Intel® 6300ESB I/O Controller Watchdog Timer Application Programming Interface

Ref. No. 0.3

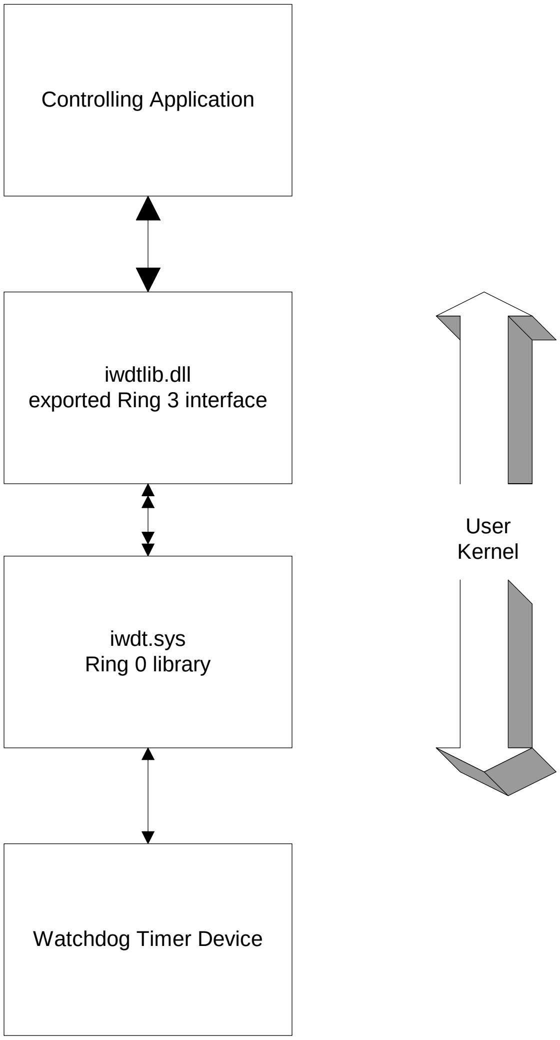
The Intel Watchdog Timer Application Programming Interface may contain design defects or errors known as errata which may cause the product to deviate from published specifications. Current characterized errata are available on request.

Watchdog Timer Windows API

The following documentation details the method of programmatically controlling the Intel® 6300ESB I/O Controller Watchdog Timer (WDT) device on the Microsoft Windows 2000*, Windows XP*, and .NET* server platforms. The Application Programming Interface (API) presented is for programs written using C\C++ language. To access the WDT API, link the iwdtlib.lib file with your project. All of the defines, structure definitions and function prototypes are included in the header file iwdt.h.

How the API Works

The API provides a user mode (Ring 3) interface to the watchdog timer driver **iwdtlib.sys**. The functions are exported from the dynamic link library **iwdtlib.dll** and act as a wrapper to the driver's IOCTL-based interface. While it is possible to control the WDT device by directly issuing device I/O function calls, the libraries API will greatly simplify the process of working with the WDT.



Interfacing the API Library to Your Code

The following lines would be added to your project to access the WDT timer functions.

```
extern “C” WDT_IMPORT HANDLE WdtInitLibrary(PWDT);
extern “C” WDT_IMPORT HANDLE WdtGetDeviceHandle();
extern “C” HANDLE WdtGetStatus( HANDLE wdHandle, SAWD_CTRL *WdtStatus );
extern “C” WDT_IMPORT ULONG   WdtGetDriverVersion(HANDLE);
extern “C” WDT_IMPORT ULONG   WdtGetLibraryVersion(VOID);
extern “C” WDT_IMPORT   bool    WdtGetCapabilities(HANDLE );
extern “C” WDT_IMPORT   bool    WdtSetPreloadValues(HANDLE,WDT);
extern “C” WDT_IMPORT   bool    WdtPing(HANDLE,ULONG);
extern “C” WDT_IMPORT   bool    WdtEnable(HANDLE, bool);
extern “C” WDT_IMPORT   bool    WdtLockDevice(HANDLE);
extern “C” WDT_IMPORT   bool    WdtStageOneNotify(HANDLE, S1FUNCTPTR );
extern “C” WDT_IMPORT   BOOL    WdtCancelNotify();
extern “C” WDT_IMPORT   int     WdtCheckTimeOutStatus(HANDLE, BOOL);
extern “C” WDT_IMPORT   BOOL    WdtSetPrescaler(HANDLE, ULONG);
extern “C” WDT_IMPORT   BOOL    WdtInterruptConnect(HANDLE, ULONG);
extern “C” WDT_IMPORT   BOOL    WdtRouteInterrupt(HANDLE, ULONG);
extern “C” WDT_IMPORT   BOOL    WdtSetMode(HANDLE xhndFile, ULONG);
extern “C” WDT_IMPORT   BOOL    WdtSetOutputEnable(HANDLE, ULONG);
```

