# AMSTRAD

# RS232C

## SERIAL INTERFACE

## USER INSTRUCTION BOOK

# AMSTRAD RS232C SERIAL INTERFACE WITH ROM SOFTWARE AND POWER SUPPLY

## AMSOFT

A division of

AMSTRAD

**CONSUMER ELECTRONICS PLC.**

AMSOFT welcome comments and suggestions relating to the product or to this manual.

All correspondence should be addressed to:

AMSOFT
Brentwood House
169 Kings Road
Brentwood
Essex CM14 4EF

All maintenance and service on the product must be carried out by AMSOFT authorised dealers. Neither AMSOFT nor AMSTRAD can accept any liability whatsoever for any loss or damage caused by service or maintenance by unauthorised personnel. This guide is intended only to assist the reader in the use of the product, and therefore AMSOFT and AMSTRAD shall not be liable for any loss or damage whatsoever arising from the use of any information or particulars in, or any error or omission in, this manual or any incorrect use of the product.


First Published 1985

Written by Roland Perry
Illustrated by Alexander Martin

Electronics and SIO drivers by MEJ Electronics

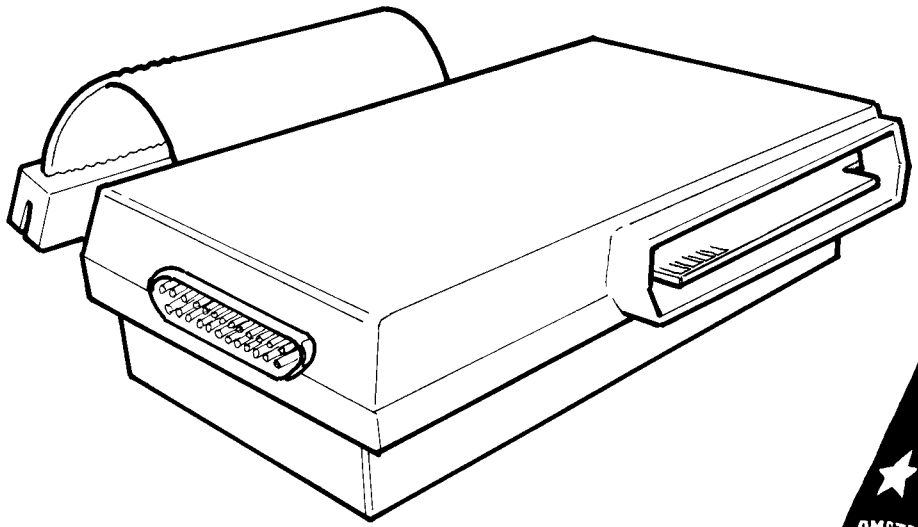
Programming by AMSOFT:

Printer re-direction, PRESTEL, and TERMINAL by David Radisic
File transfer by Vik Olliver

Concepts by Roland Perry
Consolidation and ROM housekeeping by Cliff Lawson
Edited by Ivor Spital

Published by AMSTRAD
Typeset by KAMSET typesetting graphics (Brentwood)

# Welcome to the AMSTRAD RS232C 'Book of Spells'!



When you have fitted the AMSTRAD RS232C to your computer, you have the means to communicate with printers, modems, and other computers. Implementing such a link is often regarded as a very mysterious and complicated business, if only because of the serial interface's inherent flexibility and versatility. To help you use your AMSTRAD RS232C serial interface, we have provided instructions in the form of 'magic spells'. Simply select those spells which are relevant to your application by identifying the accompanying sketches.

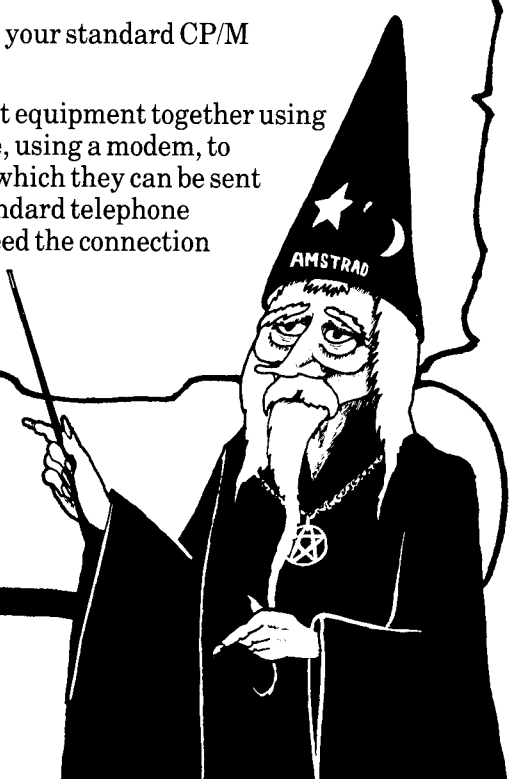
A full technical explanation is available in a series of appendices. The examples in these 'spells' are not intended to be an exhaustive survey of all the possible uses of the RS232C. They do represent, however, the majority of applications encountered.

The AMSTRAD RS232C serial interface, and these instructions, are suitable for use with the AMSTRAD CPC464, CPC664 and CPC6128 computers. The AMSTRAD RS232C works equally well with any of these computers. Some of the applications referred to work best with a disc drive, although cassette tape storage is usable as an alternative. Some disc drive applications will require CP/M 2.2 (supplied with both the CPC664, and with the DDI-1 add-on disc system for the CPC464), or CP/M Plus (supplied with the CPC6128).

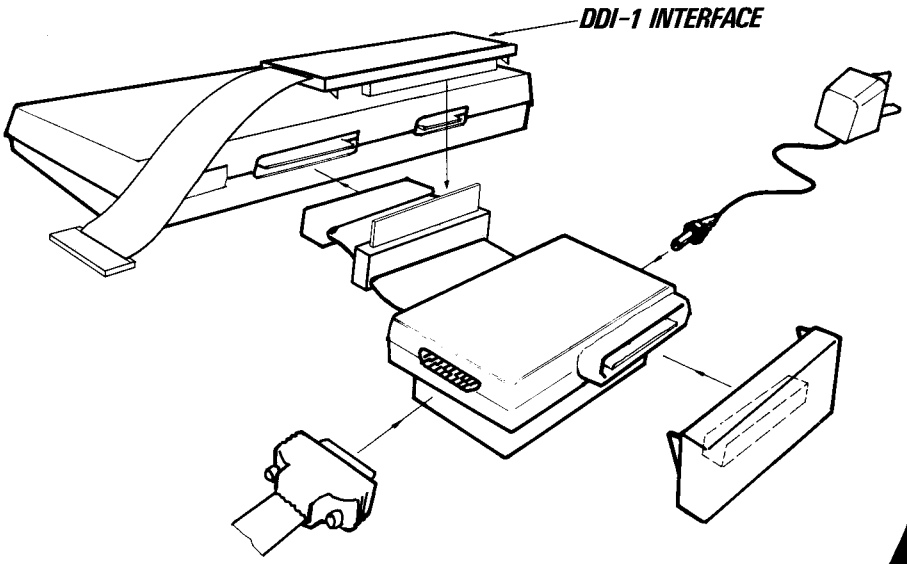
When operating in BASIC, the additional commands provided for the RS232C are programmed in a ROM supplied inside the RS232C and are accessed by keywords starting with a bar | symbol. The bar symbol is obtained by pressing **[SHIFT]@**.

The CP/M commands referred to are supplied on your standard CP/M system disc.

The use of a serial interface allows you to connect equipment together using very simple wires up to 50 feet long. It is possible, using a modem, to convert the signals in these wires into a form in which they can be sent almost any distance to another modem, over standard telephone lines. Most equipment manufacturers have agreed the connection details to a standard called 'RS232C'. It is also possible to connect devices which conform to the alternative 'RS423' standard.



