Using the Interlisp-D RS232 Facility

File:	<lisp>Library>RS232.tedit
Created:	Jan 10, 1983, by JonL White
Revised:	2/21/83, 8/18/83, 3/12/84, and 10/31/84

RS232 communications permit use of synchronous as well as asynchronous protocols, through a small piece of hardware for serial/parallel conversion called a USART or a UART; the former is necessary when synchronous protocols are used. Currently, Lisp's only interface is to the asynchronous protocol, which basically means that an 8-bit "character" may be sent, at any time, through the USART/UART, and similarly an 8-bit "character" may be received at any time.

There are three ports available under Interlisp-D for RS232 communications, two of which are on the 1108, and one on the 1100. For the 1108, one RS232 port is standard (dubbed the TTYPort) and another is optional (the E30 option, dubbed the RS232C port); for the 1100, there is an optional circuit board plugged into the parallel port. The RS232C port is a USART, terminating in a standard 25-pin connector at the back of the 1108, and an extension cable is provided which terminates in a 25-pin male connector with the standard RS232 signals; the TTYPort is a UART, which terminates in a 25-pin female connector at the back of the 1108 and for which a special adaptor cable must be built (see below); the circuit board for the 1100 terminates in a 25-pin male connector on a short cable. There are some known deficiencies with respect to the RS232 protocols in the TTYPort; see the footnote on RS232MODEMHANGUP below. Port selection on the 1108 is controlled by the global variable RS232DLionTTYP, which may be reset at initialization time; currently, the default value is T, meaning "use the TTYPort", but likely with the next release, it will be NIL meaning "use the RS232C port".

As characters arrive in at the interface port, they are stored in Lisp's memory 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. The output ring buffer is one 512-byte page, and the input ring buffer is two pages. [[Footnote: Two global variables control the initial size settings for these buffers, so it is possible to tailor the size to a specific application.]] However, when using the RS232C port, the input buffer will be set to 4, l8, or 16 pages depending on the line speed, so as to provide at least several seconds worth of buffering at full speed. [[Footnote: The 1100 has no microcode support for buffering the characters as they come in, and similarly the 1108 has no low-level support for the TTYPort; thus there are some limitations using these two ports -- 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.]]

The "host" name {RS232} is usable as a device-oriented interface to the RS232 ports. After the device has been initialized, (GETSTREAM '{RS232}) will return this bi-directional stream (which is also cached in the global variable \RS232SRTREAM). However, the stream interface is not recommended except when using the RS232C port [[Footnote: In most cases, the general stream-oriented functions will not work for the 1100 or the 1108's TTYPort because the time taken by general I/O operations is unpredictable and often quite large. However, at very slow speeds, then the RS232BACKGROUND process is turned on for input, there may be some usability here.]]

Initialization

Before using the RS232 facility, it is necessary to install certain parameters in UART or USART behind the channel, and the function RS232INIT is provided for that purpose.

```
  (RS232INIT BaudRate BitsPerSerialChar Parity NoOfStopBits ModemControl Port)
```

The arguments correspond to the desired baud rate (150, 300, 600, . . . 9600 are supported on all channels, and 19200 is supported on the RS232C channel), 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, then this argument must be either ODD or EVEN), the number of "stop" bits (except in unusual cases, 1 the default value, is used here), and the setting of any modem control lines (see description of RS232MODEMCONTROL below). The Port argument currently only selects between the TTYPort and the (optional) RS232C port on the 1108; if a non-NIL value is supplied, it must be either the litatom TTYPort or RS232C; if NIL, then the global variable RS232DLionTTYP is consulted, with non-NIL meaning to use the TTYPort and NIL meaning to use the RS232C port. The default values for all arguments are

BaudRate	1200
BitsPerSerialChar	8
Parity	NIL
NoOfStopBits	1
ModemControl	NIL
Port	NIL {i.e., consults RS232DLionTTYP}

CAUTION: the value of RS232INIT as a global variable is used by these driver functions to contain a copy of this argument list; do not reset it at any time.