NOTE: Extern "C" forces the use of the C naming convention for non C++ functions, you can omit extern “C” if your program is written in C++. The “WDT_IMPORT” is a macro defined in the **iwdtlib.h** file that resolves to the string __declspec(dllimport).

Be aware of compiler switches **/Tc** or **/TP**, which tell the compiler to ignore the filename extension and compile the file as C or C++, respectively.

Click a hyperlink below for details on a specific API.

WdtInitLibrary

WdtGetDeviceHandle

WdtGetStatus

WdtGetDriverVersion

WdtGetLibraryVersion

WdtEnable

WdtGetCapabilities

WdtLockDevice

WdtSetPreloadValues

WdtPing

WdtConfigure

WdtSetMode

WdtStageOneNotify

WdtCancelNotify

WdtSetPresecaler

WdtInterruptConnect

WdtRouteInterrupt

WdtSetOutputEnable

WdtReadPreloadValues

WdtCheckTimeOutStatus

WdtInitLibrary

Initializes the watchdog device library and returns a device handle to the caller.

```
HANDLE WdtInitLibrary(
        PWDT wdtobject
);
```

Parameters

wdtobject

[in] pointer to a WDT structure.

Return Value

If the function succeeds, the return value is an open handle to the WDT.

If the function fails, the return value is NULL.

Remarks

Use the **WdtCloseHandle** function to close an object handle returned by **WdtInitLibrary**.
The WdtInitLibrary function does the following:

Presets the user mode WDT structure.
Returns a valid driver handle to the caller.
Sets up a soft link between the watchdogs' kernel mode driver and a user supplied function.

This link is required by the WdtStageOneNotify function to process the user callback.

Requirements

Windows NT/2000/XP
Header: Declared in iwdt.h;.
Library: Use Iwdt.lib.

Example

```
// A
HANDLE wdHandle;
WDT HR_wDt;

wdHandle = WdtInitLibrary(&HR_wDt);
```

WdtGetDeviceHandle (Obsolete)

Opens the watchdog device and returns a valid device handle to the caller. This function has been replaced by the WdtInitLibrary.

```
HANDLE WdtGetDeviceHandle();
```

Parameters

No Parameters

Return Value

If the function succeeds, the return value is an open handle to the WDT.

If the function fails, the return value is WDT_FAILURE.

Remarks

Use the **WdtCloseHandle** function to close an object handle returned by **WdtGetDEviceHandle**.

Requirements

Windows NT/2000/XP
Header: Declared in iwdt.h;.
Library: Use Iwdt.lib.

Example

```
// A
HANDLE wdHandle;
wdHandle = WdtGetDeviceHandle();
```

WdtGetStatus

Queries the WDT device about its status, returning the results to the caller in the WdtControl structure.

```
HANDLE WdtGetStatus(
 HANDLE wdHandle,
 SAWD_CTRL *WdtStatus
);
```

Parameters

wdHandle

[in] Handle to the device returned by **WdtGetDeviceHandle**.

wdt

[out] pointer to structure that is populated with the watchdog status.

Return Value

If the function succeeds, the return value is WDT_SUCCESS.

If the function fails, the return value is WDT_FAILURE.

Remarks

This function returns the device status bits, configuration register bits and Lock register bits

Requirements

Windows NT/2000/XP
Header: Declared in iwdtlib.h;.
Library: Use iwdtlib.lib.
Example

```
// Query driver for the watchdog timer’s status.

WdtGetStatus(wdHandle, &WdtStatus);
```

WdtGetDriverVersion

Returns the watchdog drivers version number as an unsigned long.

