# Rust serialport-rs Crate Testing and Cross-platform Discussion Items

After performing a nominal amount of focused Serialport-rs behavior/timing tests using Windows 10 and Ubuntu Linux over the past week, here are my findings with suggestions.

1. The SerialPort trait's current set\_timeout() method only applies to the port's read() timeout. Based on testing with the read timeouts on the Windows and (Ubuntu) Linux platforms, the SerialPort read() operation varies between these two platforms when using a (read) timeout setpoint of 0 ns.

On Windows a timeout of 0 results in an infinite timeout if no data bytes have been received: i.e. read() blocks indefinitely. On the contrary, with Linux a timeout of 0 results in the read() returning immediately if no data is available.

There currently isn't an exposed port write() timeout setting.

On Linux, the SerialPort write() internal timeout setting apparently defaults to 0 ns. This results in a timeout error return value for a write() of more than a few data bytes if no matching read() is concurrently active.

On Windows, the write() timeout defaults to disabled - which means no transmit timeout occurs. Lengthier Windows write() operations which do not coincide with a concurrently matching read() operation will drop bytes - which are 'missed' by the receiving end-point.

1. Upon examining SerialPort's crate's Windows and Linux implementations of read() and write() methods, I noticed that the Posix/Linux platforms internally use wait\_read\_fd and wait\_write\_fd methods respectively. These of course are based on the Linux/Unix versatile mechanism of file/device descriptors (fd) with os system call 'select/poll' operations which support application specified timed waits for detecting changes in a given devices i/o read/write/error state.

While Windows provides a relatively equivalent set of functional features related to serial timeouts, these are designed different from Linux. Windows serial port read and write timeouts require three separate configuration integer values rather than the single integer currently supplied to the crate's exposed set\_timeout() method.

Note that many serial port communication applications are based in the instrument/machine and embedded control world, and have challenging < 100 millisecond xmt/rcv time resolution requirements. A software developer using a crate such as Serialport-rs in these domains needs access to the critical serial port system configuration parameters which are provided by each platform in order to develop fast and robust communications code.

The Serialport-rs crate's exposed API is a great foundation, but should be thoughtfully extended with certain platform specific methods and settings. These extensions should expose platform-specific receive and transmit timeout settings, as well as a more fully detailed API documentation (with platform specific sections). I also suggest considering platform-specific asynchronous i/o feature as I have concerns that relying on external crates for async serial I/O may be unnecessarily inefficient.