STTレジスタが“１”である場合（WDTエラー状態）は、その状態出力が優先され、LEDは赤点灯となります。

1：フロントLED赤点灯（デフォルト）

0：フロントLED赤消灯

**LEDG　　フロントLED緑コントロール**

1：フロントLED緑点灯

0：フロントLED緑消灯（デフォルト）

**RAS_ALM　Alarm Outputとフロントベゼル LEDをコントロール**

WDT_ALM_MASKレジスタが“０”で、かつWDT_STTレジスタが“１”の場合は、RAS_ALMレジスタの状態にかかわらず、Alarm Output ON オレンジ点灯（LED赤 ON）となります。

1：Alarm Output ON

　フロントLED、オレンジ点灯（LED 赤 ON）

0：Alarm Output OFF （デフォルト）

　フロントLED、グリーン点灯（LED 赤 OFF）

**RAS_OUT0　　DOUTコントロール**

1：DOUT0　ON

0：DOUT0　OFF（デフォルト）

**RAS_IN1　　DIN1コントロール**

DIN1がマスクされていない場合、DIN1の状態をこのビットに“１”が書き込まれるまで保持します。

RIN1の入力は、信号が0→1の遷移で取り込まれます。

1：DIN1　入力あり（Read）／ステータスクリア（Write）

0：DIN1　入力なし（Read）（デフォルト）

**RAS_IN0　　DIN0コントロール**

DIN0がマスクされていない場合、DIN0の状態をこのビットに“１”が書き込まれるまで保持します。

RIN1の入力は、信号が0→1の遷移で取り込まれます。

1：DIN0　入力あり（Read）／ステータスクリア（Write）

0：DIN0　入力なし（Read）（デフォルト）

## WDT_CR

**Watch Dog Timer　コントロールレジスタ**

**Read／Write**

| レジスタ | B7 | B6 | B5 | B4 | B3 | B2 | B1 | B0 | 備考 |
|---|---|---|---|---|---|---|---|---|---|
| WDT CR | WDT_ ALD | -- | -- | WDT_ STT | WDT_ ALM_ MASK | -- | -- | WDT_ CNTL | 15～8ビット無効読み取り“0” |

**WDT_ALD　　ウォッチドックタイマオートロード**

ウォッチドックタイマにカウントレジスタの内容を初期値としてロードします。

1：カウンター初期値ロード（Write）

0：読み取り時は常に“０”(Read）（デフォルト）

**WDT_STT　　ウォッチドックタイマタイムアウトステータス**

1：ウォッチドックタイマタイムアウト（Read）

　タイムアウト状態クリア（Write）

0：タイムアウトしていない（Read）（デフォルト）

**WDT_ALM_MASK　ウォッチドックタイマアラームマスク**

1：マスク有効（デフォルト）

0：タイムアウト時RASコネクタにAlarm出力

**WDT_CNTL　　ウォッチドックタイマカウンターコントロール**

タイムアウト時間内にスタート／ストップを繰り返すことによって、ウォッチドックタイマとして使用します。タイムアウト時間内にカウンターが停止しない場合には、Alarm/Lamp Outを出力し、WDT_STTに“１”をセットします。

1：カウントスタート

0：カウント停止／タイムアウトクリア／初期値ロード（デフォルト）

## WDT_COUNT

Watch Dog Timer　カウンタレジスタ

Read／Write

| レジスタ | B7 | B6 | B5 | B4 | B3 | B2 | B1 | B0 | 備考 |
|---|---|---|---|---|---|---|---|---|---|
| WDT COUNT | WDT_ COUNT 7 (MSB) | WDT_ COUNT 6 | WDT_ COUNT 5 | WDT_ COUNT 4 | WDT_ COUNT 3 | WDT_ COUNT 2 | WDT_ COUNT 1 | WDT_ COUNT 0 (LSB) | 15～8ビット 無効 読み取り“0” |

WDT_COUNT0～7　　タイマ初期値、1カウントは約1秒

（デフォルト：FF）

ウォッチドッグタイマにロードする初期値をセットします。

# 5　タッチパネルドライバ

タッチパネルのタッチ時のブザーと入力有効／無効のコントロールを行います。

## 5.1　タッチパネルドライバAPI一覧

| API名 | 機能 |
|---|---|
| GetTouchDriverVersion | ドライババージョン取得 |
| TouchDriverOpen | タッチパネルドライバのオープン |
| TouchDriverClose | タッチパネルドライバのクローズ |
| SetTouchClickBuzzer | タッチ時のブザー設定 |
| GetTouchClickBuzzer | タッチ時のブザー設定取得 |
| SetTouchInput | タッチ時の入力有効／無効設定 |
| GetTouchInput | タッチ時の入力有効／無効設定の取得 |

## 5.2　関数仕様詳細

### GetTouchDriverVersion

**呼び出し形式**

BOOL WINAPI GetBLDriverVersion(WORD *pMajor, WORD *pMinor)

**戻り値**

TRUE:正常

FALSE:エラー

**引数**

WORD *pMajor　バージョン情報(Major, 0～99)へのポインタ

WORD *pMinor　バージョン情報(Minor, 0～99)へのポインタ

**処理概要**

ドライババージョン情報を取得します。

例）BOOL ret = GetTouchDriverVersion(WORD &Major, WORD &Minor);

