

UNIVERSAL DLL

PC Watchdog™

Programmer's Guide

BERKSHIRE PRODUCTS, INC.

Phone: 770-271-0088

<http://www.berkprod.com/>

Rev: 1.11

© Copyright 2010, 2011

®PC Watchdog is a registered trademark of Berkshire Products

Table of Contents

1. NOTES	1
2. FUNCTIONS	2
2.1 INTERNAL & EXTERNAL SERIAL PC WATCHDOGS	2
2.2 WD_OPEN	3
2.3 WD_OPENEX	4
2.4 WD_CLOSE	5
2.5 WD_FORCECOM	6
2.6 WD_GETDLLVERSION	7
2.7 WD_GETERRORINFOMSG	8
2.8 WD_GETWDOGTYPE	9
2.9 WD_GETCOMPORTNUM	10
2.10 WD_GETDEVICEINFO	11
2.11 WD_GETTEMPTICKLE	12
2.12 WD_SETPOWERONDLYTIMES	13
2.13 WD_GETPOWERONDLYTIMES	15
2.14 WD_SETWDOGTIMES	16
2.15 WD_GETWDOGTIMES	18
2.16 WD_GETRESETCOUNT	20
2.17 WD_GETSETNVTEMPOFFSET	21
2.18 WD_GETSETNVLOWTEMPOPTION (USB ONLY)	22
2.19 WD_ENABLEDISABLE	25
2.20 WD_GETSETNVRELAYPULSE	27
2.21 WD_SETBUZZER	29
2.22 WD_GETBUZZER	31
2.23 WD_GETSETNVUSERCODE	33
2.24 WD_ENABLEDISABLEPCRESET	34
2.25 WD_GETSETAUXRELAY	36
2.26 WD_GETSETRSTTOAUXRELAY	39
2.27 WD_GETDIGITALIN	42
2.28 WD_GETANALOGIN	44
2.29 WD_GETSETPCIDIGITALINOUT (PCI ONLY)	45
2.30 WD_GETSETPCIRELAYS (PCI ONLY)	47
2.31 WD_GETSETPOWERMODULE (EXTERNAL SERIAL ONLY)	50
2.32 WD_SETIPADDRESSES (ETHERNET-USB ONLY)	51
2.33 WD_GETIPADDRESSES (ETHERNET-USB ONLY)	53
2.34 WD_GETSETNVUDPNUM (ETHERNET-USB ONLY)	55
2.35 WD_GETMACADDRESS (ETHERNET-USB ONLY)	57
2.36 WD_RESETREBOOTETHERNET (ETHERNET-USB ONLY)	58
2.37 WD_GETSETUSBSUSPENDMODE (ETHERNET-USB ONLY)	60
2.38 WD_GETSETNVETHERALLOWED (ETHERNET-USB USB ONLY)	62
3. SYSTEM STATUS / INFORMATION FLAGS	64
3.1 STATUS FLAGS	64
3.2 DIAGNOSTIC FLAGS	65
3.3 COMMANDS ALLOWED ON ETHERNET	66

1. Notes

This manual covers the interface to the functions provided in the WDog_Univsr.dll file provided on the CD. This DLL is on the CD in the ConsoleAp subdirectory. The same same directory has a sample Windows Console Ap to show how to call the functions.

All the old DLLs and static library files are still on the CD in subdirectories named for each style of PC Watchdog.

A lot of these functions store customized parameters in the non-volatile memory on the watchdog board. The memory is a type of Flash (EEPROM) memory that takes time to program. Allow 15-20 milli-seconds of time for each parameter written.

**** NOTE **** This DLL is compiled as a 32 bit ap. There are instructions at MSDN for calling 32 bit DLLs from 64 bit aps.

The latest versions of all manuals and sample code can be found on our site at:

<http://www.berkprod.com/>

If you have any questions, corrections, or feedback about this manual please contact us at:

http://www.berkprod.com/Other_Pages/Contact_Us.aspx

2. Functions

All the functions will return a status of type: `WD_STATUS`, defined in the header file: `WDog_Univrsl.dll`.

Some functions will internally write additional error or information strings into buffers within the DLL. There is a function in the DLL that can be called to get these strings for further information.

All the functions in the DLL use 32 bit integers and pointers which is a common data size for the PC and Pentium CPUs. The microprocessors on the watchdogs use 8 bit bytes and 16 bit integers. The following function definitions will let you know what the actual data size is within the 32 bit integers.

There are two additional files for use with C or C++. They are the header file, `Wdog_Univrsl.h`, and a library file, `WDog_Univrsl.lib`, that allows link time binding to the DLL instead of using a Load command in the source.

Section 3 covers the various flags that can be sent or returned by the watchdog unless the flags are very specific to a particular function. They will be covered in the function description.

NOTE: The Watchdog has two operational states – **POD** & Armed/Active:

1. The **Power-On-Delay** state where it waits for 2.5 minutes (or a user time) to allow the PC to complete a re-boot. If the POD DIP Switch is on then the board will remain in the POD state after the 2.5 minute delay until it is “tickled” the first time via the function call to get the temp.
2. The Armed and Active state is where it start counting down the timer until it gets “tickled”. When it gets “tickled” it will reload the countdown timer and start over.

This DLL works with all versions of the PC Watchdogs except the ISA PC Watchdog.

2.1 Internal & External Serial PC Watchdogs

The earlier versions of these boards had small on-board microprocessors with limited memory which resulted in limited capabilities. The current versions of these boards are hardware Rev-C for the External and Rev-D for the Internal. They now have the same capabilities as the USB and PCI watchdogs. This DLL will work with the older versions with some caveats:

- In some cases the watchdog will not handle commands at all and the DLL will report a `WD_STATUS` of `WD_SERIAL_FIRMWARE_ERROR`.
- In some case like the Internal version it can come close to using the same values. For instance the Rev-C allows you to set a timeout delay of 1-255 seconds or 1-255 minutes but not 1-65535 seconds like the PCI or USB. In cases like that, the DLL will make a best effort to convert times to a close approximate value and the return status flag will be: `WD_SERIAL_DATA_CONVERTED`.

2.2 WD_Open

Open the Watchdog board and return a Handle that must be used for all other function calls. This function should always be called first. This function can take a while since it will search the system until it finds a PC Watchdog.

WD_STATUS WD_Open(WD_HANDLE *pwdHandle)

Parameters:

pwdHandle – pointer to a variable of type WD_HANDLE.

Return Value:

WD_OK if successful or a WD error code.

Example:

```
WD_HANDLE wdHandle ;
WD_STATUS wdStatus ;

wdStatus = WD_Open(&wdHandle) ;
if(wdStatus != WD_OK)
{
    // Error Handler
}
```

The Open function will look for boards in the system in the following order:

1. USB v1 PC Watchdog (PN:1140)
2. PCI
3. PCIe
4. Reserved (Do Not Use)
5. Ethernet-USB
6. Internal Serial
7. External Serial

Once it finds the first board it stops the search. There is no provision in the DLL to handle multiple PC Watchdogs in a single PC.

2.3 **WD_OpenEx**

Open the Watchdog board and return a Handle that must be used for all other function calls. This function should always be called first. This function is used in lieu of WD_Open().

This function allows you to force the DLL to open a particular type of board. Also see the WD_Set_Com() function for forcing a single COM port.

WD_STATUS WD_OpenEx(WD_HANDLE *pwdHandle, UINT32 iType)

Parameters:

pwdHandle – pointer to a variable of type WD_HANDLE.

iType – one of the WD_TYPE_xxxx values from the header (.h) file: Wdog_Univrsl.h.

Return Value:

WD_OK if successful or a WD error code.

WD_OPEN_EX_ERROR if iType is not a valid type.

Example:

```
WD_HANDLE wdHandle ;
WD_STATUS wdStatus ;

wdStatus = WD_OpenEx(&wdHandle, WD_TYPE_PCI) ;
if(wdStatus != WD_OK)
{
    // Error Handler
}
```

2.4 **WD_Close**

Closes the Watchdog board This function should always be called last.

WD_STATUS WD_Close(WD_HANDLE wdHandle)

Parameters:

wdHandle – Handle of the device from WD_Open().

Return Value:

WD_OK if successful or a WD error code.

Example:

```
WD_HANDLE wdHandle ;
WD_STATUS wdStatus ;

wdStatus = WD_Close(wdHandle) ;
if(wdStatus != WD_OK)
{
    // Error Handler
}
```

2.5 **WD_ForceCom**

Some of the PC Watchdogs use COM ports in the PC for connectivity. The DLL will attempt to find the watchdogs with `WD_Open()` and `WD_Open_Ex()` by cycling through all the COM ports and sending out short test packets. In some cases this might conflict with other devices connected to other COM ports. This function should be called before either of the Open functions to force the DLL to only use a specific COM port. This function does not require a handle.

WD_STATUS WD_ForceCom(UINT32 iCom)

Parameters:

`iCom` – must be a value between 0 and `MAX_COM_PORTS` defined in the header (.h) file: `Wdog_Univrs1.h`. If the value is zero it resets the DLL to use all COM ports again.

Return Value:

`WD_OK` if successful or a WD error code.

`WD_COMM_VAL_ERR` - if `iCom` is greater than `MAX_COM_PORTS`, or the COM x port is not installed.

Example:

```
WD_STATUS wdStatus ;

wdStatus = WD_ForceCom(5) ;    // force COM5
if(wdStatus != WD_OK)
{
    // Error Handler
}
```

2.6 **WD_GetDllVersion**

This function returns a 32 bit encoding of the DLL version. The most significant byte will be set to **0x99** to identify the DLL as the Universal type. The middle high byte contains the major number, the middle low byte is the minor and the lower byte is the sub-minor. For example DLL version 1.09.03 would be returned as: 0x99010903. This call does not require a handle so it can be called before a device open.

In all cases a difference in the sub-minor version can be ignored by your application. This level is reserved for trivial fixes like a spelling fix in an error message.

All version change info is documented in the header file Wdog_Univrsl.h.

WD_STATUS WD_GetDllVersion(UINT32 *pDllVersion)

Parameters:

pDllVersion – pointer to a variable to save the version.

Return Value:

Always WD_OK.

Example:

```
UINT32 iVersion  
  
WD_GetDllVersion(&iVersion) ;  
printf("DLL Version: %02d.%02d.%02d\n", ((iVersion & ff0000) >> 16),  
      ((iVersion & 0xff00) >> 8), (iVersion & 0x00ff)),
```

2.7 **WD_GetErrorInfoMsg**

This function can be called to get additional information messages after each function call if any has been written to the internal DLL buffers. If there is no message to return the function will get zero length strings. This function does not require a handle.

Every DLL function clears these buffers when they are called. If you want to get the messages, then call this function before you attempt a new DLL function call.

WD_STATUS WD_GetErrorInfoMsg(char *ErrorMsg, char *InfoMsg)

Parameters:

ErrorMsg – string location for error message if present.

InfoMsg – string location for information message if present.

Return Value:

Always returns WD_OK.

Example:

```
char ErrBuff[INFO_ERR_BUFF_MAXSIZE] ;
char InfBuff[INFO_ERR_BUFF_MAXSIZE] ;
WD_STATUS wdStatus ;

wdStatus = WD_GetErrorInfoMsg(ErrBuff, InfBuff) ;
if(ErrBuff[0] != '\0')
{
    printf("%s\n", ErrBuff) ;
}
if(InfBuff[0] != '\0')
{
    printf("%s\n", InfBuff) ;
}
```

2.8 **WD_GetWDogType**

After the board has been opened successfully you can use this function to tell which type of board has been found in your system.

