



RtAttachInterruptVector (...)

RtAttachInterruptVector allows the user to associate a handler routine in user space with a hardware interrupt. Non-shared, level-triggered interrupts are supported only in the RTSS environment. They are not supported in a Win32 environment.

Prototype

```
HANDLE
RtAttachInterruptVector(
PSECURITY_ATTRIBUTES    pThreadAttributes,
ULONG                   StackSize,
VOID                    (RTFCNDCL *pRoutineIST)(PVOID ContextIST),
PVOID                   ContextIST,
ULONG                   Priority,
INTERFACE_TYPE          InterfaceType,
ULONG                   BusNumber,
ULONG                   BusInterruptLevel,
ULONG                   BusInterruptVector
);
```



RtAttachInterruptVector (...)

Parameters

<i>pThreadAttributes</i>	(ignored by RTSS)
<i>StackSize</i>	the number of bytes to allocate for the handler thread's stack. The stack size defaults to the thread's stack size.
<i>pRoutineIST</i>	a pointer to the handler routine to be run. The routine takes a single PVOID argument and returns VOID .
<i>ContextIST</i>	the argument to the handler routine, cast as a PVOID .
<i>Priority</i>	the thread priority for the handler routine. Executing equal and higher priority threads disable the interrupt; executing lower priority threads enable it.
<i>InterfaceType</i>	the type of bus interface on which the hardware is located. It can be one of the following types: Isa or PCIBus . The upper boundary on the bus types supported is always MaximumInterfaceType .
<i>BusNumber</i>	the bus that the device is on in a multiple bus environment. It is zero-based. Note that this value applies to each bus type, so for a system with one ISA bus and one PCIBus, for example, each would have a BusNumber of 0.



RtAttachInterruptVector (...)

Parameters

<i>BusInterruptLevel</i>	A bus-specific interrupt level associated with the routine that will handle the interrupt.
<i>BusInterruptVectorA</i>	bus-specific address associated with the routine that will handle the interrupt.

Return Values

The function returns an RTX-specific interrupt handle if successful. The function returns a **NULL** handle for an invalid argument, for failing to connect the interrupt, or if a device is already using the bus resources requested.



RtAttachInterruptVector (...)

```
int nContext = 1;
HANDLE hInterrupt = NULL;
#define RS485_PCI_BUS_VECTOR x08

/* ISR */
void RTFCNDCL InterruptHandler(void * nContext) {
    RtWprintf(L"index = %d\n", ((int) nContext) ++);
}

/* Set up interrupt */
void SetInterrupt() {
    hInterrupt = RtAttachInterruptVector(
        NULL, /* thread attributes */
        0, /* stack size - 0 uses default */
        InterruptHandler, /* handler routine */
        (void *) &nContext, /* context */
        1, /* priority */
        PCIBus, /* interface type */
        0, /* bus number */
        RS485_PCI_BUS_VECTOR, /* bus interrupt level */
        RS485_PCI_BUS_VECTOR ); /* bus interrupt vector */
    if (hInterrupt == NULL) {
        RtWprintf(L"RtAttachInterruptVector error = %d\n", GetLastError());
        ExitProcess(1);
    }
    Sleep(10000);
    RtReleaseInterruptVector(hInterrupt);
}
```



RtDisable/EnableInterrupts (...)

RtDisableInterrupts disables all interrupts at the processor level including timer interrupts.

RtEnableInterrupts enables user-level interrupt handling for all interrupts to which the process is attached.

Prototype

BOOL **RtDisableInterrupts**(VOID)

BOOL **RtEnableInterrupts**(VOID)

Return value

The functions return **TRUE** if successful; otherwise it return **FALSE**.