



# WB-1A

API Manual

Version 1.02

2010/07/26



Copyright © DOCOMO Systems, Inc. All Rights Reserved.

# TABLE OF CONTENTS

<b>1.</b>	<b>DIGITAL INPUT/OUTPUT</b>	<b>5</b>
1.1	OPEN	5
1.2	CLOSE	5
1.3	READ	6
1.4	WRITE	6
1.5	SAMPLE PROGRAM	6
<b>2.</b>	<b>LED</b>	<b>8</b>
2.1	OPEN	8
2.2	CLOSE	9
2.3	READ	9
2.4	WRITE	10
2.5	SAMPLE PROGRAM	10
<b>3.</b>	<b>BACKLIGHT</b>	<b>11</b>
3.1	REGISTRY	11
3.2	API CODE	11
<b>4.</b>	<b>POWER OFF</b>	<b>12</b>
4.1	INCLUDE FILES	12
4.2	SAMPLE PROGRAM	12
<b>5.</b>	<b>WATCHDOG TIMER</b>	<b>13</b>
5.1	SOFTWARE STRUCTURE	13
5.2	WATCHDOG TIMER DRIVER	14
5.3	OPEN	14
5.4	CLOSE	15
5.5	READ	15
5.6	WRITE	15
5.7	SAMPLE PROGRAM	16
<b>6.</b>	<b>BATTERY</b>	<b>17</b>
6.1	CHARGE LED COLOR	17
6.2	LONG BATTERYDRVRGETLEVELS	17
6.3	GETSYSTEMPOWERSTATUSEx2	17
6.4	SAMPLE PROGRAM	19
<b>7.</b>	<b>CAMERA</b>	<b>20</b>
7.1	API LIST	20
7.2	API EXPLANATION	20
7.2.1	<i>Create and Close the Camera Handle</i>	20
7.2.2	<i>Switch resolutions in camera driver</i>	20
7.2.3	<i>Retrieve the captured image data from camera driver</i>	20
7.2.4	<i>Preview function</i>	21
7.2.5	<i>Get the current setting of the camera chipset</i>	21
7.2.6	<i>Save the current settings on system registry</i>	22
7.2.7	<i>Setup the White Balance value to device driver</i>	22
7.2.8	<i>Setup the Exposure value to device driver</i>	23
7.2.9	<i>Setup the AGC value to device driver</i>	23
7.2.10	<i>Setup the Night Mode to device driver</i>	23
<b>8.</b>	<b>REAL TIME CLOCK</b>	<b>24</b>
8.1	SNTP	24
8.2	DAILY WAKEUP	24
8.2.1	<i>Registry</i>	24
8.2.2	<i>Sample Program</i>	25

# LIST OF FIGURES

Figure 5-2 Watchdog Timer Software Structure ..... 13

## Revision History

DATE	Change Description	REVISION
2008/01/20	Draft Release	V 0.01
2008/03/11	Camera API update	V 1.00
2008/05/20	<ol style="list-style-type: none"><li>1. Update the specification of charge LED behavior (6.1)</li><li>2. Modify some statements in chapter 7.</li><li>3. Change the default setting of reset period of WDT from 1 to 0 in section 5.2.</li></ol>	V 1.01
2008/07/01	Add comment in chapter 6.1	V 1.01b
2010/07/26	Amend the chapter 8 .1	V 1.02

# 1. Digital Input/Output

This driver will provide a standard stream driver interface for the calling software.

In order to reduce the relationship between driver and application, we will provide the stream driver for application to control the Digital Input and Digital Out.

The data format is

	BIT7	BIT6	BIT5	BIT4	BIT3	BIT2	BIT1	BIT0
DI	X	X	X	PIC_GPI5	PIC_GPI4	PIC_GPI3	PIC_GPI2	PIC_GPI1
DO	X	X	X	PIC_GPO5	PIC_GPO4	PIC_GPO3	PIC_GPO2	PIC_GPO1

Note: X means don't care.

## 1.1 Open

Opens a DI as user desired. This device only can be open once at any given time.

```
HANDLE CreateFile( LPCTSTR lpFileName,  
                  DWORD dwDesiredAccess,  
                  DWORD dwShareMode,  
                  LPSECURITY_ATTRIBUTES lpSecurityAttributes,  
                  DWORD dwCreationDisposition,  
                  DWORD dwFlagsAndAttributes,  
                  HANDLE hTemplateFile)
```

### Parameter Description

<i>lpFileName</i>	This parameter MUST be "DIO1:"
<i>dwDesiredAccess</i>	GENERIC_READ   GENERIC_WRITE
<i>dwShareMode</i>	0
<i>lpSecurityAttributes</i>	0
<i>dwCreationDisposition</i>	OPEN_EXISTING
<i>dwFlagsAndAttributes</i>	0
<i>hTemplateFileClose</i>	NULL

### Return value description

If successful, the value is the handle of the device. Otherwise, equal to INVALID\_HANDLE\_VALUE.