WD_STATUS WD_GetWDogType(WD_HANDLE wdHandle, UINT32* pType)

Parameters:

wdHandle – Handle of the device from WD_Open().

pType – pointer to store the type number of the board opened - one of the WD_TYPE_xxxx values from the header (.h) file: Wdog_Univrsl.h.

Return Value:

WD_OK if successful or a WD error code.

Example:

```
WD_HANDLE wdHandle ;
WD_STATUS wdStatus ;
UINT32 iType ;

wdStatus = WD_GetWDogType(wdHandle, &iType);
if(wdStatus == WD_OK)
{
    printf("WDog Type = %d\n", iType) ;
}
```

2.9 **WD_GetComPortNum**

After the board has been opened successfully you can use this function to tell which COM port is being used for the watchdog. If you did a `WD_ForceCom()` function call first the number returned should be the same.

WD_STATUS WD_GetComPortNum(WD_HANDLE wdHandle, UINT32* pCpn)

Parameters:

`wdHandle` – Handle of the device from `WD_Open()`.

`pCpn` – pointer to store the COM port number of the board opened.

Return Value:

`WD_OK` if successful or a WD error code.

Example:

```
WD_HANDLE wdHandle ;
WD_STATUS wdStatus ;
UINT32 iCpn ;

wdStatus = WD_GetComPortNum(wdHandle, &iCpn);
if(iCpn)
{
    printf("COM Port: COM%d is Open", iCpn) ;
}
```

2.10 **WD_GetDeviceInfo**

Gets information about the PC Watchdog.

```
WD_STATUS WD_GetDeviceInfo(WD_HANDLE wdHandle, UINT32* pStat,  
                             UINT32* pDipSw, UINT32* pVer, UINT32* pTick,  
                             UINT32* pDiag )
```

Parameters:

wdHandle – Handle of the device from `WD_Open()`.

pStat – pointer to hold status flags from the board.

pDipSw – pointer for current Dip Switch setting – lower 8 bits will match Dip Switch

pVer – pointer to firmware version info returned in two lower bytes.

(Ex: 0x0000012c = version 01.44)

pTick – pointer to 16 bit count of number of times the board has been tickled by reading the temperature. The tickle count rolls over back to zero after 0xFFFF.

pDiag – pointer to hold diagnostic status flags from the board.

Return Value:

WD_OK if successful or a WD error code.

See Section 3 for a description of the Status and Diagnostic flags that can be returned.

Example:

```
WD_HANDLE wdHandle ;  
WD_STATUS wdStatus ;  
UINT32 iStat=0, iDsw=0, iVer=0, iTick=0, iDiag=0 ;  
  
wdStatus = WD_GetDeviceInfo(wdHandle, &stat, &dsw, &ver,  
                             &tck, &iDiag) ;  
if(wdStatus == WD_OK)  
{  
    printf("Board Status Bits = 0x%04X\n", iStat) ;  
    printf("Firmware Version: %02d.%02d\n", (iVer>>8), (iVer&0x0ff));  
    printf("Current Dip Switch = 0x%02X\n", iDsw) ;  
    printf("Tickle count = %d\n", iTick) ;  
    printf("Board Diagnostic Bits = 0x%04X\n", iDiag) ;  
}
```

2.11 **WD_GetTempTickle**

This will be the most used function for the board. It reads the temperature and it “tickles” the board to make it re-load the count down timer. It also increments the tickle count by 1 before it returns the value.

WD_STATUS **WD_GetTempTickle**(**WD_HANDLE** wdHandle, **INT32*** pTempw,
UINT32* pTempf, **UINT32*** pTick)

Parameters:

wdHandle – Handle of the device from WD_Open().

pTempw – pointer to whole number part of temp in °C. Note this is an INT and can be negative!

pTempf – pointer to flag for fractional portion. Always returns zero on USB board.

pTick – pointer to 16 bit count of number of times the board has been tickled. 0x0000-0xFFFF

Return Value:

WD_OK if successful or a WD error code.

Example:

```
WD_HANDLE wdHandle ;
WD_STATUS wdStatus ;
UINT32 iTmpf=0, iTick=0 ;
INT32 iTempw=0 ;

wdStatus = WD_GetTempTickle(wdHandle, &iTempw, &iTmpf, &iTick) ;
if(wdStatus == WD_OK)
{
    if(iTmpf) // flag set for 0.5C?
        iTmpf = 5 ; // for printf
    printf("Temp = %d.%0d C\n", iTempw, iTmpf) ;
    printf("Tickle count = %d\n", iTick) ;
}
```

The tickle count rolls over back to zero after 0xFFFF.

**** NOTE **** The DLL code reads the temp first and then inserts a short software loop to allow the board to update the tickle count. It is possible for the DLL to outrun the processor on the board and show the tickle count being 1 less than you expect.

Early versions of the EXT Serial will return -100 in iTempw to indicate no capability.

2.12 **WD_SetPowerOnDlyTimes**

The standard mode of the Watchdog is to wait 2.5 minutes (150 seconds) after a **Power-On-Delay** (or reboot) of the PC before it arms itself with the watchdog countdown time. This allows time for the PC to complete the reboot sequence. If your PC or application needs more time to reboot, this function provides two alternatives. First you can write a new longer (or shorter) delay time that will only be active for the current restart of the PC. Or you can write a non-zero value to the on-board non-volatile memory that will be used at every reboot in future. If you write a zero (0x0000) to the non-volatile memory then it is disabled and the board reverts to the 2.5 minute delay at the next reboot.

WD_STATUS **WD_SetPowerOnDlyTimes**(**WD_HANDLE** wdHandle, **UINT32** iPod, **UINT32** iNvPod, **UINT32** iSetFlag, **UINT32*** pResFlag)

Parameters:

wdHandle – Handle of the device from **WD_Open()**.
iPod – a 16 bit value for the new **POD** time in seconds – 0x0000 to 0xFFFF.
iNvPod – a 16 bit value for the new non-volatile **POD** time in seconds.
iSetFlag – flags for Set operation.
pResFlag – pointer for result flag from function call.

Return Value:

WD_OK if successful or a **WD** error code.

Flags for Set operations:

WD_POD_SETPOD – must be set to enable the 16 bit value in **iPod** to be written. If clear, then **iPod** value is ignored.
WD_POD_SETNVPOD – must be set to enable the 16 bit value in **iNvPod** to be written to the non-volatile memory. If clear, then **iNvPod** value is ignored.

Result flags:

WD_POD_SET_IGNORE – returned if the Watchdog has already finished the **Power-On-Delay** and is armed and you sent the **WD_POD_SETPOD** flag. It is too late to change **POD** at this point.
WD_SERIAL_DATA_CONVERTED – data saved but it was not an exact conversion

Example:

```
WD_HANDLE wdHandle ;
WD_STATUS wdStatus ;
UINT32 iPod, iNvPod, iSetFlag, pResFlag ;

iPod = iNvPod = 600 ;          // 600 seconds - 10 minutes
iSetFlag = WD_POD_SETNVPOD | WD_POD_SETPOD ;
pResFlag = 0;
wdStatus = WD_SetPowerOnDlyTimes(wdHandle, iPod, iNvPod, iSetFlag,
                                &pResFlag);
if(wdStatus == WD_OK)
{
    if((pResFlag & WD_POD_SET_IGNORE) == 0)    // is WDog armed
        printf("POD Successfully Set\n") ;
    else
        printf("POD Set Fail - WDog already armed\n") ;
}
```

**** NOTE **** On older versions of the Serial PC Watchdogs (firmware versions less the 2.00) if you set the iPod time to 0 or 1 then the DLL will tell the board to use its ARM NOW command and end the POD if possible.

2.13 **WD_GetPowerOnDlyTimes**

This functions gets the current **POD** time if applicable and the non-volatile time. If the current **POD** time returned is 0x0000, the watchdog is no longer in **POD** and could be armed. If the status does not show armed and the DIP option was selected for an Extended Power-On-Delay then the board is waiting for the first “tickle” If the non-volatile value is not zero then this value is being used for **POD** time at every reboot instead of the fixed 2.5 minute delay.

WD_STATUS **WD_GetPowerOnDlyTimes**(**WD_HANDLE** wdHandle, **UINT32*** pPod, **UINT32*** pNvPod)

Parameters:

wdHandle – Handle of the device from **WD_Open**().

pPod – pointer for the current **POD** time in seconds – 0x0000 to 0xFFFF.

pNvPod – pointer for the current non-volatile **POD** time in seconds – 0x0000 to 0xFFFF.

Return Value:

WD_OK if successful or a **WD** error code.

Example:

```
WD_HANDLE wdHandle ;
WD_STATUS wdStatus ;
UINT32 iPod, iNvPod ;

iPod = iNvPod = 0;
wdStatus = WD_GetPowerOnDlyTimes(wdHandle, &iPod, &iNvPod);
if(wdStatus == WD_OK)
{
    printf("POD Time = %d\n", iPod) ;
    printf("NV POD Time = %d\n", iNvPod) ;
}
```

**** NOTE **** pPod = 0 on Serial PC Watchdogs with firmware versions less the 3.00.
pNvPod = 0 on Serial PC Watchdogs with firmware versions less the 2.00.

2.14 **WD_SetWdogTimes**

The standard mode of the Watchdog as shipped is to read the last three dip switches for one of eight possible countdown (timeout) delay values. These values range from 5 seconds to 1 hour. This function allows you to change the countdown timer value from 1 to 65535 (0x0001 to 0xFFFF) seconds.

The first option is to send a replacement for the watchdog time that will stay in force as long as the board is powered on (even after resets of the PC). This value is placed in a holding register on the board and will be loaded into the countdown timer the *next time you “tickle” the board*. If this value is set to zero then the board will first check to see if there is a non-volatile value to put in the holding register, otherwise the holding register is cleared. If the holding register is clear the watchdog will revert to the dip switch setting at the next “tickle”

The second option allows you to store your replacement value in non-volatile memory to be used forever. This value you send will also be placed in the holding register. If you send 0x0000 the non-volatile will be cleared, the holding register will clear and the board will revert to the dip switch option at the next “tickle”

**WD_STATUS WD_SetWdogTimes(WD_HANDLE wdHandle, UINT32 iWdTime,
UINT32 iNvWdTime, UINT32 iFlag)**

Parameters:

wdHandle – Handle of the device from WD_Open().

iWdTime – a 16 bit value for the new watchdog countdown time in seconds.

iNvWdTime – a 16 bit value for the new non-volatile watchdog time in seconds.

iFlag – flag bits for Set operations

Return Value:

WD_OK if successful or a WD error code.

Flags for Set operations:

WD_SET_TIMEOUT – must be set to enable the 16 bit value in **iWdTime** to be written to the internal holding register.

WD_SETNV_TIMEOUT – must be set to enable the 16 bit value in **iNvWdTime** to be written to the non-volatile memory and the internal holding register.

**** NOTE **** If both flags are set, the non-volatile will be processed first and the iWdTime value second. The holding register will end up getting the iWdTime value. The non-volatile value will take effect after the next power cycle.

**** NOTE **** Early versions of the serial watchdogs will return WD_STATUS set to firmware error. The Internal Serial with firmware 2.xx may return a WD_STATUS of WD_SERIAL_DATA_CONVERTED if the times can not be converted exactly.

