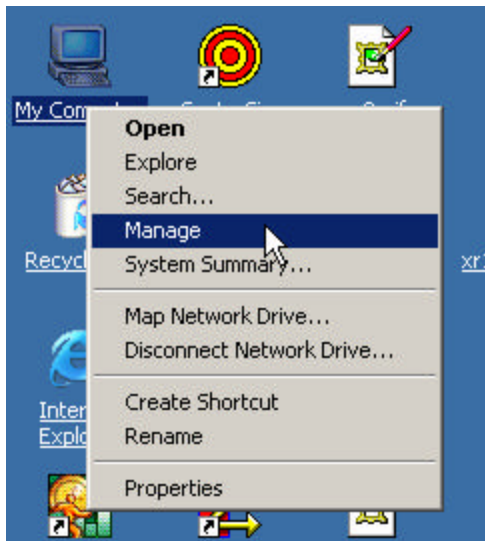


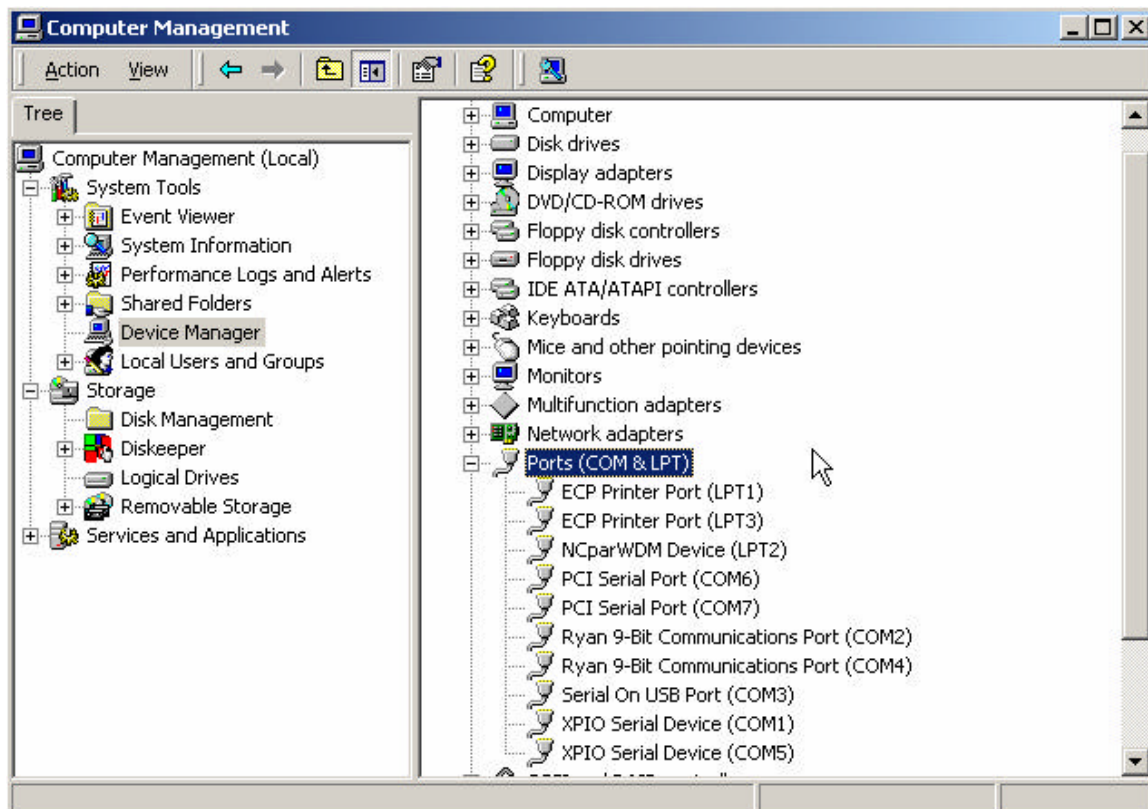
September 29, 2003

Notes about the use of serialNC.sys for Windows 2000/XP.