## 1.2 Close

Closes a device be opened, after this, the device could accept opened.

```
BOOL CloseHandle( HANDLE hObject );
```

### Parameter Description

<i>hObject</i>	handle which get from CreateFile
----------------	----------------------------------

### Return value description

None

## 1.3 Read

Read data from DI.

```
BOOL ReadFile( HANDLE hFile,  
              LPVOID lpBuffer,  
              DWORD nNumberOfBytesToRead,  
              LPDWORD lpNumberOfBytesRead,  
              LPOVERLAPPED lpOverlapped )
```

### Parameter Description

<i>hFile</i>	Handle which get from CreateFile
<i>lpBuffer</i>	Buffer which contain read data
<i>nNumberOfBytesToRead</i>	Specify the max data could be read
<i>lpNumberOfBytesRead</i>	Actually read data member count
<i>lpOverlapped</i>	0

### Return value description

None

## 1.4 Write

Write data to DO.

```
BOOL WriteFile( HANDLE hFile,  
               LPCVOID lpBuffer,  
               DWORD nNumberOfBytesToWrite,  
               LPDWORD lpNumberOfBytesWritten,  
               LPOVERLAPPED lpOverlapped )
```

### Parameter Description

<i>hFile</i>	Handle which get from CreateFile
<i>lpBuffer</i>	Buffer which contain writing data
<i>nNumberOfBytesToWrite</i>	Specify the max data could be written
<i>lpNumberOfBytesWritten</i>	Actually written data member count
<i>lpOverlapped</i>	0

### Return value description

Nonzero indicates success. Zero indicates failure. To get extended error information, call GetLastError function.

## 1.5 Sample Program

```
HANDLE hDIO;  
BOOL rt;  
DWORD NumberOfBytesWritten;  
DWORD NumberOfBytesRead;  
BYTE DI=0,DO=0;
```

```
hDIO=CreateFile( L"DIO1:",  
                GENERIC_READ | GENERIC_WRITE,  
                0,  
                0,  
                OPEN_EXISTING,  
                0,  
                NULL);
```

```
rt=ReadFile(hDIO,&DI,1,&NumberOfBytesRead,0);  
rt=WriteFile(hDIO,&DO,1,&NumberOfBytesWritten,0);  
rt=CloseHandle(hDIO);
```

## 2. LED

This driver will provide stream interface with IOCTL code to control LED display function.

The data format is 3-bytes data to contain the ID and color, they are:

**1st byte: Red LED ID, the format is**

BIT0: LED1: 1=ON, 0=OFF  
BIT1: LED2: 1=ON, 0=OFF  
BIT2: LED3: 1=ON, 0=OFF  
BIT3: LED4: 1=ON, 0=OFF  
BIT4: LED5: 1=ON, 0=OFF  
BIT5: LED6: 1=ON, 0=OFF  
BIT6: LED7: 1=ON, 0=OFF  
BIT7: LED8: 1=ON, 0=OFF

**2nd byte: Green LED ID, the format is**

BIT0: LED1: 1=ON, 0=OFF  
BIT1: LED2: 1=ON, 0=OFF  
BIT2: LED3: 1=ON, 0=OFF  
BIT3: LED4: 1=ON, 0=OFF  
BIT4: LED5: 1=ON, 0=OFF  
BIT5: LED6: 1=ON, 0=OFF  
BIT6: LED7: 1=ON, 0=OFF  
BIT7: LED8: 1=ON, 0=OFF

**3rd byte: Blue LED ID, the format is**

BIT0: LED1: 1=ON, 0=OFF  
BIT1: LED2: 1=ON, 0=OFF  
BIT2: LED3: 1=ON, 0=OFF  
BIT3: LED4: 1=ON, 0=OFF  
BIT4: LED5: 1=ON, 0=OFF  
BIT5: LED6: 1=ON, 0=OFF  
BIT6: LED7: 1=ON, 0=OFF  
BIT7: LED8: 1=ON, 0=OFF

Examples:

Light on LED 3 with red and green color, then, write data = 0x04-0x04-0x00

Light on LED 6 with red and blue color, write data = 0x20-0x00-0x20

Light OFF all, write data = 0x00-0x00-0x00

Light ON all, write data = 0xFF-0xFF-0xFF

### 2.1 Open

Opens a LED as user desired. This device only can be open once at any given time.

**HANDLE CreateFile( LPCTSTR lpFileName,  
                  DWORD dwDesiredAccess,  
                  DWORD dwShareMode,  
                  LPSECURITY\_ATTRIBUTES lpSecurityAttributes,**

**DWORD dwCreationDisposition,  
DWORD dwFlagsAndAttributes,  
HANDLE hTemplateFile)**

**Parameter Description**

<i>lpFileName</i>	this parameter MUST be "LED1:"
<i>dwDesiredAccess</i>	GENERIC_READ   GENERIC_WRITE
<i>dwShareMode</i>	0
<i>lpSecurityAttributes</i>	0
<i>dwCreationDisposition</i>	OPEN_EXISTING
<i>dwFlagsAndAttributes</i>	0
<i>hTemplateFileClose</i>	NULL

**Return value description**

If successful, the value is the handle of the device. Otherwise, equal to INVALID\_HANDLE\_VALUE

## 2.2 Close

Closes a device be opened, after this, the device could accept opened.

**BOOL CloseHandle( HANDLE hObject );**

**Parameter Description**

<i>hObject</i>	handle which get from CreateFile
----------------	----------------------------------

**Return value description**

None

## 2.3 Read

Read data from LED shadow status.

