Using the Interlisp-D RS232 Facility

File: <lispusers>RS232.tty
Created: Jan 10, 1983
Revised: Feb 21, 1983, and August 18, 1983, by JonL White

Basic RS232 Facililty Support Functions

Through a special circuit board plugged into the parallel port of a Dolphin, serial RS232 communications may be interfaced using the following basic functions. As characters arrive in at the interface port, they are stored in a ring buffer (and the output functions will buffer their data in a ring buffer until it is full, or until explicit request is made to force it out). Since there is no microcode support for buffering the characters as they come in, there are some limitations using this facility -- primarily that the user has to call one of the functions which will update the input ring buffer at intervals frequent enough to insure getting all the characters.

RS232INIT: before using the RS232 facility, it is necessary to install certain parameters in the INS8250 chip on the abovementioned board; the four arguments to this function correspond to the desired Baud rate (150, 300, 600, . . . 9600 are supported), the number of bits per serial character (i.e., 7 or 8), whether or not to use the 8th bit as a parity bit (and if so, whether parity is to be odd or even), and the number of "stop" bits (except in unusual cases, 1 the default value, is used here).
CAUTION: the value of RS232INIT as a global variable is used by these driver functions; do not reset it at any time.
A global variable, RS232XON\XOFF?, if non-NIL, causes the driver functions to look for ^S on the incoming side, and to "gag" the output transmitter until a subsequent ^Q has been received; it will probably cause undue trouble to set this flag to true if the corresponding host doesn't also obey the XON-XOFF protocols. This value is, of course, temporarily turned off by the FTP protocols, which transmit and receive random bytes.

RS232CLEARBUFFER: one argument, typically one of (IN OUTPUT BOTH) the corresponding ring buffer is cleared (and the data lost); also, return will be delayed until any character currently being sent out by the INS8250 chip has been fully transmitted (this way, not only is the ring buffer empty, but so is the one-character buffer in the INS8250).

RS232FORCEOUTPUT: no arguments. Ensures that all data in the output ring buffer is actually transmitted "on the lines". One use of this function is to ensure that all data are out -- it won't return until this is true [but also see RS23BACKGROUND below].

RS232BACKGROUND: one argument, "state"
The "state" argument must be among {OFF, INPUT, OUTPUT, BOTH, ON}; except for input of OFF, this turns on a background low-level process which will service the UART at least once every 16 milliseconds and/or flush out the output buffer. A "state" of OFF shuts this background activity off. A "state" of INPUT causes only the input buffer to be serviced; OUTPUT for only the output buffer, and either ON or BOTH for both buffers. A period

of 16 milliseconds for the input service time should give the appearance i
of asynchronous buffering, without dropping any characters, when used at
speeds of less than about 600 baud.

RS232PEEKBYTE: no arguments. Returns the next character sitting in the
input ring buffer, if any; the hardware port is checked to see if any
input characters are waiting, and if so they are put into the ring buffer
first. Calling this function is a good way to insure that characters
are moved, in a timely fashion, from the chip to the input ring buffer.

RS232READBYTE: two optional arguments. This is the basic input function,
which will return a fixp of up to 8-bits in length. If no character is
available, it will return NIL; but if the first argument is a fixp, then
it will wait up to that many time units before returning (possibly getting
an incoming character in the meantime); if the first argument is any
other non-NIL value, the it will wait (possibly forever) until some
byte comes in to be returned. The second argument determines the
length of a "time unit"; default is milliseconds, but alternatives are
SECONDS and TICKS (which is the internal Dolphin clock unit -- see the
documentation of DURATION). As with RS232PEEKBYTE, any call to this
function will update the input ring buffer as its first action.

RS232READWORD: arguments as in RS232READBYTE. If two bytes can
be read in the alloted time, they are composed into a "word"; the
first byte comprises the high-order 8 bits of a 16-bit word, and the
second byte comprises the low-order 8 bits.