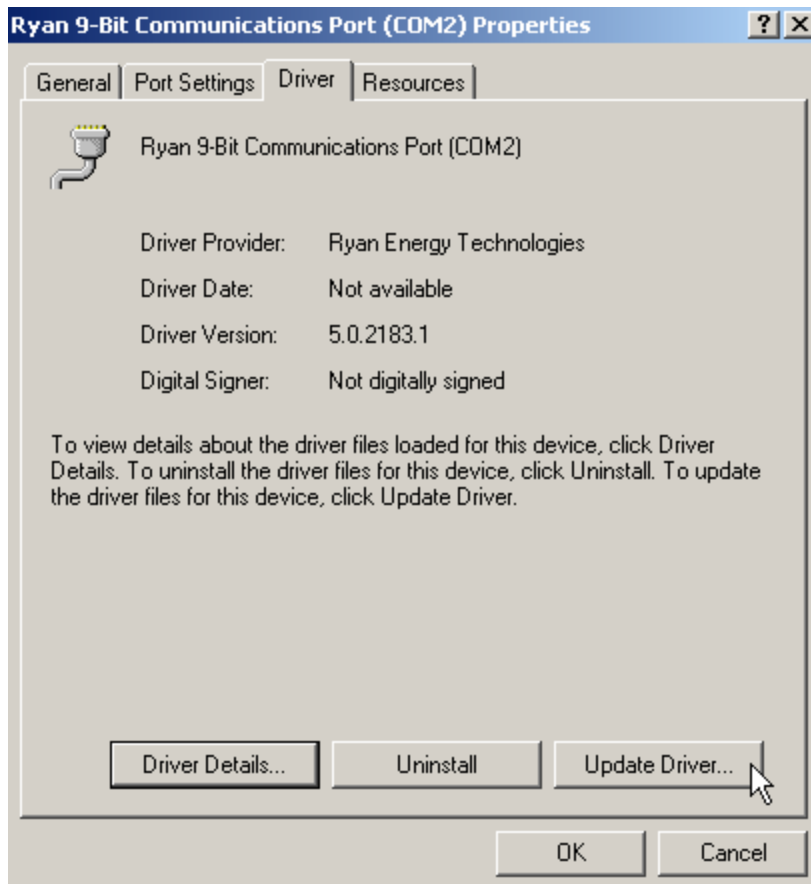
1. Place the two files serialNC.sys and Nilsenports.inf on a floppy. Install the driver using Device Manager: right click on My Computer on the desktop and go to Manager:



Next screen that comes up:

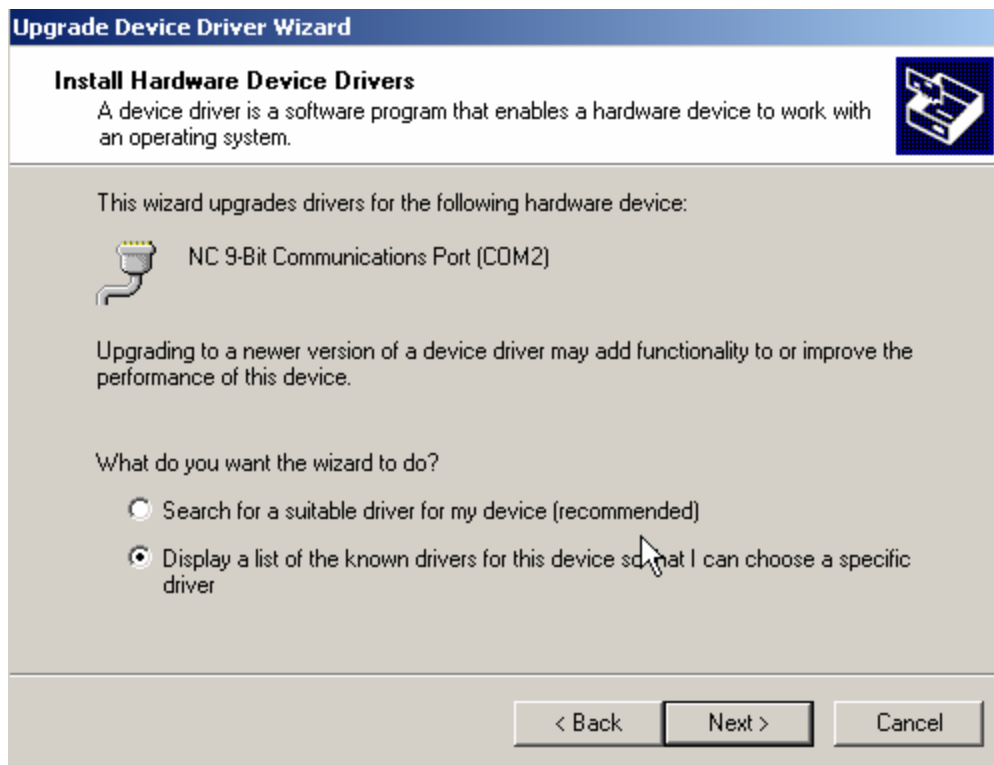


Double click on the port you want to install the driver on. Select the Driver tab:

