Kuper Serial Control

Starting in about 2002, external serial control was slowly added to the Kuper software. Initially, this was a mechanism to allow users to remotely load an ascii file via a serial link, but was eventually control expanded to the point where most of the two letter keyboard commands can be used through the serial port.

Although some versions of serial control were present in software dating back to 2002, for reliable results, use a software revision from 2/21/07 or later. Kuper preformed a major re-write of the serial handler for this revision, and it cured many of the pesky timing problems that plagued previous versions.

Enabling serial control

External serial commands can be enabled (or disabled) using the “so” command.

“so1” turns the serial command mode on, “so0” turns it off.

Additionally, the Kuper can be configured to enable the serial command mode automatically when the software starts up.

In order to do this, you create a “pragma” file in the Kuper directory called “extcom”. No contents or extensions are necessary. The Kuper software looks for this file on start-up, and if it finds the file, the software enables external control.

The simplest tool to create the extcom file out of thin air is the DOS “edit” command.

For those rusty in the ways of DOS, make sure you’re in the Kuper directory, and type “edit extcom<enter>”. You’ll open a basic but effective text editing program and a new file named extcom. The contents of extcom don’t matter, just its existence, so type a couple of spaces and save it. Quit edit and you’re done.

Please note – whichever way you turn serial control on, the Kuper will immediately start transmitting status bytes to the serial port in certain situations, especially functions related to the stop motion modes.

When the Kuper transmits bytes to the serial port, it will expect to see some sort of device out there acknowledging them. Specifically, it will expect to see a correct handshake signal on the hardware handshake lines, CTS and DSR on the RS232 port.

If there is no device responding the Kuper will dutifully wait. And wait. And wait. Computers are good at waiting, and from the operator’s point of view, it will look as though the Kuper has effectively locked up.

It is therefore *very important* that you have something connected to the Kuper's serial port before you enable external serial control, and the best thing to have is a loopback null cable since that doesn't require an active device responding at the far end. See the section on "cables" for more information.

Command formats

Kuper serial commands and regular keyboard commands enter the software through two distinctly different paths.

The serial interpreter also handles file transfers and diagnostic messages, so it has to pass many more complicated strings than the keyboard interpreter. Consequently, exhaustive error checking would be prohibitive.

The serial handler uses much less comprehensive error-checking than the keyboard handler. It basically expects that the computer at the other end typed its command correctly, and tries to do what it was told.

Consequently, it is important to get the commands correct, lest you instruct the Kuper to do something you don't want.

Almost any command you can type through the keyboard can come in through the serial port, but bear in mind that there are commands that can't easily be shut off (like fb – start frame browsing move) can cause unplanned movement (like ev – change encoder modes) or can erase unsaved data (like nm - new move).

For commands that require arguments, the same formatting used for the keyboard version applies.

Additionally, these special rules apply at all times…

- In order to maintain backward compatibility with older RTMC48 cards (which do not have on-board ports) this function is hard mapped to COM2 (2F8h, IRQ3).

- The UART chip must be at least a 16550 or better with a 16 character buffer or larger (typically not a problem for modern motherboard-mounted serial ports)

- The serial format is 9600 baud / 1 stop bit / no parity. The Kuper uses hardware flow control. This raises handshaking issues, please see the section on cables.

- All serial commands sent to the Kuper should consist of only lower-case letters, numbers, and the special characters space (20h), comma (2Ch) and decimal point (2Eh).

- All serial commands must terminate with a ASCII carriage return and line feed characters (0Dh,0Ah) in that order. That's the parsing token Kuper uses to validate a new line.

- Give the Kuper at least 125ms between commands to respond and clear its buffers.

- Additional special rules apply for loading moves through the serial port.

Special serial commands – part 1, "ms"

In addition to the most of the standard two letter commands (gh, za, etc) there are "special" commands that are used principally for serial control.
Chief among these is "ms", or motor status. Issuing an ms command (either into the port or from the keyboard) causes the Kuper to transmit a string indicating whether any of the motors is moving.

In this way, a computer that has issued commands which cause the rig to move can test to see if the rig is in fact moving, and know when it has finally stopped.

The ms command generates a serial string that typically looks like this…

"ms011000000000000000000000000000000000000000000000<cr><lf>"

…where each 1 or 0 represents a motor channel, with a 0 representing an axis that is stopped, and a 1 representing an axis in motion.

In this example, axes 2 and 3 are moving, while the rest are stopped.

A 48 channel card generates a 52 byte response, and even in computer time, it takes a pointlessly long time to transmit all these details when all we care about is if *any* motors are running, so newer versions of the software are transitioning to a "succinct" version that returns just one bit, "ms0" or "ms1" indicating if any motors are still moving.

A typical exchange might look like the following example, where a control computer moves the rig to frame #3 using the pf (position frame) command.