RS232READLINE: three optional arguments. A sequence of characters
is read, until an End-of-Line character is received; all the characters
except the EOL are returned as a string. The first two optional arugments
are interpreted exactly as the two optional arguments to RS232READBYTE;
that is, if the expected EOL is not seen "in time", then NIL is returned.
However, if the third argument is supplied, it must be a string pointer,
and it will be re-used to return the characters accumulated so far,
even if there is a timeout; note that the "characters accumulated so
far" are merely sitting in a local "RS232READLINE" buffer, so
succesive calls will reuse that buffer. One other caveat: up to 8
character times are dallied after receiving the EOL, to see if it
is followed by a line-feed, and if so, the line-feed is flushed.

RS232READSTRING: six arguments, most optional.
(#chars.limit? stopcode? noblocksflg wait? timerUnits oldstrbuffer)
This function will take input bytes from the RS232 port until one of three
conditions obtains. (1) the total number of characters taken in by this
call is equal to "#chars.limit?" [NIL means no limit]; (2) a character is
read with character code equal to the argument "stopcode?" [NIL means no
limiting charcter]; or (3) an interval of time greater than that specified
by "wait?" has passed with no bytes available at the port. If "wait?" is
non-null, it must be an integer, and "timerUnits" specifies the units
(see section 14.6 of new manual "Timers and Duratin Functions). i
If "noblocksflg" is non-null, then RS232READSTRING will consume all
the CPU cycles without offering to yield to other processes [including the

MOUSE process]; this mode is important to very-time-critical applications.
If "oldstrbuffer" is supplied, it must be a string and the result characters are smashed into it [so that no consing is done].

RS232WRITEBYTE: one argument required, one optional. An 8-bit byte is sent out; actually, if the second arg is NIL, it will just be stored in the output ring buffer, and will be forced out if the buffer starts to get full. Additionally, the ringbuffer will be forced out if the second argument is non-NIL (or whenever there is an explicit call to the function RS232FORCEOUTPUT, or from time to time when RS232BACKGROUND has specified background output from the buffer -- see documentation above.).

RS232WRITECHARS: one argument required, one optional.   First argument is either a litatom or string, and all the characters therein are "written"; second argument is interpreted the same as with RS232WRITEBYTE.

RS232SENDBREAK: one optional argument. The out-of-band BREAK signal is transmitted for a period of 0.25 seconds; if the optional argument is non-NIL, then the period is extended to 3.5 seconds.

RS232MODEMCONTROL: one argument, "signalslst". A NoSpread function which sets the modem control lines to be "on", for the signals in the list "signalslst". Returns the former setting of the lines. If "signalslst" is not supplied [which is not the same as supplying NIL], then the control lines are merely returned. The entries in "signalslst" are litatom names for standard modem control lines.   Current signal names usable are DTR and RTS.

RS232MODIFYMODEMCONTROL: two arguments "signalsonlst" and "signalsofflst" Changes only those modem control lines specified in the union of the two arguments; those in "signalsonlst" are set to be on, and those in "signalsofflst" are set off. Returns the former state just as (RS232MODEMCONTROL) does.

RS232MODEMSTATUSP: one argument, "booleanform" Returns non-null iff the reading of the modem status lines is consistent with the form "booleanform" [modem status signals currently supported are CTS, DSR, RI, and RLSD]. "booleanform" may be any AND/OR/NOT combination over the signal names. Example: (RS232MODEMSTATUSP ’(AND CTS (NOT RLSD))).

RS232MODEMHANGUP: no arguments
Takes whatever steps appropriate to cause the modem to "hang up" [mostly, this means turning the DTR signal down for about 3 seconds, or until the DSR signal has gone down].

The {RS232} device is created by RS232INIT; one can obtain a stream interface to the RS232 port by calling (GETSTREAM ’{RS232} <direction>). However, in most cases, this stream approach will not work unless the asynchronous buffering mentioned above is successful -- the time taken by general I/O operations is unpredictable and often quite large.