Example:

```
WD_HANDLE wdHandle ;
WD_STATUS wdStatus ;
UINT32 iWdgTm, iNvWdgTm, iWdgtFl ;

iWdgTm = iNvWdgTm = iWdgtFl = 0;
iWdgTm = 600 ;           // 600s = 10 minutes for new Wdog time
iNvWdgTm = 600 ;        // 10 minutes for nv memory for new Wdog time
iWdgtFl = WD_SET_TIMEOUT | WD_SETNV_TIMEOUT ;
wdStatus = WD_SetWdogTimes(wdHandle, iWdgTm, iNvWdgTm, iWdgtFl);
if(wdStatus == WD_OK)
{
    printf("-- WDG TIME -- New Countdown Timers Write OK\n") ;
}
if(wdStatus == WD_SERIAL_DATA_CONVERTED)
{
    printf("-- WDG TIME -- New Countdown Timers Write OK - Data
           Converted\n") ;
}
```

2.15 **WD_GetWdogTimes**

This function returns four 16 bit values. The first is the current watchdog countdown time remaining if the board is armed. If the watchdog is still in **POD** time then the value returned will be 0xFFFF. The second value is the stored non-volatile time that will be used at power up. If this value is 0x0000 then it is disabled. The third value is what is currently in the holding register. If it is non-zero then this value will be reloaded into the countdown timer each time the watchdog is “tickled”. The last value returned is the time selected by the dip switches that will be used if the holding register is clear.

WD_STATUS **WD_GetWdogTimes**(**WD_HANDLE** wdHandle, **UINT32*** pWdTime, **UINT32*** pNvWdTime, **UINT32*** pHoldRegTime, **UINT32*** pDipSwTime)

Parameters:

wdHandle – Handle of the device from **WD_Open**().

pWdTime – pointer to 16 bit value for the current watchdog time remaining or 0xFFFF if not armed.

pNvWdTime – pointer to 16 bit value for the non-volatile watchdog time in seconds.

pHoldRegTime - pointer to 16 value for the time in the holding register.

pDipSwTime – pointer to 16 bit value for time selected on dip switches that will be used if the holding register is clear.

Return Value:

WD_OK if successful or a **WD** error code.

**** NOTE **** Early PC Watchdog firmware on all types of boards can not read back the holding register value. The **pHoldRegTime** value will always be set to 0x0000 on these boards.

Early versions of the serial watchdogs will only return the **pDipSwTime**. The Internal Serial with firmware 2.xx can only return the **pNvWdTime** and **pDipSwTime** values.

Example:

```
WD_HANDLE wdHandle ;
WD_STATUS wdStatus ;
UINT32 iWdgTm, iNvWdgTm, iHoldReg, iDipSwTime ;

iWdgTm = iNvWdgTm = iHoldReg = iDipSwTime = 0 ;
wdStatus = WD_GetWdogTimes(wdHandle, &iWdgTm, &iNvWdgTm, &iHoldReg,
                           &iDipSwTime);

if(wdStatus == WD_OK)
{
    printf("Current Countdown Time = %d\n", iWdgTm) ;
    printf("Current NV Countdown Time = %d\n", iNvWdgTm) ;
    printf("Current Holding Register Time = %d\n", iHoldReg) ;
    printf("DipSwitch Time = %d\n", iDipSwTime) ;
}
```

2.16 **WD_GetResetCount**

Each time the watchdog resets the PC it will increment a counter that runs from 0 to 255 (0x00-0xFF). The count will not rollover, it stops at 255. This command allows you to get the reset count and optionally clear it after the current value has been retrieved. If you clear the reset count, the watchdog will also clear the bottom LED as well.

**WD_STATUS WD_GetResetCount(WD_HANDLE wdHandle, UINT32 iFlag,
UINT32* pRstCnt)**

Parameters:

wdHandle – Handle of the device from WD_Open().
iFlag – flag to clear the reset counter after retrieving.
pRstCnt – pointer to store the current reset count.

Return Value:

WD_OK if successful or a WD error code.

Flag:

WD_CLEAR_RST_CNT – set this flag to clear the reset count after it has been retrieved. It will also clear the bottom LED at the back of the board. Leave flag cleared if you just want the count.

**** NOTE **** Early versions of the serial watchdogs will stop the reset count at 128 (0x80) rather than 255.

Example:

```
WD_HANDLE wdHandle ;
WD_STATUS wdStatus ;
UINT32 iFlag, iRstCnt ;

iFlag = WD_CLEAR_RST_CNT ;           // clear count & bottom LED
wdStatus = WD_GetResetCount(wdHandle, iFlag, &iRstCnt);
if(wdStatus == WD_OK)
{
    printf("Reset Count = %d\n", iRstCnt) ;
}
```

2.17 **WD_GetSetNvTempOffset**

The watchdog board monitors the temperature and looks for high temperature trip points to start the buzzer and optionally reset the PC. The process is described in the Hardware Manuals for Temperature Reset enable Dip Switch. This function allows you to increase the trip points in 1 degree centigrade increments up to 31 degrees C (0x1F). This value is always stored in the non-volatile memory and becomes effective immediately.

WD_STATUS **WD_GetSetNvTempOffset**(**WD_HANDLE** wdHandle, **UINT32** iFlag, **UINT32** iNvOffset, **UINT32*** pCurOffset)

Parameters:

wdHandle – Handle of the device from **WD_Open()**.

iFlag – flag to enable writing the non-volatile memory

iNvOffset – new offset value for non-volatile memory

pCurOffset – pointer to current non-volatile value. In case of a write, it will equal what you just wrote.

Return Value:

WD_OK if successful or a **WD** error code.

Flag:

WD_TEMP_OFF_WREN – this flag must be set to store the new offset value. If it is clear then the function just returns the current stored value in **pCurOffset**.

**** NOTE **** Early versions of the serial watchdogs will return a status of **WD_SERIAL_FIRMWARE_ERROR** since they do not support this command.

Example:

```
WD_HANDLE wdHandle ;
WD_STATUS wdStatus ;
UINT32 iFlag, iNvOffset, iCurOffset ;

iNvOffset = 11 ;           // set 11C offset
iFlag = WD_TEMP_OFF_WREN ; // enable nv write
wdStatus = WD_GetSetNvTempOffset(wdHandle, iFlag, iNvOffset,
                                &iCurOffset);
if(wdStatus == WD_OK)
{
    printf("NV Temp Offset = %d\n", iCurOffset) ;
}
```

2.18 **WD_GetSetNvLowTempOption (USB Only)**

This is an all new option added to the DLL beginning with version 01.30.00. It allows you to set two low temperature trip points. If the temperature drops below the first trip point you have the option of turning on the Aux relay. If the temperature drops below the second trip point then you have the option of putting the PC in reset. There are also options to allow the buzzer to sound.

The values for the two temperatures must be 30°C (86°F) or lower and negative values are allowed down to -40°C (-40°F). The two trip points can be set to equal temperatures but the second trip point can not be higher than the first one.

WD_STATUS WD_GetSetNvTempOffset(WD_HANDLE wdHandle, UINT32* pFlag, INT32* pLowTempTrip1, INT32* pLowTempTrip2, UINT32* pLowTempHyster)

Parameters:

wdHandle – Handle of the device from WD_Open().

pFlag – pointer to flags to enable writing the non-volatile memory and enabling relays. Current settings will be returned in this flag value

pLowTmTrip1 – pointer to low trip 1 for the first trip point. On return it will be equal the trip temperature value if active.

pLowTmTrip2 – pointer to low trip 2 for the second trip point. It must be less than or equal to the first trip point. On return it will be equal the trip temperature value if active.

pLowTempHyster – this offset will be added to the two trip points and used as a hysteresis value. When the temperature starts going back up the relays and buzzers will turn off (if enabled) when the temperatures are higher. This value should be at least 3 to prevent sporadic action. On return it will be equal the hysteresis temperature value if active. The valid range is 2 to 15 in degrees C.

Return Value:

WD_OK if successful or a WD error code.

Flags:

WD_LOWTEMP_OPTCLR – if this flag is set then all values are cleared and the low trip point operation is canceled. This flag overrides all others. On return this flag will be set if the operation is disabled.

WD_LOWTEMP_WREN – this flag must be set to store temperature trip points, relay and buzzer options. If this bit is clear the function just returns the current status and temperatures.

WD_LOWTEMP1_AUXEN – if this flag is set then the Aux relay will turn on if the temperature is lower than the first trip point. It will stay on when the second trip point is hit as well. On return this flag will be set if the option is active.

WD_LOWTEMP2_RSTEN – if this flag is set then the Reset relay will turn on if the temperature is lower than the first trip point. On return this flag will be set if the option is active.

WD_LOWTEMP1_BUZEN – if this flag is set then the buzzer will turn on if the temperature is lower than the first trip point. On return this flag will be set if the option is active.

WD_LOWTEMP2_BUZEN – if this flag is set then the buzzer will turn on if the temperature is lower than the second trip point. On return this flag will be set if the option is active.

WD_LOWTEMP_ACTIVE – this is a return flag only. On return this flag will be set if the low trip point operation is active.

WD_LOWTEMP_ERR1 – on return this flag will be set if either trip temperature is higher than 35°C or lower than -40°C or if the second trip point has a value higher than the first.

WD_LOWTEMP_ERR2 – on return this flag will be set if the hysteresis value is outside the range of 1-9.

**** NOTES **** This function only works with the DLL version 01.30.00 or higher and only with the USB PC Watchdogs with firmware 3.10 or higher.

Be careful if you use this function along with the Aux relay function described in section 2.25 - you could end up with conflicting operation of the relay.

Example:

```
signed int pLowTempTrip1, pLowTempTrip2 ;
UINT32 pFlag, pLowTempHyster ;

pLowTempTrip1 = 2 ;           // 2 degrees C
pLowTempTrip2 = -3 ;
pLowTempHyster = 3 ;
// enable for updating
pFlag = WD_LOWTEMP_WREN ;
// AUX relay and buzzer activate at temp1
pFlag |= (WD_LOWTEMP1_AUXEN | WD_LOWTEMP1_BUZEN);
// RST relay but no buzzer at temp2
pFlag |= WD_LOWTEMP2_RSTEN ;
// send to board
wdStatus = WD_GetSetNvLowTempOption(wdHandle, &pFlag,
                                     &pLowTempTrip1, &pLowTempTrip2, &pLowTempHyster);
if(wdStatus == WD_OK)
{
    printf("-- LOW TEMP OFF -- Low Temp Option SET
           successful\n") ;
}
pLowTempTrip1 = 0 ;
pLowTempTrip2 = 0 ;
pLowTempHyster = 0 ;
pFlag = 0 ;           // just read status
wdStatus = WD_GetSetNvLowTempOption(wdHandle, &pFlag,
                                     &pLowTempTrip1, &pLowTempTrip2, &pLowTempHyster);
if(wdStatus == WD_OK)
{
    printf("\t-- LOW TEMP OFF -- Low Temp Option Read
           successful\n") ;
}
pFlag = WD_LOWTEMP_OPTCLR ;   // clear the option
wdStatus = WD_GetSetNvLowTempOption(wdHandle, &pFlag,
                                     &pLowTempTrip1, &pLowTempTrip2, &pLowTempHyster);
```

2.19 **WD_EnableDisable**

This functions allows you to Enable or Disable the watchdog. When the watchdog is disabled and it is armed it will act like it is being tickled and continuously reload the countdown timer. When it is removed from the disabled state and it is armed it will start counting down again.

If you disable the watchdog during Power-On-Delay (**POD**), the board will finish the **POD** timer and enter the armed state, but be disabled from counting down.

WD_STATUS WD_EnableDisable(WD_HANDLE wdHandle, UINT32 iFlagSet)

Parameters:

wdHandle – Handle of the device from WD_Open().
iFlagSet – flags for enable or disable

Return Value:

WD_OK if successful or a WD error code.