`HANDLE WdtGetDriverVersion();`

Parameters

hFile

[in] Handle to the device returned by **WdtGetDeviceHandle**.

Return Value

If the function succeeds, the return value is unsigned long composed of a major and minor version code.

If the function fails, the return value is 0x0FFFFFFFF.

Remarks

The lower 16 bits represent the minor version, while the upper 16 bits represent the major version number. This same version number is also reported in the WdtGetCapabilities function in the SAWD_CAPABILITY_OUT_BUFF structure.

Requirements

Windows NT/2000/XP
Header: Declared in iwdtlib.h;.
Library: Use iwdtlib.lib.
Example

```
// Query the driver for it version number then format as an ASCII
// string

Char buffer[40];
ULONG   DrvVersion;
USHORT  Lo,Hi;

 DrvVersion=WdtGetDriverVersion();
 Lo=LOWORD(DrvVersion);
 Hi=HIWORD (DrvVersion);

 sprintf(buffer, "Driver Version %u.%u",Hi,Lo);
```

WdtGetLibraryVersion

Returns the watchdog libraries version number as an unsigned long.

```
HANDLE WdtGetLibraryVersion();
```

Parameters

No Parameters

Return Value

The return value is unsigned long composed of a major and minor version code.

Remarks

The lower 16 bits represent the minor version, while the upper 16 bits represent the major version number.

Requirements

Windows NT/2000/XP
Header: Declared in iwdtlib.h;.
Library: Use iwdtlib.lib.
Example

```
// Query the library for it version number then format as an ASCII //
string

Char buffer[40];
ULONG   LibVersion;
USHORT  Lo,Hi;


 LibVersion=WdtGetLibraryVersion();
 Lo=LOWORD(LibVersion);
 Hi=HIWORD (LibVersion);

 sprintf(buffer, "Library Version %u.%u",Hi,Lo);
```

WdtEnable

Starts or stops the counter depending on the submitted control value.

```
BOOL WdtEnable(
 HANDLE wdHandle,
 BOOL control = TRUE or FALSE,
);
```

Parameters

wdHandle

[in] Handle to the device returned by **WdtGetDeviceHandle**.

control

TRUE indicate a request to start the WDT. FALSE indicates a request to stop the WDT.

Return Value

If the function succeeds, the return value is WDT_SUCCESS.

If the function fails, the return value is WDT_FAILURE. To get extended error information

Remarks

Requirements

Windows NT/2000/XP
Header: Declared in iwdtlib.h;.
Library: Use iwdtlib.lib.
Example

```
// Start Watchdog timer.
WdtEnable(wdHandle,TRUE);

// Stop Watchdog timer.
WdtEnable(wdHandle,FALSE);
```

WdtGetCapabilities (Obsolete – Do not use)

Queries the WDT device about its capabilities, returning the results to the caller in the SAWD_CAPABILITY_OUT_BUF structure.

```
HANDLE WdtGetCapabilites(
 HANDLE wdHandle,
 SAWD_CAPABILITY_OUT_BUF *WdtCapStruct
);
```

Parameters

wdHandle
[in] Handle to the device returned by **WdtGetDeviceHandle**.
wdtCapStruct
[out] pointer to structure that is populated with the WDT capabilities.

Return Value

If the function succeeds, the return value is WDT_SUCCESS.

If the function fails, the return value is WDT_FAILURE.

Remarks

This function returns the version number of the driver and the minimum and maximum preload values that can be set.

Requirements

Windows NT/2000/XP
Header: Declared in iwdtlib.h;.
Library: Use iwdtlib.lib.
Example

```
// Query driver for the watchdog timer’s capabilities

WdtGetCapabilities(wdHandle, &WdtCapStruct);
```

WdtLockDevice

Locks the watchdog device to prevent any further configuration changes. Also prevents the watchdog device from being stopped or started.

```
HANDLE WdtLockDevice (
 HANDLE wdHandle
);
```

Parameters

wdHandle

[in] Handle to the device returned by **WdtGetDeviceHandle**.

Return Value

If the function succeeds, the return value is WDT_SUCCESS.

If the function fails, the return value is WDT_FAILURE.

