Pro-face
Human Machine Interface

Pro-face

PS Series Type G Application Development Kit (ADK)

Developer Manual

Preface

Thank you for purchasing Pro-face PS Series Type G Application Development Kit (hereafter referred to as “ADK”). This manual is the Developer Manual for the ADK, which contains the libraries and definition files necessary for developing applications that will run on Pro-face PS Series Type G Programmable Operator Interface (hereafter referred to as “PS-G unit”). Please read this manual thoroughly before using the ADK.

Important

(1) The copyrights to all programs and manuals included in the “PS Series Type G Application Development Kit (ADK)” (hereafter referred to as “this product”) are reserved by the Digital Electronics Corporation. Digital grants the use of this product to its users as described in the “END-USER LICENSE AGREEMENT” documentation. Any actions violating the above-mentioned agreement is prohibited by both Japanese and foreign regulations.

(2) The contents of this manual have been thoroughly inspected. However, if you should find any errors or omissions in this manual, please inform your local representative of your findings.

(3) Regardless of article (2), the Digital Electronics Corporation shall not be held liable by the user for any damages, losses, or third party claims arising from the uses of this product.

(4) Differences may occur between the descriptions found in this manual and the actual functioning of this product. Therefore, the latest information on this product is provided in data files (i.e. Readme.txt files, etc.) and in separate documents. Please consult these sources as well as this manual prior to using the product.

(5) The specifications set out in this manual are for overseas products only. As a result, some differences may exist between the specifications given here and for those of the identical Japanese product.

(6) Even though the information contained in and displayed by this product may be related to intangible or intellectual properties of the Digital Electronics Corporation or third parties, the Digital Electronics Corporation shall not warrant or grant the use of said properties to any users and/or other third parties.

For details concerning trademarks, please refer to “About Trademarks”.

Documentation Conventions

In this manual the following warning symbols are used to indicate important points concerning safe and proper use of the ADK. Prior to operating the ADK, be sure to read these points carefully.

Warning Symbols

This manual's warning symbols indicate the following levels of danger.

⚠ WARNING	Failure to fully comply with points indicated by this symbol may result in death or serious injury.
⚠ CAUTION	Failure to fully comply with points indicated by this symbol may result in injury or equipment damage.
⃠	Indicates actions or procedures that should NOT be performed.
❗	Indicates actions or procedures that MUST be performed to ensure proper operation.

The following mark is used in this manual in addition to the safety and warning symbols.

MEMO	Refers to related information or an additional explanation.

About Trademarks

The company names and product names described in this manual are referred to in this manual by their trademarks (including registered trademarks) or service marks. These are abbreviated as follows:

Trademark	Holder
Microsoft, MS, Windows, Windows 95, Windows 98, Windows NT, Windows 2000, Windows CE, Windows Explorer, eMbedded Visual C++, eMbedded Visual Basic	Microsoft Corporation
Intel, Pentium	Intel Corporation
Pro-face	Digital Electronics Corporation
IBM, VGA, PC/AT	IBM
Adobe, Acrobat	Adobe Systems Incorporated

The abbreviated trademarks in this manual correspond to the following official trademarks:

Term used in this manual	Official Trademarks
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

PS Series Type G Model Numbers

The PS Series Type G units consist of the following models.

Series	Model	Description	International standards
PS-600G	PS600G-T11-J1	Preinstalled with Windows CE 3.0 (Japanese). Input voltage AC100V	Not supported
	PS600G-T41-J124V	Preinstalled with Windows CE 3.0 (Japanese). Input voltage DC24V	CE Marking, UL/c-UL (CSA) compliant.
	PS600G-T41-E124V	Preinstalled with Windows CE 3.0 (English). Input voltage DC24V	CE Marking, UL/c-UL (CSA) compliant.
PS-400G	PS400G-T41-J124V	Preinstalled with Windows CE 3.0 (Japanese). Input voltage DC24V	CE Marking, UL/c-UL (CSA) compliant.
	PS400G-T41-E124V	Preinstalled with Windows CE 3.0 (English). Input voltage DC24V	CE Marking, UL/c-UL (CSA) compliant.

Please note that the term PS-G in this manual refers only to the PS600G-T41-E124V and the PS400G-T41-E124V.
The Japanese version of the PS Series Type G Application Development Kit should be used to develop applications for PS-G units with preinstalled Japanese Windows CE 3.0. To develop applications for PS-G units with preinstalled English Windows CE 3.0, use the English version of the PS Series Type G Application Development Kit.

CD-ROM Contents

The CD-ROM provided with this ADK package contains the following:

Item	Description	
PS Series Type G Application Development Kit (ADK) Developer Manual (This manual)	PDF manual for the PS Series Type G Application Development Kit (ADK)	
Acrobat Reader 4.0	Software for viewing PDF files (Self-extracting file)	
PS Series Type G Application Development Kit (ADK)	Microsoft Custom SDK	Standard [Windows CE] library files
	PS Series Type G Platform SDK (Platform Developer Components for the Microsoft Custom SDK)	Library and header files that allow programs such as RAS, etc. to access specific PS-G unit hardware.
ActiveSync 3.1	Enables communication between the PS-G unit and the personal computer used for developing applications.	

Usage Precautions

⚠ WARNINGS

During Program Development

⃠ Do not design emergency stop switches that require activation via the PS-G unit's touch panel. For industrial machinery or equipment, installation of a physically activated mechanical-type emergency switch is likely to be required by law. Also, for devices or equipment other than the above, be sure to install a safety switch that complies with all relevant laws and regulations.

⚠ CAUTIONS

Handling of CD-ROM disk

❗ Be sure to remove the CD-ROM disk before turning your personal computer's power OFF.

⃠ Do not attempt to remove the CD-ROM disc while the drive lamp is lit.

⃠ Do not touch the CD-ROM data surface.

⃠ Keep this disc away from excessively high or low temperatures, and environments with prolonged high humidity and dust.

⃠ Do not turn OFF your personal computer's power while a program is running.

Table of Contents

1 Development Environment

1 Overview

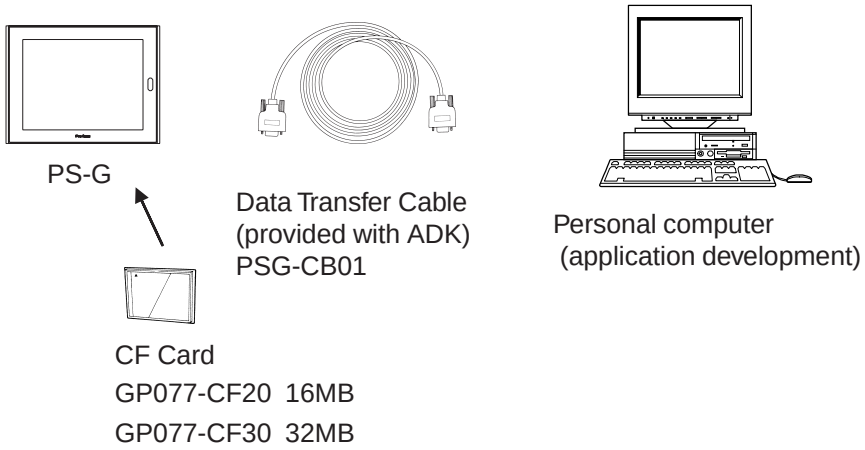
The Application Development Kit (ADK) containing libraries and definition files used for developing PS-G unit applications.
The new PS-G unit is a high-performance programmable operator interface that uses the Hitachi SH4 RISC CPU (200MHz) and is preinstalled with Windows CE 3.0.
To develop applications that will run on the PS-G unit, you will need both the eMbedded Visual Tools and the ADK-software.
eMbedded Visual Tools offer functionality similar to a regular Win32 application development environment (such as Visual Studio), and allow you to easily create applications for the PS-G unit. Since you can utilize your Win32 application programming knowledge and use familiar debugging tools, you can quickly create Windows CE applications for the PS-G unit. Also, the use of the Windows CE API, a subset of the Win32 API, helps to further improve development efficiency.

MEMO

- Since the eMbedded Visual Tools package is not included in the ADK package, you will need to purchase it separately.
- Certain Win32 APIs cannot be used with Windows CE. For details, please refer to the MSDN and check whether the APIs you intend to use are supported by Windows CE.

2 Hardware Environment

The following describes the hardware configuration required for developing and running PS-G unit applications.
Connect the PS-G unit to a personal computer via the data transfer cable provided with ADK. This allows an application developed on the personal computer to be easily download to the PS-G. If the personal computer you are using has a CF Card adaptor, you can run an application on the PS-G unit after copying it to the CF Card and then loading the card data into the PS-G unit. A CF Card is also required to save applications downloaded to the PS-G unit or data entered via the Control Panel of the PS-G unit.



MEMO

- Use only Digital Electronics Corporation CF Cards. Other vendor CF Cards may fail to meet PS-G specifications.
- Be sure to back up your CF Card data regularly.

The following describes the personal computer hardware required to develop applications. Refer to each package's hardware environment specifications for details.

<Required Personal Computer Hardware>

Operating System	Windows NT Workstation 4.0 with Service Pack 5 or later, Windows 2000 Professional, or Windows 98 Second Edition.
Computer	Windows compatible PC with Pentium processor (Pentium 150MHz or faster processor is recommended)
Memory	32MB for Windows NT 4.0 or Windows 2000 (48MB recommended) 24MB of RAM for Windows 98 (48MB recommended)
Hard-disk Space	Installing eMbedded Visual C++ and Windows CE Platform SDK (HPC Pro), or eMbedded Visual Basic and Windows CE Platform SDK (HPC Pro) requires 360MB or more. Installing the ADK software requires free space of 100MB or more.
CD-ROM Drive	A CD-ROM drive is required.

⚠ CAUTION

Windows 98 is not recommended if you want to both develop and debug applications, as problems with emulation functions or remote tools may occur during debugging. When developing applications that require performance of both the build and the debug process, be sure to use a personal computer running Windows NT 4.0 or Windows 2000.

3 Software Environment

To develop applications that will run on the PS-G unit, you need both the eMbedded Visual Tools and the ADK.
The eMbedded Visual Tools contain both eMbedded Visual C++ and eMbedded Visual Basic. ADK supports both of these visual programming tools.

3.1 Development Software

The following software should be installed when developing applications.

Item	Description	
eMbedded Visual Tools	Install eMbedded Visual C++ or eMbedded Visual Basic.	
	Windows CE Platform SDK (HPC Pro)	This program must be installed to run Windows CE applications under emulation on your computer.
PS Series Type G Application Development Kit (ADK)	Microsoft Custom SDK	Standard [Windows CE] library files
	PS Series Type G Platform SDK (Platform Developer Components for the Microsoft Custom SDK)	Library and header files that allow programs such as RAS, etc. to access specific PS-G unit hardware.
ActiveSync 3.1	Enables communication between the PS-G unit and the personal computer used for developing applications.(This program is included in the ADK package.)	
TCP/IP protocol	TCP/IP protocol must be used to download the applications to the emulator via Ethernet for debugging. (Be sure to set the appropriate network parameters for your development environment.)	

The eMbedded Visual Tools package contains the following remote tools. These tools allow the personal computer used for development to acquire necessary information from the PS-G unit (connected via Microsoft ActiveSync3.1), debug applications and check program operation.

- Remote Spy++ (Spy)
  Determines what kind of messages the target device receives or uses to check window handles, classes, and other items.
- Remote Registry Editor (Registry Editor)
  Used for editing the target device registry.
- Remote ZoomIn (Zoom)
  Enables you to capture target device screen images and view them on the personal computer used for developing applications.
- Remote File Viewer (File Viewer)
  Enables you to browse files on the target device.
- Remote Heap Walker (Heap Walker)
  Checks the heap ID and the current heap.
- Remote Process Viewer (Process Viewer)
  Enables you to check the processes, threads and modules that are currently being executed on the target device, and check for available memory areas.

3.2 ADK Files

Each ADK contains the following two software development kits: Microsoft Custom SDK and PS Series Type G Platform SDK. After an ADK is installed, individual SDK files are automatically extracted to the following folders.

Microsoft Custom SDK

Standard library files for Windows CE
\Windows CE Tools \Wce300\Bin
\Psgwce30 \Atl
\Help
\Include
\Lib
\Mfc
\Target
\Template
\Vbsdk

PS Series Type G Platform 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 Installing Application Development Tools

Before installing an ADK, be sure to check that the personal computer to be used for application development meets the specified requirements by referring to the table <Required Personal Computer Hardware> in "Chapter 1 2. Hardware Environment."

⚠ CAUTION

Windows 98 is not recommended if you want to both develop and debug applications, as problems with emulation functions or remote tools may occur during debugging. When developing applications that require performance of both the build and the debug process, be sure to use a personal computer running Windows NT 4.0 or Windows 2000.

To install the application development tools, follow the procedure below.
Be sure to follow the procedures given here, in sequence. If you do not, the tools may not operate correctly.

(1) Install the eMbedded Visual Tools.
(2) Install the ADK for eMbedded Visual C++ or eMbedded Visual.
(3) Install ActiveSync 3.1.

