Ethernet - USB
PC Watchdog™
Ethernet Interface
Programmer's Guide

BERKSHIRE PRODUCTS, INC.

Phone: 770-271-0088

http://www.berkprod.com/

Rev: 1.01
© Copyright 2008-2009

®PC Watchdog is a registered trademark of Berkshire Products

Table of Contents

1. Notes

This manuals covers the interface to the functions provided in the EUWDog-Eth-Int.dll file provided on the CD. This DLL only provides functions to interface to the Ethernet portion of the board. The USB DLL access is in the EUWDog-Int.dll file.

There are three additional files for use with C or C++. Two are the header files, EUWDog-Eth-Int.h and EUWDog-Common.h, with the defines and function definitions. The third is a library file, EUWDog-Eth-Int.lib, that allows link time binding to the DLL instead of using a Load command in the source.

There is a sample programs in C++ (Console AP) on the CD that can be used as a starting point for your own application. The C Console AP is in the Eth-TestAllDllConsoleAp directory.

There is not a sample VB.Net AP for the Ethernet side but you could easily modify the VB.NET ap provided in the VB_NET_DLL_Sample directory. Just modify the function names to WD_**E**_xxx and change the **Lib** definitions in the EU_WD_DLL_INT.vb file.

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.

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:

man1160feedback@berkprod.com

2. Functions

All the functions will return a status of type: WD_E_STATUS, defined in the header file.

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 the native 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.

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 **P**ower-**O**n-**D**elay state where it waits for 2.5 minutes (or a user time) to allow the PC to complete a re-boot.
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. The initial timer start can be delayed by a Dip Switch until the first "tickle" or the watchdog can be disabled by command function to **DISABLE** the countdown. In either case the watchdog is still considered to be in its Armed and Active state. Therefore it is possible to get the status flags: **WD_STAT_ACTIVE_ARMED & WD_STAT_CMD_DISABLED** both set.

NOTE: Most of these functions are the same as those that work through the USB interface. You will need the USB Programmer's manual for the full description of those functions and their parameters. If there are any differences they will be noted in this document. The function names have a prefix of **WD_E_** for the Ethernet functions. All flags for the common function retain the same name.

NOTE: Once the board has a valid IP address (fixed, non-volatile, or DHCP) it will respond to pings.

2.1 WD_E_Open

Open the Watchdog board and return a Handle that must used for all other function calls. This function should always be called first.

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. If you use a new port number then you must use the USB interface command (function) to change the boards default port number ahead of this operation. The PC will use a source port number one less than the board. The default then is 55107.

WD_E_STATUS WD_E_Open(WD_E_HANDLE *pwdeHandle, UINT32 iPort)

Parameters:

pwdHandle – pointer to a variable of type WD_E_HANDLE.
iPort – 16 byte replacement port number.

Return Value:

WD_E_OK if successful or a WD error code.

Example:

```
WD_E_HANDLE wdeHandle ;
WD_E_STATUS wdeStatus ;
UINT32 iPort = 0 ;

wdeStatus = WD_E_Open(&wdeHandle, iPort) ;
if(wdeStatus != WD_E_OK)
{
      // Error Handler
}
```

2.2 WD_E_Close

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

WD_E_STATUS WD_E_Close(WD_E_HANDLE wdeHandle)

Parameters:

wdHandle – Handle of the device from WD_Open().

Return Value:

WD_E_OK if successful or a WD error code.

Example:

```
WD_E_HANDLE wdeHandle ;
WD_E_STATUS wdeStatus ;

wdeStatus = WD_E_Close(wdeHandle) ;
if(wdeStatus != WD_E_OK)
{
      // Error Handler
}
```

2.3 WD_E_GetDllVersion

This function returns a 24 bit encoding of the DLL version. The high byte contains the major number, the middle byte is the minor and the lower byte is the sub-minor. For example version 1.09.03 would be returned as: 0x010903. This call does not require a handle so it can be called before a device open.

I 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 EUWDog-Int.h.

WD_E_STATUS WD_E_GetDllVersion(UINT32 *pDllVersion)

Parameters:

pDllVersion – pointer to a variable to save the version.

Return Value:

Always WD_E_OK.

Example:

```
UINT32 iVersion

WD_E_GetDllVersion(&iVersion) ;
printf("DLL Version: %02X.%02X.%02X\n", (iVersion >> 16),
      ((iVersion & 0xff00) >> 8),(iVersion & 0x00ff)),
```

2.4 WD_ E_Discover (ETH Only)

If you know the IP address of the board then pass it as a 4 byte array in network order. If you use DHCP, the board will ID itself to the DHCP server as "BPWdg---" where the "---" will match the last three hex digits of the MAC address and may be non-printable characters. You may be able to get the IP address from the DHCP client table?

If you do not know the IP address (DHCP assigned?), then set the IP address array bytes to zero and pass the MAC address and the DLL will do a broadcast inquiry to find the board and save the IP address.

When this function returns the IP address will be filled in if it was zero and/or the MAC address will be filled in if it was zero.

If you do not know either address, or you want to find multiple boards, then use the DiscoverAll described later.