Remarks

Locks the configuration register, i.e., the WDT_TOUT_CNF and WDT_ENABLE bits cannot be changed. This is also a Write-Once bit. It cannot be changed until either the power is cycled or a hard reset occurs.

Requirements

Windows NT/2000/XP
Header: Declared in iwdtlib.h;.
Library: Use iwdtlib.lib.

Example

// Lock the watchdog timer to prevent any further changes

```
BOOL status;
status = WdtLockDevice(wdHandle);
```

WdtSetPreloadValues

Load preload register 1 and preload register 2.

```
BOOL WdtSetPreload(
 HANDLE wdHandle,
 WDT wdtConfig
);
```

Parameters

wdHandle

[in] Handle to the device returned by **WdtGetDeviceHandle**.

wdtconfig

[in] pointer to structure that contains WDT preload values and other configuration values.

Return Value

If the function succeeds, the return value is WDT_SUCCESS.

If the function fails, the return value is WDT_FAILURE.

Remarks

Loading the preload register does not load the 35 bit down counter and will not prevent the WDT from timing out. Use the WdtPing function to transfer the preload values to the down counter.

Requirements

Windows NT/2000/XP
Header: Declared in iwdtlib.h;.
Library: Use iwdtlib.lib.
Example

```
wdConfig.preload1=12323;       // Set preload value # 1
wdConfig.preload2=12000        // Set preload value # 2

WdtSetPreload(wdHandle,wdtconfig);
```

WdtPing

Ping the WDT to prevent the WDT from timing out.

```
HANDLE WdtPing(
 HANDLE wdHandle,
);
```

Parameters

wdHandle

[in] Handle to the device returned by **WdtGetDeviceHandle**.

Return Value

If the function succeeds, the return value is WDT_SUCCESS.

If the function fails, the return value is WDT_FAILURE.

Remarks

This function is used to prevent a timeout occurring. During stage 1 this function moves the value in preload register 1 to the 35-bit down counter. During stage 2 this function moves the data from preload register 2 to the 35-bit down counter.

Requirements

Windows NT/2000/XP
Header: Declared in iwdtlib.h;.
Library: Use iwdtlib.lib.
Example

```
// Ping the watchdog timer.
BOOL status;
status = WdtPing(wdHandle);
```

WdtSetPresecaler

Specify which Presecalar mode of operation is used by the WDT.

```
BOOL WdtSetPresecaler(
 HANDLE wdHandle,
 BOOL   BitPat
);
```

Parameters

wdHandle

[in] Handle to the device returned by **WdtGetDeviceHandle**.

BitPat

[in] Unsigned long bit pattern defined in the iwdtlib.h.

Return Value

If the function succeeds, the return value is WDT_SUCCESS.
If the function fails, the return value is WDT_FAILURE.

Remarks

This function is used to select a Presecalar mode for the WDT's main down counter. There are 2 modes available:

- The default mode specifies that the 20-bit preload value is written into bits 34:15 of the down counter, resulting in a 1 KHz clock.
- The fast mode specifies that the preload value is loaded into bits 24:5 of the down counter, resulting in a 1 MHz clock.

You must specify one of the two defined states, **WDT_NORMAL_PRESCALER** or **WDT_FAST_PRESCALER.** Any other values will cause the function to fail.

Requirements

Windows NT/2000/XP
Header: Declared in iwdtlib.h;.
Library: Use iwdtlib.lib.

Example // Set Presecalar to default mode.

```
WdtSetPresecaler(wdHandle, WDT_NORMAL_PRESCALER);

//Set Presecalar to FAST mode
WdtSetPresecaler(wdHandle, WDT_FAST_PRESCALER);
```

WdtInterruptConnect

Manually connects the WDT IRQ to the drivers interrupt service routine.

```
BOOL WdtRouteInterrup(
 HANDLE wdHandle,

);
```

Parameters

wdHandle

[in] Handle to the device returned by **WdtGetDeviceHandle**.

Return Value

If the function succeeds, the return value is WDT_SUCCESS.

If the function fails, the return value is WDT_FAILURE.

Remarks

This function is only needed if Windows is unable to assign INTR resources at driver initialization. Intel® 6300ESB I/O Controller A1 silicon is missing two interrupt registers as per PCI spec, Interrupt Pin and Line registers logic and is unable to enumerate the WDT device resources.