Installing the eMbedded Visual Tools

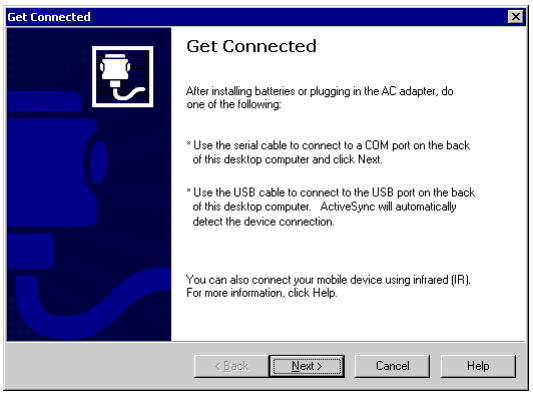
(1) Install the eMbedded Visual Tools software.
eMbedded Visual Tools contains both eMbedded Visual C++ and eMbedded Visual Basic. Install either or both, depending on your needs.

Installing the ADK

(1) Insert the ADK CD-ROM into your personal computer's CD-ROM drive and install either the ADK for eMbedded Visual C++ or eMbedded Visual Basic.
- To install ADK for Visual C++
Install ADK for Visual C++ by double-clicking on the [psgsdk.exe] file located in the CD-ROM's [\us\psg-adk\vc] folder. After the installation screen appears, follow each screen's instructions.
- To install ADK for Visual Basic
Install ADK for Visual Basic by double-clicking on the [psgsdk.exe] file located in the CD-ROM's [\us\psg-adk\vb] folder. After the installation screen appears, follow each screen's instructions.

Installing ActiveSync 3.1 and connecting with the PS-G unit

(1) Install ActiveSync 3.1 by double-clicking on the [msasync.exe] file located in the ADK CD-ROM's [\us\async] folder, and then follow each screen's instsructions.
As the installation progresses, the computer enters the PS-G connection standby mode. However, at this stage, do not establish the connection. This is done by not clicking the Next > button until step (11).

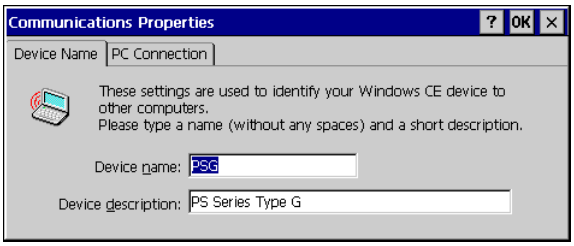


(2) Turn ON the power to the PS-G unit, and connect the data transfer cable (provided with the ADK) from the COM port on the personal computer to the serial I/F (COM1) on the PS-G unit.

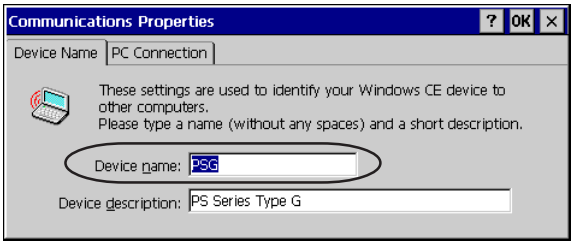
(3) To open the PS-G unit's Control Panel, click the [Start] button and then point to [Settings]. From the menu that appears, click [Control Panel].

(4) Double click 

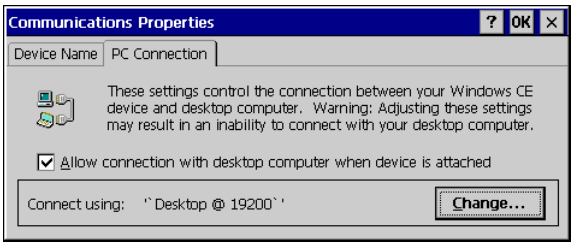
to open the [Communications Properties] screen.



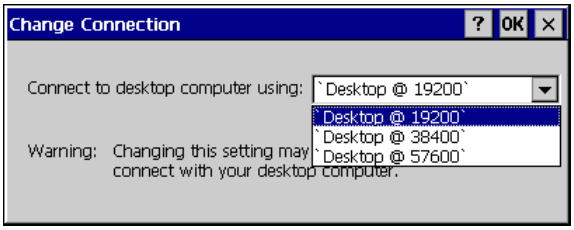
(5) Enter a name in the [Device name] field so that the PS-G unit will be recognized by the personal computer.



(6) Click the [PC Connection] tab, and then the Change... button.



(7) Select the desired baud rate setting.



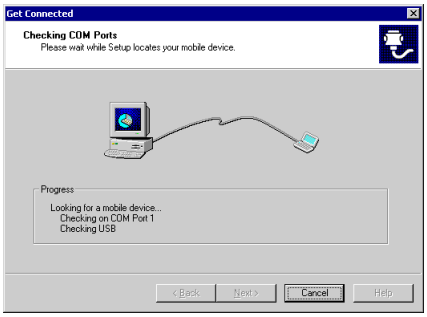
(8) Click the OK button to register the [Communications Properties] settings.

(9) In the Control Panel, click on the CF Backup icon to save the remote connection settings to the CF Card. Then, reboot the PS-G unit.

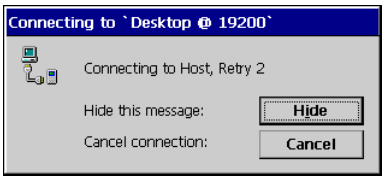
(10) After the PS-G unit is rebooted, open the [\Windows] folder, located in My Computer.

(11) Click the Next > button on the dialog box that appeared in step (1), and immediately double-click the 

icon (connection establishment program).



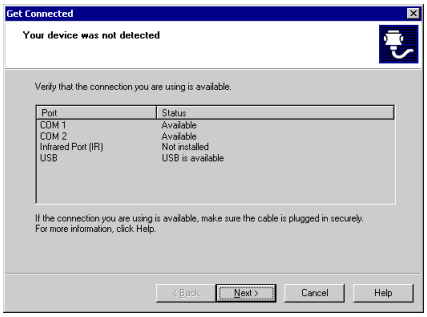
Personal computer



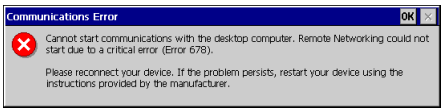
PS-G unit

The connection with the PS-G unit will not be established if the timing of the double-clicking of 

is not correct . If the connection fails, the following screen will appear on both the PC and the PS-G unit. In this case, retry step (11).

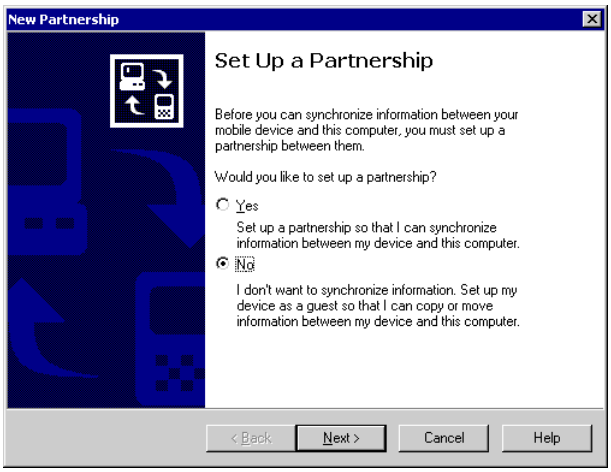


Personal computer

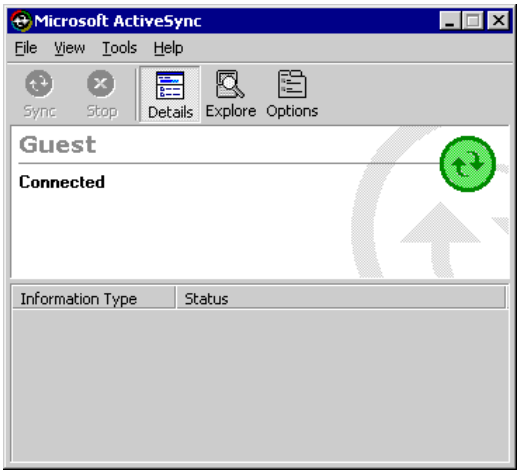


PS-G unit

(12) When the connection is successfully established, the following screen will appear on the PC. Select [No] for [Would you like to set up a partnership?] and click the Next > button.



(13) When the connection is completed, the [Microsoft ActiveSync] screen appears on the PC.



Also, an icon signifying a successful connection appears in the PS-G unit's the task bar.

MEMO

Once the connection is established, ActiveSync 3.1 stays resident in the computer memory and the connection settings are stored, allowing the PS-G unit to be connected with the personal computer from next time by simply double-clicking the PS-G unit's  icon.

5 Remote Connection Procedure

To transfer a program from the personal computer used for developing applications to the PS-G unit and then debug the program while it is running on the PS-G unit, use ActiveSync 3.1 to establish the connection between the personal computer and the PS-G unit.

MEMO

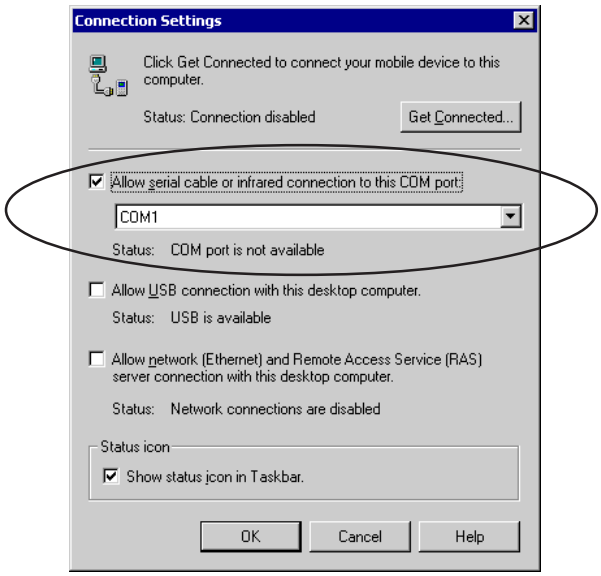
Transfering a program via an Ethernet network greatly shortens the transfer time. To use an Ethernet network, you should set the parameters appropriate for the development environment.

Turn ON the power to the PS-G unit, and connect the data transfer cable (provided with the ADK) from the COM port on the personal computer to the serial I/F (COM1) on the PS-G unit.

5.1 Personal Computer Settings

ActiveSync 3.1 - Connection Settings

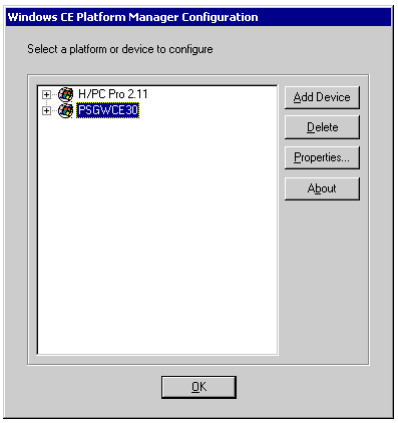
Launch ActiveSync 3.1 and select [Connection Settings] from the [File] menu. In the following screen, designate the data transfer cable port.



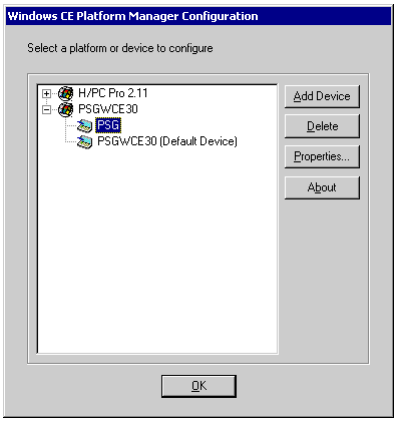
eMbedded Visual Tools - Platform Manager Configuration

From the [Platform Manager Configuration] screen, register a device name in your PC for the PS-G unit. This enables you to designate the PS-G unit as the download destination. The device properties should also be set according to the communication settings.

(1) Call up the [Platform Manager Configuration] screen following either of the procedures below. When you use eMbedded Visual C++, select [Platform Manager Configuration] from the [Tools] menu. When you use eMbedded Visual Basic, select [Project 1 - Properties] from the [Project] menu and then click the Configure Target button.



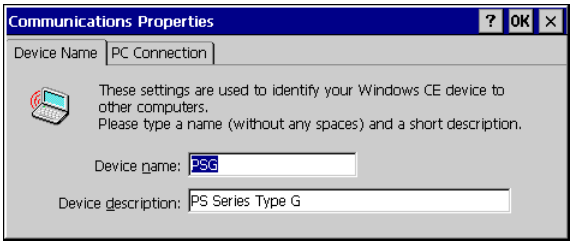
(2) Select [PSG] and click the Add Device button to add its name as a selectable device.



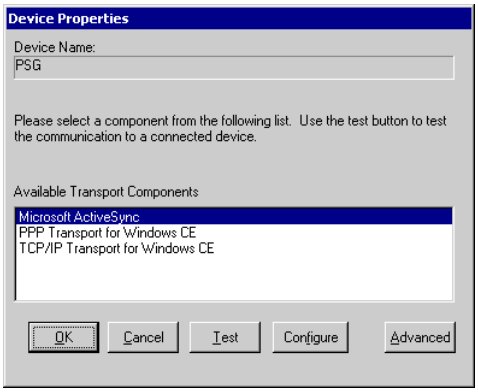
(The unit selected here is the PS-G device name.)