# Spell 1:
# Connecting the RS232C to your computer




Switch off the computer and remove anything connected to the computer's port marked 'EXPANSION' or 'FLOPPY DISC'.

Push the connector plug at the end of the RS232C's ribbon cable onto the edge connector of the computer's port marked 'EXPANSION' or 'FLOPPY DISC'.

If required, refit any additional expansion interfaces to the RS232C 'through-bus' connection.

Connect the lead from the RS232C power supply unit, into the small socket on the RS232C itself.

**WARNING - ELECTRIC SHOCK**
**NEVER ATTEMPT TO OPERATE THE RS232C POWER SUPPLY UNIT WITH ITS CASE REMOVED.**

Switch on any additional expansion interfaces, connect the power supply unit of the RS232C to the mains supply, and switch on your computer, in that order. A 'wake-up' message similar to the following will appear on the screen:

```
Amstrad 64K Microcomputer  (v2)
©1984 Amstrad Consumer Electronics plc
          and Locomotive Software Ltd.

Amstrad RS232C Serial Interface (v1)
©1985 Amstrad Consumer Electronics plc

Ready
```

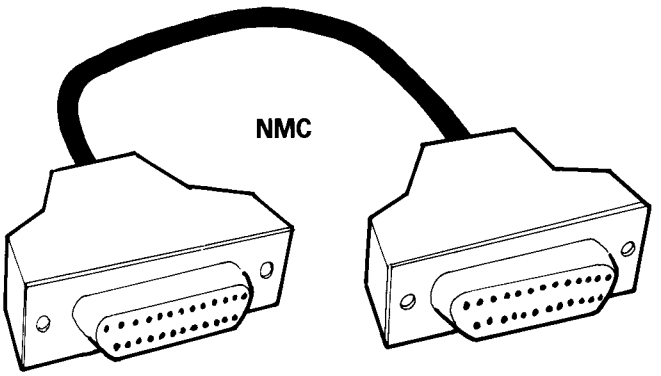
Always switch off in reverse order to the above, before disconnecting the RS232C (or any additional expansion interfaces) from the computer.

(Always disconnect the RS232C power supply unit from the mains supply socket when not in use.)

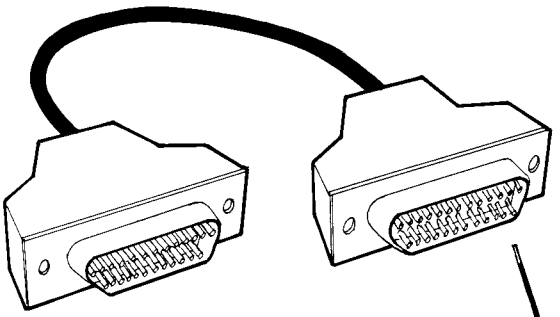
The serial output lead (the 25-way Chassis-plug) may be connected or disconnected at any time, regardless of power supply considerations.

# Spell 2:
# Cables for connecting a printer (or plotter)

The cable required to connect to a printer (or any other output device) is called a Null-modem cable (NMC). A technical explanation and drawing of the connections required is given in Appendix 1.




A Null-modem cable has Cable-sockets at each end to plug into the Chassis-plugs on the RS232C and the printer. If your printer has a Chassis-socket, then you will also require a converter cable comprising two back-to-back Cable-plugs.



# Spell 3:
# Setting the speed of your printer

It is possible to alter the speed at which characters are transmitted from the RS232C to the printer. The speed is measured by a figure known as the 'baud rate', which has a value of approximately ten times the number of characters per second.

You need to make sure that the RS232C is transmitting at the same speed as the printer is receiving. Do not confuse this serial interface speed with the speed at which the printer actually prints characters on the paper. If the printer cannot keep up with the rate at which characters are arriving from the serial interface, then it will send special signals back to the RS232C instructing it to stop sending until the printer catches up. This process is known as 'flow control'.

Your printer will probably have some switches (possibly inside its casing) to set its baud rate. The best speed to choose is 9600 baud (approximately 960 characters per second) because that is the default speed of the RS232C.

# Spell 4:
# Setting the speed of the RS232C

The RS232C operates at 9600 Baud unless instructed otherwise.

AMSTRAD BASIC: Use the command:

`|SETSIO,` ‹baud rate›

CP/M 2.2: The baud rate is set when CP/M is loaded. Refer to Appendix 2 (`SETUP.COM`) for full details on how to alter this power-on configuration.

CP/M Plus: The Baud rate can be changed by the command:

`SETSIO,` ‹baud rate›

....or by the command:

`DEVICE SIO[`‹baud rate›`]`

Examples:

AMSTRAD BASIC: `|SETSIO,1200` ....to change to 1200 baud.

CP/M Plus: `SETSIO 1200` ....or.... `DEVICE SIO[1200]`

If you require an alternative baud rate, it must be set up every time you reset the computer or move from BASIC to CP/M, or from CP/M to BASIC. When operating with CP/M Plus, it is possible to program the alternative baud rate as part of the loading process by incorporating the command given above into the '`PROFILE.SUB`' file.

NOTE - Under CP/M Plus, if the baud rate is changed by `SETSIO`, interrogation by use of the command `DEVICE` will not acknowledge the change.

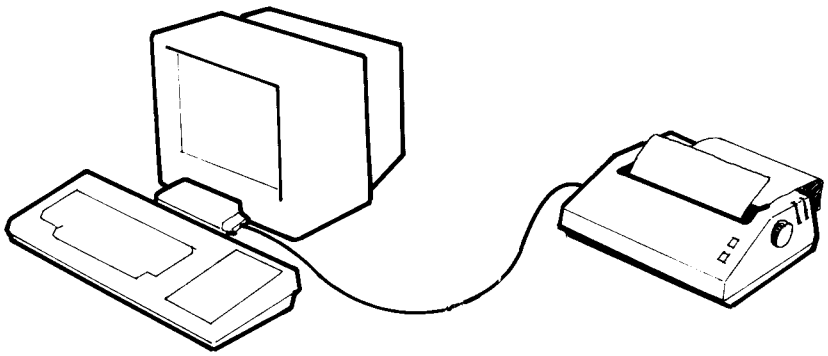
# Spell 5:
# Setting framing bits (printer)

Characters sent by the RS232C have properties other than simply 'their speed'. These properties are: the number of 'data' bits, the number of 'stop' bits, and the type of 'parity'. It is not important to understand the exact nature of these properties, but, as with the baud rate, the printer and RS232C should be arranged to match.

Most printers will have switches for these properties. It is not always essential that the RS232C and the printer match *exactly*; the final test is to try a particular configuration and see if everything works as expected.

The RS232C is set (by default) to 8 data bits, 1 stop bit, and no parity. See Appendix 2 (CP/M) or Appendix 3 (BASIC), for a complete description of the commands required to alter the RS232C framing bits.

