RFC2217 Set Xon/Xoff Extension

Grant Edwards
Comtrol Corporation

May 12, 2010

Abstract

This paper presents a set of new values for use with the telnet com port control option's SET-CONTROL sub-command defined in RFC2217. These new values provide a mechanism for the client to control and query the outbound Xon/Xoff flow control state of the device server's physical serial port. This capability is exposed in the serial port API on some operating systems and is needed by telnet clients that implement a port-redirector service which provides applications local to the redirector with transparent access to the remote serial port on the telnet server.

1 Background

To better understand the need for this extension to RFC2217, some understanding of serial device servers and their usage is required.

1.1 Serial Device Servers

Serial device servers are essentially network-connected serial ports. They are boxes with one or more serial ports and a network connection (either wired or wireless). They range from devices smaller than a deck of cards with a single serial port up to 19" wide 1U rack mount devices with 32 serial ports.

A serial device server provides client computers with remote access to the serial ports present on the serial device server. Figure 1 shows an example of device server usage. There are two basic methods for providing access to the serial ports.

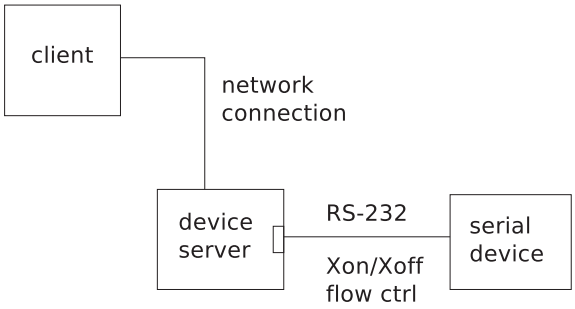

Figure 1: Device Server Usage Example

First, the serial ports can be accessed directly by user applications that open network connections to the device server. Typically a separate TCP/IP connection would be used for each serial port.

Second, a device-driver or port-redirector can be installed on the client computer where the user application is running. The driver/redirector handles the network connection(s) to the device server and provides local user applications with "virtual" serial ports that are accessed using the standard system API that is used for local serial ports. On a Windows OS, the user application sees the remote serial ports as normal "COM" ports (e.g. COM5, COM6, COM7). On a Unix OS, the user application sees the remote serial ports and normal /dev/ttyXX devices (e.g. /dev/ttySI0, /dev/ttySI1).

For a new application being written from scratch, directly accessing the serial port by opening a network connection can be the easiest method. However, the "native serial port API" approach is often the only choice for legacy applications that were previously used with local serial ports or for applications that must work with both local and remote serial ports.

1.2 Device Server Protocols

There are a variety of protocols that are used to access serial ports on device servers.

RAW TCP On the "simple" end of the spectrum a single, raw TCP/IP connection can be used with no in-band signaling. A separate connection is used for each serial port. Each byte received on the serial port is transmitted by the device server on the TCP/IP connection. Each byte received on the TCP/IP connection by the

device server is sent out the serial port. Since the connection provides no way to configure the serial port (parity, baud rate, etc.), that configuration must be done separately by another mechanism such as a web page in the device server. This approach also provides no way to read/write the modem control and status lines present on most serial ports. Such raw TCP connections could be used by either a driver/redirector or by an end-user application.

Proprietary Multiplexed A vendor-specific protocol is often used between a driver/redirector and a device server. Such protocols will often multiplex data from multiple serial ports so that only a single network connection is required between the driver and device server regardless of how many serial ports are in-use. These proprietary protocols also typically provide mechanisms to configure the serial port and to read/write modem control/status lines. This provides for the same level of functionality as provided by a local serial port.

The disadvantage of such protocols is that they tend to be complex and vendor-specific. Thus they aren't suitable for direct use by user applications, and they limit the inter-operability of device servers and drivers/redirectors.

Telnet/RFC2217 The telnet protocol [IETF STD 8: RFC854, RFC855] was developed in the early days of the Internet to define the network connection between a "terminal" program (a telnet client) and a "virtual serial port" (a telnet server) on a remote computer which was in turn connected to an application running locally on that remote computer. It provided some minimal in-band signaling that allowed for things like sending a "break" signal.

