RS232

The 1108 and 1186 each support two RS232 ports. One of these ports is configured as Data Terminal Equipment, and is intended to be connected to modems or terminal ports on other computers. (On the 1108, this port is available only with the addition of the E30 option.) The other port is configured as Data Communication Equipment, and is meant to drive printers or terminals. In this document, the DTE port is called the RS232 port, and the DCE port is called the TTY port.

Lisp provides a stream-oriented interface to the RS232 hardware. Users' programs can open streams to the hosts {RS232} or {TTY}, and perform input or output using standard Lisp I/O functions, such as READ-BYTE, READ-CHAR, etc.

Programs may use RS232 streams or TTY streams with the same programmatic interface. However, the RS232 port is preferred over the TTY if the application expects to handle large amounts of input data. In the 1108 and 1186, data entering the RS232 port is buffered independently of Lisp by the I/O processor (IOP). In addition, the RS232 software provides an additional layer of character buffering, freeing user programs from having to monitor the RS232 hardware frequently.

No independent buffering is provided for data entering the TTY port. As a result, Lisp cannot guarantee to catch all characters received on this port. For this reason, the TTY port should be used primarily to drive output devices such as printers.

Requirements

Hardware

To connect to a modem or another device, you need an RS232 or TTY cable (see the wiring diagrams in the Introduction of this manual).

For the 1108 only, you need the E-30 upgrade kit.

For the RS232 port to operate correctly, the remote device must assert the standard RS232 signals DSR and CTS. This requirement means that the appropriate pins on the 1186 be driven high, either properly by connecting the 1186 side to the corresponding device side, or permanently by jumpering the pins.

For the TTY port to operate correctly, the remote device must assert DTR and RTS. This requirement means that the appropriate pins on the 1186 be driven high, either properly by connecting the 1186 side to the corresponding device side, or permanently by jumpering the pins.

Software

You need the following .LCOM files in order to run this module successfully:

DLRS232C or DLTTY (one of these is required)
RS232MENU
RS232CHAT or TTYCHAT, and KERMIT (these are optional).

(See also the file dependencies enumerated in the Introduction of this manual.)

Installation

Load the required .LCOM modules from the library.

Run RS232C.INIT or TTY.INIT to set parameters.

Using the RS232 Port

Support for the RS232 port is contained in the file DLRS232C.LCOM. Before using the RS232 port, it is necessary to initialize the RS232 hardware. The function RS232C.INIT is provided for this purpose:

(RS232C.INIT *BAUDRATE BITSPERSERIALCHAR PARITY NOOFSTOPBITS FLOWCONTROl*) [Function]

The arguments correspond to the port parameters (see below).

Alternatively, the *BAUDRATE* argument can be an instance of the RS232C.INIT record. If *BAUDRATE* is NIL, the value of the global variable RS232C.DEFAULT.INIT.INFO is used in its place. This provides a means of automatically initializing the RS232 hardware without user intervention.

RS232C.DEFAULT.INIT.INFO [Variable]

This variable controls default initialization of the RS232 port. Its value may be set in the site INIT.LISP file, or in your INIT.LISP file. If RS232C.DEFAULT.INIT.INFO is not set when the RS232 module is loaded, its fields will be set to the following default values:

BaudRate: 1200

BitsPerSerialChar: 8

Parity: NONE

NoOfStopBits: 1

FlowControl: XOnXOff

Programs may use the Lisp function OPENSTREAM as an alternative to calling RS232C.INIT directly, with the parameters bundled up into the PARAMETERS argument.

For example, (RS232C.INIT 9600 8) can also be achieved by:

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

(RS232C.SET.PARAMETERS *PARAMETERLIST*) [Function]

This function allows applications to change the settings of the RS232 hardware while the RS232 port is in use.

PARAMETERLIST is an association list of parameter names and values. The following example sets the baud rate to 9,600 baud, and the character length to eight bits:

```
(RS232C.SET.PARAMETERS '((BaudRate . 9600) (BitsPerSerialChar
. 8)))
```

The following is a list of legal parameter names and values:

BaudRate — 1186: `50, 75, 110, 150, 300, 600, 1200, 2400, 3600, 4800, 7200, 9600, 19200.`

1108: `all of the above, and 28800, 38400, 48000, 56000, 57600.`

BitsPerSerialChar — `5, 6, 7, 8` bits of data. If `5` or `6` bits of data are sent, they should be `DATA`, not `CHARACTER`.