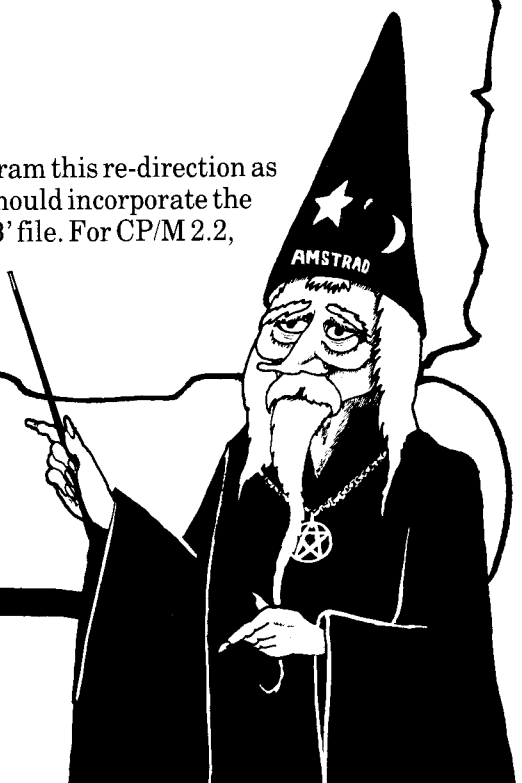
# Spell 6:
# Re-directing printer output via the RS232C



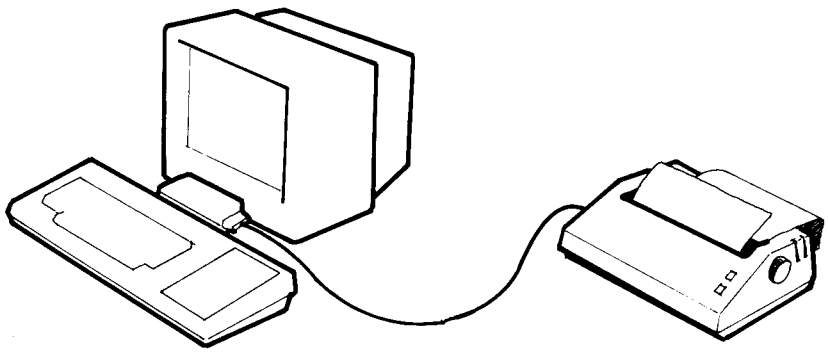
Normally, printer output is sent to the parallel Centronics port at the rear of your computer. Having connected and set up the RS232C, the following commands will cause all printer output to be sent via the serial interface:

| | |
|---|---|
| AMSTRAD BASIC: | `ISERIAL` |
| CP/M 2.2: | `STAT LST:=TTY:` |
| CP/M Plus: | `DEVICE LST:=SIO` |

When operating with CP/M, it is possible to program this re-direction as part of the loading process. For CP/M Plus, you should incorporate the command given above into the '`PROFILE.SUB`' file. For CP/M 2.2, you should use `SETUP.COM` as described in Appendix 2.



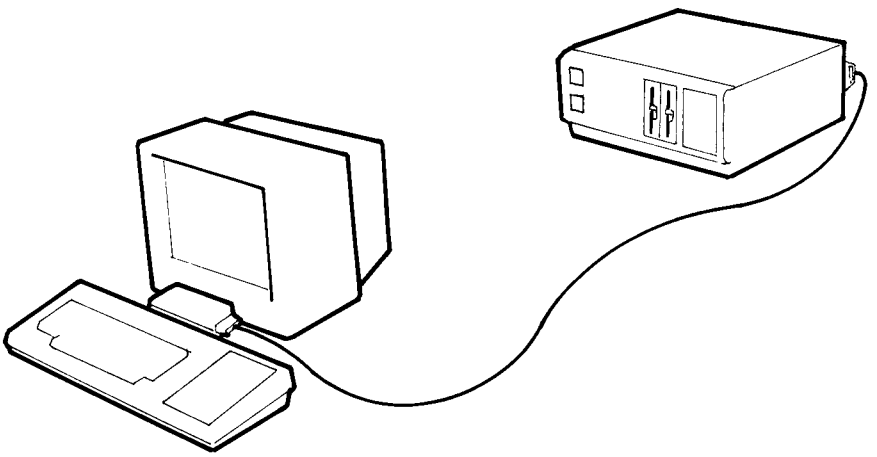
# Spell 7:
# Restoring printer output to the parallel port



Every time you reset the computer or move from BASIC to CP/M, or from CP/M to BASIC, the printer output will be restored to the parallel port. This can also be done using the commands:

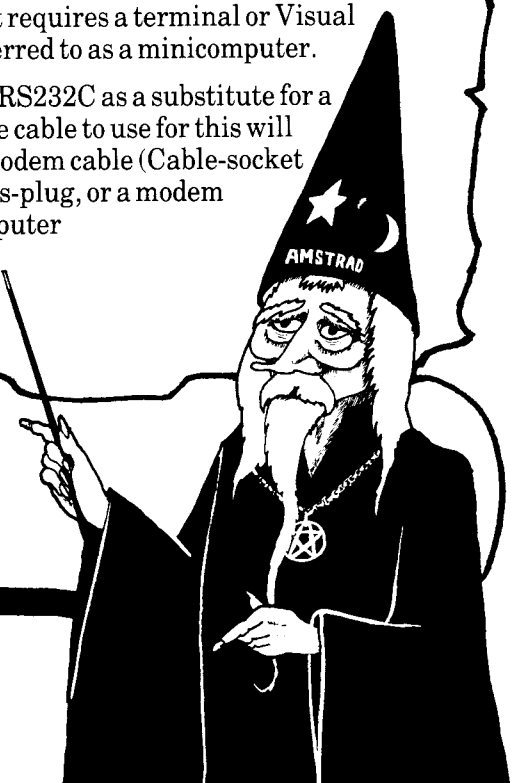
| | |
|---|---|
| AMSTRAD BASIC: | `\|PARALLEL` |
| CP/M 2.2: | `STAT LST:=LPT:` |
| CP/M Plus: | `DEVICE LST:=LPT` |

# Spell 8:
# Cables to connect Terminal Emulator to a minicomputer

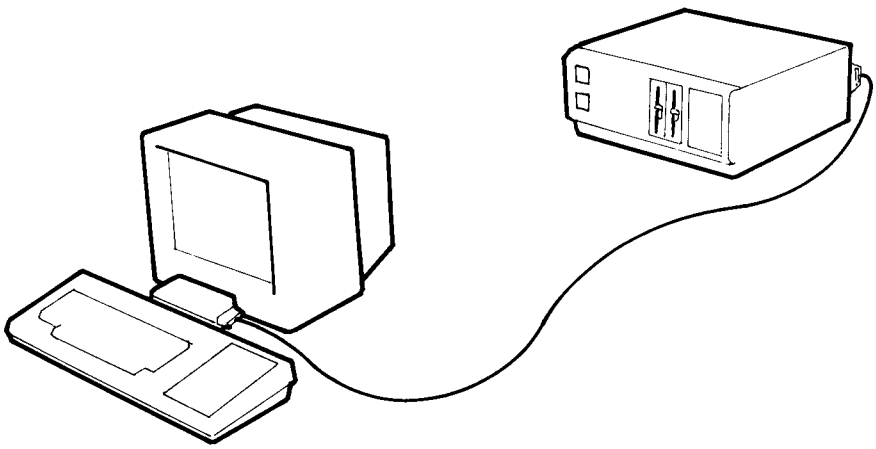


In this manual, a commercial microcomputer that requires a terminal or Visual Display Unit (VDU) in order to operate is also referred to as a minicomputer.

It is possible to connect your computer fitted with RS232C as a substitute for a terminal attached directly to a minicomputer. The cable to use for this will depend on the minicomputer. Use either a Null-modem cable (Cable-socket to Cable-socket) if the minicomputer has a Chassis-plug, or a modem cable (Cable-socket to Cable-plug) if the minicomputer has a Chassis-socket. If in doubt, consult Appendix 1 and the hardware manual of the minicomputer.



# Spell 9:
# Attaching the Terminal Emulator to a minicomputer



The baud rate and framing parameters of the RS232C and minicomputer should agree, as discussed previously when attaching to printers.