Requirements

Windows NT/2000/XP
Header: Declared in iwdtlib.h;.
Library: Use iwdtlib.lib.

Example // connect the IO-APIC IRQ to the drivers ISR

```
WdtInterruptConnect(wdHandle);
```

WdtRouteInterrupt

Specify how the WDT should report the occurrence of a stage 1 interrupt.

```
BOOL WdtRouteInterrup(
 HANDLE wdHandle,
 ULONG BitPat
);
```

Parameters

wdHandle
[in] Handle to the device returned by **WdtGetDeviceHandle**.
BitPat
[in] Unsigned long bit pattern defined in the iwdtlib.h.

Return Value

If the function succeeds, the return value is WDT_SUCCESS.

If the function fails, the return value is WDT_FAILURE.

Remarks

The WDT supports both IRQ and SMI reporting of timeouts.

You must specify **1** of the **3** defined states,

WDT_INTR_IRQ, WDT_INTR_SMI or **WDT_INTR_DISABLE**

Any other values will cause the function to fail. WARNING, the SMI define is in place to allow the user to verify that the SMI generation works correctly. However, the driver has no capabilities to handle WDT-generated SM interrupts and may result in crashing your platform if the BIOS does not implement a default SMI handler.

Requirements

Windows NT/2000/XP
Header: Declared in iwdtlib.h;.
Library: Use iwdtlib.lib.

Example // Specify the use of IOAPIC - IRQ.

```
WdtRouteInterrupt(wdHandle, WDT_INTR_IRQ);
```

WdtSetOutputEnable

Enable or Disable the toggling of the external WDT_OUTPUT pin.

```
BOOL WdtSetOutEnable(
 HANDLE wdHandle,
 ULONG BitPat
);
```

Parameters

wdHandle

[in] Handle to the device returned by **WdtGetDeviceHandle**.

BitPat

[in] Unsigned long bit pattern defined in the iwdtlib.h.

Return Value

If the function succeeds, the return value is WDT_SUCCESS.

If the function fails, the return value is WDT_FAILURE.

Remarks

This function is used to control whether the WDT_OUTPUT pin is toggled if the WDT times out. By default this feature is enabled.

You must specify **one of the two** defined states,

WDT_ENABLE_EXT_OUT or **WDT_DISABLE_EXT_OUT**

Any other values will cause the function to fail.

Requirements

Windows NT/2000/XP
Header: Declared in iwdtlib.h;.
Library: Use iwdtlib.lib.

Example // Specify that the WDT_OUTPUT pin is toggled on time out.

```
WdtSetOutEnable(wdHandle, WDT_ENABLE_EXT_OUT);
```

WdtReadPreloadValue

Read back a preload value.

```
ULONG WdtReadPreloadValue(
 HANDLE wdHandle,
 ULONG BitPat
);
```

Parameters

wdHandle

[in] Handle to the device returned by **WdtGetDeviceHandle**.

BitPat

[in] Unsigned long bit pattern defined in the iwdtlib.h.

Return Value

If the function succeeds, the return is the current value in either the first or second preload register as specified in the BitPat parameter.

If the function fails, the return value is WDT_FAILURE.

Remarks

This function is useful verifying that the value placed in one or both of the preload registers is correct.

You must specify one of the two defined counters.

WDT_PRELOAD_1 or **WDT_PRELOAD_2**

Any other values will cause the function to fail.

Requirements

Windows NT/2000/XP
Header: Declared in iwdtlib.h;.
Library: Use iwdtlib.lib.

Example // Read the current value of preload register number 1.

ULONG Preload_1=0;

```
Preload_1=WdtReadPreloadValue(wdHandle, WDT_PRELOAD_1);
```

WdtCheckTimeOutStatus

Check if a timeout of the WDT has occurred. Optionally reset the timeout flag if the Boolean *Reset* flag is TRUE.

```
int WdtCheckTimeOutStatus(
 HANDLE wdHandle,
 BOOL Reset
);
```

Parameters

wdHandle
[in] Handle to the device returned by **WdtGetDeviceHandle**.
BitPat
[in] Boolean flag, setting to TRUE will reset this bit flag.

Return Value

If the function succeeds, the return value is a 0 or 1. A 1 indicates that the second stage of the WDT reached zero after the first stage counted down to zero.