One may use OPENSTREAM as an alternative to calling RS232INIT directly, but then one must then bundle up all the parameters into the PARAMETERS argument, where the formal parameter name is paired with the value desired. For example, (RS232INIT 9600 8) can also be achived by

```
       (OPENSTREAM '{RS232} NIL NIL NIL
                       '(((BaudRate 9600) (BitsPerSerialChar 8)))
```

Regardless of whether or not OPENSTREAM is called, a stream interface is also created by the initialization, and left as the value of the global variable \RS232STREAM.

The function RS232SHUTDOWN exists in order to gracefully turn off the RS232 port; for some ports, the "gracefullness" is more important than for others. (CLOSEF '{RS232}) will merely call this function; it admits one argument, the RS232 stream, but NIL will default correctly.

Ring Buffer Management Functions

(RS232CLEARBUFFER I/O) causes the corresponding ring buffer to be cleared (and the data lost); also, return will be delayed until any character currently being sent out by the UART (or USART) have been fully transmitted (this way, not only is the ring buffer empty, but so is the one-character hardware buffer in the UART). The argument allows specifying only the input buffer, only the output buffer, or both buffers: I, IN, or INPUT for input; O, OUT, OUTPUT for output; and BOTH or I/O for both.

(RS232INPUTSTRING STRING) puts the characters of the argument, which may be a litatom or string, into the input ring buffer, just as if they had been received through the UART.

(RS232FORCEOUTPUT WAITFORFINISH) 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 "on the wires": when the argument is non-NIL, the function will not return until this condition has been met; otherwise, the funtion merely initiates a process (note: not a PROCESS in the Interlisp-D sense) to clean out the output ring buffer. This is the specific function called when one does (FORCEOUTPUT <rs232stream> WAITFORFINISH).

(RS232BACKGROUND ON? PERIOD.ms) will initiate a background low-level process (yes, a PROCESS) which will service the UART at least once every PERIOD.ms milliseconds (default value: 16ms), and/or force out the output buffer as often as possible. The first argument must be among {OFF, INPUT, OUTPUT, BOTH, ON}; OFF terminates this background activity; OUTPUT causes only the output buffer to be forced out; INPUT causes only the periodic service to the input buffer; and either ON or BOTH causes both activities. The input service is really only applicable to the ports other than RS232C, which has an independent mechanism to insure that characters are transported in a timely fashion from the UART to the ringbuffer. For the 1100, and for the 1108's TTYPort, a period of 16 milliseconds for the input service time should give the appearance of asynchronous buffering, without dropping any characters, when used at speeds of less than about 600 baud. A non-NIL value for PERIOD.ms will be coerced to be at least 16ms, and not more than 1000ms.

Input Functions

(RS232PEEKBYTE) 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 moved into the ring buffer first. For the ports other than RS232C, calling this function is a good way to insure that characters are moved, in a timely fashion, from the USART/UART to the input ring buffer. As with any "peek" function, the returned byte is not removed from the stream.

(RS232READBYTE WAIT? timerUnits) is the basic input function, which will return either NIL or a fixp between 0 and 255 inclusive. Both arguments are optional. If no character is available, RS232READBYTE will return NIL; but if WAIT? is a fixp, then it will wait up to that many time units before returning (possibly getting an incoming character in the meantime); if WAIT? 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, MILLISECONDS, and TICKS (which is the internal machine clock unit, and varies in value between the 1100 and 1108 -- see section 14.6 of the Interlisp Reference Manual "Timers and Duration Functions"). As with RS232PEEKBYTE, any call to this function will update the input ring buffer before doing anything else.

(RS232READWORD WAIT? timerUnits) 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 WAIT? timerUnits OLDSTRBUFFER) reads a sequence of characters until an End-of-Line character is received; all the characters except the EOL are returned as a string. The first arugments are interpreted exactly for 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, and its storage will be re-used to return the characters accumulated so far, even if there is a timeout. 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 #CHARS.LIMIT? STOPCODE? NOBLOCKSFLG WAIT? timerUnits OLDSTRBUFFER) is the basic "block mode" input function; the arguments are all optional, and the last three function just as in RS232READLINE. Characters will be accumulated until one of three conditions occur:

1. The total number of characters taken in by this call is equal to #CHARS.LIMIT? (NIL means no limit); or
2. A character is received with character code equal to 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 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, but most particularly when using the ports other than RS232C.