The global variable RS232XON\XOFF? controls whether or not these driver functions will participate in an XON/XOFF protocol; when non-NIL, the any incoming XOFF character (the ^S of ascii) will cause the output functions to "hang" until a releasing XON character has come in (the ^Q of ascii). Also the global variable RS232XOFF? will reflect whether or not the the port is currently in the "hanging" state.

The hardware will detect the usual error conditions (dropped characters, parity errors when so initialized, and framing errors) in addition to detecting a BREAK being sent. When a BREAK has been detected, the software will set the global variable RS232BREAKSEEN? to non-NIL; it will also check the value of RS232BREAKFN, and if non-NIL, will apply it to NIL. Similarly, if there is any error condition which causes a character to be dropped, the software will apply the value of RS232LOSTCHARFN to a litatom describing the reason for the lossage; the default value for RS232LOSTCHARFN is \RS232DING, which will "flash" the display screen a couple of times, and put the value of \RS232.DROPPEDCHARACTER.CODE into the ring buffer [initially this is set to (CHARCODE #^G)]. RS232LOSTCHARFN must, at all times, be a runnable function.

RS232CHAT Facility

The function RS232CHAT, with four optional arguments, initiates a full-duplex transmission throught the RS232 port. The first argument is coerced into a stream for printing the received characters (default is to use the window in the value of \RS232CHATWINDOW, which if null, will interactively ask the user to lay out a region for such window); second argument, if non-null, is a user-programmable interface for filtering the characters which arrive from the remote correspondent -- it must be the output of the function MAKEBINHOOK [as of August 18, 1983, this facility isn’t quite ready -- it’s primary application will be to provide a flexible means for emulation of the various semi-smart terminals like the Heath-19 etc.]; third argument, if not null, specifies that local echoing of the typed-in characters is to be done, with T meaning to use the same stream as the first argument, and any other value being coerced into a stream to use for local echoing; fourth argument is whether or not to use the XON\XOFF protocol.

While in "chat" mode, character interrupts are shut off, the keyboard is rather plainly interpreted, and characters typed in on it are sent to the correspondent. Ordinarily, a host will send a CR/LF for "newline", but some send only one; the menu selection lets you pick one or the other if this is the case (especially useful with UNIX systems). Similarly, you can specify that the RETURN key (or, EOL key) send either just CR or both CR/LF. If local echoing is being performed, and it the local echo stream is the same as the main RS232CHAT window, then the locally-generated characters will be enclosed in square-brackets, as a means of distinguishing local echo from remote output.

Caveat: Output to the the Dolphin display takes a non-trivial amount of time (e.g., just going through the character printout routines, and "painting" a character onto the screen bit-map requires over a millisecond; scrolling a modest-sized window may take well over 30 milliseconds). Without additional microcode support, to maintain the input ring buffer asynchronously, it is questionable whether rates above 2400 baud will be acceptable for RS232CHAT, and there may be ocasional problems above 1200 baud). At "slower" speeds, that is, at less than about 600 baud, the use of RS232BACKGROUND may alleviate these problems. However, RS232CHAT will pay attention to the DSPSCROLL setting of the chat window, and will do "roll" mode rather than "wrap" mode provided that it can do so without dropping characters ["roll" describes the "scroll up" action when typeout reaches the bottom of the window]. If the XON/XOFF protocol is being used, or if the background process mentioned above is in operation, then likely there will be no problem is using "roll" mode.