If the function fails, the return value is a -1.

Remarks

This function returns the current setting of the WDT_TIMEOUT bit in the memory mapped ‘Reload Register’. This bit is set to ‘1’ if the 35-bit down counter reaches zero for the second time in a row.

Requirements

Windows NT/2000/XP
Header: Declared in iwdtlib.h;.
Library: Use iwdtlib.lib.

Example // Check if a time out has occurred, then clear the bit flag to (0).

```
WdtCheckTimeOutStatus(wdHandle, TRUE);
```

WdtConfigure

Set the configuration bits in the WDT.

```
BOOL WdtConfigure(
 HANDLE wdHandle,
 WDT wdtConfig
);
```

Parameters

wdHandle

[in] Handle to the device returned by **WdtGetDeviceHandle**.

wdtconfig

[out] pointer to structure that contains WDT preload values and other configuration values.

Return Value

If the function succeeds, the return value is WDT_SUCCESS.

If the function fails, the return value is WDT_FAILURE.

Remarks

This function allows the advanced user to directly set bits in the configuration register. Users of this function are assumed to have detailed knowledge of the Intel® 6300ESB I/O Controller WDT.

Requirements

Windows NT/2000/XP
Header: Declared in iwdt.h;.
Library: Use iwdt.lib.

Example

```
/* Configure the watchdog timer to enable toggling of the WDT_OUTPUT_PIN,
route interrupts to IRQ 10 and select Presecalar 1 for loading the down
counter.
*/
wdtConfig.ConfigReg=WDT_ENABLE_TOGGLE | WDT_INTR_IRQ10 |WDT_PRE_1;
```

```
WdtConfigure(wdHandle,wdtconfig);
```

WdtSetMode

Sets the device to run in WDT mode or free running mode.

```
BOOL WdtSetMode(
 HANDLE wdHandle,
 ULONG mode
);
```

Parameters

wdHandle
[in] Handle to the device returned by **WdtGetDeviceHandle**.
mode
[in] Unsigned long bit pattern defined in the iwdtlib.h.

Return Value

If the function succeeds, the return value is WDT_SUCCESS.

If the function fails, the return value is WDT_FAILURE.

Remarks

This function allows you switch the WDT to a free running counter. In free running mode the down counter is automatically loaded every time the decrementer reaches zero.

You must specify **one of the two** defined states, **WDT_WATCHDOG_MODE** or **WDT_FREE_RUN_MODE.** Any other values will cause the function to fail.

Requirements

Windows NT/2000/XP
Header: Declared in iwdtlib.h;.
Library: Use iwdtlib.lib.

Example

```
// Set the device to watchdog mode.

WdtSetMode(wdHandle, ENABLE_WDT_MODE);

// Set the device to free running mode
```

```
WdtSetMode(wdHandle, ENABLE_FREE_MODE);
```

WdtStageOneNotify

Set up a callback to a user-supplied function that will handle interrupt processing when the WDT enters stage 1.

```
BOOL WdtStageOneNotify(
 HANDLE wdHandle,
 S1FUNCTPTR lpHandleIntProc
);
```

Parameters

wdHandle

[in] Handle to the device returned by **WdtGetDeviceHandle**.

lpHandleIntProc

[in] Application-defined function that is called by the library.

Return Value

If the function succeeds, the return value is WDT_SUCCESS.

If the function fails, the return value is WDT_FAILURE.

Remarks

This function creates a low priority thread that monitors shared user/kernel synchronization object, then returns to the caller. The new thread calls the WaitForSingleObject function to wait for the driver to signal that a stage 1 interrupt has occurred.

The function uses a semaphore 'gate' to prevent the user from creating multiple instances of the user callback thread, i.e., attempting to call this function after a previous call will result in the function returning a FALSE back to the caller. You must first cancel the waiting thread before calling this function again.

Passing a NULL for the callback address will terminate a waiting thread

Requirements

Windows NT/2000/XP
Header: Declared in iwdtlib.h;.
Library: Use iwdtlib.lib.