NOTE: This function MUST be called (typically after Open) before you try to send the board any commands. This command only works with the Ethernet DLL.

WD_E_STATUS WD_E_Discover(WD_E_HANDLE wdeHandle,
UCHAR* pIpAddress, UCHAR* pMacAddress)

Parameters:

wdeHandle – Handle of the device from WD_Open().
pIpAddress – pointer to a 4 byte array for watchdog IP address.
pMacAddress – pointer to a 6 byte array for watchdog MAC address.

Return Value:

WD_OK if successful or a WD error code.

NOTE: If the IpAddress is non-zero, it will be checked first and the MacAddress parameter will be ignored. The MAC address will be returned if the board was found.

Example:

```
WD_HANDLE wdHandle ;
WD_STATUS wdStatus ;
UCHAR IpAdd[4] = {192,168,1,60} ;
UCHAR MacAdd[6] = {0,0,0,0,0,0} ;

wdeStatus = WD_E_Discover(wdeHandle, IpAdd, MacAdd);
if(wdeStatus == WD_E_OK)
    printf("Discover IP Parameters Success.\n") ;
```

2.5 WD_ E_DiscoverAll (ETH Only)

Use this function if you do not know the IP or MAC address of the board, or if you want to see how many boards are on the network. The function will do a broadcast message on the network and collect the IP and MAC addresses from all the boards that answer the broadcast.

Call this function with a byte array pointer to get the response data. Each board response requires 10 bytes in the array (structure) to hold the 4 byte IP address and the 6 byte MAC address. The other parameter you send is an integer value for the size of the buffer in 10 byte blocks. Example: if you send the value of 4 then your array pointer must point to a byte array of at least 40 bytes.

If the iBlockCnt value is less than the number of watchdogs on the network, the function will discard the extra responses – the first ones to answer will be the ones saved. The returned count pBlockFound will have the actual count of the number of boards that responded.

This command only works with the Ethernet DLL.

WD_E_STATUS WD_E_DiscoverAll(WD_E_HANDLE wdeHandle,
UCHAR* pIpMacData, UINT32 iBlockCnt, UINT32* pBlockFound)

Parameters:

wdeHandle – Handle of the device from WD_Open().
pIpMacData – pointer to a byte array for IP – MAC address data.
iBlockCnt – count of the number of 10 byte blocks in the array.
pBlockFound – pointer to integer for count of blocks found.

Return Value:

WD_OK if successful or a WD error code.

NOTE: Once you discover which board you want to use you MUST call WD_E_DISCOVER with the IP address so that DLL knows which board to send the future commands!

Example:

```
WD_HANDLE wdHandle ;
WD_STATUS wdStatus ;
typedef struct{
    UCHAR ip[4] ;
    UCHAR mac[6];
} WD_ID ;
WD_ID id[6] ;               // room for six entries
UINT32 iBlockCnt, iBlockFound ;
UCHAR *sp ;

memset(id, 0, sizeof(id)) ;
iBlockCnt = 6 ;             // entries in strcuture
sp = (UCHAR*)&id[0] ;       // array of bytes
wdeStatus =  WD_E_DiscoverAll(wdeHandle, sp, iBlockCnt,
                &iBlockFound);
if(wdeStatus == WD_E_OK)
    printf("Discover All Success.\n") ;
```

All the commands from here on are the same as the functions (Commands) from the USB manual. The names start the WD_E_ instead. They same the same parameters unless noted.

2.6 WD_E_GetErrorInfoMsg

Same format as USB command.

2.7 WD_E_ GetDeviceInfo

Same format as USB command.

2.8 WD_ GetTempTickle

Same format as USB command.

2.9 WD_ E_SetPowerOnDlyTimes

Same format as USB command.

2.10 WD_ E_GetPowerOnDlyTimes

Same format as USB command.

2.11 WD_ E_SetWdogTimes

Same format as USB command.

2.12 WD_ E_GetWdogTimes

Same format as USB command.

2.13 WD_ E_GetAnalogDigitalIn

Same format as USB command.

2.14 WD_ E_GetSetAuxRelay

Same format as USB command.

2.15 WD_ E_EnableDisable

Same format as USB command.

2.16 WD_ E_ GetResetCount

Same format as USB command.

2.17 WD_ E_ GetSetNvTempOffset

Same format as USB command.

2.18 WD_ E_ SetBuzzer

Same format as USB command.

2.19 WD_ E_ GetBuzzer

Same format as USB command.

2.20 WD_ E_GetSetNvRelayPulse

Same format as USB command.

2.21 WD_ E_GetSetNvUsbUserCode

Same format as USB command.

2.22 WD_ E_EnableDisablePcReset

Same format as USB command.

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 - the watchdog is armed but the POD Dip Switch was set to wait for first "tickle". In this case the countdown timer has not started yet. Use the function (command) WD_ GetTempTickle to "tickle" the board and clear this status.

WD_STAT_CMD_DISABLED - the watchdog has been disabled by command.

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 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. Board should not be used and should be returned for repair.

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. Contact Berkshire about trying to recover the memory since the Ethernet MAC address is stored there.

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_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. This is not a "sticky" flag. If the next write attempt is OK this flag will be cleared.

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.