Flags Set Operations:

WD_WDOG_ENABLE – set this flag to enable the watchdog.
WD_WDOG_DISABLE – set this flag to disable the watchdog.

**** NOTE **** If both flags are set the watchdog will be disabled.

Example:

```
WD_HANDLE wdHandle ;
WD_STATUS wdStatus ;
UINT32 iStat=0, iDsw=0, iVer=0, iTick=0, iDiag=0 ;
UINT32 iEnDisFlag ;

iEnDisFlag = WD_WDOG_DISABLE ;
wdStatus = WD_EnableDisable(wdHandle, iEnDisFlag);
wdStatus |= WD_GetDeviceInfo(wdHandle, &iStat, &iDsw, &iVer,
                             &iTick, &iDiag);
if(wdStatus == WD_OK)
{
    // have to assume OK on some early firmware
    if(iStat & WD_STAT_CMD_DISABLED)
        printf("Watchdog is Disabled\n") ;
    else
        printf("Watchdog is Enabled\n") ;
}
```

2.20 **WD_GetSetNvRelayPulse**

Each time the watchdog times out and re-boots the PC, it defaults to activating the reset relay for 2.5 seconds. In a few very rare instances some older Dell machines (pre 2000 with ISA slots) have gone into setup mode if reset is held this long.

This function permits you to increase or decrease the reset relay active time. The 8 bit time is always written to non-volatile memory and the time is in increments of 50 milli-second (0.05 second) tics. Write a value of zero to clear this option and return to the default 2.5 second activation. The max value is 255 or 12.75 seconds.

On boards with an Aux Relay such as the USB or Internal serial this time is also used for that relay when you enable the option to pulse the Aux Relay at reset. This time is also used on the External serial for cycling the power on the AC Power Module.

WD_STATUS **WD_GetSetNvRelayPulse(WD_HANDLE wdHandle, UINT32 iFlag, UINT32 iNvRelayPulse, UINT32* pCurRelayPulse)**

Parameters:

wdHandle – Handle of the device from WD_Open().

iFlag – flag to enable writing the non-volatile memory

iNvRelayPulse – new 8 bit relay pulse value for non-volatile memory (0 - 255)

pCurRelayPulse – pointer to get current non-volatile value. In case of a write, it will equal what you just wrote.

Return Value:

WD_OK if successful or a WD error code.

Flag:

WD_RLY_PLS_WREN – this flag must be set to store the new relay value. If it is clear then the function just returns the current non-volatile value in **pCurRelayPulse**.

**** NOTE **** Early versions of the serial watchdogs (firmware less than 2.xx) will return a status of WD_SERIAL_FIRMWARE_ERROR since they do not support this command.

Example:

```
WD_HANDLE wdHandle ;
WD_STATUS wdStatus ;
UINT32 iFlag, iNvRelayPulse, iCurRelayPulse ;

iNvRelayPulse = 25 ;           // pulse = 25 * 0.05 = 1.25 seconds
iFlag = WD_RLY_PLS_WREN ;     // enable nv write
wdStatus = WD_GetSetNvRelayPulse(wdHandle, iFlag, iNvRelayPulse,
                                &iCurRelayPulse);
if(wdStatus == WD_OK)
{
    printf("Pulse time = %d - 50mS Tics\n", iCurRelayPulse) ;
}
```

2.21 **WD_SetBuzzer**

The buzzer on the board defaults to a 0.6 second (600 milli-second) beep at power up and after each re-boot. The power up buzzer is fixed, but this function allows you to change the buzzer time for subsequent watchdog timeout reboots. This function also allows you to turn the buzzer on or off.

WD_STATUS **WD_SetBuzzer(WD_HANDLE wdHandle, UINT32 iBuzzTime, UINT32 iNvBuzzTime, UINT32 iFlags)**

Parameters:

wdHandle – Handle of the device from WD_Open().

iBuzzTime – non-zero value to turn buzzer on. On boards with newer firmware the buzzer will sound for iBuzzTime * 10mS (0.01) up 2.54 seconds. The value of 255 – 0xFF is a special case that will turn the buzzer on continuous. Send 0x00 to turn the buzzer off.

iNvBuzzTime – reboot buzzer time value for non-volatile memory up to 2.54 seconds. The value of 255 – 0xFF is a special case that will turn the buzzer on continuous. Set this equal to zero to return to default 600ms buzzer.

iFlags – flags to enable buzzer on and write to non-volatile memory.

Return Value:

WD_OK if successful or a WD error code.

Flags:

WD_BUZZ_ON_EN – this flag must be set to turn on/off the buzzer with iBuzzTime.

WD_BUZZ_NV_WREN – this flag must be set to write non-volatile memory with the reset buzzer time in iNvBuzzTime.

WD_BUZZ_CNTL_EN – this flag must be set to do the following enable/disable option.

WD_BUZZ_DISABLE – set or clear this flag (along with WD_BUZZ_CNTL_EN) to disable or enable any buzzer activity while the board is powered on. If flag is set buzzer will be disabled.

WD_BUZZ_NV_CNTL_EN – this flag must be set to do the following enable/disable option.

WD_BUZZ_NV_DISABLE – set or clear this flag (along with WD_BUZZ_NV_CNTL_EN) to disable or enable any buzzer activity while the board is powered on. If flag is set buzzer will be disabled.

** FIRMWARE NOTES **

The External Serial does not have a buzzer so WD_STATUS will always be set to a firmware error. Internal Serial with firmware less than 3.xx will also return an error since they did not have any buzzer control.

USB boards with firmware less than 3.xx and PCI less than 3.xx have the following restrictions:

If you set the iBuzzTime variable to a non-zero value it will turn on the buzzer and leave it on until you send a value of zero.

The value sent for the non-volatile memory value allows you three options for the reboot buzzer. If the value equals 255 (0xFF) the reboot buzzer will be on continuously and you will have to use this function (iBuzzTime = 0) to turn it off. If the value equals 250 (0xFA) then the reboot buzzer time will be set to 2.5 seconds. Any other value for the non-volatile memory value causes the board reverts back to the 0.6 second buzzer at each re-boot.

There is no option to disable the USB buzzer except with DIP Switch #3.

PCI less than 1.6x do allow the buzzer to be disabled.

Example:

```
WD_HANDLE wdHandle ;
WD_STATUS wdStatus ;
UINT32 iBuzzTime, iNvBuzzTime, iFlags, i ;

iBuzzTime = 90;          // buzzer on - 900mS (0.9s)
iNvBuzzTime = 250;      // 0xfa - reboot buzzer = 2.5 seconds
iFlags = WD_BUZZ_ON_EN |WD_BUZZ_NV_WREN ;
wdStatus = WD_SetBuzzer(wdHandle, iBuzzTime, iNvBuzzTime, iFlags);
if(wdStatus == WD_OK)
{
    printf("\t-- BUZZER -- Buzzer should be ON\n") ;
}
for(i=0; i<0xffffffff; i++)
    ;
iBuzzTime = 0x0;        // buzzer off
iFlags = WD_BUZZ_ON_EN ;
wdStatus = WD_SetBuzzer(wdHandle, iBuzzTime, iNvBuzzTime, iFlags);
```

2.22 **WD_GetBuzzer**

**** NOTE **** This function has been modified for all versions of the watchdogs that have buzzers. In order to maintain compatibility with prior versions there are still just two parameters for the function.

This function returns the buzzer times set with the prior command. On newer versions of watchdog firmware it can return the buzzer time remaining if you just did a buzzer ON with a delay value. The variable pFlags can send and receive flags. In all other DLLs this parameter was pBuzzTime and it only returned flags. If the pFlags value is zero when the function is called then pBuzzTime defaults to returning the non-volatile buzzer time. If the normal 0.6 reboot buzzer is active it returns 0x00. It will return 0xFF if the continuous reboot buzzer is active.

**WD_STATUS WD_GetBuzzer(WD_HANDLE wdHandle, UINT32* pFlags,
UINT32* pBuzzTime)**

Parameters:

wdHandle – Handle of the device from WD_Open().

pFlags – pointer to flags to send and receive.

pBuzzTime – pointer to 8 bit buzzer time to return the current buzzer time left or the value for the non-volatile buzzer time.

Return Value:

WD_OK if successful or a WD error code.

Send Flag:

WD_BUZZ_GET_NOT_NV – if this flag is set when the function is called it will return the current buzzer time rather than the non-volatiles time.

Return Flags:

WD_BUZZ_DISABLE – the buzzer is disabled.

WD_BUZZ_NV_DISABLE – non-volatile buzzer disable. Buzzer is disabled all the time after every power-up.

**** FIRMWARE NOTES ****

The External Serial does not have a buzzer so WD_STATUS will always be set to a firmware error. Internal Serial with firmware less than 3.xx will also return an error since they did not have any buzzer control.

USB boards with firmware less than 2.xx and PCI less than 3.xx have the following restrictions:

If the WD_BUZZ_GET_NOT_NV is set and the buzzer is off pBuzzTime returns 0x00, if it is on it returns 0x01. If the flag is clear pBuzzTime only returns three possible non-volatile values. If the normal 0.6 reboot buzzer is active it returns 0x00, if the 2.5 second buzzer is enabled it returns 0xFA (250), and it will return 0xFF if the continuous reboot buzzer is active.

Example:

```
WD_HANDLE wdHandle ;
WD_STATUS wdStatus ;
UINT32 iFlag, iBuzzTime ;

iFlag = 0 ;          // get non-volatile time
iBuzzTime = 0;
wdStatus = WD_GetBuzzer(wdHandle, &iFlag, &iBuzzTime);
if(wdStatus == WD_OK)
{
    printf("NV Re-Boot Buzzer Time = %d\n", iBuzzTime) ;
}
```

2.23 **WD_GetSetNvUserCode**

This command allows you to store your own unique 8 byte user code or information in the non-volatile memory on the watchdog.

**WD_STATUS WD_GetSetNvUserCode(WD_HANDLE wdHandle, UINT32 iFlag,
UCHAR* pNvUserCode, UCHAR* pCurNvUserCode)**

Parameters:

wdHandle – Handle of the device from WD_Open().

iFlag – flag to enable writing the non-volatile memory

pNvUserCode – pointer to an 8 byte (char) array to be written if the flag is set.

pCurNvUserCode – pointer to an 8 byte (char) array to retrieve the current non-volatile code. In case of a write, it will equal what you just wrote.

Return Value:

WD_OK if successful or a WD error code.

Flag:

WD_USE_CODE_WREN – this flag must be set to store the new code value. If it is clear then the function just returns the current stored value in non-volatile memory.

Example:

```
WD_HANDLE wdHandle ;
WD_STATUS wdStatus ;
UINT32 iFlag, i;
UCHAR bNvUserCode[8], bCurNvUserCode[8] ;

for(i=0; i<8; i++)
    bNvUserCode[i] = i+1;
iFlag = WD_USE_CODE_WREN ;    // enable nv write
wdStatus = WD_GetSetNvUserCode(wdHandle, iFlag, bNvUserCode,
                               bCurNvUserCode);
if(wdStatus == WD_OK)
{
    printf("Code = ");
    for(i=0; i<8; i++)
        printf(" 0x%02X ", (int)bCurNvUserCode[i]) ;
    printf("\n") ;
}
```

2.24 **WD_EnableDisablePcReset**

This command allows you to force an immediate PC reset. There is also a 8 bit delay time that is passed as the number of seconds (0x00-0xFF) before the reset occurs. Values larger than 255 (0xFF) will be truncated to 255. Set this time to zero for an immediate reset.