MEMO

The device name you designate should agree with the one set in the PS-G unit. Be sure to designate the same device name as the one you set in the PS-G unit's [Communications Properties] screen. For details, refer to "Chapter 1 5.2 PS-G Unit Settings"

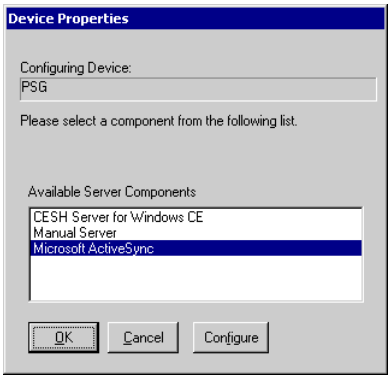


(3) Click the Properties... button to display the [Device Properties] dialog box. From [Available Transport Components], select [Microsoft ActiveSync]. If you intend to use Ethernet, select [TCP/IP Transport for Windows CE].



MEMO Transfering a program via an Ethernet network greatly shortens the transfer time. To use an Ethernet network, you should set the parameters appropriate for the development environment.

(4) Click the Advanced button and the following screen will appear. From [Available Server Components], select "Microsoft ActiveSync".



(5) Click the OK button to register the settings and complete the Platform Manager setup.

5.2 PS-G Unit Settings

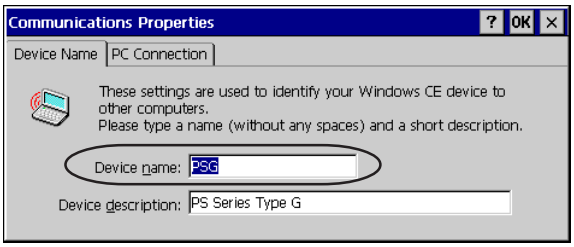
Control Panel - Communications Settings

(1) To open the PS-G unit's Control Panel, click the [Start] button and then point to [Settings]. From the menu that appears, click [Control Panel].

(2) Double Click 

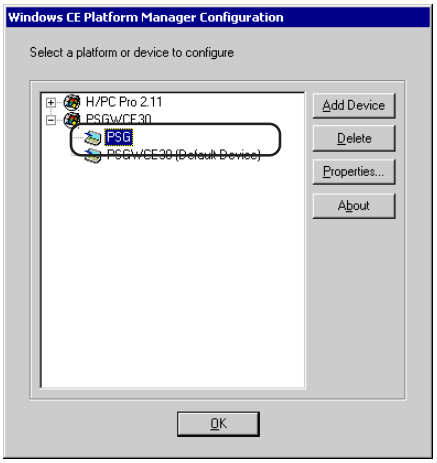
to open the [Communications Properties] screen.

(3) Enter a name in the [Device name] field so that the PS-G unit will be recognized by the personal computer it is connected to.

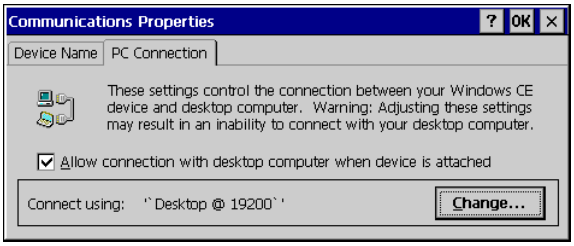


MEMO

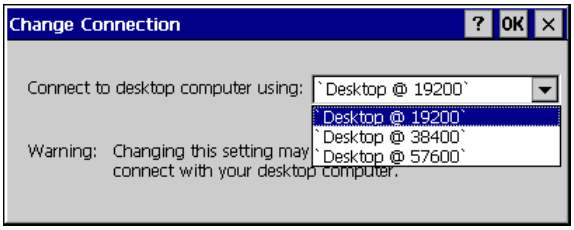
The device name you enter here should agree with the one set on the personal computer. Be sure to enter the same device name as the one you set in the [Platform Manager Configuration] screen on the personal computer. For details, refer to "Chapter 1 5.1 Personal Computer Settings"



(4) Click the [PC Connection] tab, and then the Change... button.



(5) Select the desired baud rate setting.



(6) Click the OK button to register and complete the [Communications Properties] settings.

(7) In the Control Panel, click on the 

icon to save the remote connection settings to the CF Card. Then, reboot the PS-G unit.

5.3 Connection

If you have already connected a personal computer to the PS-G unit prior installing ActiveSync 3.1, the connection settings for the personal computer and the PS-G unit can be established by simply double-clicking 

and following the procedures from step 12 in "Chapter 1 4. Installing Application Development Tools." If the connection settings have not yet been established, launch ActiveSync 3.1 and then select [Connection] from the [File] menu. The personal computer will go into PS-G connection standby mode. You can then establish the connection settings by following the procedures in "Chapter 1 4. Installing the Application Development Tools."

2 Application Development

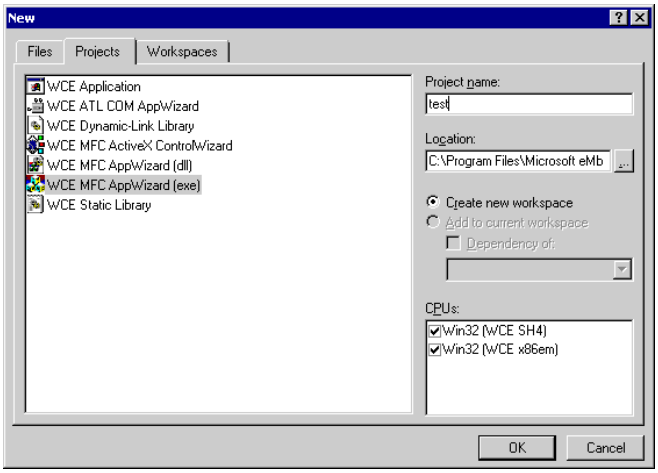
1 Development using eMbedded Visual C++
2 Development using eMbedded Visual Basic
3 Debugging Programs Using the Emulator
4 Auto Start

1 Development using eMbedded Visual C++

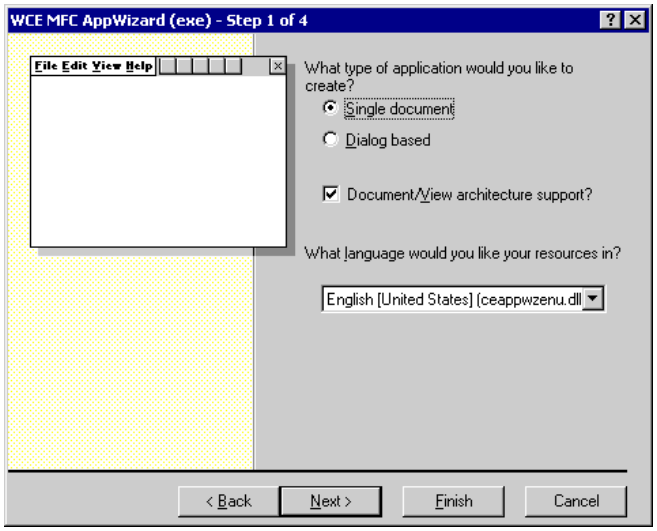
The following explains the procedure for using eMbedded Visual C++ to create a simple application to display an on-screen window.

1.1 Creating a Project

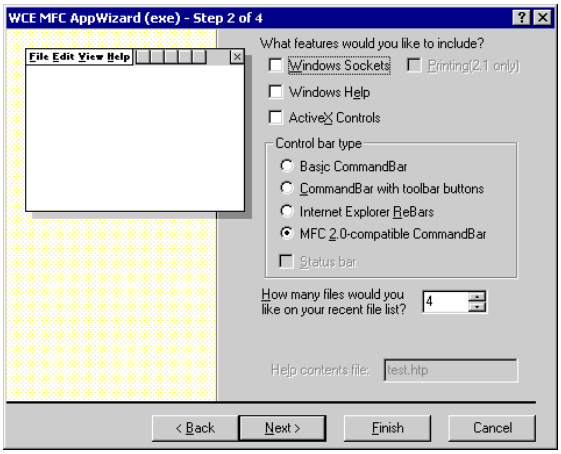
(1) Start eMbedded Visual C++ and select [New] from the [File] menu. In the [Projects] tab, select the type of program you want to create and input a project name. Select "Win32 [WCE SH4]" for "CPUs". If you intend to debug the program by running it using the emulator, also select "Win32 [WCE x86em]".
Click the OK button to proceed to the next step.



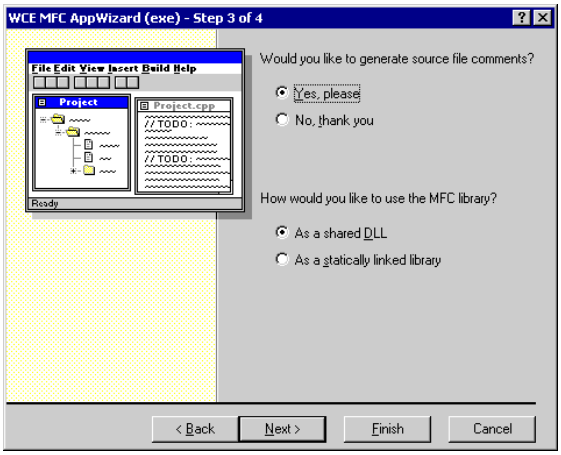
(2) Select the necessary items and click the Next > button.
To create a program using a dialog-based interface, select the [Dialog based] option.



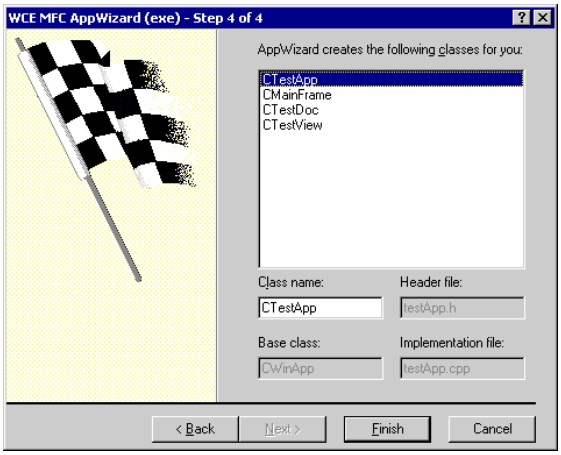
(3) Select the necessary items and click the Next > button to proceed to the next step.

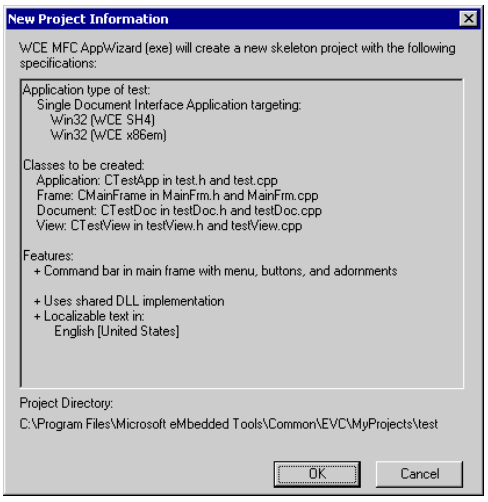


(4) Again, select the necessary items and click the Next > button to proceed to the next step.



(5) Click the Finish button, and the following page's [New Project Information] screen will appear.

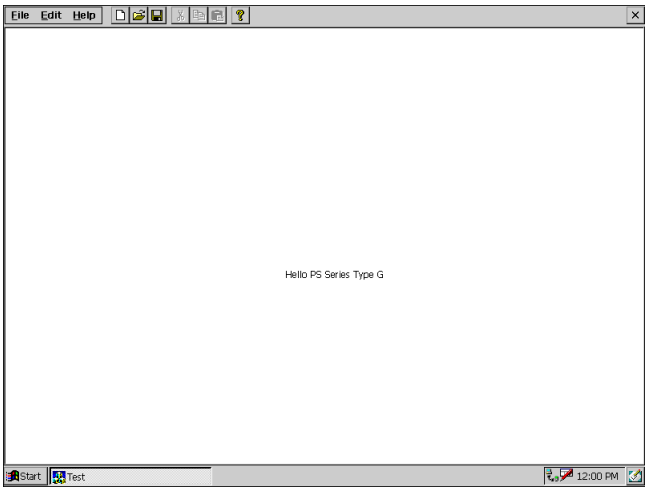




(6) Click the OK button and your new project creation is completed.

1.2 Building and Downloading a Program

Here, as an example we will create a program to display the text "Hello PS Series Type G" in the center of a window.



Add the following description to the OnDraw view class.

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

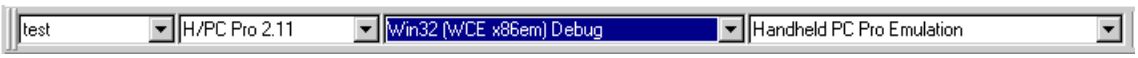
When you Build the program, it is automatically downloaded to the designated device. Before executing the build process, from the [WCE Configuration] tool bar, designate the SDK and other environment to be used in this process via the options displayed in the "Select Active WCE Configuration", "Select Active Configuration" and "Select Default Device" pop-up menus. Depending on the program’s intended use, the setting of the [WCE Configuration] tool bar will differ. The following three types of program build methods are explained below.

Programs debugged using the emulator
Programs debugged using the PS-G unit
Programs released for the PS-G unit

Program debugged using the emulator

To debug a program by running it using the emulator, the Windows CE Platform SDK (HPC Pro) is necessary. For details, refer to "Chapter 1 3.1 Development Software."

