

8n1

L.Lucius

COLLABORATORS

	<i>TITLE :</i> 8n1		
<i>ACTION</i>	<i>NAME</i>	<i>DATE</i>	<i>SIGNATURE</i>
WRITTEN BY	L.Lucius	July 28, 2024	

REVISION HISTORY

NUMBER	DATE	DESCRIPTION	NAME

Contents

1	8n1	1
1.1	8n1.device ~© by L.Lucius	1
1.2	Distribution	1
1.3	Disclaimer	1
1.4	Background	1
1.5	Installation	2
1.6	Compatibility	2
1.7	Thanks for the bug reports and suggestions	2
1.8	Feedback/Bugs	3
1.9	Note to NComm 3.0.? users	3

Chapter 1

8n1

1.1 8n1.device ~© by L.Lucius

8n1.device - By L.Lucius

A replacement for the standard serial.device
and based on v34serial.device.

```
~Distribution~~      What you can do with this
~Disclaimer~~~~     Blah Blah Blah
~Background~~~~     Why I decided to take on this task
~Installation~~     Just follow these simple instructions
~Compatibility~     Differences between 8n1.device & serial.device
~Thanks      ~      For the bug reports and suggestions
~Feedback/Bugs~    All reports good or bad are needed
~NComm Note~~~~   A note to NComm 3.0.? users
~Change~Log~~~~   A list of all changes made to 8n1.device
```

1.2 Distribution

You may use or misuse this program in any way you like.

1.3 Disclaimer

I will not be held responsible for ANY loss incurred by this program.

1.4 Background

When "v34serial.device" showed up on Aminet, I was excited, because all I ever used was 8N1 and RTS/CTS. That was it. I didn't need parity or XON/XOFF or breaks and thought that if the device didn't have to worry about all that

then it would be faster.

Well, I used it for awhile, but every now and then it would cause GURUs, so I went back to "artser.device" and forgot about it.

Until a little while ago, when I found out that I would be getting a SLIP connection to the net.

I wanted something faster than "artser.device" and with less overhead. So I went to debugging "v34serial.device" and wound up rewriting the whole thing.

The end result is a minimal serial device replacement that tries to keep system overhead at a reasonable level.

1.5 Installation

Simply copy 8n1.device to the DEVS: directory and tell your communications software the new name. NComm users read this!

If your communications software is unable to accept the name of the serial device to use, get the file on Aminet called SerPat20.lha in the hard/drivr directory. It will give you this capability.

!!!IMPORTANT!!!

If you have two or more programs that share the serial port, make sure you change ALL programs. Otherwise, you will probably get a visit from some guy called GURU. (Thanks Juan!)

1.6 Compatibility

8n1.device should be compatible with the "serial.device" as long as the following options are the only ones used: B-)

- 8 data bits
- No parity
- 1 stop bit
- RTS/CTS handshaking
- No handshaking
- EOFMODE

It also supports sending breaks, but not receiving them.

If you need something else, let me know and I'll see what I can do.

1.7 Thanks for the bug reports and suggestions

Thanks to the following people for their reports, testing, and patience:

William Crawford IV

Mans Engman

Ayan George

Paul Harrison	Harold H. Ipolyi	Mathias Juvall
Alpay Kasal	George Kourkoutas	Laura Mahony
John Millington	Brian Myers	Robert N. Olson
Greg Olstad	Ernest Otte	Koen Peetermans
Juan Ramirez	Michal L. Rybarski	George Sanderson
Orlando Santiago	Peter S. Struijk	Tinic Urou
Ashok Vadekar	Ronald van Eijck	Alexander Wild
Dwight Zenzano		

Let me know if I've missed anyone.

1.8 Feedback/Bugs

I have included the source in hopes that some other programmers can make suggestions. It's always better to have more than 1 eye (literally B^) looking at the code.

If you have problems or feedback I would be very glad to hear about it:

Email me at: llucius@millcomm.com

1.9 Note to NComm 3.0.? users

I spent over 2 weeks trying to track down a bug that I (and everyone else) thought was a bug in 8n1. I mean it made sense that 8n1 was the problem since ALL the other drivers worked, right? Well...

Using Stefan Pröls's excellent `serlog.device` and another program (and a lot of paper for debugging), I was able to determine that 8n1 was working the same as all the rest.

So, if it works the same then why the problem?

Well, the problem occurs when you try to upload using `zmodem`. One of the first things `zmodem` does is sync up with the other end. Part of this process checks to see if any characters have been transmitted by the other end. To do this, `zmodem` asks NComm how many characters are available and NComm tells `zmodem` that one character is ready to be read. But, there isn't and since `zmodem` doesn't know that, it reads a character and `get's` whatever happens to be at the beginning of NComms read buffer.

To further illustrate, here's the sequence of I/Os that `artser` goes through just prior to where the problem would occur:

```

CMD_READ (      ): OK, 1 of 1 byte
0000: 2A                                     "*"

SDCMD_QUERY (QUICK): OK
status:  $0004
buffered: 12

CMD_READ (QUICK): OK, 12 of 12 byte

```