The Terminal Emulator is entered by typing:

AMSTRAD BASIC: `|TERMINAL`

....and is exit-ed by **[CTRL] [ESC]**

Note that the Terminal Emulator 'inherits' whatever keyboard values have been previously set up by BASIC, and also responds to the screen control codes described in the AMSTRAD computer's user instruction manual. It will often be convenient to set the **[ESC]** key to return a standard value by entering:

```
KEY DEF 66,0,27
```

# Spell 10:
# Terminal Emulator echo and control codes

The Standard Terminal Emulator sends all key-strokes to the minicomputer, displays all received printable characters, and obeys received control codes.

The minicomputer will normally echo the key-strokes sent to it, so that you can see (on the screen of the terminal) what you have typed, character by character. If the minicomputer does not perform such an echo, then it will be necessary for the Terminal Emulator to do it instead. This is achieved by typing:

```
|HALFDUPLEX
```

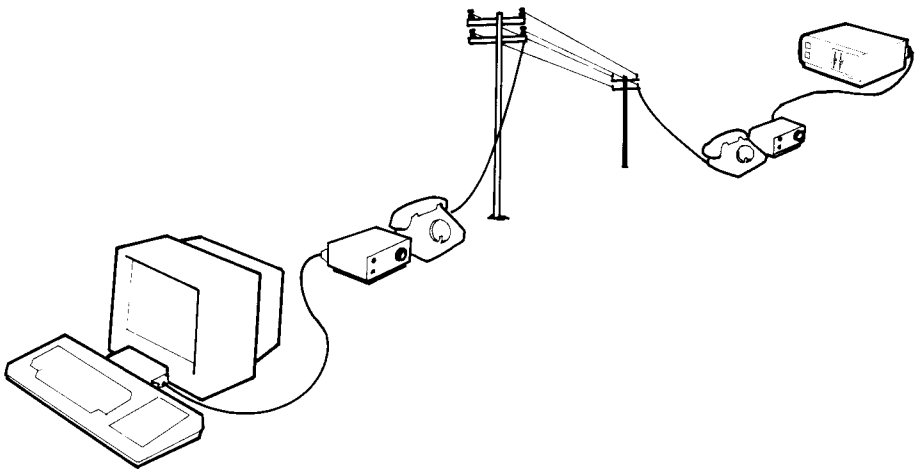
This local echo may be turned off by typing:

```
|FULLDUPLEX
```

For diagnostic purposes, it is sometimes convenient for the terminal to *display* control codes, rather than obey them. The Terminal Emulator can be switched between these two modes by the commands:

```
|CTRLDISPLAY    ....and....    |CTRLACTION
```

# Spell 11:
# Connecting to another computer via a modem



A modem is simply a way of extending the length of the serial connection between two computers - normally via the public telephone network. The connection to the modem is made by a Modem-cable (Cable-socket to Cable-plug) if the modem is fitted with a suitable Chassis-socket. Otherwise, a special cable is required, and Appendix 1 gives explanations and examples of this.

The baud rate and framing parameters of the RS232C should be set to match the modem and distant computer.

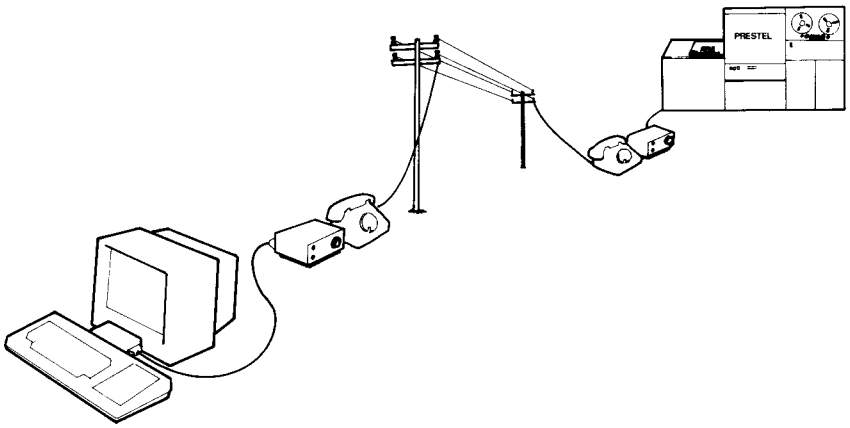
For example, two common speeds for modems are 300 baud (receive and transmit), or 1200 baud (receive)/75 baud (transmit).

The relevant commands for these are:

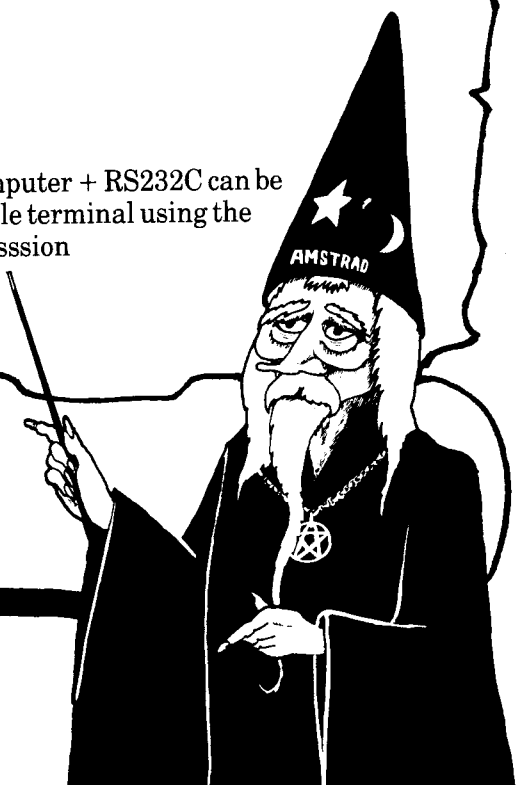
```
|SETSIO,300
```
....which sets both transmit and receive to 300 baud.

```
|SETSIO,75,1200,1,7,1,0
```
....which is suitable for PRESTEL.

# Spell 12:
# Attaching Terminal Emulator/PRESTEL Emulator




Having set the relevant baud rate, your computer + RS232C can be connected to the distant computer as a simple terminal using the command |TERMINAL. Refer to the discusssion of connection to a (local) minicomputer for further details.



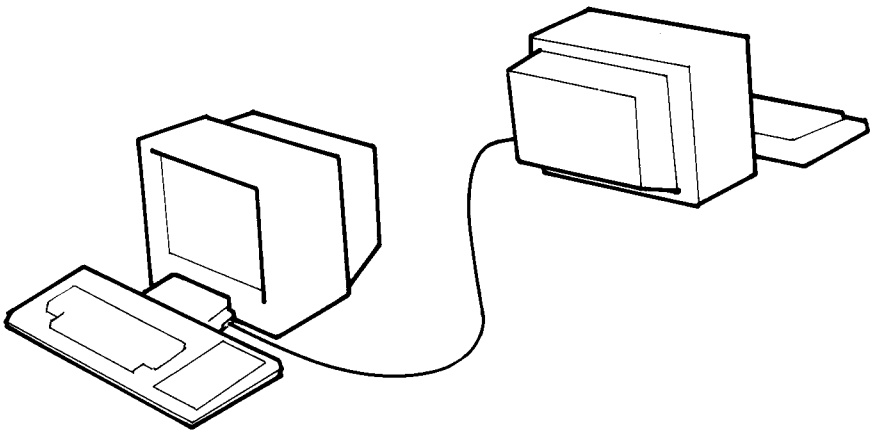
A PRESTEL Emulator is also available, and this is entered by

`|PRESTEL` ....and is exit-ed by **[CTRL] [ESC]**.