(1) Set the tool bar as follows and then execute the build.



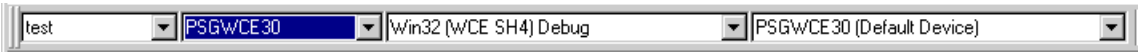
(2) When the build completes successfully, the program is automatically downloaded to the emulator. If the emulator is not already running, it will be launched and then it will load the program automatically.

(3) After downloading is completed, the program is saved in the [\My Handheld PC] folder. Run the program by double-clicking on the program file's icon. To debug this program, start the debugger. For details, refer to "Chapter 2 3 Debugging Programs Using the Emulator."

Program debugged using the PS-G unit

(1) Connect the personal computer to be used for development and the PS-G unit via ActiveSync3.1. For details, refer to "Chapter 1 5 Remote Connection Procedure."

(2) Set the [WCE Configuration] tool bar as follows, and execute the build.



(3) When the build completes successfully, program downloading to the PS-G unit starts automatically.

(4) After downloading is completed, the program is saved in the PS-G unit's [\My Computer] folder. Run the program by double-clicking on the program file's icon. To debug this program, start the PC's debugger.

Programs released for the PS-G unit

(1) Connect the personal computer to be used for development and the PS-G unit via ActiveSync3.1. For details, refer to "Chapter 1 5 Remote Connection Procedure."

(2) Set the [WCE Configuration] tool bar as follows, and execute the build.

(3) When the build completes successfully, program downloading to the PS-G unit starts automatically.

(4) After downloading is completed, the program is saved in the PS-G unit's [\My Computer] folder. Run the program by double-clicking on the program file's icon.

1.3 Running a Program

The program is downloaded to the specified target device. Run the downloaded program by double-clicking on the program file's icon. To debug this program, start the PC's debugger.
It is also possible to run a program that has been copied to a CF Card. To run a program from a CF Card, a backup copy of the PS-G's environment settings for running applications should also be saved on the CF-Card. Insert the CF Card into the PS-G unit and double-click 

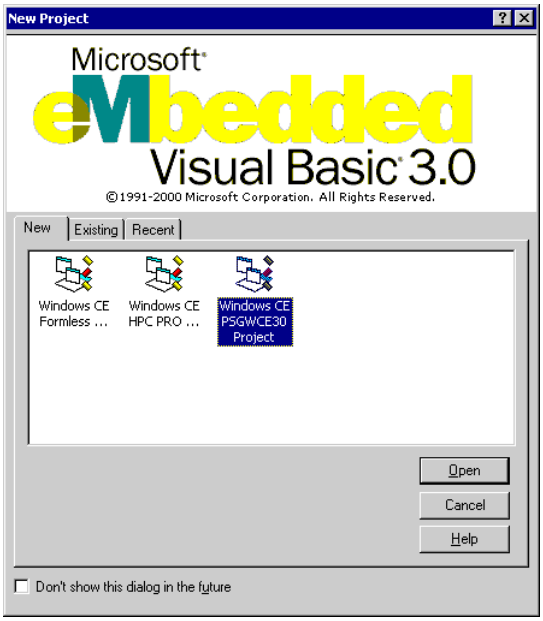
in the Control Panel. Then, double-click the program file's icon located in the [\My Computer\Storage Card1] folder to run the program.

2 Development using eMbedded Visual Basic

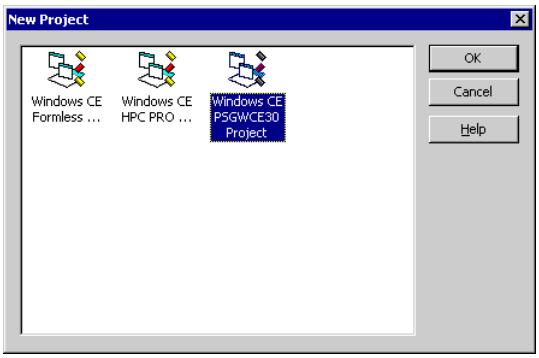
The following explains the procedure for using eMbedded Visual Basic to create a simple application to display an on-screen window.

2.1 Creating a Project

(1) After starting Visual Basic, the following [New Project] dialog box appears.



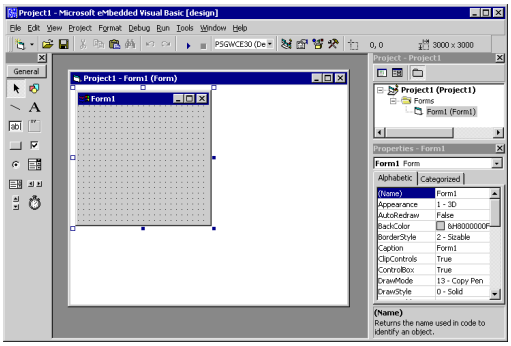
You can also select [New Project] from the [File] menu to display the [New Project] dialog box.



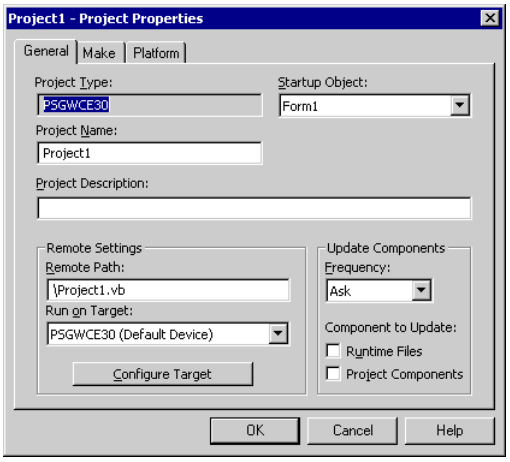
(2) Select [Windows CE PSGWCE30 Project] to create a PS-G program. If you want to create a program that will run using the emulator, select [Windows CE HPC PRO Project].

MEMO To run a program (debugged using the emulator) on the PS-G, be sure to add the forms and modules that were created for the Windows CE HPC Pro project to the Windows CE PSGWCE30 project.

(3) After the project type is selected, the following screen will appear.

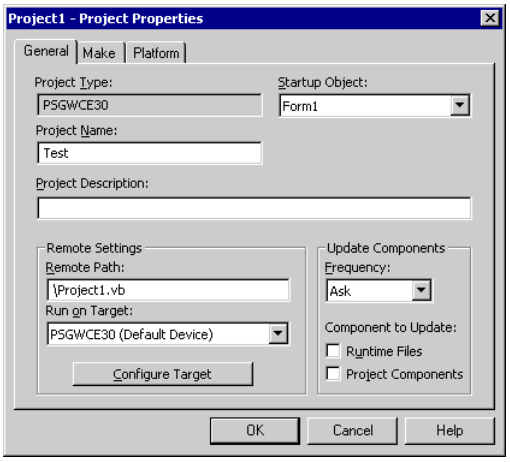


(4) (4) Select [Project 1 - Properties] from the [Project] menu, and the [Project Properties] dialog box will appear.



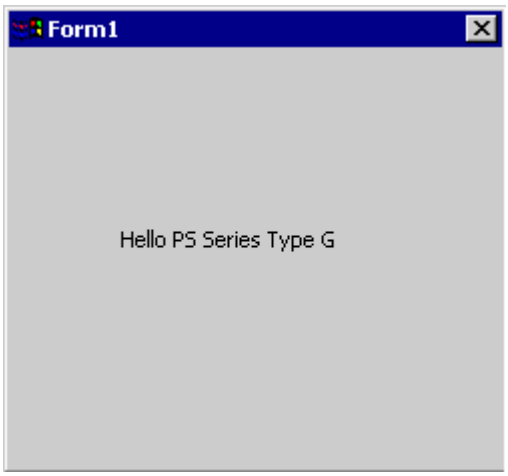
(Here, the project type has been set to "Windows CE PSGWCE30".)

(5) Change the name of the project, if necessary.

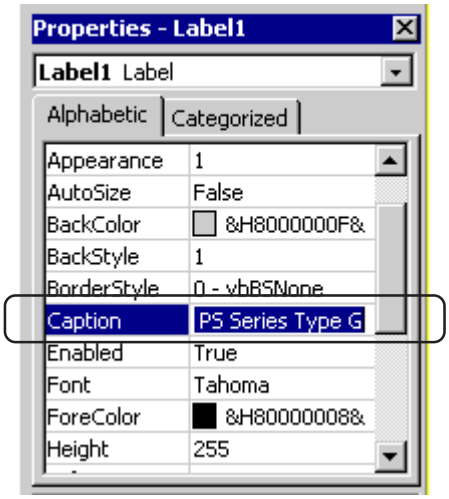


2.2 Downloading and Running a Program

Here, we will create a program to display the text "Hello PS Series Type G" in the center of a window.

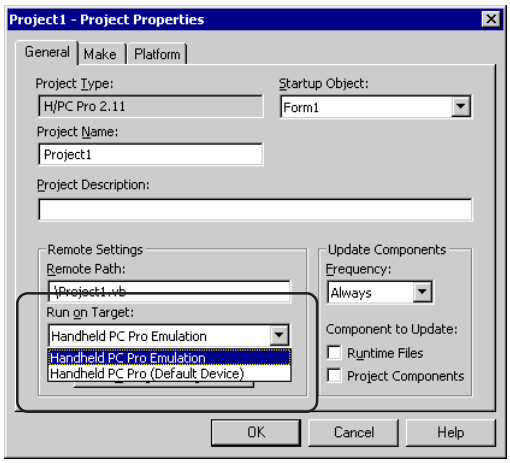


(1) Select a label from the toolbox, and paste it on the form. Input “Hello PS Series Type G” into the “Caption” property.

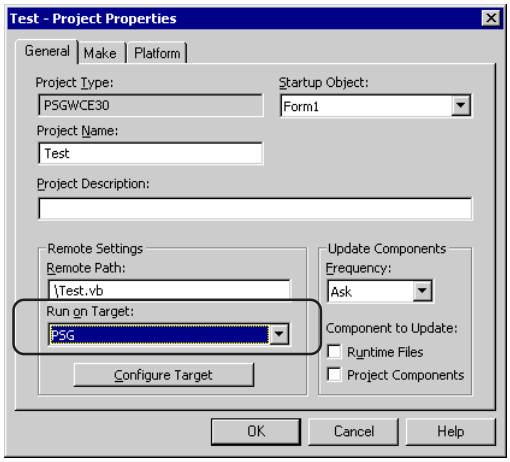


(2) Select [Project 1 - Properties] from [Project]. In the [Project Properties] screen that appears, designate a target device from the [Run on Target] pull-down menu.

If you intend to debug a program (Windows CE HPC Pro project) by running it using the emulator, select [Handheld PC Pro Emulation].



If you intend to debug a program (Windows CE PSGWCE30 project) by running it on the PS-G unit, select the device name for the PS-G that was registered using the Platform Manager. For details, refer to "Chapter 1 4.1 Settings on the Personal Computer."



(3) When you designate the PS-G unit as the data download target device, connect your PC to the PS-G unit using the data transfer cable.

(4) Selecting [Start Debug] or [Execute] from the [Run] menu initiates downloading of the program to the designated target device. After downloading of the program completes, the debugging will start.
If a program (Windows CE HPC Pro project) is to be debugged using the emulator, the program is downloaded to the [\My Handheld PC] folder.
If a program (Windows CE PSGWCE30 project) is to be run on the PS-G unit, the program is downloaded to the [My Computer] folder of the PS-G unit.

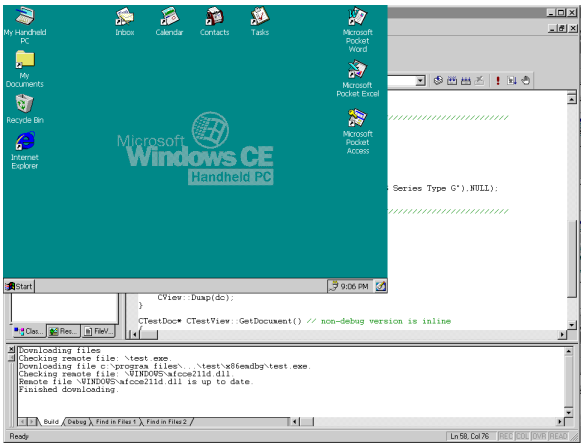
(5) After program download is completesd, you can run the program by simply double-clicking the program file's icon. To debug the program, execute the PC's debugger. For details, refer to "Chapter 2 3. Debugging Programs Using the Emulator."
It is also possible to run a program that has been copied to a CF Card on the PS-G unit. To run a program from a CF Card, a backup copy of the PS-G's environment settings for running applications should also be saved on the CF-Card. Insert the CF Card into the PS-G unit and double-click 

in the Control Panel. Then, double-click the program file's icon located in the [\My Computer\Storage Card1] folder to run the program.

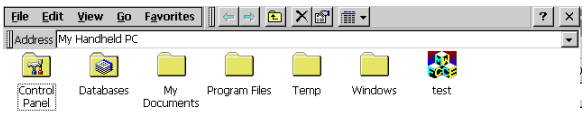
3 Debugging Programs Using the Emulator

If you intend to debug a program without using the PS-G unit's special hardware interfaces (such as RAS or the touch panel), you can effectively debug the program using the emulator included in the Windows CE Platform SDK (HPC Pro) in your PC. Please note, however, that the Windows CE Platform SDK (HPC Pro) only runs under Windows NT or Windows 2000.