**BOOL ReadFile( HANDLE hFile,  
LPVOID lpBuffer,  
DWORD nNumberOfBytesToRead,  
LPDWORD lpNumberOfBytesRead,  
LPOVERLAPPED lpOverlapped )**

**Parameter Description**

<i>hFile</i>	Handle which get from CreateFile
<i>lpBuffer</i>	Buffer which contain read data
<i>nNumberOfBytesToRead</i>	Specify the max data could be read
<i>lpNumberOfBytesRead</i>	Actually read data member count
<i>lpOverlapped</i>	0

**Return value description**

None

## 2.4 Write

Write data to LED buffer.

```
BOOL WriteFile( HANDLE hFile,  
                LPCVOID lpBuffer,  
                DWORD nNumberOfBytesToWrite,  
                LPDWORD lpNumberOfBytesWritten,  
                LPOVERLAPPED lpOverlapped )
```

### Parameter Description

<i><b>hFile</b></i>	Handle which get from CreateFile
<i><b>lpBuffer</b></i>	Buffer which contain writing data
<i><b>nNumberOfBytesToWrite</b></i>	Specify the max data could be written
<i><b>lpNumberOfBytesWritten</b></i>	Actually written data member count
<i><b>lpOverlapped</b></i>	0

### Return value description

Nonzero indicates success. Zero indicates failure. To get extended error information, call GetLastError function.

## 2.5 Sample Program

```
HANDLE hLED;  
BOOL rt;  
BYTE LEDDATA[5];  
DWORD NumberOfBytesWritten;
```

```
LEDDATA[0]=RData;  
LEDDATA[1]=GData;  
LEDDATA[2]=BData;
```

```
hLED=CreateFile( L"LED1:",GENERIC_READ | GENERIC_WRITE,0,0,OPEN_EXISTING,0,NULL);  
rt=WriteFile(hLED,&LEDDATA,3,&NumberOfBytesWritten,0);  
rt=CloseHandle(hLED);
```

## 3. Backlight

A driver is required to adjust the intensity of the back light from 0 being off, 1 being on with minimum brightness, to 255 being full brightness.

An applet will be provided for control panel utility to control the backlight of LCD panel; also it provides the registry to allow application to control the backlight with the registry modification.

### 3.1 Registry

[HKEY\_CURRENT\_USER\ControlPanel\Backlight]

"BattBacklightLevel"=dword:EF ; Backlight level settings. 0xFF = Full On

"ACBacklightLevel"=dword:EF ; Backlight level settings. 0xFF = Full On

Whenever a change occurs in the timeout values, the application signals an event named **BackLightChangeEvent**. This event eliminates the need for your display driver to constantly read the registry to monitor the timeout values for changes.

### 3.2 API code

```
HANDLE hEvent = CreateEvent(NULL, FALSE, FALSE, TEXT("BackLightChangeEvent"));
SetEvent(hEvent); // Inform backlight driver to change the backlight level
```

## 4. Power Off

This chapter describes a process that user can power system off via software procedure. PIC controls the main power of system

There are two major steps in this procedure:

1. Send "Power Off" command to PIC.
2. Configure system into suspend mode.

### 4.1 Include Files

```
mpicomm.h  
Pm.h
```

### 4.2 Sample Program

```
HANDLE hMpi;  
HANDLE evMpiVer;  
DWORD waitStatus;  
  
// 1. Send "PowerOff" command to PIC  
evMpiVer = CreateEvent(NULL, FALSE, FALSE, KMpiPwrEventName);  
hMpi = CreateFile(KMpiPortName,  
                GENERIC_READ | GENERIC_WRITE,  
                0,  
                NULL,  
                OPEN_EXISTING,  
                0,  
                NULL);  
  
DeviceIoControl(hMpi, IOCTL_MPI_SetPWR_CMD, NULL, 0, NULL, 0, NULL, NULL);  
waitStatus = WaitForSingleObject(evMpiVer, 10000);  
if(waitStatus == WAIT_TIMEOUT){  
    SetDlgItemText(IDC_Status, L"Power Off Fail");  
}  
else{  
    // 2. Transmit command success, set system to suspend mode  
    SetSystemPowerState(NULL, POWER_STATE_SUSPEND, POWER_FORCE);  
    SetDlgItemText(IDC_Status, L"Power Off Ok");  
}  
  
CloseHandle(hMpi);  
CloseHandle(evMpiVer);
```

## 5. Watchdog Timer

In this system, PIC as a watchdog timer of CPU. If CPU cannot reset this watchdog timer in a specific period, PIC will reset whole system.

This driver will provide a standard stream driver interface for the calling software.

### 5.1 Software Structure

For watchdog timer function, there are some mechanisms have to implement.