Parity — `NONE, ODD, EVEN` (1 parity bit).

NoOfStopBits — `1, 1.5, 2` (i.e., the stop bit may have a period equal to 1, 1.5 or 2 bit-widths).

FlowControl — `NIL, NONE, XOnXOff, list.`

`NIL` and `NONE`: no flow control.

`XOnXOff`: flow control using Xon and Xoff characters. For applications requiring XOn and XOff characters other than ^Q and ^S respectively, this parameter may be supplied as a list in the form: `(1 <XOn> <XOff>)`, where `<XOn>` and `<XOff>` represent the character codes (ASCII 1 - 127) of the characters which are to be treated as the XOn and XOff characters. The leading "1" signifies that flow control should be enabled; a leading "0" will program the RS232 port with the appropriate XOn and XOff characters, but leave flow control disabled.

ModemControl — This parameter should be a list of modem control signals to be enabled, such as DSR, CTS, DTR, DTR, RI and HS.

The functions `RS232MODEMCONTROL`, `RS232MODEMSTATUSP`, and `RS232MODIFYMODEMCONTROL` provide finer control over the settings of modem signals (see below).

DTR — This parameter enables or disables the data terminal ready signal; it may be specified as `T` or `NIL`.

RTS — This parameter enables or disables the request to send signal; it may be specified as `T` or `NIL`.

`(RS232C.GET.PARAMETERS` *PARAMETERLIST*`)` [Function]

The current settings for the RS232 port may be obtained at any time by calling this function.

PARAMETERLIST should be a list of parameter names. `RS232C.GET.PARAMETERS` returns an association list of parameter names and values, in a format acceptable to `RS232C.SET.PARAMETERS`.

`(RS232C.SHUTDOWN)` [Function]

The RS232 port is turned off by calling this function. It disables the RS232 port and closes any open streams on the devices.

Using RS232 Streams

Programs may open streams to the RS232 port by calling OPENSTREAM with the file name {RS232}. The *ACCESS* argument to OPENSTREAM controls whether an INPUT or OUTPUT stream is returned. RS232 streams are unidirectional; to obtain a second stream open for the opposite access, call the function RS232C.OTHER.STREAM.

Only one pair of RS232 streams may be open at a time; an error will result if you attempt to open more.

(RS232C.OTHER.STREAM *STREAM*) [Function]

STREAM should be an RS232 stream. If *STREAM* is open for INPUT, an RS232 stream open for OUTPUT is returned; conversely, if *STREAM* is open for OUTPUT, an RS232 stream open for INPUT is returned.

The following Lisp functions are defined to work on RS232 streams open for the appropriate access: BIN, BOUT, READP, OPENP, CLOSEF, and FORCEOUTPUT.

(RS232C.CLOSE-STREAM *DIRECTION*) [Function]

This function closes one or both RS232 streams, so you don't need to have access to the streams to close them.

DIRECTION can be one of INPUT, OUTPUT, BOTH, or NIL. The function closes the RS232 stream open in *DIRECTION* mode; if *DIRECTION* is BOTH or NIL the input and output streams will be closed, if they exist.

RS232 streams are buffered. Input and output are performed in units of packets of data. The I/O processor collects incoming data into a packet, and makes that packet available to Lisp when one of the following conditions is true:

The packet is filled.
The frame time-out has expired.

The frame time-out is the number of hundredths of a second that are allowed to occur between the reception of single characters. This parameter is automatically set by the RS232 module. Its value depends on the baud rate of the RS232 port. If the value is set too large, interactive applications such as Chat will suffer from uneven typeout; if the value is too small, a larger number of shorter packets may be exchanged between Lisp and the I/O processor, resulting in increased processing overhead.

Lisp buffers data for output in packets of up to 578 characters. Output packets are sent to the RS232 port when one of the following conditions is true:

The current output packet is full.

The user program calls the FORCEOUTPUT function to force the current output packet to be sent.

The output stream is closed.