Example// Assign the user-defined function that the watchdog library will call

```
WdtStageOneNotify(wdHandle, S1FUNCTPTR(Stage1CallBack));
```

WdtCancelNotify

Cancel CALLBACK notification that was set up using the WdtStageOneNotify call.

```
BOOL WdtCancelNotify(
void
);
```

Parameters

No Parameters needed

Return Value

If the function succeeds, the return value is WDT_SUCCESS.

If the function fails, the return value is WDT_FAILURE.

Remarks

This function frees the waiting stage 1 monitor thread synchronization object, then returns to the caller. Any new stage interrupt activity will still be serviced by the drivers ISR; however, the user stage 1 function will **not** be called.

Requirements

Windows NT/2000/XP
Header: Declared in iwdtlib.h;.
Library: Use iwdtlib.lib.

Example// cancel the user-defined function that the watchdog library will call

```
WdtCancelNotify();
```

Format of Stage One Handler

```
ULONG Stage1CallBack(LPVOID UserData)
{
ULONG *iData;
BOOL status=FALSE;
char buffer[100];

 //
 // User can cast UserData to which every memory type is appropriate
 //

 iData = (ULONG *) UserData;

 //
 // Place code here that will decide what to do if a stage 1
 // Interrupt occurs
 //



return status;
}
```

Appendix A: Header File Constructs

```
typedef struct WdtControl
{
   bool      Running;           // 0 WDT is idle, 1 WDT is Running
   bool      Mode;              // 0 (default) Run as Watchdog timer, 1 = Free running mode
   bool      Presecalar;        // 0 (default) use 34:15 prescalar setting, 1 = 24:5
   bool      OutputEnable;      // 0 (default) Enable WDT out pin 1 = disable

   ULONG     InterruptRouting;  // 00 = IRQ, 01 (Reserved) 10 = SMI 11 = DISABLED
   ULONG     Refresh;           // Number of seconds before WDT must be refreshed
   ULONG     PreLoad1;          // primary down counter value Stage 1
   ULONG     PreLoad2;          // Secondary down counter value Stage 2
   USHORT    ConfigReg;         // Bit pattern to be written to the configuration register
   USHORT    LockReg;           // Bit pattern to be written to the LOCK register
   USHORT    DeviceStatus;      // Bits read from the Device Status register

   HANDLE    wdtHandle;         // copy of the active handle to the device driver
   ULONG     InitStatus;        // Initialization results; 0= OK
   BOOL      ErrorMsgBox;       // TRUE = allow msg box generation, FALSE disable Msg box
   UHSORT    DriverVersionLo;   // lo portion of driver version
   UHSORT    DriverVersionHi;   // Hi portion of driver version
   UHSORT    LibraryVersionLo;  // lo portion of Library version
   UHSORT    LibraryVersionHi;  // Hi portion of Library version


}WDT, *PWDT;
```

Note: The WdtInitLibrary function call will initialize the preload registers, set the initial refresh rate to once per second, set the execution flag to ‘IDLE’ and save off a copy of the drivers handle.

```
typedef struct _SAWD_CAPABILITY_OUT_BUFF
{
 ULONG Version;          // version of driver used
 ULONG Capability;       // bit field indicating capabilities
 ULONG min;              // minimum value in msecs
 ULONG max;              // maximum value in msecs

} SAWD_CAPABILITY_OUT_BUFF, *PSAWD_CAPABILITY_OUT_BUFF;
```

Appendix B: Program Defines

Return constants:

WDT_SUCCESS	Function Passed
WDT_FAILURE	Function Failed

Input constants:

WDT_ENABLE	
WDT_DISABLE	
WDT_INTR_IRQ	0x00
WDT_INTR_SMI	0x02
WDT_INTR_DISABLE	0x03
WDT_NORMAL_PRESCALER	0x0FFFFFFFB // Bit pattern to set bit 2 to a zero
WDT_FAST_PRESCALER	0x04 // Bit pattern to set bit 2 to a one
WDT_DISABLE_EXT_OUT	0x0FFFFFFDF // Bit pattern to set bit 5 to a zero
WDT_ENABLE_EXT_OUT	0x20 // Bit pattern to set bit 5 to a one
WDT_PRELOAD_1	0x01
WDT_PRELOAD_2	0x02
WDT_WATCHDOG_MODE	0x0FFFFFFFB // Bit pattern to set bit 2 to a zero
WDT_FREE_RUN_MODE	0x04