**補足**

バージョンが1.10の場合は、

Major:1　(10進数)

Minor:10　(10進数)

となります。

## TouchDriverOpen

**呼び出し形式**

BOOL WINAPI TouchDriverOpen(void)

**戻り値**

TRUE:正常

FALSE:エラー

**引数**

なし

**処理概要**

タッチパネルドライバをオープンし、設定動作可能とします。

例）BOOL ret = TouchDriverOpen( );

## TouchDriverClose

**呼び出し形式**

BOOL WINAPI TouchDriverClose(void)

**戻り値**

TRUE:正常

FALSE:エラー

**引数**

なし

**処理概要**

タッチパネルドライバをクローズし、設定動作を無効とします。

例）BOOL ret = TouchDriverClose( );

## SetTouchClickBuzzer(BOOL bState)

**呼び出し形式**

BOOL WINAPI SetTouchClickBuzzer(BOOL bState)

**戻り値**

TRUE:正常

FALSE:エラー

**引数**

BOOL bState　　TRUE:ブザー音ON

　　FALSE:ブザー音OFF

**処理概要**

クリック時のブザー音の設定を行います。

例）// クリック時のブザー音ONの設定

```
BOOL ret = SetTouchClickBuzzer(TRUE);
```

## GetTouchClickBuzzer(BOOL *bState)

**呼び出し形式**

BOOL WINAPI GetTouchClickBuzzer(BOOL *bState)

**戻り値**

TRUE:正常

FALSE:エラー

**引数**

BOOL *bState　　クリック時のブザー音の設定へのポインタ

**処理概要**

クリック時のブザー音の設定を取得します。

例）//クリック時のブザー音設定の取得

```
BOOL State;
BOOL ret = GetTouchClickBuzzer(&State);
```

## SetTouchInput(BOOL bState)

**呼び出し形式**

BOOL WINAPI SetTouchInput(BOOL bState)

**戻り値**

TRUE:正常

FALSE:エラー

**引数**

BOOL bState　　TRUE:入力有効

　　FALSE:入力無効

**処理概要**

タッチパネルのタッチ時の有効／無効を設定します。

例）//タッチ時の入力有効の設定

```
BOOL ret = SetTouchInput(TRUE);
```

## GetTouchInput(BOOL *bState)

**呼び出し形式**

BOOL WINAPI GetTouchInput(BOOL *bState)

**戻り値**

TRUE:正常

FALSE:エラー

**引数**

BOOL *bState　　タッチ時の入力設定へのポインタ

**処理概要**

タッチ時の入力の設定を取得します。

例）//タッチ時の入力設定の取得

```
BOOL State;
BOOL ret = GetTouchInput(&State);
```

# 索引


## C

CD-ROM の構成 ..... 5

## D

DIN ファンクション ..... 3-21, 3-35

## G

GMU-BUS ドライバ ..... 3-14
GMU-BUS ドライバ API 一覧 ..... 3-14

## R

RAS ドライバ ..... 3-20
RAS ドライバ API 一覧 ..... 3-20

## S

SRAM ドライバ ..... 3-9
SRAM ドライバ API 一覧 ..... 3-9

## V

Visual C++ ..... 2-2

## W

Win32 API ..... 1-2

## ア

アプリケーション開発ツールのインストール ..... 1-8

## ウ

ウォッチドッグタイマファンクション ..... 3-20, 3-21, 3-26

## エ

エミュレーション ..... 1-4, 1-8
エミュレーションモードでのデバッグ ..... 2-12

## オ

オートスタート ..... 2-14

## カ

関数仕様詳細 ..... 3-4, 3-9, 3-14, 3-22, 3-46

## キ

共通ファンクション ..... 3-20, 3-22

## シ

使用上の注意 ..... 6
商標権などについて ..... 3

## ソ

ソフトウェア環境 ..... 1-5

## タ

ダウンロードケーブル ..... 1-3
タッチパネルドライバ ..... 3-46
タッチパネルドライバ API 一覧 ..... 3-46

## ハ

ハードウェア環境 ..... 1-3
バックライトドライバ ..... 3-3
バックライトドライバ API 一覧 ..... 3-3

## ヒ

ビルドとダウンロード ..... 2-5

## フ

プラットフォームマネージャの構成 ..... 1-14

## ヘ

ヘッダーファイル ..... 3-2

## リ

リモート接続手順 ..... 1-13
リモートツール ..... 1-4, 1-6, 1-8
リモートリセットファンクション ..... 3-21, 3-41

## レ

レジスタ詳細 ..... 3-42

Pro-face®

株式会社デジタル

**本　　社**
TEL: (06) 6613-1101 (代)　FAX: (06) 6613-5888

**東京支店**
TEL: (03) 5821-1101　FAX: (03) 5821-1110

**中部支店**
TEL: (052) 932-6610　FAX: (052) 932-6802

**西日本支店**
TEL: (06) 6613-3111　FAX: (06) 6613-5888

URL: http://www.proface.co.jp