Applications that generate a large amount of output slowly may wish to reduce the size of outgoing packets. Although this will require additional processing overhead, it will cause output to occur more frequently without the program explicitly calling FORCEOUTPUT.

(RS232C.OUTPUT.PACKET.LENGTH *NEWVALUE*) [Function]

This function returns the current setting of the variable that controls the maximum size of output packets. If *NEWVALUE* is supplied, the setting is changed to *NEWVALUE*. *NEWVALUE* may be a number between 1 and 578.

Specifying a value is a matter of how long you are willing to wait at input time before you are assured of seeing incoming data. You can do a straightforward division to set the length (e.g., at 9600 baud, you get about 1 char/millisecond, so the maximum delay is 578ms; if you can tolerate only 1/4 second, then set it to 250 or so).

(RS232C.READP.EVENT *STREAM*) [Function]

Many RS232 applications are time-dependent. File transfer protocols such as Kermit and Modem depend on one or both sides of a file transfer detecting connection problems by means of time-outs. This function allows user programs to detect time-out conditions efficiently.

STREAM should be an RS232 stream open for input. This function returns an event that a program may wait upon for input data to become available on the stream. The following example illustrates how a program could wait up to 10 seconds for a character to become available:

```
[LAMBDA (STREAM)
      (LET ((EVENT (RS232C.READP.EVENT STREAM))
      (TIMER (SETUPTIMER 10000)))
      (until (OR (READP STREAM) (TIMEREXPIRED? TIMER))
      do (AWAIT.EVENT EVENT TIMER T)
      finally (RETURN (COND ((READP STREAM) (BIN STREAM]
```

Using Modems

The following functions are useful for controlling modems:

(RS232SENDBREAK *EXTRALONG?*) [Function]

This function sends the out-of-band BREAK signal, for a period of 0.25 seconds; if *EXTRALONG?* is non-NIL, then the period is extended to 3.5 seconds.

(RS232MODEMCONTROL *SIGNALSONLST*) [Function]

This function is a Lambda-NoSpread function that 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 symbol names for standard modem control lines; currently usable signal names are DTR and RTS.

(RS232MODIFYMODEMCONTROL *SIGNALSONLST SIGNALSOFFLST*) [Function]

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*) [Function]

Returns non-null if the reading of the modem status lines is consistent with the boolean form specified by *SPEC*; modem status signals currently supported are

DSR, RI, and RLSD. SPEC may be any AND/OR/NOT combination over these signal names.

Example:

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

(RS232MODEMHANGUP) [Function]

This function takes whatever steps are appropriate to cause the modem to hang up. Generally, this means turning the DTR signal down for about three seconds, or until the DSR signal has gone down.

Error Condition Reporting

The RS232 port detects parity errors, character framing errors, lost characters, and a number of other unusual conditions. As the I/O processor delivers each input packet to Lisp, it reports when the packet was received without error. If an error did occur while the packet was being received, Lisp will report this fact by writing a message to RS232C.ERROR.STREAM.

RS232C.ERROR.STREAM [Variable]

RS232 error conditions are reported on this stream. This stream is initially the PROMPTWINDOW.

(RS232C.REPORT.STATUS *NEWVALUE*) [Function]

There are circumstances in which the RS232 hardware believes it has encountered an error, when in fact it has not. A frequent cause is an incorrect parity setting in the RS232 port. Continually reporting RS232 errors is likely to slow RS232 processing severely. In cases where error reporting is not important, it is possible to disable error reports with the RS232C.REPORT.STATUS function. This function returns the current setting of status reporting, which may be one of the following:

T	Errors are reported on both input and output.
NIL	Errors are never reported.
OUTPUT	Errors are reported on output only.
INPUT	Errors are reported on input only.

In addition, if *NEWVALUE* is supplied, the current setting of status reporting is changed to *NEWVALUE*.