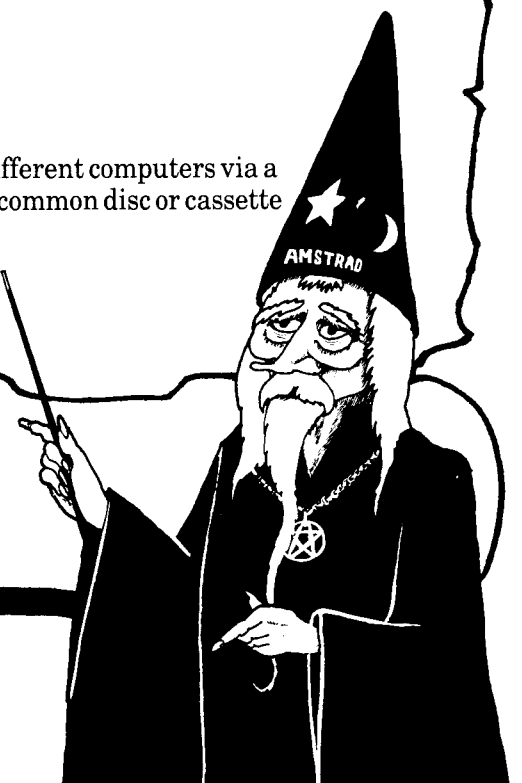
The screen is operated in `MODE Ø` (sixteen colours) with eight continuous colours and eight flashing colours. The number of screen pixels available in this mode restricts the resolution of the characters on the screen, but each of the available characters is nevertheless distinct.

The PRESTEL Emulator simultaneously receives characters from the modem and updates the screen in stages, in a clearly recognisable manner.

# Spell 13:
# Connecting two computers for file transfers



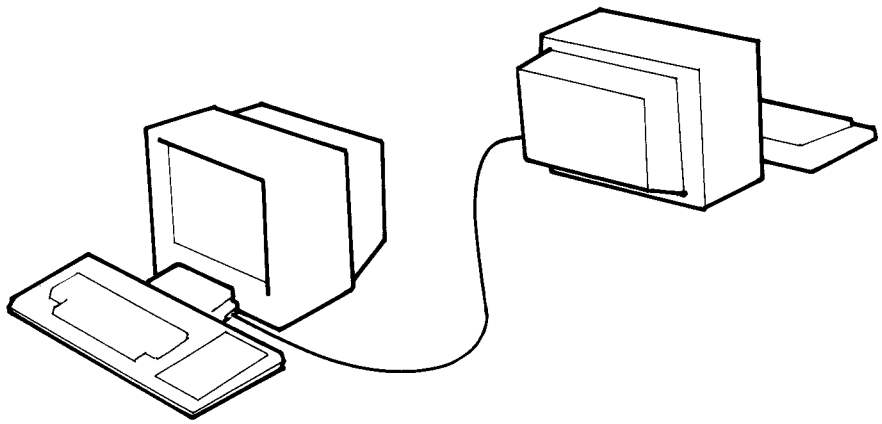
It is often convenient to transfer files between different computers via a serial connection, particularly when there is no common disc or cassette format between the two machines.



It is assumed that you will normally wish to receive files on your AMSTRAD computer, transmitted from some other computer. Should you wish to program an alternative computer to receive files from the AMSTRAD computer, the protocol is published in Appendix 3. A transmitting program will also be required if your transmitting computer is not included in the subsequent examples.

Follow the guidelines for connecting your computer to the transmitting computer or modem (as appropriate). Examples of cable connections are given with some of the subsequent examples. Make sure that the baud rates and framing parameters match. In this instance you must use the default 8 'data' bits, and it is useful to set odd or even parity (rather than none) as this ensures a far greater integrity of data.

# Spell 14:
# Receiving the file on your AMSTRAD computer



Having selected tape or disc as the destination for the file, type:

I SUCK, ‹filename› ....in one of the following ways:

All versions of BASIC (CPC464,CPC664, and CPC6128):

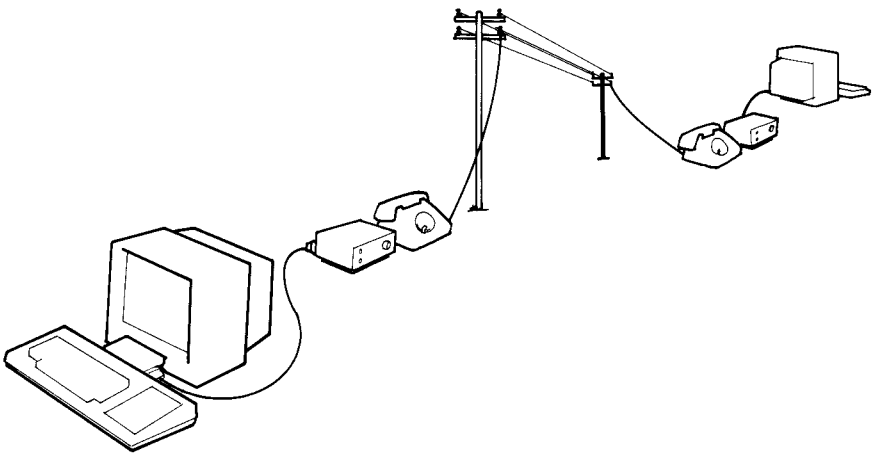
```
a$="FILE.TYP": |SUCK,@a$
```

....or (CPC664 and CPC6128 only):

```
|SUCK,"FILE.TYP"
```

The file will be stored on the disc/tape as an ASCII file. This can then be read by either BASIC or CP/M programs. The section of your (CPC664, CPC6128, or DDI-1) user instruction manual which describes disc to tape transfers, contains much background information on the nature of such an ASCII file.

# Spell 15:
# Transmitting the file from an AMSTRAD computer



This is particularly appropriate when two distant AMSTRAD computers are connected by modem, making a simple physical transfer of cassette or disc impossible. The receiving program checks the incoming data to ensure that there are no errors, and requests re-transmission if necessary.

Having selected tape or disc as the source for the file, type:

```
|BLOW,‹filename›
```
....in one of the following ways:

All versions of BASIC (CPC464,CPC664, and CPC6128):

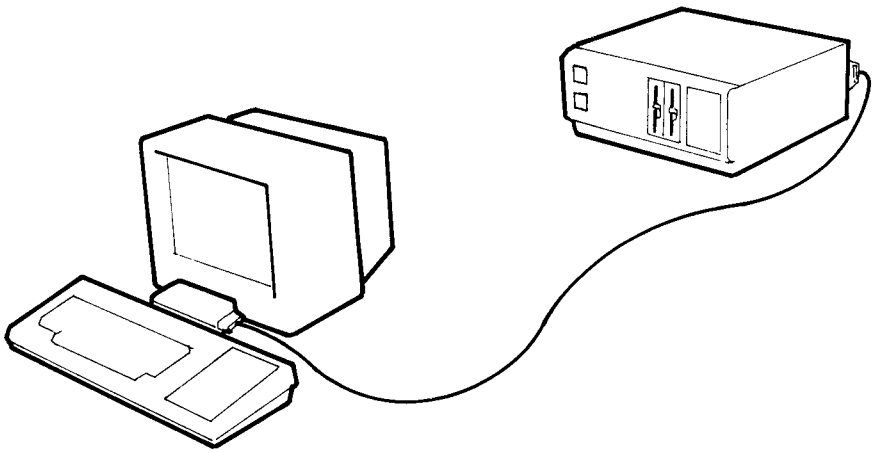
```
a$="FILE.TYP": |BLOW,@a$
```

....or (CPC664 and CPC6128 only):

```
|BLOW,"FILE.TYP"
```

Once again the file must be an ASCII file (remember that all CP/M files appear to be ASCII files to AMSDOS), and there are similarities with the process of tape to disc transfers.