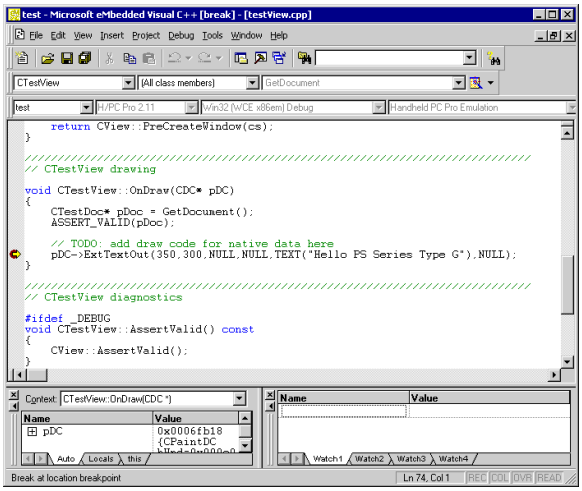
- Start the emulator and the following screen will appear.



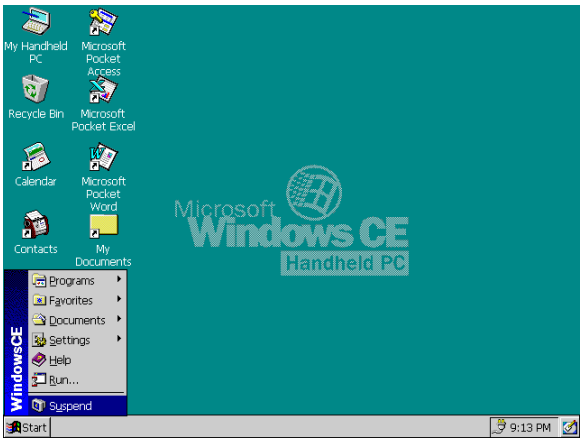
- After the program is loaded to the emulator, the executable file is saved in the [\My Handheld PC] folder.



- Debugging can be started by simply clicking the saved exec file.
  You can also debug the program by running it to a specified break point, set earlier via the eMbedded Visual Tools debug feature.



- To close the emulator, click [Suspend] in the [Start] menu.



4 Auto Start

An application can be automatically started when the PS-G unit is turned ON. The following explanation describes this procedure.

To automatically start an application in object store memory

(1) Create a shortcut for the application and place it in the PS-G's [\Windows\Startup] folder, or place a copy of the application there.

(2) Insert a CF Card into the PS-G, and click the 

icon in the Control Panel.

(3) Reboot the PS-G unit with the CF Card inserted.

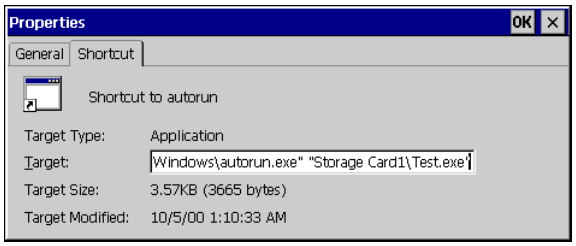
(4) Once Windows CE starts, the PS-G unit will read the environment data from the CF Card and launch the application registered in the [\Windows\Startup] folder.

To automatically start a CF Card application

(1) Create a shortcut in the [\Windows\Startup] folder pointing to [AutoRun.exe] in the PS-G unit's [\Windows] folder.

(2) Open the Properties of the shortcut created in step (1), open the [Shortcut] tab and set the application to run automatically by changing the link destination in the "Target" field as follows.

Target: "\Windows\AutoRun.exe"_"Storage _Card1\Application name"



MEMO If "/F" is added to the link destination setting, e.g., "Windows\AutoRun.exe"_/F_"Storage _Card1\application name", an application can be launched without showing the task bar at the bottom of the screen display. Use this feature to disable the Explorer Shell of Windows CE in the user application.

(3) Click the OK button.

(4) Insert a CF Card into the PS-G, and click the 

icon in the Control Panel.

(5) Reboot the PS-G with the CF Card inserted.

(6) Once Windows CE starts, the PS-G unit will read the environment data from the CF Card and launch the application registered in the [\Windows\Startup] folder.

3 Library Interface Reference

To use the PS-G unit's special hardware interfaces in your applications, the required header file(s) should be included in the application's source file so that the necessary interface library files can be linked to the object file.

- Backlight driver
- SRAM driver
- GMU-BUS driver
- RAS driver
- Touch panel driver

Driver Name	Library file	eMbedded Visual C++ header file	eMbedded Visual Basic header file
Backlight driver	BlDrvIf.lib	BlDrvApi.h	BlDrvDef.bas
SRAM driver	SramDrvIf.lib	SramDrvApi.h	SramDrvDef.bas
GMU-BUS driver	GmuDrvIf.lib	GmuDrvApi.h	GmuDrvDef.bas
RAS driver	RasDrvIf.lib	RasDrvApi.h	RasDrvDef.bas
Touch panel driver	TouchDrvIf.lib	TchDrvApi.h	TchDrvDef.bas

MEMO Detailed interface specifications for eMbedded Visual C++ are given in the following pages. When developing applications using eMbedded Visual Basic, although the application APIs are common to eMbedded Visual C++ APIs, the arguments will differ. For more information, refer to the available eMbedded Visual Basic header files.

To use an individual API contained in the ADK, it is necessary to set the drivers to Open before using the API and to change the setting back to Close after the API is used. Use the following example as a reference when creating a program.

Example)
```
// Example of the settings for drivers
// Not only the SRAM but also other drivers need the Open/Close settings.
// Declaration for variables
BOOL Ret;
DWORD SramOffset;
BYTE SramData;
// Application reinstate processing
// Before using the drivers, each driver should be set to Open.
// Call up the corresponding "xxx DriverOpen( )" setting for each driver.
Ret = SramDriverOpen();                          // SRAM driver Open
        :
// Access to the SRAM
Ret = SramByteRead(SramOffset, &SramData);  // Reads data from the SRAM.
        :
Ret = SramByteWrite(SramOffset, SramData);     // Writes data into the SRAM.
        :
// Application exit processing
// After the drivers are used, each driver should be set to Close.
// Call up the corresponding "xxx DriverOpen( )" setting for each driver.
Ret = SramDriverClose();
```

1 Backlight Driver

The function of this driver is to set and control the backlight status. This driver's functionality is equivalent to the control panel icon.

1.1 Backlight Driver APIs

API name	Description
GetBLDriverVersion	Retrieves driver version information
BLDriverOpen	Opens the backlight driver
BLDriverClose	Closes the backlight driver
GetBLStatus	Retrieves the backlight status
SetBLOnOff	Turns the backlight ON/OFF
SetBLBright	Sets the brightness of the backlight
SetBLAction	Enables/disables input in the case of backlight burnout.
GetBLAction	Retrieves backlight burnout settings.
SetBLTmOut	Sets the timer value used for turning the backlight OFF when the PS-G is idle.
GetBLTmOut	Retrieves the timer value used for turning the backlight OFF when the PS-G is idle.

1.2 Function Specifications

GetBLDriverVersion

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

Return value
TRUE: Normal
FALSE: Error

Argument
WORD *pMajor Pointer to the version information (Major, 0 to 99)
WORD *pMinor Pointer to the version information (Minor, 0 to 99)

Processing performed
Retrieves information on the driver version.
E.g.) BOOL ret = GetBLDriverVersion (WORD &Major, WORD &Minor);

Note
When the driver version is 1.10,
Major is 1 (in decimal) and;
Minor is 10 (in decimal).

BLDriverOpen

Call format
BOOL WINAPI BLDriverOpen (void)

Return value
TRUE: Normal
FALSE: Error

Argument
None

Processing performed
Opens the backlight driver, enabling features to be set.
E.g.) BOOL ret = BLDriverOpen ( );

BLDriverClose

Call format
BOOL WINAPI BLDriverClose (void)

Return value
TRUE: Normal
FALSE: Error

Argument
None

Processing performed
Closes the backlight driver, disabling setting operations. However, the thread operation of the driver continues.
E.g.) BOOL ret = BLDriverClose ( );

GetBLStatus

Call format
BOOL WINAPI GetBLStatus (BYTE *pStatus)

Return value
TRUE: Normal
FALSE: Error

Argument
BYTE *pStatus Pointer to address where Backlight Status data is stored.

Backlight status
D7: Error
D4: Backlight burnout 1: Burnout 0: Normal
D3: Backlight status 1: ON 0: OFF
D2 to D0: Backlight brightness

Processing performed
Retrieves the ON/OFF status and brightness of the backlight.
E.g.) // Retrieves the backlight status.
```
BYTE Status;
BOOL ret = GetBLStatus (&Status);
```

SetBLOnOff

Call format
BOOL WINAPI SetBLOnOff (BOOL bSwitch)

Return value
TRUE: Normal
FALSE: Error

Argument
BOOL bSwitch TRUE: Turns the backlight ON.
FALSE: Turns the backlight OFF.

Processing performed
Controls the backlight.
E.g.) // Turns the backlight OFF.
BOOL ret = GetBLOnOff (FALSE);

SetBLBright

Call format
BOOL WINAPI SetBLBright (BYTE Bright)

Return value
TRUE: Normal
FALSE: Error

Argument
BOOL Bright 0 to 3: 0 (darkest) to 3 (brightest)

Processing performed
Controls the brightness of the backlight.
E.g.) // Sets the backlight to its brightest level.
BOOL ret = SetBLBright (3);

SetBLAction

Call format
BOOL WINAPI SetBLAction (BYTE Action)

Return value
TRUE: Normal
FALSE: Error

Argument
BYTE Action

D2: Input from touch panel	1: Enabled	0: Disabled
D1: Input from keyboard	1: Enabled	0: Disabled
D0: Input from mouse	1: Enabled	0: Disabled

Processing performed
Enables/disables input from the touch panel, keyboard, and mouse when backlight burnout is detected.
The data for this setting will be stored in the registry.
E.g.) // Disables input from the touch panel, keyboard, and mouse in the case of backlight burnout.

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

GetBLAction

Call format
BOOL WINAPI GetBLAction (BYTE *pAction)

Return value
TRUE: Normal
FALSE: Error

Argument
BYTE *pAction Pointer to the setting for whether or not input is enabled in the case of backlight burnout.

BYTE Action

D2: Input from touch panel	1: Enabled	0: Disabled
D1: Input from keyboard	1: Enabled	0: Disabled
D0: Input from mouse	1: Enabled	0: Disabled

Processing performed
Retrieves the setting for whether or not input from the touch panel, keyboard, and mouse is enabled when backlight burnout is detected.
E.g.) // Retrieves the setting for input from the touch panel, keyboard, and mouse in the case of backlight burnout.

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

SetBLTmOut

Call format
BOOL WINAPI SetBLTmOut (DWORD TmOut)

Return value
TRUE: Normal
FALSE: Error

Argument
DWORD TmOut — Specified timer value for turning the backlight OFF in the case of idle time (0, 15, 30, 60, 120, 300, 600, 900, or 1800)

Processing performed
Sets the timer value used for turning the backlight OFF when there is no input via the touch panel, keyboard, or mouse.
E.g.) DWORD TmOut = 120;
BOOL ret = SetBLTmOut (TmOut);

GetBLTmOut

Call format
BOOL WINAPI GetBLTmOut (DWORD *pTmOut)

Return value
TRUE: Normal
FALSE: Error

Argument
DWORD *pTmOut — Pointer to the specified timer value for turning the backlight OFF in the case of idle time

Processing performed
Retrieves the timer value used for turning the backlight OFF when there is no input via the touch panel, keyboard, or mouse.
E.g.) DWORD TmOut;
BOOL ret = GetBLTmOut (&TmOut);

2 SRAM Driver

The function of this driver is to read/write data from/to SRAM.
Up to 256KB is available. If a function's return value exceeds 256KB during access, an error will occur.

2.1 SRAM Driver APIs

API name	Description
GetSramDriverVersion	Retrieves information on the driver version.
SramDriverOpen	Opens the SRAM driver.
SramDriverClose	Closes the SRAM driver.
SramByteRead	Reads data from the SRAM. (Byte)
SramWordRead	Reads data from the SRAM. (Word)
SramLongRead	Reads data from the SRAM. (Long)
SramByteWrite	Writes data into the SRAM. (Byte)
SramWordWrite	Writes data into the SRAM. (Word)
SramLongWrite	Writes data into the SRAM. (Long)

2.2 Function Specifications

GetSramDriverVersion

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

Return value
TRUE: Normal
FALSE: Error

Argument
WORD *pMajor Pointer to the version information (Major, 0 to 99)
WORD *pMinor Pointer to the version information (Minor, 0 to 99)

Processing performed
Retrieves the driver version.
E.g.) BOOL ret = GetSramDriverVersion (WORD &Major, WORD &Minor);

Note
When the driver version is 1.10,
Major is 1 (in decimal) and;
Minor is 10 (in decimal).