Output Functions

(RS232WRITEBYTE BYTE FORCEOUT? IGNOREXOFF?) will send out one 8-bit byte; if FORCEOUT? is NIL, the byte will simply be stored in the output ring buffer, and will be not actually be "sent" until some condition arises to force out the buffer. The output 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). Especially with the RS232C port, it incurs a high overhead cost to do a force output on every character sent.

(RS232WRITECHARS STRING FORCEOUTPUT?) sends out all the characters in the first argument, which may be either a litatom or string. Second argument is interpreted the same as with RS232WRITEBYTE.

Modem Interface Functions

(RS232XON\XOFF? ON?) controls the processing of the in-band XON and XOFF characters sent by the correspondent; when ON? is non-NIL, such processing will be enabled; when ON? is NIL, it will be disabled. When the RS232 port is initialized, this feature is disabled. [[Footnote: Up until the fall of 1984, this action was determined by the value of the global variable RS232XON\XOFF?; in order to ensure a smooth transition, there wil be a period of time during which the functional interface will merely set this global variable.   This cross-over period will extend at least until the next release of Interlisp-d.   But ultimately, it will have to carry out actions that are not consistent with a Lisp variable setting.]] This means that the low-level driver functions will look for ^S on the incoming side, and will "gag" the output transmitter until a subsequent ^Q has been received. When in the "gag"’d state, the value of the global variable RS232XOFF? will be T; when not "gag"’d, it will be NIL. It should hardly be necessary to mention that it will cause undue trouble to turn this feature on when the corresponding host doesn’t also obey the XON-XOFF protocols. [[Footnote: Since the RS232C port contains additional buffering "outboard" of the 1108, Lisp will probably not see an incoming XOFF character in the time required by some correspondents; this problem will be fixed in a future release of software, but it is the "outboard" I/O processor that is at fault here rather than Lisp]] This protocol is, of course, temporarily turned off by the RS232FTP functions, which transmit and receive random bytes.

(RS232SENDBREAK EXTRALONG?) sends the out-of-band BREAK signal, for a period of 0.25 seconds; if the optional argument EXTRALONG? is non-NIL, then the period is extended to 3.5 seconds.

(RS232MODEMCONTROL SIGNALSONLST) is a lambda-nospread function which sets the modem control lines to be "on", for the signals named in the list SIGNALSONLST; it returns the former setting of the lines. If SIGNALSONLST is not supplied (which is not the same as supplying NIL), then the control lines remain unchanged. The entries in SIGNALSONLST are litatom names for standard modem control lines; currently usable signal names are DTR and RTS.

(RS232MODIFYMODEMCONTROL SIGNALSONLST 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 SPEC) Returns non-null iff the reading of the modem status lines is consistent with the boolean form specified by SPEC; modem status signals currently supported are CTS, DSR, RI, and RLSD. SPEC may be any AND/OR/NOT combination over the signal names. Example:

```
        (RS232MODEMSTATUSP ’(AND CTS (NOT RLSD))).
```

(RS232MODEMHANGUP) takes whatever steps appropriate to cause the modem to "hang up"; generally, this means turning the DTR signal down for about 3 seconds, or until the DSR signal has gone down [[Footnote: There is a bug in the low-level code for the 1108’s "outboard" I/O processor such that the TTYPort can’t actually turn off the DTR signal; a consequence of this is that RS232MODEMHANGUP will not work on the TTYPort.]]

Error Condition Reporting

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. Additionally, certain low-level errors may occur in either the hardware or software which can only be classified as device errors. There are three global variables providing a functional interface to the handling of these errors: RS232BREAKFN, RS232LOSTCHARFN, and RS232DEVICEERRORFN.

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 low-level drivers will apply the value of RS232LOSTCHARFN, if non-NIL, to a litatom describing the reason for the lossage. Currently the "reasons" are: DroppedCharacter, ParityError, FramingError, RingBufferFull, LineBufferFull, and MultipleErrors (which is a catch-all for any combination of the foregoing "reasons"). 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)].

Finally, when some inscrutable "device error" occurs, the value of RS232DEVICEERRORFN, if non-NIL, will be applied to a litatom reason. Currently the "reasons" are: TransmitterWedged and RS232CDisaster. The former is typical of a TTYPort use without the proper adaptor cable; the latter seems to indicate a bug in the low-level I/O processor software. The default value for RS232DEVICEERRORFN is \RS232.DEVICEERROR, which will "flash" the display screen a couple of times, and print a message in the PROMPTWINDOW.