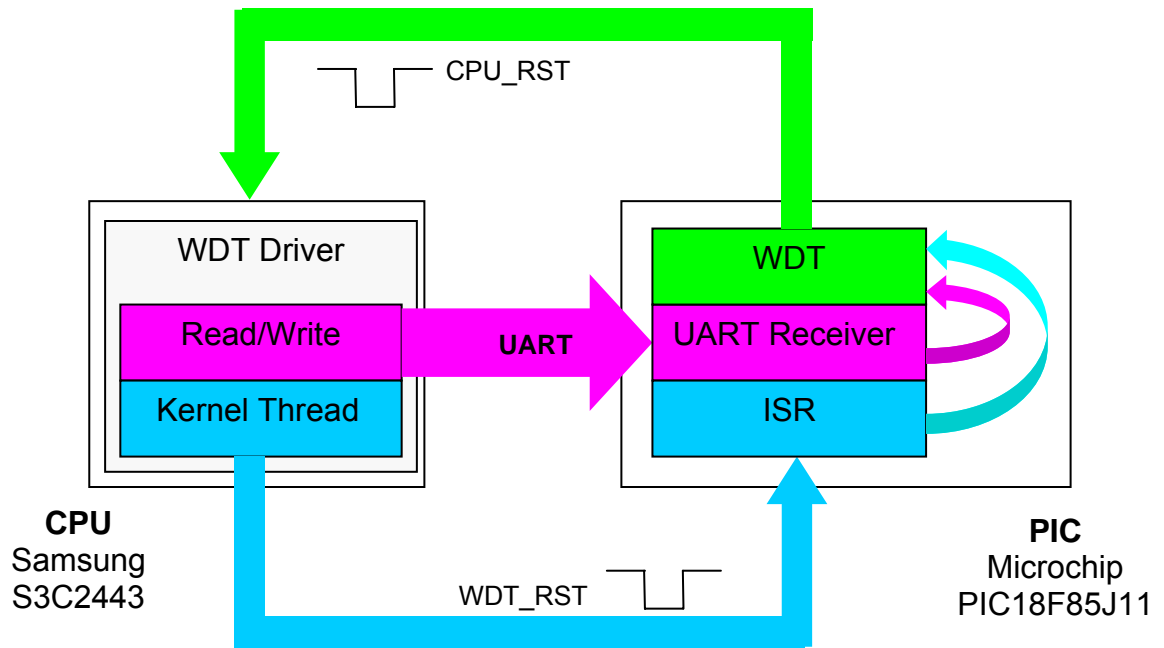


Figure 5-1 Watchdog Timer Software Structure

## 5.2 Watchdog Timer Driver

In order to reduce the relationship between driver and application, we will provide the stream driver for application to control the Digital Input and Digital Out.

**Data type:**

DWORD

**Data format:**

4 <sup>th</sup> Byte	3 <sup>rd</sup> Byte	2 <sup>nd</sup> Byte	1 <sup>st</sup> Byte
Reset Period		Watchdog Timer Interval	

**Reset Period :**

- 0: 1/4 interval, reset watchdog timer in 1/4 interval
- 1: 1/2 interval, reset watchdog timer in 1/2 interval
- 2: 3/4 interval, reset watchdog timer in 3/4 interval

**Watchdog Timer Interval:**

Value: 0~65535

Unit: 500ms

**Default Setting:**

Reset Period: 0 (1/4)

Watchdog Timer Interval: 600 (600x500ms)

## 5.3 Open

```
HANDLE CreateFile( LPCTSTR lpFileName,
                  DWORD dwDesiredAccess,
                  DWORD dwShareMode,
                  LPSECURITY_ATTRIBUTES lpSecurityAttributes,
                  DWORD dwCreationDisposition,
                  DWORD dwFlagsAndAttributes,
                  HANDLE hTemplateFile)
```

**Parameter Description**

<i>lpFileName</i>	this parameter MUST be "WDT1:",
<i>dwDesiredAccess</i>	GENERIC_READ   GENERIC_WRITE
<i>dwShareMode</i>	0
<i>lpSecurityAttributes</i>	0
<i>dwCreationDisposition</i>	OPEN_EXISTING
<i>dwFlagsAndAttributes</i>	0
<i>hTemplateFileClose</i>	NULL

**Return value description**

If successful, the value is the handle of the device. Otherwise, equal to INVALID\_HANDLE\_VALUE

## 5.4 Close

Closes a device be opened, after this, the device could accept opened.

**BOOL CloseHandle( HANDLE hObject );**

### Parameter Description

*hObject* handle which get from CreateFile

### Return value description

None

## 5.5 Read

Read data from WDT shadow status.

**BOOL ReadFile( HANDLE hFile,  
LPVOID lpBuffer,  
DWORD nNumberOfBytesToRead,  
LPDWORD lpNumberOfBytesRead,  
LPOVERLAPPED lpOverlapped )**

### Parameter Description

<i>hFile</i>	Handle which get from CreateFile
<i>lpBuffer</i>	Buffer which contain read data
<i>nNumberOfBytesToRead</i>	Specify the max data could be read
<i>lpNumberOfBytesRead</i>	Actually read data member count
<i>lpOverlapped</i>	0

### Return value description

None

## 5.6 Write

Write data to WDT buffer.

**BOOL WriteFile( HANDLE hFile,  
LPCVOID lpBuffer,  
DWORD nNumberOfBytesToWrite,  
LPDWORD lpNumberOfBytesWritten,  
LPOVERLAPPED lpOverlapped )**

### Parameter Description

<i>hFile</i>	Handle which get from CreateFile
<i>lpBuffer</i>	Buffer which contain writing data
<i>nNumberOfBytesToWrite</i>	Specify the max data could be written
<i>lpNumberOfBytesWritten</i>	Actually written data member count
<i>lpOverlapped</i>	0

### Return value description

Nonzero indicates success. Zero indicates failure. To get extended error information, call GetLastError function.

The settings that write to WDT will be saving as the default value after registry flush. Otherwise, the settings are the temporary data and it will return to default value after system reset.

## 5.7 Sample Program

```
HANDLE hWDT;
BOOL rt;
DWORD NumberOfBytesRead;
DWORD dwWDT;
DWORD NumberOfBytesWritten;
DWORD iInterval;
int iMode;

hWDT=CreateFile( L"WDT1:",
                GENERIC_READ | GENERIC_WRITE,
                0,
                NULL,
                OPEN_EXISTING,
                0,
                NULL);

rt=ReadFile(hWDT,&dwWDT,sizeof(dwWDT),&NumberOfBytesRead,NULL);

dwWDT = (((DWORD)iMode <<16) & 0xFFFF0000) | (DWORD)(iInterval & 0x0000ffff);
rt=WriteFile(hWDT,&dwWDT,sizeof(dwWDT),&NumberOfBytesWritten,NULL);
```

## 6. Battery

This driver will provide battery information and power status with standard Windows CE API.