Escape from "duplex" mode is made by typing the the character which is found in the value of \RS232ESCAPE.CHARCODE, currently initialized to (CHARACTER #B) [this happens to be middle-blank]. Typing "?" just after the escape character will give a small "help message; the commands to be used in this mode are all one-letter:

```
   B - send a BREAK  (0.25 seconds)
   E - change the escape character
   F - deactivate the XON\XOFF protocol
   H - call the function (HELP), with interrupts re-activated
   L - call the function (RS232.PROMPT&LOGIN)
   O - set the XON\XOFF protocol active
   Q - for quit and exit, presumably back to LISPX
   S - set the speed of the RS232 port; ? will display choices.
   <CR> - 'Return' key sends <CR> to remote host
   <LF> - 'Return' key sends <CR><LF> to remote host
   ^B - run a "break" or HELP loop
   R  - call RAID
   7 - truncate incoming characters to 7 bits (this is necessary when you
        have opened an 8-bit connection ignoring the parity bit; you really
        only want to see the lower 7 bits interpreted as a character to be
        printed on the window).
   8 -  undo the "7-bit" mode above (just in case you actually wanted to
        see the eight bit -- typically the printout will be just the same
as
        as the 7-bit printout, but preceeded by a "#" when the eighth bit
        is on in a character.)
```

Additional control may be exercised with the pop-up menu obtained by pressing the middle mouse button with the cursor in the RS232CHAT window while RS232CHAT is active; its commands are essentially self-documenting, and are a super-set of the above-mentioned commands available from the keyboard. In particular, it's possible to alter what RS232CHAT thinks is the "NewLine" character; Interlisp-D's default is to choose CR, but for connections to some systems, LF is a much better choice.

Two other commands permit "toggling" (that is, switching the state from one choice to an alternate, and vice-versa): ~LocalEcho and ~RollMode. The latter will change the DSPSCROLL of \RS232CHATWINDOW; the former will "toggle" the use of the local echo stream provided in the call to RS232CHAT (or will use \RS232CHATWINDOW if no stream was provided).

The RS232CHAT window tries to play the "TTY-process passing" protocol described
in the recent documentation for multiple-process and TTY interactions.

A number of variables control certain characteristics; in addition to \RS232ESCAPE.CHARCODE mentioned above, there are:

\RS232PERMITTED.INTERRUPTS -- a list of items such as returned by INTERRUPTCHAR (or such as would be input to RESET.INTERRUPT), which will be "active" during RS232CHAT; initially this list is null.
\RS232CHAT.IgnoreCharcodes -- a list of character codes that will be ignored by the input side of RS232CHAT; initially this list contains only the single code (CHARCODE NULL).
\RS232CHAT.EOLsequence -- A string of characters to be sent out whenever the RETURN key is typed on the keyboard; initially this just contains the one character CR, and is changeable by a menu command.
\RS232CHAT.NEWLINECHAR -- Normally set to LF, which will cause RS232CHAT to work right for systems which send both CR/LF for newline as well as for those that send only LF (e.g., UNIX). However, some hosts send only CR, and thus to get RS232CHAT to advance to a new line, CR must be recognized as the "newline" character. This option is changeable by a menu command. Note that this section isn't talking about the EOL character.
\RS232CHAT.BellSequence -- Since the standard Interlisp-D action for printing a "bell" to a display stream takes too long (much longer than the inter-character time at 1200 baud), then unless the XON\XOFF protocol is active, this string of characters will be substituted for the (CHARCODE BELL). Initially, this is "^<bell>".

The following two functions are most useful when trying to "chat" to a host through an RS232 connection running at a speed higher than Interlisp-D can support for display stream activities:

RS232LOGIN: 6 arguments, most optional. First is the name of a host machine with which the RS232 port is corresponding, second is the desired username/login.id on that machine, third is the password needed there, and fourth is the "host system type"; the remaining two arguments are concerned where to echo the activity caused by this function, and are mainly of interest to other system-level functions. If either "username" or "password" is NIL, the will be obtained via PROMPTFORWORD from the keyboard (this is so that you don't have to have passwords in code files); there is also an internal cache of the information about host/username/password, just as is kept for logins over the Ethernet. [see documentation of PROMPTFORWORD]
When the Host's system type is known, then a database of login protocols is consulted to figure out how to send (automatically and blindly) the necessary characters to effect a login. At convenient moments, the output from the host, which is accumulated in the RS232 line buffer, will be output for the user's perusal (the fifth argument is a stream for this printout: NIL defaults to the primary output, NONE gags this type-out; window, files ets. all are acceptable here). A primary reason for this function's existence, besides the cacheing of such information as login.id and password, is to permit loggin in at speeds which cannot support RS232CHAT (see documentation below).