# Spell 16:
# Transmitting the file from a CP/M computer



Enter the following HEX dump into your CP/M computer using a text editor or PIP, e.g:

```
PIP BLOW.HEX=CON:
```

Now enter the following HEX dump (one line at a time), each line terminated by pressing **[ENTER]** followed by **[CTRL]J**. After entering the final line (together with **[ENTER] [CTRL]J**), press **[CTRL]Z** to terminate the operation:

```
:180100003A5D00FE20CA0502115C000E0FCD0500FEFFCA0E02CD26013A
:180118000E10CD0500118E020E09CD0500C721FFFF22A5021E020E0474
:18013000CD0500CD8101D22C012AA5022322A502CD9C01CDB001CDD055
:180148000B7C26A01CDB801CDC001CDC801CD810138DECD9C01CDB0C4
:18016000001C34D01CD9C01CDB0011E000E04CD0500210000022A702CDD2
:18017800C801CD8101D26401C90E03CD0500FE03C29601F1F1116A02BB
:18019000E09CD0500C9FE0637C83FC9215C007EF6405F0E04E5CD0541
:1801A80000E123060FC3F50121A5020602C3F5011E800E04CD0500C999
:1801C000218000680C3F50121A7020602C3F501210000022A7020E14AE
:1801D800115C00CD0500218000680E516005E2AA7021922A702E12395
:1801F00005C2E301C9E5C55E0E04CD0500C1E12305C2F501C911170222
:18020800E09CD0500C71142020E09CD0500C70A074E6F2066696C659B
:1802200020737065636966696564E0D0A0A5472616E736665722061E5
:1802380626F72746564D0A0A240A0746696C65206E6F7420666F757D
:18025000E642E0D0A0A5472616E7366657220616F72746564D0A18
:18026800A240A075472616E7366657220416F72746420627920FE
:18028000F746865722056E642E0D0A0A240A5472616E736665722006B
:1102980636F6D706C6574652E0D0A0A240000000089
:00000000000
```

Convert the .HEX file to a .COM file with the command LOAD, e.g:

```
LOAD BLOW
```

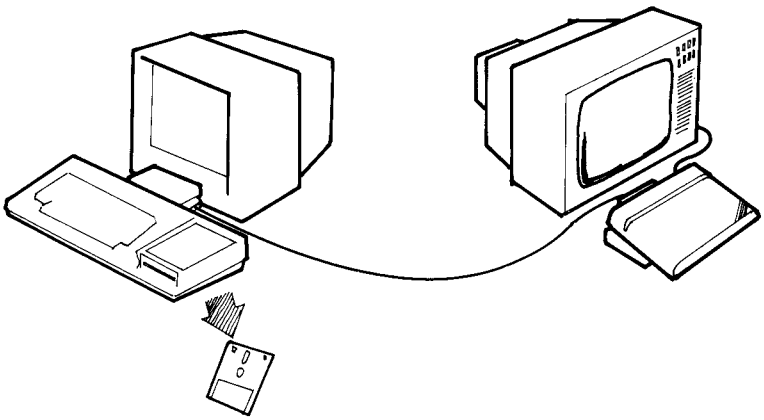
The program assumes that the serial input/output from the CP/M computer is configured to the Reader/Punch logical devices.

Run the program by typing:

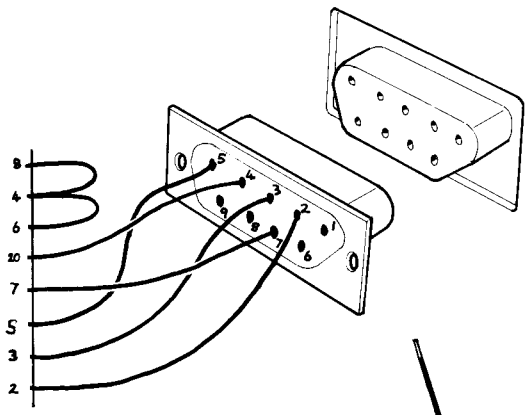
```
BLOW FILE.TYP
```

(*) SEE ADDENDUM, PAGE 67

# Spell 17:
# Transmitting the file from a Sinclair Spectrum



This transmitting program transfers areas of memory from a Spectrum fitted with an Interface 1.



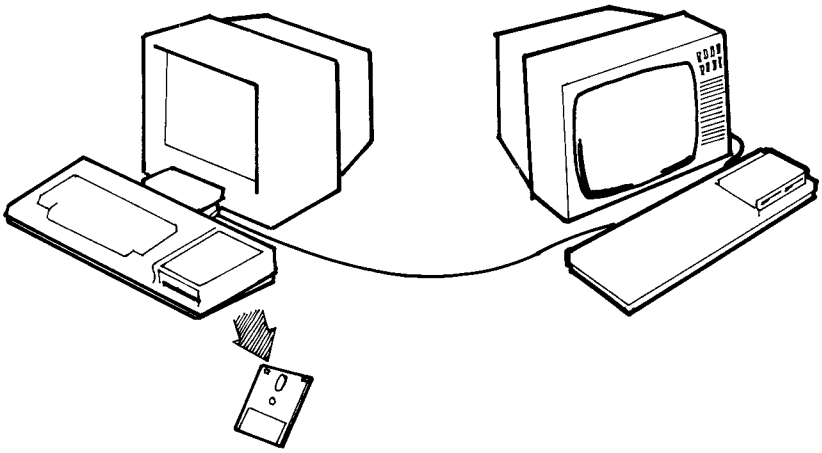

The data to be transmitted should be loaded as binary data into the highest free memory possible. Simple errors in transmission will be automatically re-tried.

```
  5 INPUT "Baud rate : ";baud
 10 FORMAT "b";baud
 20 OPEN #4;"b"
 25 OPEN #3;"b"
 30 INPUT "Type in start of text in Memory : ";mstart
 40 LET initl=mstart
 50 INPUT "Type  in  length  of  text   : ";mlen
 55 INPUT "Filename ";f$
 56 IF LEN f$>16 THEN GO TO 55
 57 IF LEN f$<16 THEN LET f$=f$+" ": GO TO 57
 60 PRINT "Commencing Transfer"
100 LET blkno=0
105 LPRINT CHR$ (2);
110 LET a$=INKEY$#4: IF a$="" THEN GO TO 110
120 IF a$=CHR$ (3) THEN PRINT "TRANSFER ABORTED": BEEP
    1,5: GO TO 9000
130 IF a$<>CHR$ (6) THEN GO TO 105
140 LPRINT f$;
150 LET bkh=INT (blkno/256)
160 LET bkl=blkno-bkh*256
170 LPRINT CHR$ (bkl);CHR$ (bkh);
180 LET chksm=0
200 LET ftn=mlen+initl-mstart
220 IF ftn>=128 THEN LET ftn=128
260 LPRINT CHR$ (ftn);
265 IF ftn=0 THEN GO TO 300
270 PRINT ftn: FOR t=1 TO ftn
280 LET mloc=PEEK (mstart): LPRINT CHR$ (mloc);:
      LET chksm=chksm+mloc
```