### 6.1 Charge LED color

OS State(*1)	Battery	Battery Capacity	DC/PoE	Power LED	Charge LED
OFF	-	-	Input	○	○
OFF	ON	Do not care	-	○	○
OFF	ON	0% ~ 80%	Input	○	●
OFF	ON	81% ~ 100%	Input	○	●
ON	-	-	Input	●	○
ON	ON	100% ~ 20%	-	●	●
ON	ON	19% ~ 0%	-	●	●
ON	ON	0% ~ 80%	Input	●	●
ON	ON	80% ~ 100%	Input	●	●

\*1: Suspend mode and Off mode have the same behavior.

\*1: Idle mode and On mode have the same behavior.

### 6.2 LONG BatteryDrvGetLevels

LONG BatteryDrvGetLevels(void)

The number of levels that this function can return ranges from zero through 3. For example, if only the two values **BATTERY\_FLAG\_HIGH** and **BATTERY\_FLAG\_LOW** can be returned, the return value is 2. If this function can also return **BATTERY\_FLAG\_CRITICAL**, the value is 3. The **BATTERY\_FLAG\_CHARGING**, **BATTERY\_FLAG\_NO\_BATTERY**, and **BATTERY\_FLAG\_UNKNOWN** values are ignored for the purposes of this count.

This function returns 3 in this system.

### 6.3 GetSystemPowerStatusEx2

```
DWORD GetSystemPowerStatusEx2(  
    PSYSTEM_POWER_STATUS_EX2 pSystemPowerStatusEx2,  
    DWORD dwLen,  
    BOOL fUpdate  
)
```

This function retrieves battery status information.

```
typedef struct _SYSTEM_POWER_STATUS_EX2 {  
    BYTE ACLineStatus;  
    BYTE BatteryFlag;  
    BYTE BatteryLifePercent;  
    BYTE Reserved1;
```

```

DWORD BatteryLifeTime;
DWORD BatteryFullLifeTime;
BYTE Reserved2;
BYTE BackupBatteryFlag;
BYTE BackupBatteryLifePercent;
BYTE Reserved3;
DWORD BackupBatteryLifeTime;
DWORD BackupBatteryFullLifeTime;
DWORD BatteryVoltage;
DWORD BatteryCurrent;
DWORD BatteryAverageCurrent;
DWORD BatteryAverageInterval;
DWORD BatterymAHourConsumed;
DWORD BatteryTemperature;
DWORD BackupBatteryVoltage;
BYTE BatteryChemistry;
} SYSTEM_POWER_STATUS_EX2, *PSYSTEM_POWER_STATUS_EX2,
*LPSYSTEM_POWER_STATUS_EX2;

```

We used the smart battery in this system and we do not use backup battery. Hence, you can only use these parameters:

#### ***ACLineStatus***

AC power status. It is one of the following values:

```

AC_LINE_OFFLINE
AC_LINE_ONLINE
AC_LINE_BACKUP_POWER
AC_LINE_UNKNOWN

```

#### ***BatteryFlag***

Battery charge status. It is one of the following values:

```

BATTERY_FLAG_HIGH
BATTERY_FLAG_LOW
BATTERY_FLAG_CRITICAL
BATTERY_FLAG_CHARGING
BATTERY_FLAG_NO_BATTERY
BATTERY_FLAG_UNKNOWN

```

#### ***BatteryLifePercent***

Percentage of full battery charge remaining. Must be in the range 0 to 100, or BATTERY\_PERCENTAGE\_UNKNOWN if percentage of battery life remaining is unknown.

#### ***BatteryVoltage***

Number of millivolts (mV) of battery voltage. It can range from 0 to 65535.

#### ***BatteryCurrent***

Number of milliamps (mA) of instantaneous current drain. It can range from 0 to 32767 for charge and 0 to -32768 for discharge.

#### ***BatteryTemperature***

Average number of milliamps of short term device current drain. It can range from 0 to 32767 for charge and 0 to -32768 for discharge.

### ***BatteryChemistry***

Type of battery. It can be one of the following values:

BATTERY\_CHEMISTRY\_ALKALINE  
BATTERY\_CHEMISTRY\_NICD  
BATTERY\_CHEMISTRY\_NIMH  
***BATTERY\_CHEMISTRY\_LION***  
BATTERY\_CHEMISTRY\_LIPOLY  
BATTERY\_CHEMISTRY\_UNKNOWN

In above data structure; the others of the parameter are not supported.

## **6.4 Sample Program**

```
SYSTEM_POWER_STATUS_EX2 pwr;  
DWORD level;  
  
level=BatteryDrvGetLevels();  
GetSystemPowerStatusEx2(&pwr,sizeof(pwr),TRUE);
```

# 7. Camera

## 7.1 API List

1. Create and Close the Camera Handle
2. Switch resolutions in camera driver
3. Retrieve the captured image data from camera driver
4. Preview function
5. Get the current setting of the camera chipset
6. Save the current settings on system registry
7. Setup the White Balance value to device driver
8. Setup the Exposure value to device driver
9. Setup the AGC value to device driver
10. Setup the Night Mode to device driver

## 7.2 API explanation

### 7.2.1 Create and Close the Camera Handle

- a. Camera module named as "CaptOV7660.dll", it will be loaded when WinCE boots up
- b. Create :  
`m_Handle = CreateFile("CSI1:", GENERIC_READ|GENERIC_WRITE, 0, NULL, OPEN_EXISTING, 0, 0);`
- c. Close :  
`CloseHandle(m_Handle);`