Since the telnet protocol was used to connect to remote virtual serial ports, it did not include provisions for configuring and accessing features present only on physical serial ports: baud rate, parity, modem control and status lines, hardware and software flow control, etc.

RFC2217 extends the telnet protocol to provide those mechanisms that are required for effective remote use of physical serial ports. It provides commands to configure baud, parity, stop bits, flow control and modem control lines. It also provides a mechanism for the device server to notify the telnet client of changes on modem status lines.

With the RFC2217 extensions, the telnet protocol has become a popular "standard" protocol for use between serial port driver/redirectors and serial device servers. Compared to proprietary multiplexed protocols telnet/RFC2217 has some disadvantages. It requires a separate TCP connection for each serial port. For de-

vice servers with high port densities, this may be a minor issue. It is also somewhat inefficient in its handling of binary data streams that contain a high percentage of bytes with a value of FF_{16}.

Despite these minor drawbacks, it has become the *de facto* standard protocol for non-proprietary communication between serial device servers and port redirectors, and it is being seen more frequently as one of the options for accessing remote serial ports.

1.3 Xon/Xoff Flow Control

Xon/Xoff flow control is an in-band signaling method used between a serial port and a serial device to control the flow of data. If the serial device connected to a serial port sends an Xoff character, the serial port will stop sending data until the serial device sends an Xon character. This allows the device to control the flow of data in cases where it's possible that the data could be sent faster than it could be processed.

Operating systems generally provide in the serial port API a way to enable and disable Xon/Xoff flow control. Such a mechanism is also defined in RFC2217.

In addition to being able to enable/disable the Xon/Xoff flow control feature, some operating systems also provide in the serial port API a way to control the state of the Xon/Xoff feature after it has been enabled. In other words a user application can "force" a serial port to behave as if it has received either an Xoff character (stopping transmission) or an Xon character (resuming transmission).

2 Problem

There is no way using telnet/RFC2217 to control the state of Xon/Xoff flow control once it has been enabled. The feature can be enabled or disabled, but while enabled its state can't be forced. While this may seem to be a trivial omission, it is something that can be done with local serial ports and with many drivers/redirectors using proprietary protocols. It is actually something that is required by many customers, and a real world example of its use is detailed at the end of this paper.

How do we provide this capability when using a driver/redirector and device server that use the telnet/RFC2217 protocol?

Do it in the driver/redirector? One suggestion might be to handle the Xon/Xoff flow control in the driver/redirector instead of remotely in the device server. At first glance this seems like a good solution. If it is handled in the driver/redirector, then the network protocol between the driver/redirector and the device server doesn't need to accommodate such a feature. When the driver/redirector sees the Xoff character that was sent by the serial device, the driver/redirector will stop sending data to the device server. However, the device server generally has an internal transmit data buffer, and any buffered data will continue to be sent by the server to the serial device. In order to allow for efficient network usage and smooth data transmission, that transmit buffer might hold several thousand bytes. That means that when the serial device sends an Xoff, it might continue to receive several thousand bytes of data before the device server's buffers are empty. When a serial device sends an Xoff, it usually expects data to stop quickly – receiving only a few more bytes. If it receives several thousand bytes, it might not have room to store them.

Use the commands defined in RFC1372? RFC1372 "Telnet Remote Flow Control Option" sounds like it might provide for control of the Xon/Xoff state. On close reading it is apparent that it does not. It provides for enabling and disabling the Xon/Xoff flow control feature, but does not provide for controlling the state of the feature while it is enabled. Thus RFC1372 merely duplicates functionality present in RFC2217.