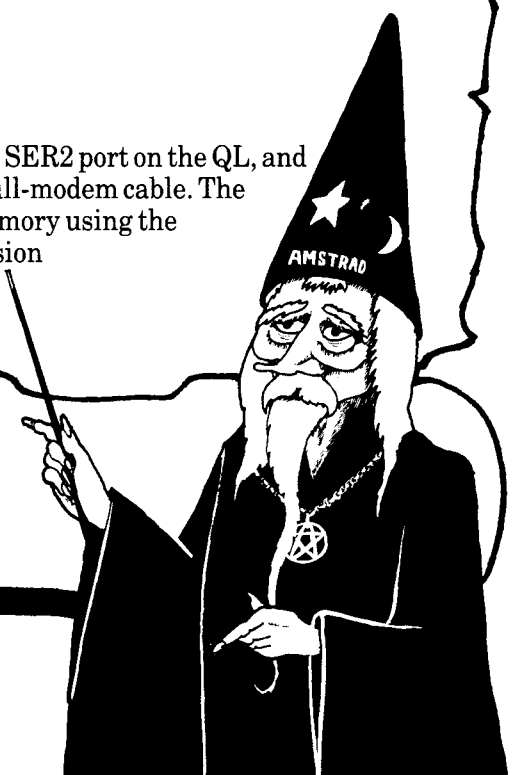
....continued next page

```
 290 LET mstart=mstart+1: NEXT t
 300 LET ckh=INT (chksm/256)
 310 LET ckl=chksm-(ckh*256)
 320 LPRINT CHR$ (ckl);CHR$ (ckh);
 330 LET a$=INKEY$#4
 335 IF a$=CHR$ (6) AND ftn=0 THEN GO TO 500
 340 IF a$=CHR$ (6) THEN LET blkno=blkno+1: GO TO 140
 350 IF a$=CHR$ (21) THEN LET mstart=mstart-ftn: GO TO
     140
 360 IF a$<>CHR$ (3) THEN GO TO 330
 370 GO TO 120
 500 PRINT "TRANSFER COMPLETE"
 510 BEEP 1,1
9000 CLOSE #3: CLOSE #4
```

# Spell 18:
# Transmitting the file from a Sinclair QL



This transmitting program sends data from the SER2 port on the QL, and requires the standard QL RS232 cable and a Null-modem cable. The data to be transmitted must first be put into memory using the LBYTES command. Simple errors in transmission will be automatically re-tried.

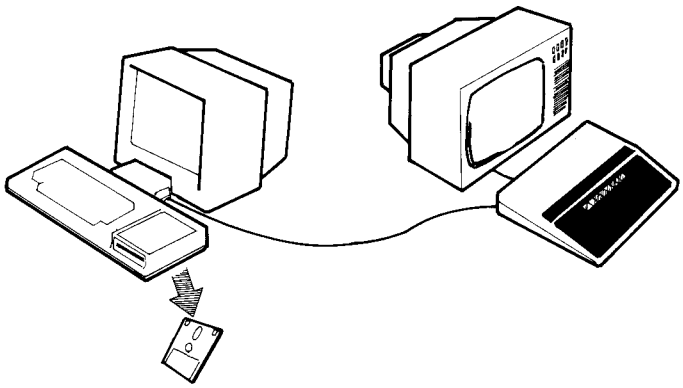


```
100 CLS
110 INPUT "Baud Rate: ";b
120 BAUD b
130 OPEN #3,ser2
140 INPUT "Type in start of text in memory:     ";mstart
150 initl=mstart
160 INPUT "Type in length of text  : ";mlen
170 INPUT "Filename : ";f$
180 IF LEN (f$)>16 THEN GO TO 170
190 IF LEN (f$)<16 THEN f$=f$&" " : GO TO 190
200 PRINT "Commencing Transfer"
210 blkno=0
220 PRINT #3,CHR$(2);
230 a$=INKEY$(#3): IF a$="" THEN GO TO 230
240 IF a$=CHR$(3) THEN PRINT "TRANSFER ABORTED!!": BEEP
    1000,50: GO TO 490
250 IF a$<>CHR$(6) THEN GO TO 220
260 PRINT#3,f$;
270 bkh=INT (blkno/256)
280 bkl=blkno-bkh*256
290 PRINT#3,CHR$(bkl);CHR$(bkh);
300 chksm=0
310 ftn=mlen+initl-mstart
320 IF ftn>=128 THEN ftn=128
330 PRINT#3,CHR$(ftn);
340 IF ftn=0 THEN GO TO 380
350 FOR t=1 TO ftn
360 mloc=PEEK(mstart): PRINT#3, CHR$(mloc);: chksm=
    chksm+mloc
370 mstart=mstart+1: NEXT t
380 ckh=INT (chksm/256)
390 ckl=chksm-(ckh*256)
400 PRINT#3,CHR$(ckl);CHR$(ckh);
```

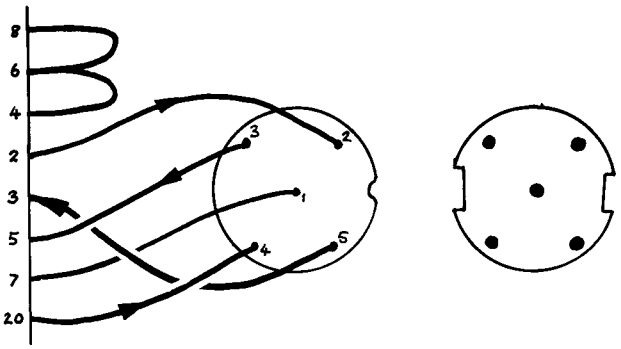
....continued next page

```
410 a$=INKEY$(#3)
420 IF a$=CHR$(6) AND ftn=0 THEN GO TO 470
430 IF a$=CHR$(6) THEN blkno=blkno+1: GO TO 260
440 IF a$=CHR$(21) THEN mstart=mstart-ftn : GO TO 260
450 IF a$<>CHR$(3) THEN GO TO 410
460 GO TO 240
470 PRINT "Transfer Complete"
480 BEEP 6000,12
490 CLOSE #3
```

# Spell 19:
# Transmitting the file from an Acorn BBC Micro



This transmitting program requires that OS1.1 (or later versions) be used. Data from a specified file is transmitted through the RS423 port.




```
100 PRINTCHR$(12)''
110 INPUT"File to transfer";F$
```

....continued next page

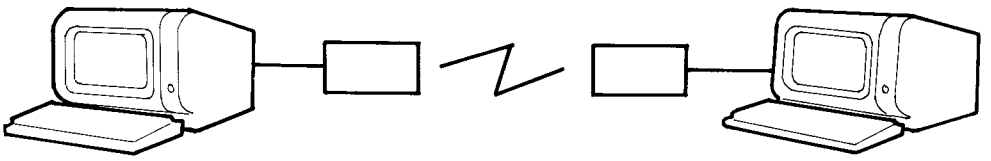
```
120 X=OPENIN F$
130 *FX3,7
140 REPEAT: F$=F$+" ": UNTIL LEN(F$)=16
150 BN=-1
160 REPEAT
170  PRINT CHR$(2);
180  PROCwait
190 UNTIL W<>0
200 REPEAT
210  BN=BN+1
220  PROCgetblock
230   REPEAT
240    PRINT F$;
250    PROCdbyte(BN)
260    PRINT CHR$(N);
270    PRINT A$;
280    PROCdbyte(CK)
290    PROCwait
300   UNTIL W<>0
310 UNTIL EOF#X
320 BN=BN+1
330 REPEAT
340  PRINT F$;
350  PROCdbyte(BN)
360  PRINT CHR$(0);
370  PROCdbyte(0)
380  PROCwait
390 UNTIL W<>0
400 *FX3,0
```

....continued next page

```
410 PRINT'"TRANSFER COMPLETE"'
420 *FX2,0
430 CLOSE#X: END
440 DEFPROCgetblock
450 A$="": N=0: CK=0
460 REPEAT
470  N=N+1
480  A=BGET#X
490  CK=CK+A
500  A$=A$+CHR$(A)
510 UNTIL (N=128)OR(EOF#X)
520 ENDPROC
530 DEFPROCdbyte(B%)
540 J=B% DIV 256
550 B%=B%-J*256
560 PRINT CHR$(B%);CHR$(J);
570 ENDPROC
580 DEFPROCwait
590  *FX2,1
600  K=GET
610  IF K=3 THEN 650
620  IF K=6 THEN W=6 ELSE W=0
630  *FX2,0
640 ENDPROC
650 *FX3,0
660 PRINT CHR$(7);"TRANSFER ABORTED": GOTO 420
```

# Appendix 1: RS232C Connections

For a complete understanding of the connections required between the RS232C and the outside world, it is important to realise that all devices with a serial interface can be classified either as a modem or as a terminal. Modems are merely a way of extending the length of the connection (often via a telephone wire), and Fig 1 (below) shows a simplified, idealised, connection between two terminals.