Control computer sends...	Kuper Replies…
pf 3<cr><lf>	(none)
ms<cr><lf>	ms1<cr><lf>
ms<cr><lf>	ms1<cr><lf>
ms<cr><lf>	ms0<cr><lf>

In this case, the computer issued the position command, specifying frame #3 ("pf3" - position to frame 3). This initiates the move, but the control computer does not know when the rig has stopped moving, so the computer then pings the Kuper every quarter second or so with an "ms" to see if the command has finshed yet.

After the last "ms" came back with no motors still moving, the control computer would know that the reposition was finished, and it could go on to the next step, like making an exposure.

Special serial commands – part 2, "P1", "P2", "P3"

Because there is no good way to deal with animation sub-frames via serial commands, triggerbits and editbits are more difficult to use for things like front-light/back-light passes. To compensate for this, there are three new commands, P1, P2 and P3, to control these ports directly.

P1 controls LPT1 (typically 378h) P2 controls LPT2 (278h) and P3 controls the RTMC logic port on the Kuper card (triggerbits)

These commands are identical, they open with a port number token, then list the polarity of each bit, starting low and working high. For example…

P1 11110000<cr><lf> sets1's on the low 4 bits, 0's on the high 4 bits of LPT1.

P2 00000001<cr><lf> sets a 1 on the high bit, 0's on all other bits of LPT2.

Note that in this mode, 1 and 0 refer to the actual logic level of the physical port. A 1 is a logic high, typically more than 3 volts, and a 0 is a logic low, typically less than 0.7 volts. This applies on both the LPT ports and the RTMC triggerbits port, even though we've traditionally thought of these ports as "active low".

Active port control through serial commands won't work correctly when other parts of the software are also trying to use those ports, so you have to turn the editbit function off, and set all the triggerbits off in the animation exposure menu.

Special serial commands – part 3, "S1"

The special command "s1" is equivalent to pressing the shoot switch, and swill initiate any cued cycle, including shoot frame, go motion or shoot-and-move, as appropriate to whatever is set up on that sub exposure.

Animation mode status

When in the external serial port is turned on, and the Kuper is shooting in animation mode, every time it's ready to shoot a frame, it will report it's status to the serial port.

Say, for example, you're shooting a 10 frame move with two subframes on each frame.

When you hit the forward switch, the Kuper will move to frame 0 and report over the serial line that it is ready to shoot the first exposure of frame zero. Specifically, it will issue the string…

```
fr0: ex0<cr><lf>
```

As you continue to shoot exposures, and the Kuper readies itself for the next exposure, it will keep the serial port updated …

```
fr0: ex1<cr><lf>
fr1: ex0<cr><lf>
fr0: ex1<cr><lf>
```

All the way up to the last frame…

```
fr9: ex1<cr><lf>
```

Then, when the move ends…

```
STOP <cr><lf>  (in capital letters)
```

If the move is stopped by the operator before the end, the Kuper will also issue STOP<cr><lf>.

Once again, regardless of whether or not you're sendign serial commands into the Kuoer, if the serial function is turned on, the the kuper will try to issue status updates serially.

Since the Kuper is issuing characters from the serial port, it expects hardware handshaking from an external device and will likely hang if it doesn't get any.

See the next section.

Cables

Because it has to interface with so many other devices, of varying vintage and capability, the Kuper uses hardware handshaking. This is an older protocol whereby byte transfer between machines is synchronized with separate timing signals in the cable. This was necessary in ye olde days of slow computers, because it was possible to transmit bytes faster than the receiving computer could deal with them. It survives as a legacy format.

This leads to an unintended consequence, because the Kuper can be induced to hang if you tell it to transmit a string from the serial port and it never gets a handshake response from the other end of the line. It just sits and waits, and waits, and waits for a reply that never comes.

This can happen if you try to use serial commands and there's nothing attached to the serial port, because there's no device to generate the response. It should be noted that once you turn the Kuper serial mode "on" the Kuper will regularly transmit status information when in animation mode, even if not prodded. Therefore a serial enabled Kuper can lock itself up all by itself even if there is there's nothing connected to the serial port.

It's pretty difficult to fill up the port buffer on a modern processor board, especially at a mere 9600 characters a second, but it's still possible if something happens in the receiving machine so that the software applications there stops servicing the serial port.

In this case, the Kuper will continue to run, sending status messages down the serial line, and the incoming characters fill up the remote machine's port buffer. Eventually, the buffer will overrun, and the port in the remote machine will issue a hardware command down the serial line, on the CTS and DSR lines meaning "I'm full, stop transmitting".

