Why are there two incompatible ways of specifying a serial port baud rate?

devblogs.microsoft.com/oldnewthing/20170426-00

April 26, 2017

Raymond Chen

One of my colleagues tracked down a bug in their code that communicates over a serial port. (Remember serial ports?)

The DCB structure specifies the baud rate as an integer. To request 2400 baud, you set the BaudRate to 2400. There are some convenient defined constants for this purpose.

```
#define CBR_110             110
#define CBR_300             300
#define CBR_600             600
#define CBR_1200            1200
#define CBR_2400            2400
#define CBR_4800            4800
#define CBR_9600            9600
#define CBR_14400           14400
#define CBR_19200           19200
#define CBR_38400           38400
#define CBR_56000           56000
#define CBR_57600           57600
#define CBR_115200          115200
#define CBR_128000          128000
#define CBR_256000          256000
```

Meanwhile, the COMMPROP structure also has a way of specifying the baud rate, but it is done by setting the dwMaxBaud to a bitmask:

```
#define BAUD_075          ((DWORD)0x00000001)
#define BAUD_110          ((DWORD)0x00000002)
#define BAUD_134_5        ((DWORD)0x00000004)
#define BAUD_150          ((DWORD)0x00000008)
#define BAUD_300          ((DWORD)0x00000010)
#define BAUD_600          ((DWORD)0x00000020)
#define BAUD_1200         ((DWORD)0x00000040)
#define BAUD_1800         ((DWORD)0x00000080)
#define BAUD_2400         ((DWORD)0x00000100)
#define BAUD_4800         ((DWORD)0x00000200)
#define BAUD_7200         ((DWORD)0x00000400)
#define BAUD_9600         ((DWORD)0x00000800)
#define BAUD_14400        ((DWORD)0x00001000)
#define BAUD_19200        ((DWORD)0x00002000)
#define BAUD_38400        ((DWORD)0x00004000)
#define BAUD_56K          ((DWORD)0x00008000)
#define BAUD_128K         ((DWORD)0x00010000)
#define BAUD_115200       ((DWORD)0x00020000)
#define BAUD_57600        ((DWORD)0x00040000)
#define BAUD_USER         ((DWORD)0x10000000)
```

My colleague accidentally set the `DCB.BaudRate` to a `BAUD_xxx` value, and since these values are untyped integers, there was no compiler warning.

My colleague asked for the historical background behind why there are two easily-confused ways of doing the same thing.

The `DCB` structure dates back to 16-bit Windows. It tracks the feature set of the 8250 UART, since that is what came with the IBM PC XT.[1] In particular, there is no need to ask what baud rates are supported by the serial chip because you already know what baud rates are supported by the serial chip: The 8250 and 16650 support baud rates that are divisors of 115200.[2]

Enter Windows NT. This operating system wanted to run on things that weren't IBM PCs. Crazy. In particular, those systems may have serial communications chips that support a different set of baud rates. That's where the `COMMPROP` structure came in: It reports baud rates as a bitmask that is filled out by the `GetCommProperties` function. That way, the program that wants to do serial communications can find out what baud rates are supported by the current hardware. And since it's reporting a set of values, a bitmask seems the natural way of representing it.

The program inspects the bitmask, decides which of the available baud rates it wants to use, and puts the desired value (as an integer, not a bitmask) in the `BaudRate` member of the `DCB`.

That's my attempt to reverse-engineer the history of the two incompatible ways of representing baud rates.

[1] The PS/2 line introduced the 16550 UART which is backward-compatible with the 8250. In particular, it supports the same baud rates.

[2] Other baud rates like 110 are approximations. For example 110 is really 115200 ÷ 1048 = 109.92 baud. This article claims that microcontrollers “rarely offer an internal oscillator that has accuracy better than ±1.5%,” so an error of 0.07% is easily lost in the jitter.

Raymond Chen

Follow