### 7.2.2 Switch resolutions in camera driver

- a. Camera supports four resolutions. They are SXGA(1280x1024)、VGA(640x480)、QVGA(320x240)、QQVGA(160x120).
- b. It uses the key code of "\_IOCTL\_CAPTURE\_SET\_RESOLUTION" to switch resolutions in camera driver.
- c. Sample Code :  
`DWORD dwResolution = 0 ; //set to SXGA mode  
DWORD* pdwIn = &dwResolution;  
DeviceIoControl(m_Handle, _IOCTL_CAPTURE_SET_RESOLUTION, pdwIn, sizeof(DWORD), 0, 0, 0, NULL)`
- d. "pdwIn" setting :  
`*pdwIn = 0 //SXGA  
*pdwIn = 1 //VGA  
*pdwIn = 2 //QVGA  
*pdwIn = 3 //QQVGA`

### 7.2.3 Retrieve the captured image data from camera driver

- a. It uses the key code of "\_IOCTL\_CAPTURE\_GET\_IMAGE\_DATA" to retrieve the captured data from camera driver.

- b. Sample Code:

#### 1. Send capture command

```
TCameraCaptureParameter CamCapSize;
```

```
CamCapSize.iSizeX = iCaptureX;
```

```
// Please set 1280 if you will capture 1280x1024 image size
```

```
CamCapSize.iSizeY = iCaptureY;
// Please set 1024 if you will capture 1280x1024 image size
```

```
DeviceloControl(m_Handle, _IOCTL_CAPTURE_GET_IMAGE_DATA, &CamCapSize,
sizeof(TCameraCaptureParameter), 0, 0, 0, NULL);
```

## **2.Read image data**

```
While(iCaptureY-->0)
{
    ReadFile(m_Handle, iScanLine, buffer_size, &n, NULL)
    If (n == 0)
        Break;
}
```

- c. Function parameter:

“(unsigned char \* )***iScanLine***” is the data buffer in order to save captured image data.

“***buffer size***” is the size of ***iScanLine***. “ ***buffer size = iCaptureX \* 2*** “  
“**DWORD *n***” return the read data size.

You will get one line of image data if you call ReadFile( ..) function. Therefore, you must call it “iCaptureY” times to get all image data. The data you get is YUV format; you could refer to our sample AP to transfer it to JPEG format.

## **7.2.4 Preview function**

- It uses the key code of “\_IOCTL\_PREVIEW\_START\_PREVIEW” to start preview function.
- It uses the key code of “\_IOCTL\_PREVIEW\_STOP\_PREVIEW” to stop preview function.
- Sample Code:

### **Start to preview**

```
TCameraPreviewParameter cCamPreview;
```

```
cCamPreview.iSizeX = preview frame width
cCamPreview.iSizeY = preview frame height
cCamPreview.iPosX = preview frame X position
cCamPreview.iPosY = preview frame Y position
```

```
DeviceloControl(m_Handle, _IOCTL_PREVIEW_START_PREVIEW, &cCamPreview,
sizeof(TCameraPreviewParameter), 0, 0, 0, NULL);
```

### **Stop to preview**

```
DeviceloControl(m_Handle, _IOCTL_PREVIEW_STOP_PREVIEW,0,0,0,0,0,0,NULL)
```

## **7.2.5 Get the current setting of the camera chipset**

- It uses the key code of “\_IOCTL\_CAPTURE\_GET\_REGISTRY” to get the current setting of the camera chipset.
- Sample Code:

```
_SETTING_INFO_CHIP stReg;
_PSETTING_INFO_CHIP pReg;
```

```
pReg = &stReg;
```

```
BOOL bRes = DeviceIoControl(m_Handle, _IOCTL_CAPTURE_GET_REGISTRY, 0, 0, (PBYTE)pReg, sizeof(_SETTING_INFO_CHIP), 0, NULL);
```

c. bRes = TRUE or FALSE

### 7.2.6 Save the current settings on system registry

- The camera driver would not save the current settings automatically even you have changed it. User needs to call this function to save current settings.
- It uses the key code of “\_IOCTL\_CAPTURE\_PUT\_REGISTRY” to save the current settings on system registry
- Save the value in [HKEY\_LOCAL\_MACHINE\Drivers\Camera\Setting]

```
"WhiteBalance_R"=dword:40  
"WhiteBalance_G"=dword:40  
"WhiteBalance_B"=dword:50  
"ExposureTime"=dword:3DE  
"NightMode"=dword:0  
"AGC"=dword:1F  
"Resolution"=dword:1
```

d. Sample Code:

e. DeviceIoControl(m\_Handle, \_IOCTL\_CAPTURE\_PUT\_REGISTRY, 0, 0, 0, 0, 0, NULL)

### 7.2.7 Setup the White Balance value to device driver

- It uses the key code of “\_IOCTL\_CAPTURE\_SET\_WHITE\_BALANCE” to setup the White Balance value to camera drive
- White Balance setting consists of Red, Green and Blue colors. The max value of each color is 0xFF and the min is 0x00.
- Following settings are recommend :

Place	Red	Green	Blue
Cloudy	0x68	0x46	0x40
IndoorHome	0x3C	0x41	0x62
IndoorOffice	0x40	0x40	0x50
Sunny	0x52	0x40	0x35

d. Sample Code :

```
DWORD dwRGB = 0x405072;
```

```
// It implies that red color is 0x40, green color is 0x50 and blue is 0x72.
```

```
DWORD* pdwIn = &dwRGB ;
```

```
BOOL bRes = DeviceIoControl(m_Handle, _IOCTL_CAPTURE_SET_WHITE_BALANCE, pdwIn , sizeof(DWORD), 0, 0, 0, NULL);
```

e. bRes = TRUE or FALSE

### 7.2.8 Setup the Exposure value to device driver

- a. It uses the key code of “\_IOCTL\_CAPTURE\_SET\_EXPOSURE” to setup the Exposure value to camera drive
- b. Exposure Value :

	SXGA	VGA	QVGA	QQVGA
Exposure Max Value	1052	526	526	1052
Exposure Min Value	1	1	1	1

P.S. : For example, if you set the exposure value to 1052 in VGA mode, the effect equals to set as 526.