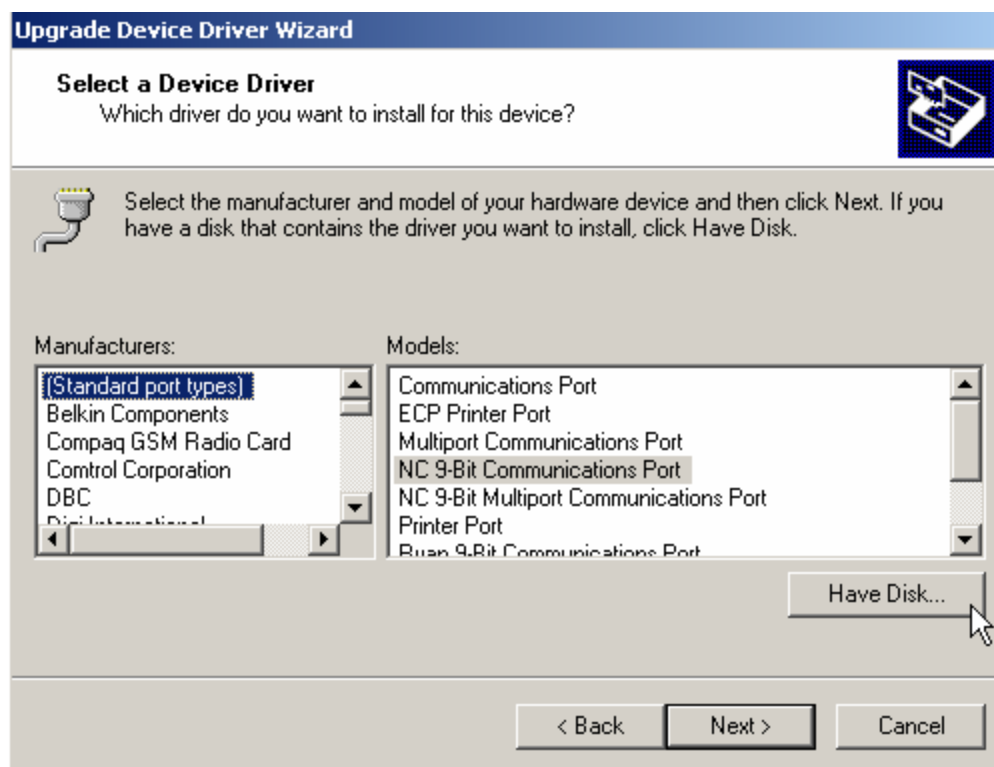


Select Update Driver...

Press next until you get to this screen:

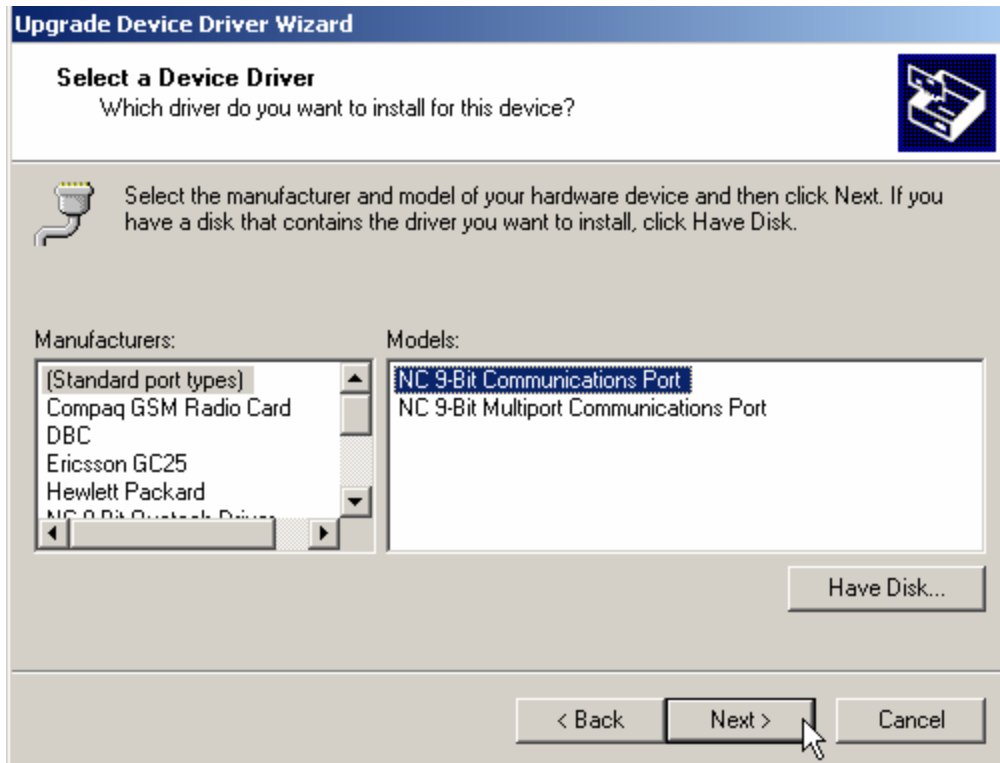


Change the selection to Display a list etc. Press next.