SramDriverOpen

Call format
BOOL WINAPI SramDriverOpen (void)

Return value
TRUE: Normal
FALSE: Error

Argument
None

Processing performed
Opens SRAM driver, enabling driver operations.
E.g.) BOOL ret = SramDriverOpen ( );

SramDriverClose

Call format
BOOL WINAPI SramDriverClose (void)

Return value
TRUE: Normal
FALSE: Error

Argument
None

Processing performed
Closes SRAM driver, disabling driver operations. However, driver thread operation continues.
E.g.) BOOL ret = SramDriverClose ( );

SramByteRead

Call format
BOOL WINAPI SramByteRead (DWORD Offset, BYTE *pData)

Return value
TRUE: Normal
FALSE: Error

Argument

DWORD Offset	Offset from the beginning of SRAM to the data to be read (in byte units) (0 to 3ffffh)
BYTE *pData	Pointer to the address where readout will be stored

Processing performed
Reads data from the specified address of SRAM.
E.g.) // Reads data of the byte 16th from the beginning of SRAM.

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

SramWordRead

Call format
BOOL WINAPI SramWordRead (DWORD Offset, WORD *pData)

Return value
TRUE: Normal
FALSE: Error

Argument

DWORD Offset	Offset from the beginning of SRAM to the data to be read (in word units) (0 to 1ffffh)
WORD *pData	Pointer to the address where readout will be stored

Processing performed
Reads data from the specified address of SRAM.
E.g.) // Reads data of the word 16th from the beginning of SRAM.

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

SramLongRead

Call format
BOOL WINAPI SramLongRead (DWORD Offset, DWORD *pData)

Return value
TRUE: Normal
FALSE: Error

Argument

DWORD Offset	Offset from the beginning of SRAM to the data to be read (in double word units) (0 to ffffh)
WORD *pData	Pointer to the address where the readout will be stored

Processing performed
Reads data from the specified address of SRAM.
E.g.) // Reads double word data, 16th from the beginning of SRAM.

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

SramByteWrite

Call format
BOOL WINAPI SramByteWrite (DWORD Offset, BYTE Data)

Return value
TRUE: Normal
FALSE: Error

Argument

DWORD Offset	Offset from the beginning of SRAM to the destination of the data to be written (in byte units) (0 to 3ffffh)
BYTE Data	Data to be written

Processing performed
Writes the designated data into the specified address of SRAM.
E.g.) // Writes data (aah) to the byte 16th from the beginning of SRAM.

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

SramWordWrite

Call format
BOOL WINAPI SramWordWrite (DWORD Offset, WORD Data)

Return value
TRUE: Normal
FALSE: Error

Argument

DWORD Offset	Offset from the beginning of SRAM to the destination of the data to be written (in word units) (0 to 1ffffh)
WORD Data	Data to be written

Processing performed
Writes the given data into the specified SRAM address.
E.g.) // Writes data (55aah) to the word 16th from the beginning of SRAM.
BOOL ret = SramWordWrite (0x10, 0x55aa);

SramLongWrite

Call format
BOOL WINAPI SramLongWrite (DWORD Offset, DWORD Data)

Return value
TRUE: Normal
FALSE: Error

Argument

DWORD Offset	Offset from the beginning of SRAM to the destination of the data to be written (in double word units) (0 to ffffh)
DWORD Data	Data to be written

Processing performed
Writes the given data to the specified SRAM address.
E.g.) // Writes the data (55aa55aah) into the double word 16th from the beginning of SRAM.
BOOL ret = SramLongWrite (0x10, 0x55aa55aa);

3 GMU-BUS Driver

This driver provides GMU-BUS control. Interrupt support is provided via registration of the user callback routine.

3.1 GMU-BUS Driver APIs

API name	Description
GetGmuDriverVersion	Retrieves the driver version.
GmuDriverOpen	Opens the GMU-BUS driver.
GmuDriverClose	Closes the GMU-BUS driver.
GmuByteRead	Reads data from the GMU. (Byte)
GmuWordRead	Reads data from the GMU. (Word)
GmuLongRead	Reads data from the GMU. (Long)
GmuByteWrite	Writes data to the GMU. (Byte)
GmuWordWrite	Writes data to the GMU. (Word)
GmuLongWrite	Writes data to the GMU. (Long)
SetGmuCallbackIntA	Registers the GMU IntA Callback.
SetGmuCallbackIntB	Registers the GMU IntB Callback.

3.2 Function Specifications

GetGmuDriverVersion

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

Return value
TRUE: Normal
FALSE: Error

Argument
WORD *pMajor Pointer to version information (Major, 0 to 99)
WORD *pMinor Pointer to version information (Minor, 0 to 99)

Processing performed
Retrieves information on the driver version.
E.g.) BOOL ret = GetGmuDriverVersion (WORD &Major, WORD &Minor);

Note
When the driver version is 1.10,
Major is 1 (in decimal) and;
Minor is 10 (in decimal).

GmuDriverOpen

Call format
BOOL WINAPI GmuDriverOpen (void)

Return value
TRUE: Normal
FALSE: Error

Argument
None

Processing performed
Opens the GMU-BUS driver, enabling driver operations.
E.g.) BOOL ret = GmuDriverOpen ( );

GmuDriverClose

Call format
BOOL WINAPI GmuDriverClose (void)

Return value
TRUE: Normal
FALSE: Error

Argument
None

Processing performed
Closes the GMU-BUS driver, disabling driver operations. However, driver thread operation continues.
E.g.) BOOL ret = GmuDriverClose ( );

GmuByteRead

Call format
BOOL WINAPI GmuByteRead (DWORD Offset, BYTE *pData)

Return value
TRUE: Normal
FALSE: Error

Argument
DWORD Offset	Offset from the beginning of the GMU to the data to be read (in byte units)
BYTE *pData	Pointer to the address where readout will be stored

Processing performed
Reads data from the specified address of the GMU.
E.g.) // Reads data of the byte 16th from the beginning of the GMU.

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

GmuWordRead

Call format
BOOL WINAPI GmuWordRead (DWORD Offset, WORD *pData)

Return value
TRUE: Normal
FALSE: Error

Argument
DWORD Offset	Offset from the beginning of the GMU to the data to be read (in word units)
WORD *pData	Pointer to the address in which the readout will be stored

Processing performed
Reads data from the specified address of the GMU.
E.g.) // Reads data of the word 16th from the beginning of the GMU.

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

GmuLongRead

Call format

BOOL WINAPI GmuLongRead (DWORD Offset, DWORD *pData)

Return value

TRUE: Normal
FALSE: Error

Argument

DWORD Offset	Offset from the beginning of the GMU to the data to be read (in double word units)
DWORD *pData	Pointer to the address where readout will be stored

Processing performed

Reads data from the specified GMU address.
E.g.) // Reads data of the double word 16th from the beginning of the GMU.

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

GmuByteWrite

Call format

BOOL WINAPI GmuByteWrite (DWORD Offset, BYTE Data)

Return value

TRUE: Normal
FALSE: Error

Argument

DWORD Offset	Offset from the beginning of the GMU to the destination of the data to be written (in byte units)
BYTE Data	Data to be written

Processing performed

Writes the specified data into the specified GMU address.
E.g.) // Writes data (aah) to the byte 16th from the beginning of the GMU.

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

GmuWordWrite

Call format
BOOL WINAPI GmuWordWrite (DWORD Offset, WORD Data)

Return value
TRUE: Normal
FALSE: Error

Argument

DWORD Offset	Offset from the beginning of the GMU to the destination of the data to be written (in word units)
WORD Data	Data to be written

Processing performed
Writes the designated data to the specified GMU address.
E.g.) // Writes the data (55aah) into the word 16th from the beginning of the GMU.
BOOL ret = GmuWordWrite (0x10, 0x55aa);

GmuLongWrite

Call format
BOOL WINAPI GmuLongWrite (DWORD Offset, DWORD Data)

Return value
TRUE: Normal
FALSE: Error

Argument

DWORD Offset	Offset from the beginning of the GMU to the destination of the data to be written (in double word units)
DWORD Data	Data to be written

Processing performed
Writes the designated data into the specified GMU address.
E.g.) // Writes data (55aa55aah) to the double word 16th from the beginning of the GMU.
BOOL ret = GmuLongWrite (0x10, 0x55aa55aa);

SetGmuCallbackIntA

Call format
BOOL WINAPI SetGmuCallbackIntA (GMUPROC lpCallbackFunc)

Return value
TRUE: Normal
FALSE: Error

Argument
GMUPROC lpCallbackFunc	Long pointer to the interrupt processing function When set to NULL, cancels the registration of the function selected for interrupt processing.

Processing performed
Registers the function used for interrupt processing when a GMU IntA occurs.

SetGmuCallbackIntB

Call format
BOOL WINAPI SetGmuCallbackIntB (GMUPROC lpCallbackFunc)

Return value
TRUE: Normal
FALSE: Error

Argument
GMUPROC lpCallbackFunc	Long pointer to the interrupt processing function When set to NULL, cancels the registration of the function selected for interrupt processing.

Processing performed
Registers the function used for interrupt processing when a GMU IntB occurs.

4 RAS Driver

This driver provides RAS function control. It supports the watchdog timer, remote reset input, standard signal inputs (two), standard signal output (one), and alarm output (one).
Each of the actions to be performed can also be designated from the control panel. This driver provides functions equivalent to the control panel.

4.1 RAS Driver APIs

<Common functions>

	Setting/display dialog control	API name	Description	H/W resister/registry
Common		GetRasDriverVersion	Retrieves information on the driver version.	
		RasDriverOpen	Opens the RAS driver.	
		RasDriverClose	Closes the RAS driver.	
		BuzzerControl	Controls ON/OFF of the buzzer.	
		AlarmControl	Controls ON/OFF of the ALARMOUT.	RAS IN/OUT
		DoutControl	Controls ON/OFF of the DOUT.	RAS IN/OUT
		SetRasRegister	Writes data into the RAS resister.	
		GetRasRegister	Reads data from the RAS resister.	

<Watchdog timer functions>

	Setting/display dialog control	API name	Description	H/W resister/registry
WDT	Enable	SetWdtEnable	Enables/disables the watchdog timer.	Registry W
		GetWdtEnable	Retrieves the setting for whether the watchdog timer is enabled.	Registry R
	Counter	SetWdtCounter	Sets the counter value.	Registry W
		GetWdtCounter	Retrieves the counter value setting.	Registry R
	Alarm	SetWdtMask	Sets the alarm mask.	Registry W
		GetWdtMask	Retrieves the alarm mask setting.	Registry R
	Buzzer	SetWdtBuzzer	Enables/disables the buzzer activation.	Registry W
		GetWdtBuzzer	Retrieves the buzzer activation setting.	Registry R
	Reset	SetWdtReboot	Enables/disables reboot.	Registry W
		GetWdtReboot	Retrieves the reboot setting.	Registry R
	Popup Enable	SetWdtPopup	Enables/disables the pop-up message.	Registry W
		GetWdtPopup	Retrieves the setting for whether the pop-up message is enabled.	Registry R
	Popup Message	SetWdtMessage	Sets the pop-up message.	Registry W
		GetWdtMssage	Retrieves the pop-up message setting.	Registry R

<Watchdog timer functions>

	Setting/display dialog control	API name	Description	H/W resister/registry
WDT	User application	SetWdtControl	Starts/stops the watchdog timer.	WDT_CNTL
		GetWdtControl	Retrieves the start/stop setting for the watchdog timer.	WDT_CNTL
		GetWdtTimeout	Retrieves the information on occurrence of the watchdog timer time-out.	WDT_STT
		ClearWdtTimeout	Clears the watchdog timer time-out state.	WDT_STT
		RefleshWdtTime	Resets the watchdog timer.	WDT_COUNT

<DIN functions>

	Setting/display dialog control	API name	Description	H/W resister/registry
DIN	Enable	SetDinEnable	Enables/disables DIN port input.	Registry W
		GetDinEnable	Retrieves the setting for whether DIN port input is enabled.	Registry R
	Alarm	SetDinAlarmOut	Enables/disables alarm output.	Registry W
		GetDinAlarmOut	Retrieves the setting for whether alarm output is enabled.	Registry R
	Buzzer	SetDinBuzzer	Enables/disables buzzer output.	Registry W
		GetDinBuzzer	Retrieves the setting for whether buzzer output is enabled.	Registry R
	RAS Output	SetDinDout	Enables/disables DOUT output.	Registry W
		GetDinDout	Retrieves the setting for whether DOUT output is enabled.	Registry R
	Popup Enable	SetDinPopup	Enables/disables pop-up message output.	Registry W
		GetDinPopup	Retrieves the setting for whether pop-up message output is enabled.	Registry R
	Popup Message	SetDinMessage	Sets the pop-up message.	Registry W
		GetDinMessage	Retrieves the pop-up message setting.	Registry R

<Remote reset functions>

	Setting/display dialog control	API name	Description	H/W resister/registry
Remote Reset	Enable	SetRstEnable	Enables/disables remote reset.	Registry W
		GetRstEnable	Retrieves the setting for whether remote reset is enabled.	Registry R

4.2 Details of Function Specifications

<Common functions>

GetRasDriverVersion

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

Return value
TRUE: Normal
FALSE: Error

Argument

WORD *pMajor	Pointer to the version information (Major, 0 to 99)
WORD *pMinor	Pointer to the version information (Minor, 0 to 99)

Processing performed
Retrieves driver version information.
E.g.) BOOL ret = GetRasDriverVersion (WORD &Major, WORD &Minor);

Note
When the driver version is 1.10,
Major is 1 (in decimal) and;
Minor is 10 (in decimal).