Extend RFC2217? Another suggestion is to extend the commands defined in RFC2217 by the addition of commands to force the Xon/Xoff flow control state. This would take advantage of RFC2217 command parsing code already present in both driver/redirector and device servers. This is the approach that was chosen by Comtrol. The details of the proposal are presented below. They have been submitted to the IETF as an "Internet-Draft" in hopes of becoming an RFC, and Comtrol would like to encourage other vendors of RFC2217 compliant products to adopt this extension. This Internet-Draft is available for review at the URL below, and comments are invited from interested parties.

http://datatracker.ietf.org/doc/draft-edwards-telnet-xon-xoff-state-control/

3 RFC2217 Extension Proposal

The relevant portion of RFC2217 is the SET-CONTROL command defined in section 3. The command has this format with decimal byte values shown:

IAC	SB	COM-PORT-OPTION	SET-CONTROL	*value*	IAC	SE
255	250	44	5	*value*	255	240

Where *value* is defined by RFC2217 as:

Value	Control Command
0	Request Com Port Flow Control Setting (outbound/both)
1	Use No Flow Control (outbound/both)
2	Use XON/XOFF Flow Control (outbound/both)
3	Use HARDWARE Flow Control (outbound/both)
4	Request BREAK State
5	Set BREAK State ON
6	Set BREAK State OFF
7	Request DTR Signal State
8	Set DTR Signal State ON
9	Set DTR Signal State OFF
10	Request RTS Signal State
11	Set RTS Signal State ON
12	Set RTS Signal State OFF
13	Request Com Port Flow Control Setting (inbound)
14	Use No Flow Control (inbound)
15	Use XON/XOFF Flow Control (inbound)
16	Use HARDWARE Flow Control (inbound)
17	Use DCD Flow Control (outbound/both)
18	Use DTR Flow Control (inbound)
19	Use DSR Flow Control (outbound/both)
20-127	Available for Future Use

The new values proposed by Comtrol for this command are:

20	Request Xon/Xoff state
21	Set XOFF state
22	Set XON state

The value 20 “Request Xon/Xoff state” would be sent by the driver/redirector to request the current Xon/Xoff flow control state. The value 21 “Set Xoff state” would be sent by the driver/redirector to force the device server to act as if it has received an Xoff character from the serial device (stopping data transmission). The value 22 “Set Xon state” would be sent by the driver/redirector to force the device server to act as if it had received an Xon character (resuming data transmission).

The responses to all three of the newly defined sub-command values will reflect the flow control state after execution of the command. Examples are shown below (byte values are in decimal). “Command” is client → server and “Response” is server → client:

Query Xon/Xoff State Command:

IAC	SB	COM-PORT-OPTION	SET-CONTROL	REQUEST-XON/XOFF-STATE	IAC	SE
255	250	44	5	20	255	240

Response:

IAC	SB	COM-PORT-OPTION	SET-CONTROL	SET-X[ON|OFF]-STATE	IAC	SE
255	250	44	105	[21/22]*	255	240

*value in response is 21 if state is XOFF (tx suspended) and is 22 if state is XON (tx allowed).

Set XOFF state (stops tx data) Command:

IAC	SB	COM-PORT-OPTION	SET-CONTROL	SET-XOFF-STATE	IAC	SE
255	250	44	5	21	255	240

Response:

IAC	SB	COM-PORT-OPTION	SET-CONTROL	SET-XOFF-STATE	IAC	SE
255	250	44	105	21	255	240

Set XON state (starts tx data) Command:

IAC	SB	COM-PORT-OPTION	SET-CONTROL	SET-XON-STATE	IAC	SE
255	250	44	5	22	255	240

Response:

IAC	SB	COM-PORT-OPTION	SET-CONTROL	SET-XON-STATE	IAC	SE
255	250	44	105	22	255	240

The addition of these three commands allows drivers and redirectors to properly implement the serial port API that provides user applications a method to control the Xon/Xoff flow control state.

For example, on a Win32 system, a redirector would send a “set-xoff-state” command to the device server when the user application calls EscapeCommFunction(handle,SETXOFF) on the virtual COM port. Likewise, it would send

a “set-xon-state” command when the user application calls EscapeCommFunction(handle,SETXON) on the virtual COM port.

4 Real World Usage

One real-world use of the above feature that has been encountered regularly by Comtrol is the downloading of program files to computer numeric controlled (CNC) machine tools such as lathes, milling machines, drill presses, etc. Such tools are often distributed throughout a facility with the controlling programs stored centrally on a single computer. When a machine tool is being set up for a particular job, the appropriate program data must be downloaded to the machine tool from the computer where the program data are stored.