RS232.PROMPT&LOGIN: one argument. Prompts the user (via PROMPTFORWORD) to type in the necessary information, in the PROMPTWINDOW, to call and use RS232LOGIN; the argument is handed to RS232LOGIN as its fifth argument (the "Type-out stream" argument).

RS232 "FTP" Facility

Two functions exist for interfacing to a new protocol for doing file transfers over an RS232 connection; the primary version of this new protocol
was developed in the micro/home computer world, where there was a need to transfrer files between a "home" computer and some major, RS232 accessible host. Since the RS232 connection was most often made through a telephone modem, this protocol has come to be known as MODEM; unfortunately, since CP/M
was the predominant operating system on these "home" computers, the protocol
does not provide a totally secure way of knowing how long a file really is; furthermore, the packet size is fixed at 128 bytes, and some systems have input buffers for which this is frequently too large. [But I have some variants
on this protocol which solve these problems, and I intend to certify their implementation in Interlisp and suggest them to the other MODEM users]. Nevertheless, the protocol does have packetizing, checksumming, and timeouts;
so there is only a very small probability that a file so transmitted will have
undetected errors.

RS232GETFILE: three required arguments, and one optional. First argument is the name of a file to store the file being transmitted *from* the correspondent; second arg is either TEXT or BINARY (ASCII permissible in place of TEXT), indicating the file type (in the Interlisp-D sense); third argument is the protocol being used (currently, only MODEM is acceptable here); fourth argument, when given is just transmitted first (typically, this would be the series of characters you would type at the
remote executive to cause the desired file to be run).
RS232PUTFILE: arguments the same as for RS232GETFILE, except that the direction of transmission is from the existing file on the Dolphin *to* the correspondent.

Examples: (assuming connection to a TOPS-20 host)

```
   (RS232GETFILE  '{DSK}MUMBLE  'TEXT  'MODEM
          "MODEM SA <LISPUSERS>MUMBLE
")

   (RS232PUTFILE  '{DSK}RUN.DCOM  'BINARY  'MODEM
          "MODEM RB <JONL>RUN.DCOM
")
```

Both ends of the MODEM protocol have "synchronizing" features, so a typical scenario of usage would be to use RS232CHAT to login to the host, and then simply put the MODEM program in its wait state, by typing whatever arguments it needs, and finally exiting from RS232CHAT and calling RS232GETFILE (or RS232PUTFILE) directly, without the fourth argument. The fourth-argument facility is provided so that one may use RS232FTP at speeds greater than would be available for RS232CHAT; login could thus be achieved through use of the function RS232.PROMPT&LOGIN.

The global variable RS232FTPTRACEFLG, if non-null, causes a trace of activity to be printed out on the file/stream specified by the global variable RS232TRACEFILE; it the value is PEEK, then only a "+" will be printed for successful transit of packets, and "-" for unsuccessful ones;
any other non-null value causes a more verbose output.

Several implementations of the MODEM protocol for other machines are
available:
one for the IBM/PC is available on floppy disk through XSIS (and is also on
[MAXC]<XEOS>IBMFTP.ASC). Several files are available also on [MAXC]<XEOS>
for
VAX/VMS users: XMODEM.FOR and QIO.DCK are an implementation in FORTRAN;
TOXMOD.FOR and FMXMOD.FOR are helpful for dealing with the structure of
files
in VMS's record management system.