Select Have Disk...

Browse for the floppy and Nilsenports.inf



Select NC 9-Bit Communications Port

You'll likely have to reboot the PC.

The driver is now installed on the port. It will work just like the normal serial.sys for other programs and you can now use the special modes.

Using the special driver modes.

The normal serial.sys has the following parity modes defined (in DDK's ntddser.h):

```
#define NO_PARITY          0
#define ODD_PARITY        1
#define EVEN_PARITY       2
#define MARK_PARITY       3
#define SPACE_PARITY      4
```

The above are used when setting the port's baudrate, stop bits, parity, etc. The driver includes new modes:

```
#define MULTI_PARITY      5
#define SPACE_AND_RTS    6
#define NONE_AND_RTS     7
#define EVEN_AND_RTS     8
#define ODD_AND_RTS      9
#define MARK_AND_RTS    10
```

I determine if the driver is installed by testing if it accepts one of the above parities. The normal serial.sys will return an error. These modes control the RTS line as following:

When a message* is written to the port it first turns on the RTS line (+12VDC) and then waits for ½ ms (or so) before the first character is sent. The idea is to allow the bus to electrically settle after turning on the transmitter before transmission starts, but not wait for "too" long. When the last character enters the UART, the driver will wait for the UART to fully empty and at that very moment will turn off the RTS line (-12VDC). The idea here is to quickly turn off the RTS line at the completion of the message (i.e. transmitter off) before a responding slave starts its transmission. So a couple of milliseconds error is OK.

MULTI_PARITY is special in that the first character of the message will have the parity bit stuck at '1' and all other characters will exit the UART with the parity bit stuck at '0'. The other modes have standard parity behavior,

*(A message is one or more characters sent to the driver at one time using WriteFile().)

To open the port:

```
HANDLE porthandle;
OpenComPort(1, 19200, NONE_AND_RTS); //open with no parity and tight control
of the RTS line
```

```
BOOL OpenComPort(int port, ULONG baudrate, UCHAR parity)
{
    DCB portdcb;
    char portname[32];
    //*****
    //Create string of comport name
    //*****
    sprintf(portname, "\\.\COM%d", port);
    //*****
    //Open handle to comport
    //*****
}
```

```

porthandle = CreateFile(portname,
    GENERIC_READ | GENERIC_WRITE,
    NULL, //do NOT allow sharing of the port
    NULL,
    OPEN_EXISTING,
    NULL,
    NULL);
//*****
//Check to see if open succeeded
//*****
if (INVALID_HANDLE_VALUE != porthandle)
{
    //*****
    //Reset port - safe
    //*****
    ULONG len =0;
    //*****
    //Find out if port has Ryan Driver on it
    //*****
    //*****
    //Get the commstate
    //*****
    GetCommState(porthandle,&portdcb);

    portdcb.Parity                = parity;
    portdcb.fParity                = TRUE;
    portdcb.fBinary                = TRUE;
    portdcb.BaudRate                = baudrate;
    portdcb.fDtrControl            = DTR_CONTROL_DISABLE;
    portdcb.fDsrSensitivity        = FALSE;
    portdcb.fInX                    = FALSE;
    portdcb.fOutX                  = FALSE;
    portdcb.fNull                  = FALSE;
    portdcb.fOutxCtsFlow           = FALSE;
    portdcb.fAbortOnError          = FALSE;
    portdcb.fOutxDsrFlow           = FALSE;
    portdcb.fErrorChar              = FALSE;
    portdcb.fRtsControl            = RTS_CONTROL_DISABLE;
    portdcb.fTXContinueOnXoff      = FALSE;
    //*****
    //Set the commstate with special PARITY
    //*****
    if (SetCommState(porthandle, &portdcb))
    {
        //*****
        //Turn on DTR line to power Walt Helm dongles
        //*****
        DeviceIoControl(porthandle,IOCTL_SERIAL_CLR_RTS,NULL,0,NULL
            ,0,&len,NULL);
        DeviceIoControl(porthandle,IOCTL_SERIAL_SET_DTR,NULL,0,NULL
            ,0,&len,NULL);
        return(TRUE); //port is open and has Nilsen driver on it
    }
    //*****
    //Serial driver is not Nilsen's
    //*****
    CloseHandle(porthandle);
}
porthandle = NULL;
//*****
//We failed
//*****
return(FALSE);
}

```