This download is often done via a serial port on the machine tool. In the case of interest, the download process begins when a management application on the central computer opens a serial port. Xon/Xoff flow control is enabled on that serial port and the port is forced into the “Xoff” state so that it won’t transmit data until it receives an Xon character from the machine tool. The application then starts writing the program data to the serial port. The data isn’t transmitted out the serial port at this point in time but rather is buffered by the serial port driver.

The machine tool is then placed in programming mode and the transfer of data is initiated by the machine tool by sending an Xon character.

In the past, individual serial cables were run – one to each machine tool. In addition to the expense of running the serial cables, this required either the installation of many serial ports on the central computer or a switching mechanism to select the machine tool with which the computer needs to communicate. More recently, the individual serial cables have been eliminated by installing a single-port serial device server on each of the machine tools – allowing the machine tools to be placed on a local area network. In some installations use of single-port device servers with built-in 802.11 wireless networking allows the elimination of networking/data cabling entirely. Figure 2 shows a machine shop installation where DeviceMaster FreeWire device servers have been installed on CNC milling machines so that an 802.11 WiFi network can be used to eliminate the need to route either serial or networking cables to the machine tools on the shop floor.

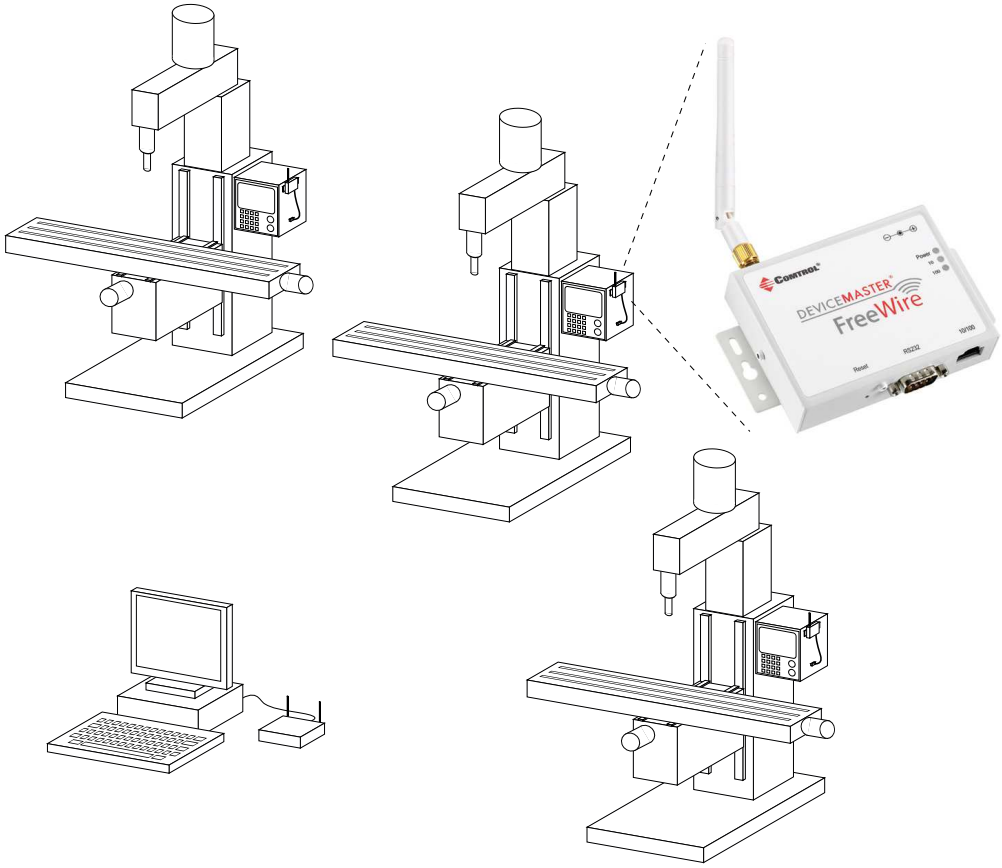

Figure 2: Machine Shop Installation Example