RS232CHAT Facility

The function RS232CHAT, with four optional arguments, initiates a full-duplex transmission throught the RS232 port.

```
(RS232CHAT TypeScriptStream BINHOOK LocalEchoStream XON\XOFF?)
```

The first argument, TypeScriptStream, 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); if BINHOOK is non-NIL, iat is a user-programmable interface for filtering the characters which arrive from the remote correspondent; NB: it must be the output of the function MAKEBINHOOK [[Footnote: this facility isn't fully ready yet -- it's primary application will be to provide a flexible means for emulation of the various semi-smart terminals like the Heath-19 etc.]; if LocalEchoStream is non-NIL, it 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; XON\XOFF? controls 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; a menu selection from the chat window 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 applicable to the non-RS232C ports only: Since output to the the 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, and scrolling a modest-sized window may take well over 30 milliseconds) it is questionable whether baud rates above 2400 will be acceptable for RS232CHAT, and there may even 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).

A number of internal global 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 when a non-RS232C port is in
      use (unless the XON\XOFF protocol is enabled), this string of characters
      will be substituted for the (CHARCODE BELL).  Initially, it is just
      "^<bell>".
```

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

```
  (RS232LOGIN HOST UserName PASSWORD HostSYSTEMTYPE TypeScriptStream
                      PROMPTFORWORDstream URGENCY.OPTION)
```

Of the 7 arguments, most are optional. HOST is a name for the machine with which the RS232 port is corresponding; UserName is the desired username/login.id on that machine; PASSWORD is the password needed there; HostSYSTEMTYPE is the host's operating system type (currently, only TOPS-20, TENEX, VMS, UNIX, WAITS, ITS, and TOPS10 are know system types; TENEX is the default); the remaining three arguments are concerned where to echo the activity caused by this function, and are mainly of interest to other system-level functions. The global variable RS232OSTYPES is parallel to NETWORKOSTYPES, but the names are not in any way coordinated with a name database, such as occurs with ethernet protocols; in fact, NETWORKOSTYPES will also be consulted by RS232LOGIN, so it would not be wise to use a name for an RS232 correspondent that is in conflict with NETWORKOSTYPES.

If either UserName or PASSWORD is NIL, they 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. 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 above); also, it is convenient to be able to select "Login" from the RS232CHAT menu and have the login sequence sent automatically.

(RS232.PROMPT&LOGIN TypeScriptStream) 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 protocol for doing file transfers over an RS232 connection; the primary version of this protocol was developed in the micro/home computer world, where there was a need to transfer 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. 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 FILE FILETYPE PROTOCOL REMOTE.COMMAND.STR) initiates a file transfer from the remote host, and stores it in the file named by FILE (which can also be an already-open stream); FILETYPE is either TEXT or BINARY (ASCII is permissible in place of TEXT), indicating the file type (in the Interlisp-D sense); PROTOCOL is the protocol being used (currently, only MODEM is acceptable here, but someday KERMIT may be usable); REMOTE.COMMAND.STR, when non-NIL, is just transmitted first (typically, this would be the series of characters you would type at the remote executive to cause the desired "modem" program to be run on the remote host).

(RS232PUTFILE FILE FILETYPE PROTOCOL REMOTE.COMMAND.STR XON\XOFF?) initiates a file transfer to the remote host, from the file FILE (which can also be an already-open stream); FILETYPE, PROTOCOL, and REMOTE.COMMAND.STR are the same as with RS232GETFILE; XON\XOFF?, when non-NIL, causes the output functions to watch for XOFF commands from the correspondent (not every host computer generates them, especially when running a "modem" protocol program).

Examples: (assuming connection to a TOPS20 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. One reason for the fourth-argument capability is so that one may use RS232FTP at speeds greater than would be available for RS232CHAT -- login could be achieved through use of the function RS232.PROMPT&LOGIN, and file transfer through the use of RS232GETFILE or RS232PUTFILE, and no need arises at all for a terminal protocol.

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 Xerox Artificial Intelligence Systems; others are available for VAX/VMS and VAX/UNIX users.