The pGetTime parameter is used for three purposes. When the WD_PC_RESET_EN flag is used it will return the iResetTime and no flags indicating the board accepted the command. If both send flags are clear and the firmware is newer then it will return the time left before a reset will occur (in seconds) if you originally passed a non-zero value in iResetTime in a prior function call. Otherwise it returns flag bits that are greater than 0x00FFFF.

**** NOTE **** The board must be in the **ARMED** state for this command to work!

**** NOTE **** PcReset will override a WDog disable command!

WD_STATUS WD_EnableDisablePcReset(WD_HANDLE wdHandle, UINT32 iFlag, UINT32 iResetTime, UINT32* pGetTime)

Parameters:

wdHandle – Handle of the device from WD_Open().

iFlag – flag to enable or disable the reset. If both flags are set then the function does nothing. If both flags are clear it will return the remaining time before reset on boards with newer firmware (see the firmware notes).

iResetTime – 8 bit delay value in seconds before reset occurs. The Ethernet-USB allows 16 bit values (up to 65535 seconds).

pGetTime – Pointer for returned time or flags.

Return Value:

WD_OK if successful or a WD error code.

Note: if you send a time of zero then you will not get the return – the PC should reset almost immediately.

Send Flags:

WD_PC_RESET_EN – this flag must be set to enable the reset to start.

WD_PC_RESET_DIS – this flag is must be set to cancel the reset command.

Return Flags (in iGetTime):

WD_PC_RESET_REJCT – this flag will be set if the board did not accept the command. This means the board was not yet ARMED.

WD_PC_RESET_NO_DIS – this flag is will be set if you try to disable the reset command on boards with earlier firmware.

WD_PC_RESET_NO_TR – this flag is will be set if you try to read back the time remaining before reset on older rev boards.

WD_PC_RESET_NO_PEND – this flag is will be set if you try to read back the time remaining before reset if there was not a reset pending. Issue a reset enable command first.

**** FIRMWARE NOTES ****

The Serial versions with firmware less than 3.xx had a Reset Now and a Reset in 10 Seconds command. If you set iResetTime to a value less than 10 it will use the Reset Now. A value in iResetTime greater than 10 will become a Reset in 10 command and pGetTime will be set equal to 10. There is no capability to stop a reset command once it has been sent.

Once this command has been sent with a non-zero delay time, the USB & PCI PC Watchdogs with firmware less than 3.xx will no longer respond to any more commands. There is no capability to stop a reset command once it has been sent.

Example:

```
WD_HANDLE wdHandle ;
WD_STATUS wdStatus ;
UINT32 iFlag, iResetTime, iGetTime ;

iResetTime = 20 ;           // 20 seconds till PC reset
iFlag = WD_PC_RESET_EN ;   // enable reset
wdStatus = WD_EnableDisablePcReset(wdHandle, iFlag, iResetTime,
                                     &iGetTime);
if((wdStatus == WD_OK) && ((iGetTime & WD_PC_RESET_REJCT) == 0))
{
    printf("\t-- Reset PC - Command Accepted\n");
}
```

2.25 **WD_GetSetAuxRelay**

The USB, Ethernet-USB, and Internal Serial PC Watchdogs have an auxiliary relay. This function gives you some control of the auxiliary relay on the board. The returned status is the state of the relay **before** any changes are applied except for relay on status.

There is a non-volatile memory option to force immediate inversion every time the board powers up (the board will power up with the relay on). Then if power is lost on the PC the relay will turn off. There is also an option to store a non-volatile memory options to pulse the relay or have it latch on after the watchdog reboots the PC.

Since this function allows multiple relay options the flags have been designed to allow a subset of actions without disturbing prior settings. Therefore each action requires an ENable flag and an action flag. As an example: if you just want to turn on the relay then call the function with the WD_AUX_RLY_EN flag set along with the WD_AUX_RLY_ON set. To turn off the relay just set the WD_AUX_RLY_EN flag and leave the WD_AUX_RLY_ON flag clear.

**WD_STATUS WD_GetSetAuxRelay(WD_HANDLE wdHandle, UINT32 iRelaySet,
UINT32* pRelayGet)**

Parameters:

wdHandle – Handle of the device from WD_Open().
iRelaySet – flags sent for control.
pRelayGet – pointer to flags returned from the board

Return Value:

WD_OK if successful or a WD error code.

Flags for Set operations:

WD_AUX_RLY_HTEMP_EN – Set this flag to enable changing (setting or clearing) the the relay action when the first high temperature trip point is hit.
WD_AUX_RLY_INVRT_EN – Set this flag to enable changing (setting or clearing) the inversion.
WD_AUX_RLY_RST_EN – Set this flag to enable changing the relay action at reboot.
WD_AUX_RLY_EN – Set this flag to enable changing (on or off) the relay.

Flags for Get & Set operations:

WD_AUX_RLY_HTEMP_ON – set this flag (along with WD_AUX_RLY_HTEMP_EN) for aux relay activation at the first high temperature trip point. This option is stored in non-volatile memory. This flag will be returned in **pRelayGet**.

WD_AUX_RLY_INVRT – set this flag (along with WD_AUX_INVRT_EN) for aux relay inversion enable at power-up only. This option is stored in non-volatile memory. This flag will be returned in **pRelayGet**.

WD_AUX_RLY_LATCH – set this flag (along with WD_AUX_RLY_RST_EN) to make the relay latch on after the watchdog reboots the PC. This option is stored in non-volatile memory. This flag will be returned in **pRelayGet**.

WD_AUX_RLY_PULSE – set this flag (along with WD_AUX_RLY_RST_EN) to make the relay pulse (same length as reset relay) after the watchdog reboots the PC. This option is stored in non-volatile memory. This flag will be returned in **pRelayGet**.

WD_AUX_RLY_ON – this flag must be set (along with WD_AUX_RLY_EN) to turn the relay on. If clear the relay will be turned off. This flag will be returned in **pRelayGet**.

**** FIRMWARE NOTES ****

The Internal Serial version with firmware less than 2.xx only allowed the relay On/Off option and it did not return status. Firmware less than 3.xx added relay inversion (relay on) at power up and it did return status.

The Ethernet-USB firmware less than 3.xx did not have the power up relay on option.

Be careful with Aux relay settings if you are also using the new low temperature trip options. You could end up with conflicting operation of the relay. See section 2.18

The high temperature relay on option requires DLL version 01.30.00 or higher and USB firmware 3.10 or higher. Currently the USB PC Watchdog Aux relay is the only one that supports this option for now.

Example:

```
WD_HANDLE wdHandle ;
WD_STATUS wdStatus ;
UINT32 iRlySet, iRlyGet, i ;

// Turn On relay
iRlySet = (WD_AUX_RLY_ON | WD_AUX_RLY_EN) ;
// store pulse at reset
iRlySet |= (WD_AUX_RLY_PULSE | WD_AUX_RLY_RST_EN) ;
wdStatus = WD_GetSetAuxRelay(wdHandle, iRlySet, &iRlyGet);
if(wdStatus == WD_OK)
{
    for(i=0; i<0x00ffffff; i++) // short loop - hear relay click
        ;
    printf("-- AUXRELAY -- Relay Status was = 0x%04X\n", iRlyGet) ;
}

// leave relay on & nv store pulse at reset
// This time we will see the new status we wrote last time
iRlySet = 0 ; // no new output actions
wdStatus = WD_GetSetAuxRelay(wdHandle, iRlySet, &iRlyGet);
if(wdStatus == WD_OK)
{
    printf("-- AUXRELAY -- Relay Status is = 0x%04X\n", iRlyGet);
}

// Turn Off relay
iRlySet = WD_AUX_RLY_EN ;
// turn off pulse at reset
iRlySet |= WD_AUX_RLY_RST_EN ;
wdStatus = WD_GetSetAuxRelay(wdHandle, iRlySet, &iRlyGet);

// Enable Aux Relay ON at first high temp trip point
iRlySet = WD_AUX_RLY_HTEMP_EN | WD_AUX_RLY_HTEMP_ON ;
wdStatus = WD_GetSetAuxRelay(wdHandle, iRlySet, &iRlyGet);
```

2.26 **WD_GetSetRstToAuxRelay**

The External Serial PC Watchdog has a reset relay available on the 1/8" stereo jack. This function allows a user to control the relay and make it operate like the Aux relay on the other versions of the PC Watchdog. The returned status is the state of the relay **before** any changes are applied except for relay on status.

There is a non-volatile memory option to force immediate inversion every time the watchdog powers up (the watchdog will power up with the relay on). Then if power is lost the relay will turn off. There is also an option to store a non-volatile memory options to pulse the relay or have it latch on after the watchdog times out.

Since this function allows multiple relay options the flags have been designed to allow a subset of actions without disturbing prior settings. Therefore each action requires an ENable flag and an action flag. As an example: if you just want to turn on the relay then call the function with the WD_RSTAUX_RLY_ONOFF_EN flag set along with the WD_RSTAUX_RLY_ON set. To turn off the relay just set the WD_RSTAUX_RLY_ONOFF_EN flag and leave the WD_RSTAUX_RLY_ON flag clear.

**** NOTE ****

Once this function has been called with any flags for set operations (except the two option Clear flags), the relay will no longer act like a timeout reset relay. There are two flags provided to force the watchdog to use the relay as a timeout relay again. One flag clears the options as long as the watchdog is powered on. The other flag clears all the non-volatile options as well for the next power up.

**WD_STATUS WD_GetSetRstToAuxRelay(WD_HANDLE wdHandle,
UINT32 iRelaySet, UINT32* pRelayGet)**

Parameters:

wdHandle – Handle of the device from WD_Open().
iRelaySet – flags sent for control.
pRelayGet – pointer to flags returned from the board

Return Value:

WD_OK if successful or a WD error code.

Flags for Set operations:

- WD_RSTAUX_RLY_INVRT_EN – Set this flag to enable changing (setting or clearing) the inversion.
- WD_RSTAUX_RLY_RST_EN – Set this flag to enable changing the relay action at timeout.
- WD_RSTAUX_RLY_ONOFF_EN – Set this flag to enable changing (on or off) the relay.
- WD_RSTAUX_CLEAR_OPTION – if this flag is set then all other flags (except WD_RSTAUX_CLEAR_NVOPT) are ignored and the relay will revert back to a normal timeout relay operation as long as the watchdog is powered. If non-volatile memory options are present they will activate on the next power up cycle.
- WD_RSTAUX_CLEAR_NVOPT – clears the no-volatile memory options as well. All other flags (except WD_RSTAUX_CLEAR_OPT) are ignored.

Flags for Get & Set operations:

- WD_RSTAUX_RLY_ISAUX – if this flag is set after a function call then it indicates that the relay is acting as an Aux relay, not a timeout relay.
- WD_RSTAUX_RLY_INVRT – set this flag (along with WD_RSTAUX_INVRT_EN) for aux relay inversion enable at power-up only. This option is stored in non-volatile memory. This flag will be returned in **pRelayGet**.
- WD_RSTAUX_RLY_LATCH – set this flag (along with WD_RSTAUX_RLY_RST_EN) to make the relay latch on after the watchdog reboots the PC. This option is stored in non-volatile memory. This flag will be returned in **pRelayGet**.
- WD_RSTAUX_RLY_PULSE – set this flag (along with WD_RSTAUX_RLY_RST_EN) to make the relay pulse (same length as reset relay) after the watchdog times out. This option is stored in non-volatile memory. This flag will be returned in **pRelayGet**. In essence this actually makes the relay act like the normal timeout relay.
- WD_RSTAUX_RLY_ON – this flag must be set (with WD_RSTAUX_RLY_ONOFF_EN) to turn the relay on. If clear the relay will be turned off. This flag will be returned in **pRelayGet**.

