

```

/*****
/*          I R Q U T I L . C          */
/**-----**/
/* task          : assigns functions for the hardware interrupt */
/*               : programming at your disposal                */
/**-----**/
/* author        : Michael Tischer / Bruno Jennrich          */
/* developed on   : 3/12/1994                                  */
/* last update    : 4/06/1995                                  */
/**-----**/
/* COMPILER      : Borland C++ 3.1, Microsoft Visual C++ 1.5 */
/*****
#ifdef __IRQUTIL_C          /* may also be #included */
#define __IRQUTIL_C

/*-- bind in include files -----*/
#include <dos.h>
#include <conio.h>

#include "irqutil.h"

/*****
/* irq_Enable : admit hardware interrupt          */
/**-----**/
/* request : iIRQ - number for the hardware interrupt (0-15), which */
/*           should be admitted                                */
/**-----**/
/* info : connecting and disconnecting IRQ's takes place via */
/*         transmission of an OCW1 (Operation Control Word) labeled */
/*         byte to the IRQ-Controller base port.                */
/*****
VOID irq_Enable( INT iIRQ )

```

```

{ INT iPort;          /* establish the port address for the appropriate PIC */
                        /* ( 0-7 = MASTER_PIC , 8-15 = SLAVE_PIC ) */
    iPort = ( iIRQ <= 7 ) ? MASTER_PIC : SLAVE_PIC ;
    iPort += IRQ_MASK;          /* select masking port */

    iIRQ &= 0x0007;          /* establish PIC interrupt number (0-7) */
                                /* erase bit -> interrupt admitted */
    outp( iPort, inp( iPort ) & ~( 1 << iIRQ ) );
}

/*****
/* irq_Disable : hardware interrupt refused */
/**-----*/
/* request : iIRQ - number of the hardware interrupts (0-15), */
/*          that will be refused */
*****/
VOID irq_Disable( INT iIRQ )
{ INT iPort;          /* establish the port address for the appropriate PIC */
                        /* ( 0-7 = MASTER_PIC , 8-15 = SLAVE_PIC ) */
    iPort = ( iIRQ <= 7 ) ? MASTER_PIC : SLAVE_PIC ;
    iPort += IRQ_MASK;          /* select masking port */

    iIRQ &= 0x0007;          /* PIC-establish interrupt number (0-7) */
                                /* set bit -> interrupt locked */
    outp( iPort, inp( iPort ) | ( 1 << iIRQ ) );
}

/*****
/* irq_SendEOI : signal "End Of Interrupt" */
/**-----*/
/* input : iIRQ : number of the hardware interrupt (0-15), which */
/*          will be completely utilized. */
*****/

```

```

/* info : - an interrupt can again be released */
/*         if the previous release has been completely utilized. */
/*         In order to signal the PIC's, that an interrupt */
/*         has completed all of the tasks, an EOI must be */
/*         sent to the PIC's. */
/*         EOI - without the IRQ number because it is in nested mode */
/*****
VOID irq_SendEOI( INT iIRQ )
{
    /* Also notify the slave with IRQ 8 - 15 */
    if ( iIRQ > 7 ) outp( SLAVE_PIC, EOI );
    outp( MASTER_PIC, EOI ); /* Always signal the MASTER EOI */
}

/*****
/* irq_SetHandler : install new interrupt handler */
/*-----*/
/* input : iIRQ      - number of the interrupt, which should contain */
/*              the new handler. */
/*          lpHandler - Address of the new handler. */
/*          print out : Address of the old handler. */
/*****
VOID ( _interrupt _FP *irq_SetHandler( INT iIRQ,
                                       VOID ( _interrupt _FP *lpHandler )() ) ) ( )
{ VOID ( _interrupt _FP *lpOldHandler )();
  INT iVect;

      /* determine the interrupt vector of the hardware interrupt */
      /* IRQ 0 - 7 = vectors 0x08 - 0x0F */
      /* IRQ 8 - 15 = vectors 0x70 - 0x77 */

  iVect = ( iIRQ <= 7 ) ?
    ( MASTER_FIRST_VECTOR + iIRQ ) :
    ( SLAVE_FIRST_VECTOR + ( iIRQ & 0x0007 ) );

```

```

irq_Disable( iIRQ );    /* block hardware and software interrupts */
_disable();

lpOldHandler = _dos_getvect( iVect );    /* save old handler */
_dos_setvect( iVect, lpHandler );    /* set new handler */

_enable();    /* allow software interrupts */
if( lpHandler )    /* In the event a handler is turned in once again */
    irq_Enable( iIRQ );    /* allow the respective hardware interrupt */

return lpOldHandler;    /* return the address of the old handler */
}

/*****
/* irq_ReadMask : read an IRQ controller's masking    */
/**-----*/
/* input : iController - Base port for the IRQ controller (0x20/0xA0) */
/* print out : masking of the IRQ's served by the IRQ controller.    */
/*          set bits : IRQ is not released.    */
/**-----*/
/* Info : A PIC's masking can be transmitted by simply    */
/*          selecting the mask register. (0x21/0xA1)    */
*****/
BYTE irq_ReadMask( INT iController )
{
    return ( BYTE )inp( iController + IRQ_MASK );
}

/*****
/* irq_ReadISR : read the Irq controller's status register    */
/**-----*/
/* input : iController - IRQ controller base port (0x20/0xA0)    */

```

```

/* print out : status register of the IRQ's served          */
/*      by the IRQ controller.                               */
/*      set bits : the corresponding IRQ routine is         */
/*      briefly executed or interrupted by a an IRQ         */
/*      having a higher priority.                            */
/**-----**/
/* Info : Before the IRQ controller can transmit the contents of the */
/*      ISR register the OCW3 labeled byte must be transmitted */
/*      to the controller                                     */
/*******/
BYTE irq_ReadISR( INT iController )
{
    /* OCW3: 0x0B = no operation , no poll, read ISR */
    outp( iController, 0x0B );
    return ( BYTE )inp( iController );
}

/*******/
/* irq_ReadIRR : read an Irq-Controller's request register */
/**-----**/
/* input : iController - IRQ controller's base port(0x20/0xA0) */
/* print out : request register of the IRQ's */
/*      served by the IRQ controller. */
/*      set bits : the corresponding IRQ routine should be */
/*      run next, in as far as a higher */
/*      priority request doesn't precede it. */
/**-----**/
/* Info : before the IRQ controller can transmit the contents of the */
/*      ISR register the OCW3 labeled byte must be transmitted */
/*      to the controller                                     */
/*******/
BYTE irq_ReadIRR( INT iController )
{
    /* OCW3: 0x0B = no operation , no poll, read IRR */

```

```
    outp( iController, 0x0A );  
    return ( BYTE )inp( iController );  
}  
  
#endif
```