- c. Sample Code:  
`BOOL bRes = DeviceIoControl(m_Handle,_IOCTL_CAPTURE_SET_EXPOSURE,pdwIn, sizeof(DWORD),0,0,0,NULL)`
- d. bRes = TRUE or FALSE

### 7.2.9 Setup the AGC value to device driver

- a. It uses the key code of “\_IOCTL\_CAPTURE\_SET\_AGC” to setup the AGC value to camera drive.
- b. The max value of AGC is 0xFF and min value is 0x00.
- c. Sample Code:

```
DWORD dwAGC = 0x05;  
DWORD* pdwIn =(DWORD*) &dwAGC;  
BOOL bRes = DeviceIoControl(m_Handle,_IOCTL_CAPTURE_SET_AGC,pdwIn, sizeof(DWORD),  
0,0,0 ,NULL);
```

- d. bRes = TRUE or FALSE.

### 7.2.10 Setup the Night Mode to device driver

- a. It uses the key code of “\_IOCTL\_CAPTURE\_SET\_NIGHT\_MODE” to setup the Night Mode to camera drive.
- b. We send value “0” to disable night mode function. In other words, we send value “1” to enable night mode.
- c. Sample Code:  
`DWORD dwNightMode = 0; // Disable night mode  
DWORD* pdwIn = &dwNightMode;  
BOOL bRes = DeviceIoControl(m_Handle,_IOCTL_CAPTURE_SET_NIGHT_MODE,pdwIn, sizeof(DWORD),  
0,0,0,NULL)`
- d. bRes = TRUE or FALSE.

## 8. Real Time Clock

The driver gives us the ability to set the time/date and to retrieve the time/date. This driver adheres to the common window CE 5.0 RTC driver specification.

### 8.1 SNTP

Software timer is expected to synchronize with RTC every hour; Save SNTP server URL in registry (**ntp.jst.mmfeed.ad.jp & time.windows.com**) as default in registry. The SNTP registry settings are located under the **HKEY\_LOCAL\_MACHINE\Services\Timesvc** registry key.

Upon system startup, services.exe uses the information located under this key to initialize the SNTP service.

[お詫びと訂正]

誠に申し訳ございませんが、SNTP サーバのレジストリの初期設定値にスペル誤りがございます。

下記が正しい URL となります。

(正) 「ntp.jst.mfeed.ad.jp」

(誤) 「 ntp.jst.mmfeed.ad.jp」

上記の初期設定値のままお使いいただいた場合、時刻は「ntp.jst.mfeed.ad.jp」ではなく、「time.windows.com」を使用して同期いたします。

time.windows.com 以外の NTP サーバをご利用の場合は、システムベンダー様にて変更いただきますようお願い申し上げます。

### 8.2 Daily Wakeup

The Real Time Clock also provides a special function "Daily Wakeup". User can specific a date/time or a regularly time of everyday to wake system up if system in suspend mode.

#### 8.2.1 Registry

The all settings of daily wakeup function can be changed by registry access.

**[HKEY\_LOCAL\_MACHINE\SOFTWARE\Carreara]**

**//Specific a data/time to wake system up**

"DwYear"=dword:7DF //The year of the specific data  
"DwMonth"=dword:1 //The month of the specific data  
"DwDay"=dword:1 //The day of the specific data  
"DwHour"=dword:1 //The hour of the specific time  
"DwMinute"=dword:1 //The minute of the specific time  
"DwSecond"=dword:1 //The second of the specific time  
"DwFlag"=dword:0 //1: enable 0:disable

**//Specific a regularly time of everyday to wake system up**

"DwAlarmFlag"=dword:0 //1: enable 0:disable  
"DwAlarmHour"=dword:1 //The hour of the specific time  
"DwAlarmMinute"=dword:1 //The second of the specific time  
"DwAlarmSecond"=dword:1 //1: enable 0:disable

If "DwAlarmFlag" and "DwFlag" are both 1, the DWFlag will be ignored, and only daily alarm function will be executed.

**[HKEY\_LOCAL\_MACHINE\SOFTWARE\Carreara\RegChanged]**

"dwRegChanged"=dword:0

//If any setting changed, this value must change to notify the RTC driver to update the data.