**** FIRMWARE / DLL NOTES ****

The External Serial PC Watchdog must have firmware 3.10 or higher to use this function.

The Universal DLL must be 01.20.00 or higher to support this function.

Example:

```
WD_HANDLE wdHandle ;
WD_STATUS wdStatus ;
UINT32 iRlySet, iRlyGet, i ;

// Turn On relay
iRlySet = (WD_RSTAUX_RLY_ON | WD_RSTAUX_RLY_ONOFF_EN) ;
// store pulse at reset
iRlySet |= (WD_RSTAUX_RLY_PULSE | WD_RSTAUX_RLY_RST_EN) ;
// store invert at power on
//iRlySet |= (WD_RSTAUX_RLY_INVRT | WD_RSTAUX_RLY_INVRT_EN) ;
wdStatus = WD_GetSetRstToAuxRelay(wdHandle, iRlySet, &iRlyGet);
if(wdStatus == WD_OK)
{
    for(i=0; i<0x00ffffff; i++) // short loop to hear relays click
        ;
    printf("\t-- RSTAUXRLY -- Relay Status was = 0x%04X\n", iRlyGet);
}

// leave relay on & nv store pulse at reset
// This time we will see the new status we wrote last time
iRlySet = 0 ; // no new output actions
wdStatus = WD_GetSetRstToAuxRelay(wdHandle, iRlySet, &iRlyGet);
if(wdStatus == WD_OK)
{
    printf("\t-- RSTAUXRLY -- Relay Status is = 0x%04X\n", iRlyGet) ;
}

// Turn Off relay
iRlySet = WD_RSTAUX_RLY_ONOFF_EN ;
// turn off pulse at reset
iRlySet |= WD_RSTAUX_RLY_RST_EN ;
wdStatus = WD_GetSetRstToAuxRelay(wdHandle, iRlySet, &iRlyGet);

// Reset relay back to a timeout (Reset) relay
iRlySet = WD_RSTAUX_CLEAR_OPTION ;
iRlySet |= WD_RSTAUX_CLEAR_NVOPT ;
wdStatus = WD_GetSetRstToAuxRelay(wdHandle, iRlySet, &iRlyGet);
```

2.27 **WD_GetDigitalIn**

The Ethernet-USB and the USB hardware Rev-C boards have a second Auxiliary jack with a digital input. This input can be tested for the current input level or it can be used as an external trigger to reset the watchdog countdown timer. This function allows you to program the input as a re-trigger (“tickle”) input and you can also save this setting in non-volatile memory to enable it at every power up.

On the PCI board this input is called the External Trigger input on Pin 21 of the DB-25. PCI boards with firmware greater than 3.xx now allow you to get more control of this pin with this function. On the PCI boards the normal operation of this input is always enabled as an external “tickle” input. You can now use this function to read the input level and disabled the trigger option.

The inputs on the boards accept levels from 0.0 to +5.0V and they have a pull up resistor to keep the input high when it is unconnected. The pull up also allows the user to connect a dry contact switch to the jack to pull the input low.

The only value passed to this function is a set of flags to configure the digital input for edge sensing for “tickling” the watchdog.

**WD_STATUS WD_GetDigitalIn(WD_HANDLE wdHandle, UINT32* pExtCount
UINT32* pDi, UINT32 iFlag)**

Parameters:

wdHandle – Handle of the device from WD_Open().

pExtCount – pointer to 16 bit value for the count of the number of external “tickle” triggers received. This count rolls over to zero when it hits 0xffff. This function also allows you to clear the counter.

pDi – pointer to get digital input flags returned

iFlag – flag bits for Set edge configure options

Return Value:

WD_OK if successful or a WD error code.

Flags for Set operations:

WD_DIG_CLR_EXTCNT – if this flag is set the board will clear the External Trigger Counter after it returns the current value.

WD_DIG_IN_EDGE_EN – this flag (bit) must be set to change the current edge sensing.

WD_DIG_IN_EDGE – if this flag is set (along with DIG_IN_EDGE_EN), the board will enable edge sense “tickles” on the digital input. Clear this flag to disable them.

WD_DIG_IN_NVWR_EN – this bit must be set in order to update the non-volatile memory setting for edge triggers.

WD_DIG_IN_NVEDGE – if this flag is set (along with DIG_IN_NVWR_EN), the board will enable edge trigger in non-volatile memory. Clear this flag to clear non-volatile memory.

Flags returned in pDi:

WD_DIG_IN_EDGE_ACT – edge detection is currently active

WD_DIG_IN_NVEDGE_ACT – non-volatile edge enable is active

WD_DIGITAL_IN_EDGE – edge (falling) captured on digital input. Note that the board also checks and clears this status if it is doing edge detection for “tickles”.

WD_DIGITAL_IN_HIGH – digital input is high – otherwise it is low.

WD_DIG_ERROR – Board is wrong firmware or hardware revision.

**** NOTE **** The firmware on the board polls the digital input. If you supply input pulses too quickly, the board may miss some and report a lower External Count.

Example:

```
WD_HANDLE wdHandle ;
WD_STATUS wdStatus ;
UINT32 iExtCnt, iDi, iFlag;

iFlag = 0 ;          // no flags for edge or nv edge enable
wdStatus = WD_GetDigitalIn(wdHandle, &iExtCnt, &iDi, iFlag) ;
if(wdStatus == WD_OK)
{
    printf("Digital input bits = 0x%04X\n", iDi) ;
    if(iDi & WD_DIGITAL_IN_HIGH)
        printf("-- DIGITALIN -- Input is High\n") ;
    else
        printf("-- DIGITALIN -- Input is Low\n") ;
}
```

2.28 **WD_GetAnalogIn**

This function returns the reading from the analog input on the PCI DB-25 (DSub) connector or the analog input on the Ethernet-USB connector. The analog value is 8 bits.

On the PCI board the span range is 0.0 to +5.0V and the Ethernet-USB allows 0.0 to+3.3V. Do not exceed these levels on the inputs.

WD_STATUS WD_GetAnalogIn(WD_HANDLE wdHandle, UINT32* pAi)

Parameters:

wdHandle – Handle of the device from WD_Open().
pAi – pointer to 8 bit value for the analog input value.

Return Value:

WD_OK if successful or a WD error code.

Example:

```
WD_HANDLE wdHandle ;
WD_STATUS wdStatus ;
UINT32 iAnIn, iType;

wdStatus = WD_GetAnalogIn(wdHandle, &iAnIn) ;
wdStatus += WD_GetWDogType(wdHandle, &iType);
if(wdStatus == WD_OK)
{
    if(iType == WD_TYPE_ETH_USB)
        iAnIn = ((iAnIn * 330)/255) ;           // 330 = 3.3V * 100
    else
        iAnIn = ((iAnIn * 500)/255) ;         // 500 = 5.0V * 100
    printf("Analog In = %d.%02dV\n", (iAnIn/100), (iAnIn % 100));
}
```

2.29 **WD_GetSetPciDigitalInOut** (PCI Only)

This function allows you to read the 4 general purpose digital inputs and set the four general purpose digital outputs on the PCI DB-25 (DSub) connector. The digital output data is in the lowest four bits of the iDigOut variable. The data read back in pDigIn is 8 bits. The lower 4 bits reflect what was on the output port **before** your new data output write. The upper four bits of iDigIn reflect the current input values on the digital inputs. The digital input pins DI3-0 map to D7-4 of pDigIn.

**** NOTE **** This function only works on PCI boards.

WD_STATUS **WD_GetSetPciDigitalInOut**(WD_HANDLE wdHandle, UINT32 iDigOut, UINT32* pDigIn, UINT32 iFlag)

Parameters:

wdHandle – Handle of the device from WD_Open().

iDigOut – 4 bit value to set the digital output pins.

pDigIn – lower 4 bits (D3-0) are the current digital output values **before** the write. The next 4 bits (D7-4) are the digital inputs.

iFlag – flag to enable writing output port data.

Return Value:

WD_OK if successful or a WD error code.

Flag:

WD_PCI_DIG_OUT_EN – this flag must be set to enable writing the digital outputs.

**** NOTE **** The IO pins on the PCI board have electrical inversion. The output pins are driven by open-collector drivers with pull-up resistors. The reset state of the board sets the output bits to zero (0) which gets inverted by the drivers so the outputs are pulled high. Writing a one (1) to an output pin cause the driver to turn on and pull to ground potential.

The input pins have their inputs pulled up with resistors to +5.0V. Inversion causes the input bits to read as zero. If you pull an input pin low to ground (ex: dry contact switch) then the input bit will read as a one (1).

Example:

```
WD_HANDLE wdHandle ;
WD_STATUS wdStatus ;
UINT32 iDigOut, pDigIn, iFlag ;

iDigOut = 0x06 ;           // set two of the bits
iFlag = WD_PCI_DIG_OUT_EN ;
wdStatus = WD_GetSetPciDigitalInOut(wdHandle, iDigOut, &pDigIn,
                                     iFlag);
if(wdStatus == WD_OK)
{
    printf("-- DIG OUT -- Prior Digital Out = 0x%02X\n",
           (pDigIn & 0x0f)) ;
    printf("-- DIG IN  -- Current Digital In = 0x%02X\n",
           ((pDigIn & 0xf0) >> 4)) ;
}
iFlag = 0 ;               // now do read only
wdStatus = WD_GetSetPciDigitalInOut(wdHandle, iDigOut, &pDigIn,
                                     iFlag);
if(wdStatus == WD_OK)
{
    printf("-- DIG OUT -- Current Digital Out = 0x%02X\n",
           (pDigIn & 0x0f)) ;
}
}
```

2.30 **WD_GetSetPciRelays** (PCI Only)

This function gives you some control of the two relays on the board. The operation of the two relays is also governed by the setting of the DIP Switch options as well. Turning the relays on and off is done by firmware on the board. The returned status is the state of the relays **before** any changes are applied.

Relay #1 has the option of being inverted. When it is inverted it will do the opposite of the DIP Switch options at the next watchdog reboot. There is also a non-volatile memory option to force immediate inversion every time the board powers up. Note that inversion does not apply to the ON/OFF operation in this function. If you send a relay #1 ON command then the relay actually turns on. If you know that you have inversion in effect make appropriate changes in your call to this function.

The board also has a hardware option to allow you to get exclusive control of relay 2 completely locking out the processor on the board. This function also provides the hardware options which always override the software.

WD_STATUS WD_GetSetPciRelays(WD_HANDLE wdHandle, UINT32 iRelaySet, UINT32* pRelayGet)

Parameters:

wdHandle – Handle of the device from WD_Open().
iRelaySet – flags sent for control.
pRelayGet – pointer to flags returned from the board

Return Value:

WD_OK if successful or a WD error code.

Flags for Set operations:

WD_PCI_INVRT_EN – set this flag for relay #1 inversion enable. Note that the DIP Switch options will be inverted at the next watchdog reboot and the board will power up with Relay #1 on. If you need inversion immediately you should use the flags to set the relay #1 on as well.

WD_PCI_HDW2_EN – this flag must be set to force exclusive use of Relay #2. It will be used to set or clear the hardware Port #1 options on the board.

WD_PCI_RLY2_EN – this flag must be set to turn the relay #2 on or off. This Relay #2 flag is the software option done by firmware on the board.

WD_PCI_RLY1_EN – this flag must be set to turn the relay #1 on or off. Relay #1 is always a software option done by firmware on the board.