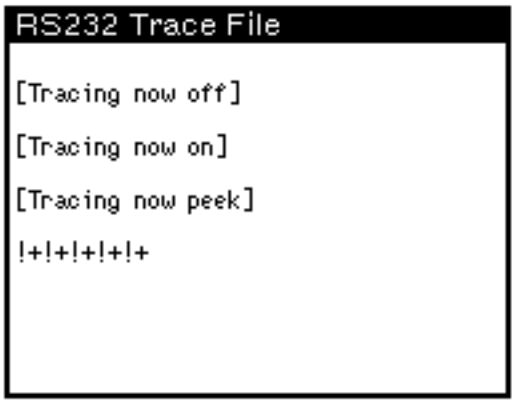
RS232TRACE

To help in debugging RS232 applications, it is possible to trace the data that is being sent out via the port. RS232 packets are traced using:

(RS232C.TRACE *MODE*) [Function]

MODE is one of PEEK, T, or NIL. If *MODE* is either T or PEEK, RS232C.TRACE opens a trace window in the mode selected. T indicates a full trace, with every byte being shown. PEEK is a less verbose trace, with every incoming packet shown as a "+" and every outgoing packet shown as a "!". NIL turns off the

tracing. Clicking the left mouse button in the trace window will cycle between the modes.



RS232CHAT

The RS232CHAT module is a facility that permits the Chat library module to communicate over the RS232 port. Once it is loaded, you may chat to the host named RS232 to open a connection to the RS232 port.

RS232CMENU

RS232CMENU is a utility that provides a menu interface to controlling the settings of the RS232 port. When loaded along with RS232CHAT, this utility may be invoked by choosing the "Set Line Parameters" entry in the options menu that appears if you select “Options” in the middle-button Chat menu.

The following commands are available in the RS232 menu:

Apply	This menu item changes the RS232 port settings according to the values of the fields in the rest of the menu. No changes are made to the RS232 hardware until this command is given.
Abort	Closes the RS232 menu and aborts any changes that could have been made.
SendBreak	Sends a normal (0.25 second) break signal. Requires confirmation, as this command could cause the modem connection to be broken.
SendLongBreak	Sends a long (3.5 second) break signal. As above, this requires confirmation.
Hangup	Tries to hang up the modem (by dropping DTR). Requires confirmation.

The following fields are multiple choice items; when their labels are selected with the mouse, a menu of possible values for the field appears. Choosing a value from the menu will change the field; clicking off the menu will leave the field unchanged.

Baud Rate	Changes the BaudRate of the RS232 port.
Parity	Changes the Parity setting of the RS232 port.
Character Length	Changes the BitsPerSerialChar setting of the RS232 port.
Flow Control	Changes the FlowControl setting of the RS232 port.
Stop Bits	Changes the NoOfStopBits setting of the RS232 port.
Report Errors	Changes the RS232C.REPORT.STATUS setting of the RS232 port.

File Transfer Using RS232

Files may be transferred using RS232CHAT and either the Kermit or Modem protocols.

Using the TTY Port

Support for the TTY port is contained on the file DLTTY.LCOM. The TTY port is designed to support low-speed communications with RS232 devices. The I/O processor offers no low-level support for input buffering. As a result, it is quite possible for newly received input characters to overwrite previously received but unread characters in the input hardware. The TTY port provides exactly one character's worth of buffering; each character must be read by Lisp before the next character is completely received.

Note: No hardware flow control is provided on the TTY port. The Lisp TTY port service routines will obey received flow control commands, but will not generate flow control commands in response to increased input data rate.

(TTY.INIT *BAUDRATE BITSPERSERIALCHAR PARITY NOOFSTOPBITS FLOWCONTROL*) [Function]

This function is very similar to the function RS232C.INIT.

Before using the TTY port, the TTY hardware must be programmed with the proper characteristics for your application.

Alternatively, the *BAUDRATE* argument can be an instance of the RS232C.INIT record. If *BAUDRATE* is NIL, the value of the global variable TTY.DEFAULT.INIT.INFO is used in its place. This provides a means of automatically initializing the TTY port hardware without user intervention.

TTY.DEFAULT.INIT.INFO [Variable]

This variable controls default initialization of the TTY port. Its value may be set in the site INIT.LISP file, or in your INIT.LISP file. If TTY.DEFAULT.INIT.INFO is not set when the TTY package is loaded, its fields will be set to the following default values:

BaudRate: 1200
BitsPerSerialChar: 8
Parity: NONE
NoOfStopBits: 1
FlowControl: XOnXOff

Programs may use OPENSTREAM as an alternative to calling TTY.INIT directly, with the parameters bundled up into the PARAMETERS argument as shown below.