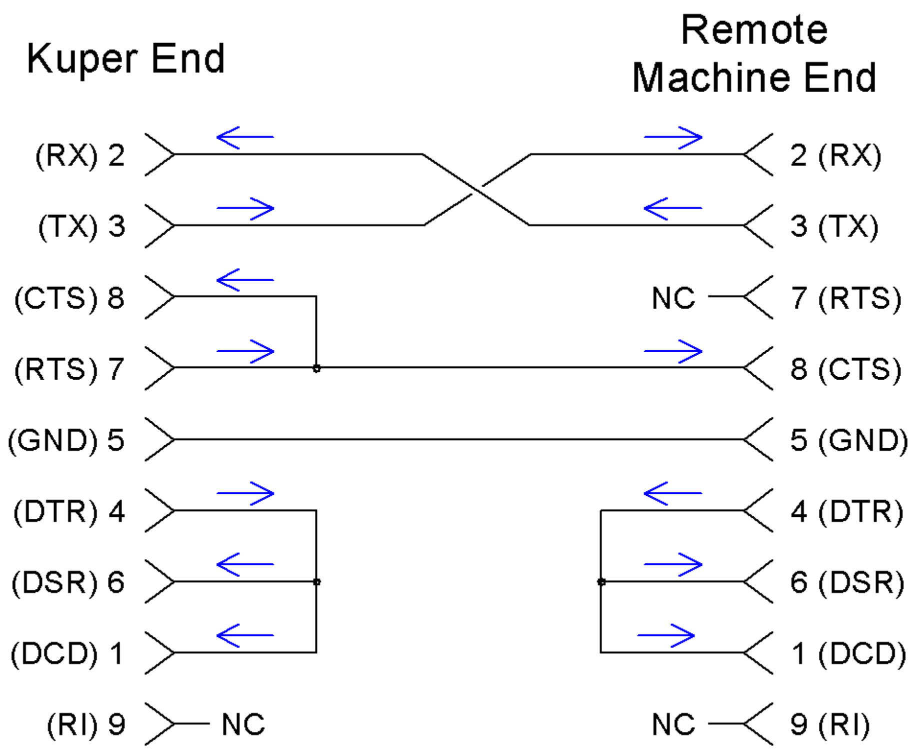
The Kuper will then dutifully stop and wait, forever if need be, until the hold clears.

For these reasons, you should always use a "loopback" null modem cable. Unlike the full 9-wire type, loopback null cables wrap back the handshake lines, so in effect the transmitting side issues its own clearance and will never see a "stop transmitting" command coming from the remote end.

This does defeat the purpose of hardware handshaking, but modern machines are so fast hardware handshaking is an anachronism. Since byte packets are very small and the serial speeds are very low in this application, in any reasonably current system there should be more than enough processing power available in the remote machine to clear out the buffers before they come close to overflow.

And frankly, even if there wasn't, you still want to drop status packets rather than hang up the Kuper.

Ultimately, I would suggest that if you're going to do a lot of serial control, you go so far as to make your own specialty cables, wired thusly…



And leave the Kuper end permanently connected to the Kuper's serial port.

Wired thusly, the Kuper RTS to remote CTS line allows the Kuper to pause a long serial stream coming up from the remote end, but the Kuper RTS loopback will not allow the remote machine to pause the Kuper.

This addresses the reality that the remote machine has more processing power available for error checking and recovery, and if all goes badly anyhow, it’s better to hang up the remote end (where the application can easily be closed and restarted) than it is to hang up the Kuper (which requires a hardware re-boot and re-finding zero).

By leaving the Kuper end permanently attached to the motion control computer, the Kuper can issue serial status strings even when there’s no remote attached and it won’t lock up the system if the remote computer is disconnected or de-powered while the Kuper serial mode is engaged.

Other serial ports in the Kuper system

There are several other entry points into the Kuper software where data come in and goes out, and though these aren't as directly applicable to serial control, they should still be noted.

The oldest serial port is the serial output routine in the hardware setup menu. Originally, this port was designed to provide a real-time frame count to external video systems, and rig position to "smart" devices such as motion bases.

It is capable of real-time export or frame number, timecode number, virtual machine position or real machine position at up to 115200 baud through either COM1 or COM2.

It's particularly important to us because, if mapped to COM2 it can interfere with external commands by sending a great deal of information to the serial port that we're not expecting, or by fighting for baud rate control.

This port is controlled through menus under hardware setup. Access <HardSet>/<Setup Serial Port> to open these menus. Typically, choose "stop serial port" before you close the menu.

In addition, to the motherboard, 4 additional high-speed serial ports are available on a k2001 board. 3 of these ports are isolated and support RS422, and the last one supports RS232.

One of these 422 ports is typically used to stream data to and from a Kupernode card, one is used to stream to a DMX dimmer and one is used to stream to a Preston FiZ.

The RS232 port is typically used as a dedicated version of the Serial Port command described above, to stream real-time machine position data to graphics machines doing real-time background rendering for virtual background composite shots.

The actual data sent to any of these ports is soft-mapped and controlled by pragma files. K2Ser, for example, controls the virtual background data streaming function.

More data on these functions can be found in the Kuper update bulletins.