Flags for Set and Get operations:

- WD_PCI_NV_INVRT – set this flag (along with WD_PCI_INVRT_EN) if you want the inversion option you select with WD_PCI_INVRT_ON to be saved in non-volatile memory also.
- WD_PCI_INVRT_ON - set this flag (along with WD_PCI_INVRT_EN) if you want relay #1 operation to become inverted at the next watchdog reboot.. **pRelayGet** will contain this flag if the option is enabled.
- WD_PCI_HDW_EXCL_ON - set or clear this flag (along with WD_PCI_HDW2_EN) if you want to gain exclusive control of relay #2. **pRelayGet** will contain this flag if relay #2 exclusive control is currently on. This is the hardware option to gain control of Relay #2 and uses the **R2DS** bit in hardware Port #2.
- WD_PCI_HDW_RLY2_ON - set or clear this flag (along with WD_PCI_HDW2_EN) if you want to turn relay #2 on. If it is clear the relay will be turned off. **pRelayGet** will contain this flag if relay #2 is currently on. This is the hardware option to turn on Relay #2 and uses the **RLY2** bit in hardware Port #2.
- WD_PCI_RLY2_ON - set or clear this flag (along with WD_PCI_RLY2_EN) if you want to turn relay #2 on or off. If it is clear the relay will be turned off. **pRelayGet** will contain this flag if relay #2 is currently on. This is the firmware option to turn on Relay #2.
- WD_PCI_RLY1_ON - set or clear this flag (along with WD_PCI_RLY1_EN) if you want to turn relay #1 on or off. If it is clear the relay will be turned off. **pRelayGet** will contain this flag if relay #1 is currently on.

Example:

```
WD_HANDLE wdHandle ;
WD_STATUS wdStatus ;
UINT32 iRlySet, iRlyGet, i ;

// Firmware turn On relay #1
iRlySet = (WD_PCI_RLY1_EN | WD_PCI_RLY1_ON) ;
// Hardware turn on relay #2 & exclusive
iRlySet |= (WD_PCI_HDW2_EN | WD_PCI_HDW_RLY2_ON
           | WD_PCI_HDW_EXCL_ON) ;
// Ask for relay #1 inversion and save in nv memory
iRlySet |= (WD_PCI_INVRT_EN | WD_PCI_INVRT_ON | WD_PCI_NV_INVRT) ;
wdStatus = WD_GetSetPciRelays(wdHandle, iRlySet, &iRlyGet);
if(wdStatus == WD_OK)
{
    for(i=0; i<0x00ffffff; i++) // short loop to hear relays click
        ;
    printf("\t-- PCIRELAY -- Relay Status was = 0x%04X\n",
           iRlyGet) ;
}
// just read new status now
iRlySet = 0 ;
wdStatus = WD_GetSetPciRelays(wdHandle, iRlySet, &iRlyGet);
if(wdStatus == WD_OK)
{
    printf("\t-- PCIRELAY -- New Relay Status = 0x%04X\n",
           iRlyGet) ;
}
// finally undo everything
// Firmware turn Off relay #1
iRlySet = WD_PCI_RLY1_EN ;
// Hardware turn off relay #2 & exclusive
iRlySet |= WD_PCI_HDW2_EN ;
// turn relay #1 inversion and clear nv memory
iRlySet |= (WD_PCI_INVRT_EN | WD_PCI_NV_INVRT) ;
wdStatus = WD_GetSetPciRelays(wdHandle, iRlySet, &iRlyGet);
```

2.31 **WD_GetSetPowerModule** (External Serial Only)

This functions allows you to turn on or turn off the AC power module and get the status.

**** NOTE **** This function only works with an External Serial PC Watchdog. Status is only returned with firmware that is 3.05 or higher.

WD_STATUS WD_GetSetPowerModule(WD_HANDLE wdHandle, UINT32 iFlagSet, UINT32* pFlagGet)

Parameters:

wdHandle – Handle of the device from WD_Open().

iFlagSet – flags for module off and on.

pFlagGet – flags that reflect the status on the power module *prior* to the function call.

Return Value:

WD_OK if successful or a WD error code.

Flags Get/Set Operations:

WD_POWERMODULE_ON – set this flag to turn on the AC power module or status was on.

WD_POWERMODULE_OFF – set this flag to turn it off or status was off.

**** NOTE **** If both flags are set the module will be on. If both flags are clear then function just returns the current power module status.

Example:

```
WD_HANDLE wdHandle ;
WD_STATUS wdStatus ;
UINT32 iFlagSet, iFlagGet ;

iFlagSet = WD_POWERMODULE_OFF ;
iFlagGet = 0 ;
wdStatus = WD_GetSetPowerModule(wdHandle, iFlagSet, &iFlagGet);
if(wdStatus == WD_OK)
{
    if(iFlagGet & WD_POWERMODULE_ON)
        printf("Power Module now Off. Prior state was ON\n") ;
}
```

2.32 **WD_SetIpAddresses** (Ethernet-USB Only)

This function allows you to set a new IP address for the Ethernet interface. It also allows you to store a fixed set of IP addresses in the non-volatile memory on the board and use the DIP Switch option to make them active at power up. This function will allow you to set IP addresses and enable the Ethernet port even if the Ethernet Mode switches are both off for: DISABLED.

NOTE: This command (function) will not work over the Ethernet interface. The addresses can only be set through the USB interface.

This function will allow any values for the IP, Subnet, and Gateway addresses. It is up to you to make sure they are valid.

These byte arrays of addresses should be in Network order – not Intel (PC) reversed.

```
WD_STATUS WD_SetIpAddresses(WD_HANDLE wdHandle, UINT32 iFlag,  
    UCHAR* pIpAddress, UCHAR* pSubMskAddress,  
    UCHAR* pGwAddress, UCHAR* pUnused)
```

Parameters:

wdHandle – Handle of the device from WD_Open().

iFlag – flags to enable activation of new addresses and non-volatile memory write.

pIpAddress – pointer to a 4 byte IP address array to be written.

pSubMskAddress – pointer to a 4 byte Subnet Mask address.

pGwAddress – pointer to a 4 byte Gateway address.

pUnused – Unused pointer

Return Value:

WD_OK if successful or a WD error code.

Flag:

WD_IP_ACTIVATE_NOW – if this flag is set the watchdog will reset the Ethernet interface and start using these addresses.

WD_NV_IPADD_WREN – if this flag is set the addresses will be written to non-volatile memory.

Example:

```
WD_HANDLE wdHandle ;
WD_STATUS wdStatus ;
UCHAR IpAddress[4] = {192, 168, 1, 75} ;
UCHAR SubMskAddress[4] = {255, 255, 255, 0};
UCHAR GwAddress[4] = {192, 168, 1, 1};

iFlag = WD_IP_ACTIVATE_NOW ;          // enable Ethernet now
wdStatus = WD_SetIpAddresses(wdHandle, iFlag, IpAddress,
                             SubMskAddress, GwAddress, NULL);
if(wdStatus == WD_OK)
{
    printf("-- IP ADD -- Set to: %0d.%0d.%0d.%0d\n",
           (int)IpAddress[0], (int)IpAddress[1],
           (int)IpAddress[2], (int)IpAddress[3]) ;
}
```

2.33 **WD_GetIpAddresses** (Ethernet-USB Only)

This function allows you to get the current active IP address for the Ethernet interface or get the ones stored in the non-volatile memory.

NOTE: This command (function) will not work over the Ethernet interface.

These byte arrays of addresses will be in Network order – not Intel (PC) reversed.

**WD_STATUS WD_GetIpAddresses(WD_HANDLE wdHandle, UINT32 iFlag,
UCHAR* pIpAddress, UCHAR* pSubMskAddress,
UCHAR* pGwAddress, UCHAR* pUnused)**

Parameters:

wdHandle – Handle of the device from WD_Open().
iFlag – flag for read of current or non-volatile memory
pIpAddress – pointer to a 4 byte IP address array to be read.
pSubMskAddress – pointer to a 4 byte Subnet Mask address.
pGwAddress – pointer to a 4 byte Gateway address.
pUnused – Unused pointer (set to NULL)

Return Value:

WD_OK if successful or a WD error code.

Flag:

WD_NV_IPADD_RDEN – if this flag is set the addresses will be read from non-volatile memory, otherwise they will be the current in effect.

Example:

```
WD_HANDLE wdHandle ;
WD_STATUS wdStatus ;
UCHAR IpAddress[4] = {0, 0, 0, 0} ;
UCHAR SubMskAddress[4] = {0, 0, 0, 0};
UCHAR GwAddress[4] = {0, 0, 0, 0};

iFlag = WD_NV_IPADD_RDEN ; // get the non-volatile
wdStatus = WD_GetIpAddresses(wdHandle, iFlag, IpAddress,
                             SubMskAddress, GwAddress, NULL);
if(wdStatus == WD_OK)
{
    printf("\t-- NV IP ADD -- %0d.%0d.%0d.%0d\n", (int)IpAddress[0],
           (int)IpAddress[1], (int)IpAddress[2], int)IpAddress[3]) ;
}
```

2.34 **WD_GetSetNvUdpPortNum** (Ethernet-USB Only)

This function allows you to get and set the port number that the board uses for UDP communications on the Ethernet. The watchdog board defaults to listening to the UDP port number 55108. If there is a conflict on your network then pass a new port number in the **iPort** parameter. Pass a value of zero to keep the default or go back to the default. The PC will use a source port number one less than the board. The default then is 55107.

This value is stored in non-volatile memory. If you make a change to the port number the board will automatically restart the UDP socket with the new port number.

NOTE: This command (function) will not work over the Ethernet interface.

WD_STATUS WD_GetSetNvUdpPortNum(WD_HANDLE wdHandle, UINT32 iFlag, UINT32 iPort, UINT32* pCurPort)

Parameters:

wdHandle – Handle of the device from WD_Open().

iFlag – flag for write enable of new port number

iPort – new port number. Set to zero to use default port number 55108.

pCurPort – pointer to return port number in use – may be what you just wrote.

Return Value:

WD_OK if successful or a WD error code.

Flag:

WD_NV_UDP_PORT_WREN – if this flag is set, the value in iPort will be written to non-volatile memory and the UDP socket will be restarted with the new number. If it is clear then the function just returns the current port number.

Example:

```
WD_HANDLE wdHandle ;
WD_STATUS wdStatus ;
UINT32 iFlag, iPort, iCurPort ;

iFlag = WD_NV_UDP_PORT_WREN ;           // write new port
iPort = 65000 ;
wdStatus = WD_GetSetNvUdpPortNum(wdHandle, iFlag, iPort, &iCurPort);
if(wdStatus == WD_OK)
{
    printf("-- UDP Port -- Port set to %0d\n", iCurPort) ;
}
```

2.35 **WD_GetMacAddress** (Ethernet-USB Only)

This function gets the MAC address for the Ethernet interface.

NOTE: This command (function) will not work over the Ethernet interface.

The byte array for the address will be in Network order – not Intel (PC) reversed.

**WD_STATUS WD_GetMacAddress(WD_HANDLE wdHandle,
CHAR* pMacAddress)**

Parameters:

wdHandle – Handle of the device from WD_Open().

pMacAddress – pointer to a 6 byte MAC address array to be read.

Return Value:

WD_OK if successful or a WD error code.

Example:

```
WD_HANDLE wdHandle ;
WD_STATUS wdStatus ;
UCHAR pMacAddress[6] ;

wdStatus = WD_GetMacAddress(wdHandle, pMacAddress) ;
if(wdStatus == WD_OK)
{
    printf("\t-- MAC ADD -- %02X:%02X:%02X:%02X:%02X:%02X\n",
        pMacAddress[0], pMacAddress[1], pMacAddress[2],
        pMacAddress[3], pMacAddress[4], pMacAddress[5]) ;
}
```

2.36 **WD_ResetRebootEthernet** (Ethernet-USB Only)

This function allows you to reset the Ethernet hardware interface and leave it disabled, or reset it then reboot to restart.