For example:

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

(TTY.SET.PARAMETERS *PARAMETERLIST*) [Function]

Applications may change the settings of the TTY hardware while the TTY port is in use.

PARAMETERLIST is an association list of parameter names and values. For example, let's set the baud rate to 9600 baud and the character length to eight bits:

```
(TTY.SET.PARAMETERS '((BaudRate . 9600) (BitsPerSerialChar .
8)))
```

The following is a list of legal parameter names and values:

BaudRate	50, 75, 110, 150, 300, 600, 1200, 2400, 3600, 4800, 7200, 9600, 19200.
BitsPerSerialChar	5, 6, 7, 8. If 5 or 6 bits of data are sent, they should be DATA, not CHARACTERS.
Parity	NONE, ODD, EVEN (1 parity bit).
NoOfStopBits	1, 2 (This parameter should be 1 except at 110 baud).
FlowControl	NIL, XOnXOff. For applications requiring XOn and XOff characters other than ^Q and ^S respectively, this parameter may be supplied as a list in the form: (1 <XOn> <XOff >), where <XOn> and <XOff> are replaced by the character values of the XOn and XOff characters. The leading 1 signifies that flow control should be enabled; a leading 0 will program the TTY port with the appropriate XOn and XOff characters, but leave flow control disabled. Note: XOnXOff flow control is known to be reliable only up to 4800 baud.
DSR	This parameter enables or disables the data set ready signal; it may be specified as T or NIL.
CTS	This parameter enables or disables the clear to send signal; it may be specified as T or NIL.

(TTY.GET.PARAMETERS *PARAMETERLIST*) [Function]

The current settings for the TTY port may be obtained at any time by calling this function.

PARAMETERLIST should be a list of parameter names. TTY.GET.PARAMETERS returns an association list of parameter names and values, in a format acceptable to TTY.SET.PARAMETERS.

(TTY.SHUTDOWN) [Function]

This function turns off (disables) the TTY port and closes any open streams on the device.

Using TTY Streams

Programs may open streams to the TTY port by calling OPENSTREAM with the file name {TTY}. The ACCESS argument to OPENSTREAM may be INPUT, OUTPUT, or BOTH.

Unlike RS232 streams, TTY port streams are not buffered, and a single stream may be used for both input and output. The generic Lisp input/output functions BIN, BOUT, READP, OPENP, and CLOSEF may be used on TTY port streams.

TTYCHAT

TTYCHAT is a module that enables the Chat library module to communicate over the TTY port. No user-callable functions or user-settable variables are available in the TTYCHAT module. Once it is loaded, you may chat to the host named TTY to open a connection to the TTY port. TTYCHAT is contained on the file TTYCHAT.LCOM.

Because the TTY port does no low-level input buffering, it is quite likely that many input characters will be lost while chatting at 1200 baud or higher. This package should only be used for non-critical applications, such as testing connections between the TTY port and low-speed printers.

RS232CMENU

RS232CMENU is compatible with the TTY port as well. Certain RS232 port commands, such as SendBreak! are not available with the TTY port, and hence do not appear in the menu.

Examples

Testing the Connection Between Two Xerox Lisp Machines

To test for a working RS232 connection between Machine A (a Xerox 1108) and Machine B (a Xerox 1100, 1108, or 1186) by moving a file between them, proceed as follows:

1. Load RS232CHAT.LCOM on both machines.
2. Call RS232C.INIT to set up parameters.
3. Do RS232CHAT on both machines. You will be prompted for a window.

Whatever you type on machine A should be echoed on machine B and vice versa.

Testing the Connection Between Xerox Lisp Machines and a VAX Running VMS

VAX side: Set baud rate at which files will be transferred on the VAX side using the VMS command SET. For example, to set to 1200 baud, type:

```
SET TERM TTA1:/SPEED=1200/PERM
```

1108 side: Load `RS232CHAT.LCOM`

Initialize: `(RS232C.INIT 1200 8 NIL 1 NIL'RS232C)`

Then call `(RS232CHAT)`

You should be able to use the Chat window like a VAX teletype terminal.

By matching the baud rate on the VAX (through `SET TERM`) with that on the 1108 (through `RS232C.INIT`), you can use any speed up to 9600 baud.

[This page intentionally left blank]