RasDriverOpen

Call format
BOOL WINAPI RasDriverOpen (void)

Return value
TRUE: Normal
FALSE: Error

Argument
None

Processing performed
Opens the RAS driver, enabling setting operations.
E.g.) BOOL ret = RasDriverOpen ( );

RasDriverClose

Call format
BOOL WINAPI RasDriverClose (void)

Return value
TRUE: Normal
FALSE: Error

Argument
None

Processing performed
Closes the RAS driver, disabling setting operations. However, driver thread operation continues.
E.g.) BOOL ret = RasDriverClose ( );

BuzzerControl

Call format
BOOL WINAPI BuzzerControl (short length)

Return value
TRUE: Normal
FALSE: Error

Argument
Short length ON (-1): Buzzer is ON.
OFF (0): Buzzer is OFF.
1 to 32767: Buzzer length (in ms)

Processing performed
Controls ON/OFF of the buzzer.
E.g.) // Sounds the buzzer for 100 ms.
BOOL ret = BuzzerControl (100);

AlarmControl

Call format
BOOL WINAPI AlarmControl (BOOL bSwitch)

Return value
TRUE: Normal
FALSE: Error

Argument
BOOL bSwitch TRUE: ALARMOUT output
FALSE: Stops ALARMOUT output

Processing performed
Controls the ALARMOUT port.
E.g.) // Turns the ALARMOUT OFF.
BOOL ret = AlarmControl (FALSE);

DoutControl

Call format
BOOL WINAPI DoutControl (BOOL bSwitch)

Return value
TRUE: Normal
FALSE: Error

Argument
BOOL bSwitch TRUE: DOUT output
FALSE: Stops DOUT output

Processing performed
Controls the DOUT port.
E.g.) // Turns the DOUT ON.
BOOL ret = DoutControl (TRUE);

SetRasRegister

Call format
BOOL WINAPI SetRasRegister (WORD RegNo, BYTE Data)

Return value
TRUE: Normal
FALSE: Error

Argument
WORD RegNo	Register designation *1
BYTE Data	Data to be set in the register

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

Processing performed
Sets the application's designated data to the designated port.
E.g.) // Sets 0 in the RAS_IN_MASK register.

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

GetRasRegister

Call format
BOOL WINAPI GetRasRegister (WORD RegNo, BYTE *pData)

Return value
TRUE: Normal
FALSE: Error

Argument
WORD RegNo	Number of the resister to be read
BYTE *pData	Pointer to the data to be read

Processing performed
Reads the port specified by the application.
E.g.) // Reads the RAS_IN_MASK register.

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

*1: For details on the registers, refer to "4.3 Details of the Registers" in Chapter 3.

<Watchdog timer functions>

SetWdtEnable

Call format
BOOL WINAPI SetWdtEnable (BOOL bEnable)

Return value
TRUE: Normal
FALSE: Error

Argument
BOOL bEnable TRUE: Enables the watchdog timer.
FALSE: Disables the watchdog timer.

Processing performed
Enables/disables the watchdog timer.
E.g.) BOOL ret = SetWdtEnable(TRUE);

GetWdtEnable

Call format
BOOL WINAPI GetWdtEnable (BOOL *pEnable)

Return value
TRUE: Normal
FALSE: Error

Argument
BOOL *pEnable Pointer to the watchdog timer setting

Processing performed
Retrieves the setting for whether or not the watchdog timer is enabled.
E.g.) BOOL Enable;
BOOL ret = GetWdtEnable(&Enable);

SetWdtCounter

Call format
BOOL WINAPI SetWdtCounter (BYTE Counter)

Return value
TRUE: Normal
FALSE: Error

Argument
BYTE Counter Initial counter value (5 to 255) for the watchdog timer (in sec.)

Processing performed
Sets the initial counter value for the watchdog timer. (WDT COUNT)
E.g.) // Sets 10 seconds as the initial counter value for the watchdog timer.
BOOL ret = SetWdtCounter (10);

GetWdtCounter

Call format
BOOL WINAPI GetWdtCounter (BYTE *pCounter)

Return value
TRUE: Normal
FALSE: Error

Argument
BYTE *pCounter Pointer to the initial counter value (in sec.) for the watchdog timer

Processing performed
Retrieves the initial counter value currently set for the watchdog timer. (WDT COUNT)
E.g.) Byte Counter;
BOOL ret = GetWdtCounter (&Counter);

SetWdtMask

Call format
BOOL WINAPI SetWdtMask (BOOL bMask)

Return value
TRUE: Normal
FALSE: Error

Argument
BOOL bMask TRUE: Sets the mask for the watchdog timer alarm.
FALSE: Removes the mask for the watchdog timer alarm.

Processing performed
Sets/removes the mask for the front LED control and the alarm to be output in the event of watchdog timer time-out.
E.g.) // Removes the mask for the ALARM output.

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

GetWdtMask

Call format
BOOL WINAPI GetWdtMask (BOOL *pMask)

Return value
TRUE: Normal
FALSE: Error

Argument
BOOL *pMask Pointer to the mask setting for the watchdog timer

Processing performed
Retrieves the mask setting for the front LED control and the alarm to be output in the event of watchdog timer time-out.
E.g.) // Retrieves the mask setting for the ALARM.

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

SetWdtBuzzer

Call format
BOOL WINAPI SetWdtBuzzer (BOOL bBuzzer)

Return value
TRUE: Normal
FALSE: Error

Argument
BOOL bBuzzer TRUE: Enables the buzzer.
FALSE: Disables the buzzer.

Processing performed
Enables/disables the buzzer to operate in the event of watchdog timer time-out.
E.g.) BOOL ret = SetWdtBuzzer (TRUE);

GetWdtBuzzer

Call format
BOOL WINAPI GetWdtBuzzer (BOOL *pBuzzer)

Return value
TRUE: Normal
FALSE: Error

Argument
BOOL *pBuzzer Pointer to the buzzer activation setting

Processing performed
Retrieves the setting for whether or not the buzzer is enabled to operate in the event of watchdog timer time-out.
E.g.) BOOL Buzzer;
BOOL ret = GetWdtBuzzer (&Buzzer);

SetWdtReboot

Call format
BOOL WINAPI SetWdtReboot (BOOL bReboot)

Return value
TRUE: Normal
FALSE: Error

Argument
BOOL bReboot TRUE: Enables reboot.
FALSE: Disables reboot.

Processing performed
Enables/disables reboot in the event of watchdog timer time-out.
E.g.) BOOL ret = SetWdtReboot (TRUE);

GetWdtReboot

Call format
BOOL WINAPI GetWdtReboot (BOOL *pReboot)

Return value
TRUE: Normal
FALSE: Error

Argument
BOOL *pReboot Pointer to the reboot setting

Processing performed
Retrieves the setting for whether or not reboot is enabled in the event of watchdog timer time-out.
E.g.) BOOL Reboot;
BOOL ret = SetWdtReboot (&Reboot);

SetWdtPopup

Call format
BOOL WINAPI SetWdtPopup (BOOL bPopup)

Return value
TRUE: Normal
FALSE: Error

Argument
BOOL bPopup TRUE: Enables the pop-up message.
FALSE: Disables the pop-up message.

Processing performed
Enables/disables the pop-up message to appear in the event of watchdog timer time-out.
E.g.) BOOL ret = SetWdtPopup (TRUE);

GetWdtPopup

Call format
BOOL WINAPI GetWdtPopup (BOOL *pPopup)

Return value
TRUE: Normal
FALSE: Error

Argument
BOOL *pPopup Pointer to the pop-up message setting

Processing performed
Retrieves the setting for whether or not the pop-up message is enabled to appear in the event of watchdog timer time-out.
E.g.) BOOL Popup;
BOOL ret = GetWdtPopup (&Popup);

SetWdtMessage

Call format
BOOL WINAPI SetWdtMessage (wchar_t *pMessage)

Return value
TRUE: Normal
FALSE: Error

Argument
wchar_t *pMessage Pointer to the pop-up message

Processing performed
Sets the pop-up message which appears in the event of watchdog timer time-out.
E.g.) BOOL ret = SetWdtMessage (L” Watchdog timer time-out”);

GetWdtMessage

Call format
BOOL WINAPI GetWdtMessage (wchar_t *pMessage)

Return value
TRUE: Normal
FALSE: Error

Argument
wchar_t *pMessage Pointer to the pop-up message storage area

Processing performed
Retrieves the pop-up message which appears in the event of watchdog timer time-out.
E.g.) wchar_t msg[32];
BOOL ret = GetWdtMessage (msg);

SetWdtControl

Call format
BOOL WINAPI SetWdtControl (BOOL bCont)

Return value
TRUE: Normal
FALSE: Error

Argument
BOOL bCont TRUE: Starts the watchdog timer.
FALSE: Stops the watchdog timer.

Processing performed
Starts/stops the watchdog timer. Start, stop and refresh of the watchdog timer are not triggered via the driver. They are performed via the application's timing of the driver function call. As soon as the driver detects the time-out, it takes the specified action(s).
E.g.) BOOL ret = SetWdtControl (TRUE);

GetWdtControl

Call format
BOOL WINAPI GetWdtControl (BOOL *bCont)

Return value
TRUE: Normal
FALSE: Error

Argument
BOOL *bCont	Pointer to the start/stop setting of the watchdog timer TRUE: The watchdog timer has been started. FALSE: The watchdog timer has been stopped.

Processing performed
Retrieves the start/stop setting of the watchdog timer.
E.g.) BOOL Cont;
BOOL ret = GetWdtControl (&Cont);

GetWdtTimeout

Call format
BOOL WINAPI GetWdtTimeout (BOOL *pTOut)

Return value
TRUE: Normal
FALSE: Error

Argument
BOOL *pTOut	Pointer to the watchdog timer time-out information TRUE: The watchdog timer time-out has occurred. FALSE: The watchdog timer time-out has not occurred.

Processing performed
After reading the WDT_STT, this function retrieves the information on whether or not the watchdog timer time-out has occurred.
E.g.) BOOL TOut;
BOOL ret = GetWdtTimeOut (&TOut);

ClearWdtTimeout

Call format
BOOL WINAPI ClearWdtTimeout (void);

Return value
TRUE: Normal
FALSE: Error

Argument
Nothing

Processing performed
Clears the watchdog timer time-out state. (WDT_STT)
E.g.) // Clears the watchdog timer time-out state.
BOOL ret = CleanWdtTimeout ( );

RefleshWdtTime

Call format
BOOL WINAPI RefleshWdtTime (void);

Return value
TRUE: Normal
FALSE: Error

Argument
Nothing

Processing performed
Resets the watchdog timer counter (WDT_STT) to the initial counter value preset by the SetWdtCounter.
E.g.) // Refreshes the watchdog timer.
BOOL ret = RefleshWdtTime ( );

<DIN functions>

SetDinEnable

Call format
BOOL WINAPI SetDinEnable (WORD inp, BOOL bEnable)

Return value
TRUE: Normal
FALSE: Error

Argument

WORD inp	0 or 1: DIN port number
BOOL bEnable	TRUE: Enables DIN port input. FALSE: Disables DIN port input.

Processing performed
Enables/disables DIN0/DIN1 port input.
E.g.) BOOL ret = SetDinEnable (0, TRUE);

GetDinEnable

Call format
BOOL WINAPI GetDinEnable (WORD inp, BOOL *pEnable)

Return value
TRUE: Normal
FALSE: Error

Argument

WORD inp	0 or 1: DIN port number
BOOL *pEnable	Pointer to the DIN port setting

Processing performed
Retrieves the setting for whether or not DIN0/DIN1 port input is enabled.
E.g.) // Retrieves the DIN port setting.

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

SetDinAlarmOut

Call format
BOOL WINAPI SetDinAlarmOut (WORD inp, BOOL bAlm)

Return value
TRUE: Normal
FALSE: Error

Argument

WORD inp	0 or 1: DIN port number
BOOL bAlm	TRUE: Enables alarm output. FALSE: Disables alarm output.

Processing performed
Enables/disables alarm output.
E.g.) BOOL ret = SetDinAlarmOut (0, TRUE);

GetDinAlarmOut

Call format
BOOL WINAPI GetDinAlarmOut (WORD inp, BOOL *pAlm)

Return value
TRUE: Normal
FALSE: Error

Argument

WORD inp	0 or 1: DIN port number
BOOL *pAlm	Pointer to the setting for whether or not alarm output is enabled while the DIN0/DIN1 port is active

Processing performed
Retrieves the setting for whether or not alarm output is enabled while the DIN0/DIN1 port is active.
E.g.) // Retrieves the setting for alarm output through the DIN port.

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

SetDinBuzzer

Call format

BOOL WINAPI SetDinBuzzer (WORD inp, BOOL bBuzzer);

Return value

TRUE: Normal
FALSE: Error

Argument

WORD inp	0 or 1: DIN port number
BOOL bBuzzer	Enables/disables buzzer output.

Processing performed

Enables/disables buzzer output while the DIN0/DIN1 port is active.
E.g.) // Disables buzzer output through the DIN port.

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

GetDinBuzzer

Call format

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

Return value

TRUE: Normal
FALSE: Error

Argument

WORD inp	0 or 1: DIN port number
BOOL *pBuzzer	Pointer to the setting for whether or not buzzer output is enabled.

Processing performed

Retrieves the setting for whether or not buzzer output is enabled while the DIN0/DIN1 port is active.
E.g.) // Retrieves the setting for buzzer output through the DIN port.