NOTE: This command (function) will not work over the Ethernet interface.

**WD_STATUS WD_ResetRebootEthernet(WD_HANDLE wdHandle,
UINT32 iFlag)**

Parameters:

wdHandle – Handle of the device from WD_Open().

iFlag – Flags for reset and reboot

Return Value:

WD_OK if successful or a WD error code.

Flags:

WD_RESET_ETHERNET – if this flag is set the Ethernet hardware will be reset and left disabled unless the REBOOT flag is set.

WD_REBOOT_ETHERNET – if this flag is set the Ethernet hardware will be software reset and then rebooted per the dip switch settings if the following OverRide flags are clear.

WD_OVRR_FIXED_IP – if this flag is set the Ethernet will reboot with the fixed IP address and ignore the dip switches.

WD_OVRR_NVMEM_IP – if this flag is set the Ethernet will reboot with the non-volatile memory IP address and ignore the dip switches. *Make sure you have a valid IP address stored in non-volatile memory!*

WD_OVRR_DHCP_IP – if this flag is set the Ethernet will reboot looking for a DHCP server to provide the IP address and ignore the dip switches.

NOTE: The #defines for the overrides just match the dip switch values. The overrides must be used with the REBOOT flag – by themselves they will do nothing.

Example:

```
WD_HANDLE wdHandle ;
WD_STATUS wdStatus ;
UINT32 iFlag;

iFlag = WD_RESET_ETHERNET ;           // force hardware reset
iFlag |= WD_REBOOT_ETHERNET ;         // then software reset & reboot
iFlag |= WD_OVRR_FIXED_IP ;           // override dip switch
wdStatus = WD_ResetRebootEthernet(wdHandle, iFlag);
if(wdStatus == WD_OK)
{
    printf("-- ETH RST -- Ethernet Reset/Reboot OK\n") ;
}
```

2.37 **WD_GetSetUsbSuspendMode** (Ethernet-USB Only)

This function deals with the USB suspend mode which happens when the PC restarts. In the normal mode the board will detect the suspend and re-enter the **POD** (Power-On-Delay) mode and wait for the suspend to finish. This allows you to reset the computer with the windows option without disabling the board first or having the board inadvertently reset the PC. If the suspend lasts longer than 30 seconds the board will reset the computer assuming it locked up during reboot. While the suspend is active the board will flash both the USB-COMM & ETH-COMM LEDs.

This function provides two non-volatile memory options. One to ignore suspend completely and the other to change the suspend wait time. The time can be between 1 and 255 (0x01 – 0xff) seconds. A value of zero returns the board back to the 30 Seconds default.

NOTE: This command (function) will not work over the Ethernet interface.

```
WD_STATUS WD_GetSetUsbSuspendMode(WD_HANDLE wdHandle,  
                                     UINT32 iFlag, UINT32 iSuspTime,  
                                     UINT32* pCurFlag, UINT32* pCurSuspTime)
```

Parameters:

wdHandle – Handle of the device from `WD_Open()`.

iFlag – Flags for disable and time set.

iSuspTime – the new suspend wait time value.

pCurFlag – pointer to get the `SUSPEND_IGNORE` flag status back.

pCurSuspTime – pointer to get back the current suspend wait time. If the return is zero then the board is using the default of 30 seconds.

Return Value:

`WD_OK` if successful or a `WD` error code.

Flags:

`WD_SUSPEND_IGNORE` – if this flag is set the USB board will ignore the USB suspend signal and continue its current operation. This flag can be returned if enabled.

`WN_NV_IGNORE_WREN` – flag must be set to store the ignore status. If clear then the board just returns the current status.

`WN_NV_SUSP_TIME_WREN` – if this flag is set the new time value will be written to non-volatile memory. If the time is zero and this flag is set, the board sets the 30 second default.

Example:

```
WD_HANDLE wdHandle ;
WD_STATUS wdStatus ;
UINT32 iFlag, iSuspTime, iCurFlag, iCurSuspTime;

iFlag = WN_NV_SUSP_TIME_WREN ; // write suspend time
iSuspTime = 45 ; // new time
iFlag |= WN_NV_IGNORE_WREN ; // test only
iFlag |= WD_SUSPEND_IGNORE ; // ignore suspend
wdStatus = WD_GetSetUsbSuspendMode(wdHandle, iFlag, iSuspTime,
                                     &iCurFlag, &iCurSuspTime) ;
if(wdStatus == WD_OK)
{
    printf("-- USB MODE -- New Suspend Timer = %d\n",
           iCurSuspTime) ;
    if(iCurFlag & WD_SUSPEND_IGNORE)
        printf("-- USB MODE -- Suspend Ignored\n") ;
}
}
```

2.38 **WD_GetSetNvEtherAllowed** (Ethernet-USB USB Only)

Most of the commands that you can send the Ethernet-USB board via the USB side interface can also be sent via the Ethernet port. This function allows you to tell the board which functions (commands) should be allowed on the Ethernet side. See section 3.3 for the Structure of bits definition for commands allowed.

This structure value is always stored in the non-volatile memory of the board. The boards are shipped with the whole structure set to ones so that all commands (and future commands) are allowed.

The structure is composed of one 64 bit integer (64 functions/commands). The address of the structure is passed as a byte array of 8 bytes.

WD_STATUS WD_GetSetNvEtherAllowed(WD_HANDLE wdHandle, UINT32 iFlag, UCHAR* pNvCmdAllow, UCHAR* pNvCurCmdAllow)

Parameters:

wdHandle – Handle of the device from WD_Open().

iFlag – flag to enable writing the non-volatile memory codes

pNvCmdAllow – pointer to a 8 byte (char) array to be written if the flag is set.

pNvCurCmdAllow – pointer to a 8 byte (char) array retrieve the current non-volatile allow bits.

In case of a write, it will equal what you just wrote.

Return Value:

WD_OK if successful or a WD error code.

Flag:

WD_ETH_ALLOW_WREN – this flag must be set to store the new code value. If it is clear then the function only returns the current stored value in non-volatile memory

Example:

```
WD_HANDLE wdHandle ;
WD_STATUS wdStatus ;
UINT32 iFlag;
UINT64 *sp ;
WD_E_COMMANDS_ALLOWED_1 structEthAllow , structCurEthAllow;

iFlag = WD_ETH_ALLOW_WREN ;          // enable write to non-volatile
sp = (UINT64*)&structEthAllow ;
*sp = 0xffffffffffffffff ;          // enable all
structEthAllow.E_Allow_WD_GetTempTickle = 0 ; // not allow a tickle
wdStatus = WD_GetSetEtherAllowed(wdHandle, iFlag,
                                  (UCHAR*)&structEthAllow, (UCHAR*)&structCurEthAllow);
if(wdStatus == WD_OK)
{
    printf("-- ETH ALLOW -- Allow bits write OK - Bits =
           %016I64X\n", structCurEthAllow) ;
}
```

3. System Status / Information Flags

These flags and their actual bits are also listed in the header (.h) file.

3.1 Status Flags

WD_STAT_ACTIVE_ARMED - the watchdog is armed and done with POD time.

WD_STAT_POD_ACTIVE - the watchdog is still in 2.5 minute (or user time) delay.

WD_STAT_POD_DSW_DELAY – indicates that the board has finished the POD time and is waiting for the first “tickle”.

**** NOTE **** Early firmware versions of the USB and Serials will not report this status.

WD_STAT_CMD_DISABLED - the watchdog has acknowledged the disable command.

**** NOTE **** Early firmware versions of the USB and Serials will not report this status.

WD_STAT_RESET_PEND – set by the EnableDisablePcReset function to show that a command to reset the PC has been issued with a delay time.

WD_STAT_ETH_ENABLED – shows that the Ethernet IC has been configured and enabled. In case of DHCP, an IP address may not have been assigned.

WD_STAT_ETH_IP_SET – shows that the Ethernet interface now has a valid IP address.

3.2 Diagnostic Flags

WD_DIAG_TEMP_OK – This bit should always be set if the watchdog has temperature reporting. Early versions of the serial watchdog did not have this. If it is clear the temperature sensor IC on the board has failed.

WD_DIAG_NVMEM_OK – This bit should always be set. If it is clear the non-volatile memory IC on the board has failed or a checksum error was found resulting in the memory being cleared. If this bit stays set even after cycling power on the board then the memory has failed and should be returned for repair. Early versions of the serial watchdog did not have this.

WD_DIAG_ETHER_OK – This bit should always be set. If it is clear the Ethernet IC on the board has failed. Board should not be used and should be returned for repair.

WD_DIAG_NV_CORRUPTED – This bit should never be set. If it is set then the Non-Volatile memory data has been corrupted and failed a checksum test. The firmware will try to reset the EEPROM non-volatile data and rewrite the checksum. All non-volatile options will be cleared. If you reset the watchdog and this error persists then the memory has failed and the board needs to be repaired.

WD_DIAG_NV_WRITE_FAIL – This flag will be set if there is a failure writing to the non-volatile memory. The IC and the firmware have built in safeguards to prevent bad writes. If the firmware detects any problems – including a low voltage situation - it will not write the data. If the next write attempt is OK this flag will stay set. A reset clears the flag.

WD_DIAG_MAC_INVALID – This bit should never be set. If it is set then the board has found a problem with the MAC address and will not start the Ethernet interface. Contact Berkshire about trying to recover the MAC address.

WD_DIAG_ARP_ERROR – If this bit is set the board has found another device on the network with the same IP address. The Ethernet interface will be reset and left inactive. You will need to correct the problem and then call the Ethernet Reset/Reboot function.

3.3 Commands Allowed on Ethernet

This is the structure for the commands allowed bits to enable commands to be processed from the Ethernet port. Obviously you could also do these with #defines if you are careful to avoid duplicates. A bit set to one (1) means the command is enabled.

```
typedef struct{
    UINT64 E_Allow_WD_GetDeviceInfo           :1;    // Bit 0
    UINT64 E_Allow_WD_GetTempTickle         :1;    // Bit 1
    UINT64 E_Allow_WD_SetPowerOnDlyTimes    :1;    // Bit 2
    UINT64 E_Allow_WD_GetPowerOnDlyTimes    :1;    // Bit 3
    UINT64 E_Allow_WD_SetWdogTimes          :1;    // Bit 4
    UINT64 E_Allow_WD_GetWdogTimes          :1;    // Bit 5
    UINT64 E_Allow_WD_GetAnalogDigitalIn    :1;    // Bit 6
    UINT64 E_Allow_WD_GetSetAuxRelay        :1;    // Bit 7
    UINT64 E_Allow_WD_EnableDisable         :1;    // Bit 8
    UINT64 E_Allow_WD_GetResetCount         :1;    // Bit 9
    UINT64 E_Allow_WD_GetSetNvTempOffset    :1;    // Bit 10
    UINT64 E_Allow_WD_SetBuzzer             :1;    // Bit 11
    UINT64 E_Allow_WD_GetBuzzer             :1;    // Bit 12
    UINT64 E_Allow_WD_GetSetNvRelayPulse   :1;    // Bit 13
    UINT64 E_Allow_WD_GetSetNvUsbUserCode  :1;    // Bit 14
    UINT64 E_Allow_WD_EnableDisablePcReset :1;    // Bit 15
} WD_E_COMMANDS_ALLOWED_1 ;
```

Bits 16 - 63 have not been assigned yet.