## 8.2.2 Sample Program

### *//Define registry structure*

```
typedef struct _WAKStruct
{
    DWORD   m_DwYear;
    DWORD   m_DwMonth;
    DWORD   m_DwDay;
    DWORD   m_DwHour;
    DWORD   m_DwMinute;
    DWORD   m_DwSecond;
    DWORD   m_DwFlag;
    DWORD   m_dwAlarmFlag;
    DWORD   m_dwAlarmHour;
    DWORD   m_dwAlarmMinute;
    DWORD   m_dwAlarmSecond;
} WAKStruct, *PWAKStruct;
```

```
WAKStruct g_WAKINFO, g_oldWAKINFO;
```

### *//Define registry path*

```
const TCHAR szregRootKey[]   = TEXT("Software\\Carreara");
const TCHAR szregChangedRootKey[] = TEXT("Software\\Carreara\\RegChanged");
const TCHAR szregChanged[]   = TEXT("dwRegChanged");
const TCHAR szregYear[]      = TEXT("DwYear");
const TCHAR szregMonth[]     = TEXT("DwMonth");
const TCHAR szregDay[]       = TEXT("DwDay");
const TCHAR szregHour[]      = TEXT("DwHour");
const TCHAR szregMinute[]    = TEXT("DwMinute");
const TCHAR szregSecond[]    = TEXT("DwSecond");
const TCHAR szregFlag[]      = TEXT("DwFlag");
const TCHAR szregAlarmFlag[] = TEXT("dwAlarmFlag");
const TCHAR szregAlarmHour[] = TEXT("dwAlarmHour");
const TCHAR szregAlarmMinute[] = TEXT("dwAlarmMinute");
const TCHAR szregAlarmSecond[] = TEXT("dwAlarmSecond");
```

### *//Update registry*

```
void WAK_WriteRegistry(WAKStruct *pWAKInfo)
{
    HKEY   hKey;
    LONG   IResult;
    DWORD  dwType;
    DWORD  dwVal;
    DWORD  dwLen;
```

### *//Update Registry value*

```
IResult = RegOpenKeyEx(HKEY_LOCAL_MACHINE, szregRootKey, 0, KEY_ALL_ACCESS, &hKey);
if(ERROR_SUCCESS == IResult)
```

```

{
    dwType = REG_DWORD;
    dwLen = sizeof(DWORD);
    dwVal = pWAKInfo->m_dwAlarmFlag;
    IResult = RegSetValueEx(hKey, szregAlarmFlag, NULL, dwType,
        (LPBYTE)&dwVal, dwLen);
    if(ERROR_SUCCESS == IResult) {
        //Error handling here}

    dwVal = pWAKInfo->m_DwFlag;

    IResult = RegSetValueEx(hKey, szregFlag, NULL, dwType,
        (LPBYTE)&dwVal, dwLen);
    if(ERROR_SUCCESS == IResult) {
        //Error handling here}

    dwVal = pWAKInfo->m_dwAlarmHour;
    IResult = RegSetValueEx(hKey, szregAlarmHour, NULL, dwType,
        (LPBYTE)&dwVal, dwLen);
    if(ERROR_SUCCESS == IResult) {
        //Error handling here}

    dwVal = pWAKInfo->m_dwAlarmMinute;
    IResult = RegSetValueEx(hKey, szregAlarmMinute, NULL, dwType,
        (LPBYTE)&dwVal, dwLen);
    if(ERROR_SUCCESS == IResult) {
        //Error handling here}

    dwVal = pWAKInfo->m_dwAlarmSecond;
    IResult = RegSetValueEx(hKey, szregAlarmSecond, NULL, dwType,
        (LPBYTE)&dwVal, dwLen);
    if(ERROR_SUCCESS == IResult) {
        //Error handling here}

    dwVal = pWAKInfo->m_DwYear;
    IResult = RegSetValueEx(hKey, szregYear, NULL, dwType,
        (LPBYTE)&dwVal, dwLen);
    if(ERROR_SUCCESS == IResult) {
        //Error handling here}

    dwVal = pWAKInfo->m_DwMonth;
    IResult = RegSetValueEx(hKey, szregMonth, NULL, dwType,
        (LPBYTE)&dwVal, dwLen);
    if(ERROR_SUCCESS == IResult) {
        //Error handling here }

    dwVal = pWAKInfo->m_DwDay;
    IResult = RegSetValueEx(hKey, szregDay, NULL, dwType,
        (LPBYTE)&dwVal, dwLen);
    if(ERROR_SUCCESS == IResult) {
        //Error handling here}

```

```

dwVal = pWAKInfo->m_DwHour;
IResult = RegSetValueEx(hKey, szregHour, NULL, dwType,
                        (LPBYTE)&dwVal, dwLen);
if(ERROR_SUCCESS == IResult) {
    //Error handling here}

dwVal = pWAKInfo->m_DwMinute;
IResult = RegSetValueEx(hKey, szregMinute, NULL, dwType,
                        (LPBYTE)&dwVal, dwLen);
if(ERROR_SUCCESS == IResult) {
    //Error handling here}

dwVal = pWAKInfo->m_DwSecond;
IResult = RegSetValueEx(hKey, szregSecond, NULL, dwType,
                        (LPBYTE)&dwVal, dwLen);
if(ERROR_SUCCESS == IResult) {
    //Error handling here}
RegFlushKey(hKey);
RegCloseKey(hKey);
}
else
{
    //Error handling here}

```

***//Notify RTC Driver***

```

IResult = RegOpenKeyEx(HKEY_LOCAL_MACHINE, szregChangedRootKey, 0, KEY_ALL_ACCESS,
                      &hKey);
if(ERROR_SUCCESS == IResult)
{
    dwVal = pWAKInfo->m_dwAlarmFlag;
    IResult = RegSetValueEx(hKey, szregChanged, NULL, dwType,
                            (LPBYTE)&dwVal, dwLen);
    if(ERROR_SUCCESS == IResult) {
        //printf("Set dwAlarmSecond to %d", dwVal);
    }
    RegFlushKey(hKey);
    RegCloseKey(hKey);
}
else
{
    printf("WAK : HKEY_LOCAL_MACHINE\\%s key doesn't exist!\r\n", szregChangedRootKey);}
}

```