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

SetDinDout

Call format
BOOL WINAPI SetDinDout (WORD inp, BOOL bRasout);

Return value
TRUE: Normal
FALSE: Error

Argument
WORD inp	0 or 1: DIN port number
BOOL bRasout	Enables/disables DOUT output.

Processing performed
Enables/disables DOUT output while the DIN0/DIN1 port is active.
E.g.) // Disables DOUT output.

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

GetDinDout

Call format
BOOL WINAPI GetDinDout (WORD inp, BOOL *pRasOut);

Return value
TRUE: Normal
FALSE: Error

Argument
WORD inp	0 or 1: DIN port number
BOOL *pRasOut	Pointer to the setting for whether or not DOUT output is enabled.

Processing performed
Retrieves the setting for whether or not DOUT output is enabled while the DIN0/DIN1 port is active.
E.g.) // Retrieves the DOUT output setting.

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

SetDinPopup

Call format
BOOL WINAPI SetDinPopup (WORD inp, BOOL bPopup);

Return value
TRUE: Normal
FALSE: Error

Argument
WORD inp	0 or 1: DIN port number
BOOL bPopup	Enables/disables pop-up message output.

Processing performed
Enables/disables pop-up message output while the DIN0/DIN1 port is active.
E.g.) // Enables pop-up message output.
```
BOOL ret = SetDinPopup (0, TRUE);
```

GetDinPopup

Call format
BOOL WINAPI GetDinPopup (WORD inp, BOOL *pPopup);

Return value
TRUE: Normal
FALSE: Error

Argument
WORD inp	0 or 1: DIN port number
BOOL *pPopup	Pointer to the setting for whether or not pop-up message output is enabled.

Processing performed
Retrieves the setting for whether or not pop-up message output is enabled while the DIN0/DIN1 port is active.
E.g.) // Retrieves the pop-up message output setting.
```
BOOL Popup;
BOOL ret = GetDinPopup (0, &Popup);
```

SetDinMessage

Call format
BOOL WINAPI SetDinMessage (WORD inp, wchar_t *pMessage);

Return value
TRUE: Normal
FALSE: Error

Argument
WORD inp	0 or 1: DIN port number
wchar_t *pMessage	Pointer to the pop-up message.

Processing performed
Sets the pop-up message while the DIN0/DIN1 port is active.
E.g.) BOOL ret = SetDinMessage (1, L" Universal Input Active");

GetDinMessage

Call format
BOOL WINAPI GetDinMessage (WORD inp, wchar_t *pMessage);

Return value
TRUE: Normal
FALSE: Error

Argument
WORD inp	0 or 1: DIN port number
wchar_t *pMessage	Pointer to the pop-up message

Processing performed
Retrieves the pop-up message while the DIN0/DIN1 port is active.
E.g.) wchar_t msg [32];
BOOL ret = GetDinMessage (1, msg);

<Remote reset functions>

SetRstEnable

Call format
BOOL WINAPI SetRstEnable (BOOL bEnable)

Return value
TRUE: Normal
FALSE: Error

Argument
BOOL bEnable TRUE: Enables remote reset input.
FALSE: Disables remote reset input.

Processing performed
Enables/disables remote reset input.
E.g.) BOOL ret = SetRstEnable (TRUE);

GetRstEnable

Call format
BOOL WINAPI GetUniEnable (BOOL *pEnable)

Return value
TRUE: Normal
FALSE: Error

Argument
BOOL *pEnable Pointer to the setting for whether or not remote reset input is enabled.

Processing performed
Retrieves the setting for whether or not remote reset input is enabled.
E.g.) // Retrieves the remote reset input setting.
BOOL Enable;
BOOL ret = GetRstEnable (&Enable);

4.3 Register Details

RAS_IN_MASK

RAS IN MASK register
Read/Write

Register	B7	B6	B5	B4	B3	B2	B1	B0	Note
RAS IN MASK	--	--	RMT_ RST_ MASK	--	--	--	IN1_ MASK	IN0_ MASK	8th to 15th bits invalid "0".

RMT_RST_MASK Remote reset mask
1: External remote reset input is enabled. (Default)
0: External remote reset input is disabled.

IN1_MASK DIN1 input mask
1: DIN1 input is enabled.
0: DIN1 input is disabled. (Default)

IN0_MASK DIN0 input mask
1: DIN0 input is enabled.
0: DIN0 input is disabled. (Default)

RAS_IN_OUT

RAS IN/OUT register
Read/Write

Register	B7	B6	B5	B4	B3	B2	B1	B0	Note
RAS IN/OUT	LEDR	LEDG	--		RAS_ ALM	RAS_ OUT0	RAS_ IN1	RAS_ IN0	8th to 15th bits invalid "0".

LEDR Front red LED control (Independent functioning of Alarm Output)
When the RAS_ALM register is 0 and the WDT_ALM_MSK register is 1 (independent of the WDT_STT register), the control is enabled.
When the RAS_ALM register is 1 or the WDT_ALM_MSK register is 0, and the WDT_STT register is 1 (WDT error status), the red LED is ON as priority is given to the latter status.
1: The front red LED is ON. (Default)
0: The front red LED is OFF.

LEDG Front green LED control
1: The front green LED is ON.
0: The front green LED is OFF. (Default)

RAS_ALM Control of Alarm Output and front bezel LED
When the WDT_ALM_MASK register is 0 and the WDT_STT register is 1, Alarm Output is ON and the orange LED is ON (the red LED is also ON), regardless of the RAS_ALM register status.
1: Alarm Output is ON.
Front LED: The orange LED is ON. (The red LED is ON.)
2: Alarm Output is OFF. (Default)
Front LED: The green LED is ON. (The red LED is OFF.)

RAS_OUT0 DOUT control
1: DOUT0 is ON.
2: DOUT0 is OFF. (Default)

RAS_IN1 DIN1 control
When DIN1 is not masked, the DIN1 status is retained until 1 is written in this bit.
DIN1 input is accepted as soon as the signal changes from 0 to 1.
1: There is a DIN1 input. (Read) / Status clear (Write)
0: There is no DIN1 input. (Read) (Default)

RAS_IN0 DIN0 control
When DIN0 is not masked, the DIN0 status is retained until 1 is written in this bit.
DIN0 input is accepted as soon as the signal changes from 0 to 1.
1: There is a DIN0 input. (Read) / Status clear (Write)
0: There is no DIN0 input. (Read) (Default)

WDT_CR

Watchdog timer control register
Read/Write

Register	B7	B6	B5	B4	B3	B2	B1	B0	Note
WDT CR	WDT_ ALD	--	--	WDT_ STT	WDT_ ALM_ MASK	--	--	WDT_ CNTL	8th to 15th bits invalid "0".

WDT_ALD Watchdog timer automatic load
The data in the counter resister is loaded into the watchdog timer as the initial value.
1: The initial value for the counter is loaded. (Write)
0: This bit remains set to 0 at the time of reading. (Read) (Default)

WDT_STT Watchdog timer time-out status
1: The watchdog timer time-out has occurred. (Read)
The time-out state is cleared. (Write)
0: The watchdog timer time-out has not occurred. (Read) (Default)

WDT_ALM_MASK Watchdog timer alarm mask
1: The mask is enabled. (Default)
0: The alarm is output to the RAS connector in the event of time-out.

WDT_CNTL Watchdog timer counter control
The operation of the watchdog timer is supported by repeated start and stop before the specified time-out limit passes. If the counter is not stopped before the specified time-out limit passes, Alarm/Lamp Out is output and 1 is set in the WDT_STT.
1: The counter is started.
2: The counter is stopped./The time-out status is cleared./The initial value is loaded. (Default)

WDT_COUNT

Watchdog timer counter register
Read/Write

Register	B7	B6	B5	B4	B3	B2	B1	B0	Note
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)	8th to 15th bits invalid "0".

WDT_COUNT 0 to 7 Initial value for the timer. One count is approximately equal to one second.

(Default: FF)

Sets the initial value to be loaded into the watchdog timer.

5 Touch Panel Driver

The function of this driver is to enable/disable buzzer activation and input when the touch panel is touched.

5.1 Touch Panel Driver APIs

API name	Description
GetTouchDriverVersion	Retrieves information on the driver version.
TouchDriverOpen	Opens the touch panel driver.
TouchDriverClose	Closes the touch panel driver.
SetTouchClickBuzzer	Enables/disables the buzzer when the touch panel is touched.
GetTouchClickBuzzer	Retrieves the setting for buzzer activation when the touch panel is touched.
SetTouchInput	Enables/disables input when the touch panel is touched.
GetTouchInput	Retrieves the setting for input when the touch panel is touched.

5.2 Function Specifications

GetTouchDriverVersion

Call format
BOOL WINAPI GetTouchDriverVersion (WORD *pMajor, WORD *pMinor)

Return value
TRUE: Normal
FALSE: Error

Argument
WORD *pMajor Pointer to the version information (Major, 0 to 99)
WORD *pMinor Pointer to the version information (Minor, 0 to 99)

Processing performed
Retrieves information on the driver version.
E.g.) BOOL ret = GetTouchDriverVersion (WORD &Major, WORD &Minor);

Note
When the driver version is 1.10,
Major is 1 (in decimal) and;
Minor is 10 (in decimal).

TouchDriverOpen

Call format
BOOL WINAPI TouchDriverOpen (void)

Return value
TRUE: Normal
FALSE: Error

Argument
None

Processing performed
Opens the touch panel driver, enabling setting operations.
E.g.) BOOL ret = TouchDriverOpen ( );

TouchDriverClose

Call format
BOOL WINAPI TouchDriverClose (void)

Return value
TRUE: Normal
FALSE: Error

Argument
None

Processing performed
Closes the touch panel driver, disabling setting operations. However, the thread operation of the driver continues.
E.g.) BOOL ret = TouchDriverClose ( );

SetTouchClickBuzzer (BOOL bState)

Call format
BOOL WINAPI SetTouchClickBuzzer (BOOL bState)

Return value
TRUE: Normal
FALSE: Error

Argument
BOOL bState TRUE: Turns the buzzer ON.
FALSE: Turns the buzzer OFF.

Processing performed
Enables/disables the buzzer to activate when the touch panel is clicked.
E.g.) // Enables the buzzer to operate when the touch panel is clicked.

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

GetTouchClickBuzzer (BOOL *bState)

Call format
BOOL WINAPI GetTouchClickBuzzer (BOOL *bState)

Return value
TRUE: Normal
FALSE: Error

Argument
BOOL *bState Pointer to the buzzer activation setting when the touch panel is clicked

Processing performed
Retrieves the setting for whether or not the buzzer is enabled when the touch panel is clicked.
E.g.) // Retrieves the buzzer activation setting when the touch panel is clicked.

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

SetTouchInput (BOOL bState)

Call format
BOOL WINAPI SetTouchInput (BOOL bState)

Return value
TRUE: Normal
FALSE: Error

Argument
BOOL bState TRUE: Enables input.
FALSE: Disables input.

Processing performed
Enables/disables input when the touch panel is touched.
E.g.) // Enables input when the touch panel is touched.
BOOL ret = SetTouchInput (TRUE);

GetTouchInput (BOOL *bState)

Call format
BOOL WINAPI GetTouchInput (BOOL *bState)

Return value
TRUE: Normal
FALSE: Error

Argument
BOOL *bState Pointer to the setting for input when the touch panel is touched.

Processing performed
Retrieves the setting for whether or not input is enabled when the touch panel is touched.
E.g.) // Retrieves the setting for input when the touch panel is touched.
BOOL State;
BOOL ret = GetTouchInput (&State);

Index

Pro-face

Global Head Office
Digital Electronics Corporation
8-2-52 Nanko-higashi,
Suminoe-ku, Osaka 559-0031 JAPAN
Tel:+81 (0)6 6613 3116 Fax:+81(0)6 6613 5888
http://www.pro-face.com support@digital.co.jp

South Korea
Pro-face Korea Co., Ltd.
Room #701, Jaeyoung Building
678-10 Deungchon-dong
Kandgseo-Ku, Seoul 157-030 KOREA
Tel: +82 (0)2 658 6835 Fax: +82 (0)2 3664 6839
http://www.proface.co.kr proface@proface.co.kr

Taiwan
Pro-face Taiwan Co., Ltd.
2F, No.69, Fushing North Road
Taipei 105 TAIWAN R. O. C.
Tel: +886 (0)2 2772 5208 Fax: +886 (0)2 8773 7892
http://www.proface.com.tw proface@proface.com.tw

North/South America
Pro-face America, Inc.
2190-E Gladstone Court
Glendale Heights, IL 60139 U.S.A.
Tel: +1 630 351 1101 Fax: +1 630 351 1102
http://www.profaceamerica.com
support@profaceamerica.com

European Head Office
Pro-face HMI B.V.
Amsteldijk 166
1079 LH Amsterdam THE NETHERLANDS
Tel: +31 (0)20 6464 134 Fax: +31(0)20 6464 358
http://www.proface.com support@proface.com

Italy
Pro-face HMI B.V. Italy
Via Carcano 44
20033 Desio (MI)
ITALY
Tel: +39 0362 33 71 63 Fax: +39 0362 30 77 25
supporto_tecnico@proface.com

Germany
Pro-face Deutschland Gmbh
Albertus-Magnus-Straße 11
42719 Solingen
GERMANY
Tel: +49 (0)212 258 260 Fax: +49 (0)212 258 2640
http://www.pro-face.de support@pro-face.de