IDEALISED TERMINAL TO TERMINAL CONNECTION

Fig 1

The standard connector used for serial interfaces has 25 pins although only up to 7 are required in most cases. When connecting a terminal to a modem, a 'one to one' cable is used, i.e. pin 1 to pin 1, pin 2 to pin 2....pin 25 to pin 25. Assuming such cables are in use, data is transferred as follows:

Following the signal path from left to right, characters from the keyboard are sent out of pin 2 of the left-hand terminal, to pin 2 of the modem (the connection marked 'transmit data'). The left-hand modem then sends the characters, via the telephone line, to the right-hand modem. The characters are received at pin 3 of the right-hand modem (the connection marked 'receive data') which sends them to pin 3 of the right-hand terminal. On receipt of the characters, the right-hand terminal displays them on the screen.

Notice how the names of the connections 'transmit data' and 'receive data' are expressed from the point of view of the terminal, not the modem.

The data path from left to right just described, is exactly matched by a data path from right to left which uses the same numbered connections, i.e. pin 2 from terminal to modem (transmitting), and then pin 3 from modem to terminal (receiving). This arrangement is perfectly symmetrical, and there is no confusion over who is using which pin number, and for what direction of data transfer.

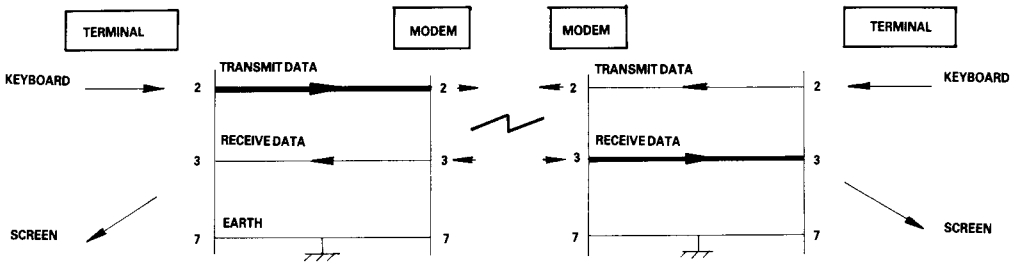



Fig 2

Problems of definition arise, however, when we wish to connect two terminals together locally, without the intervening pair of modems. We cannot connect pin 2 to pin 2 because both keyboards will be transmitting head-on, and neither screen is connected to anyone who is sending. The obvious solution is to cross over pins 2 and 3 so that the transmit pin of each terminal is connected to the receive pin of the other. A cable containing such a cross-over connection is known as a 'Null-modem' cable because of the way in which it replaces the pair of back to back modems.

The earth pin (pin 7) is still common to both terminals using this arrangement.

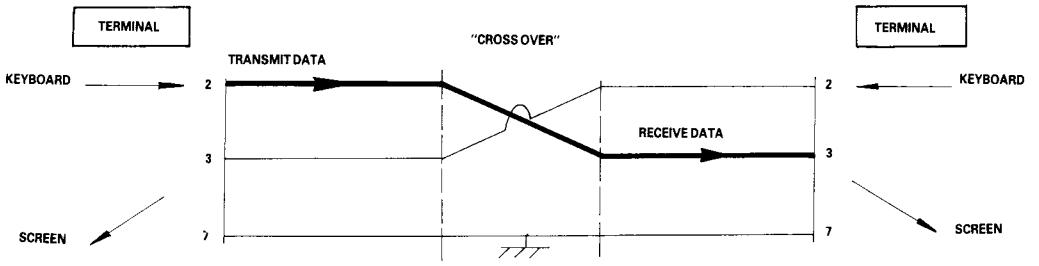



Fig 3

Naturally, the combination of an Amstrad computer+RS232C is considered a terminal, and therefore to connect to a modem, (for example, to access a dial-up database) requires a simple one-to-one cable.

The Null-modem cable is required for connecting to other terminals. The sort of equipment we mean by other terminals is: a second Amstrad computer+RS232C, a conventional Visual Display Unit (VDU), a printer with a serial interface, or perhaps a desk-top computer which requires a VDU.

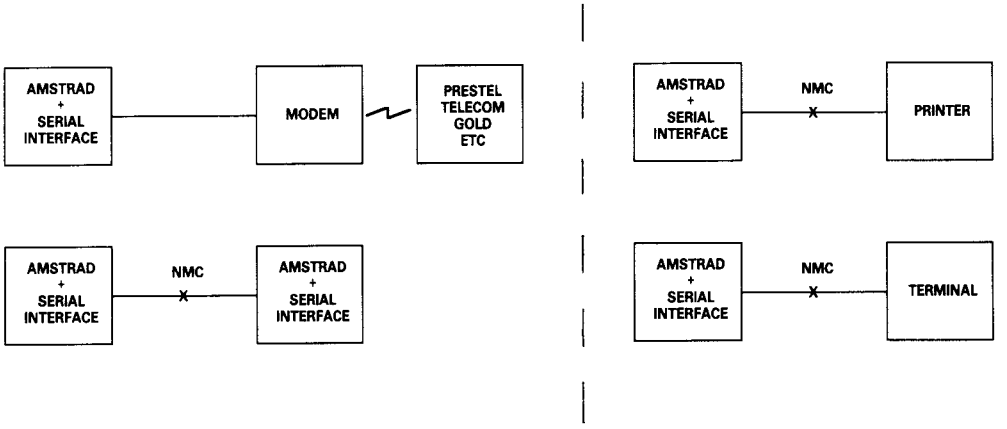



Fig 4

There is a point to be noted here: many manufacturers of desk-top computers wire up the serial interface (for a VDU or a printer) as if it were a modem, not a terminal. This is in the belief that life will therefore be simpler because VDU's and printers can be connected to that computer with one-to-one cables.

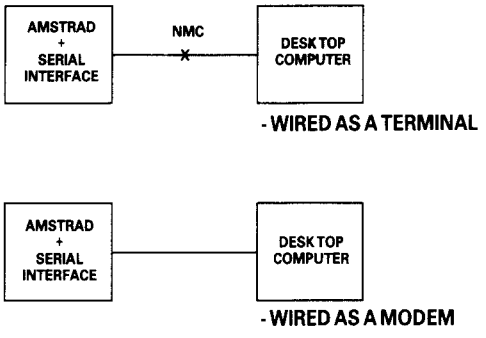



Fig 5

In a perfect world, it would be possible to identify which serial devices behave like modems and which behave like terminals, by examining the 'sex' of the 25-way connector - terminals should have a 'male' connector, and modems a 'female' connector. This is not, unfortunately, as reliable a guide as it should be, as many manufacturers of terminals and printers equip them with 'female' connectors, mostly for reasons of electrical safety.

If in doubt, the ultimate test is to examine the user manual and determine the function of PIN 2 - if the description includes the word 'TRANSMIT' then the equipment is wired as a Terminal, and if it includes the word 'RECEIVE' then the equipment is wired as a modem.

# Hardware flow control

The simplified connection described so far does not allow any control of the data flow. In practice, we often wish the receiving device to have some control over the transmitting device, thus preventing the receiving device from being overwhelmed (where it is slower in digesting the input than the rate at which the input is arriving). In addition, if the transmitting device has reason to mistrust the data which it is sending, there should be provision for it to disable the receiving device.

In the case of modem to terminal connection; when the terminal is happy to transmit it activates pin 4 - the RTS pin (Request To Send). When the modem is ready to receive input, it activates pin 5 - the CTS pin (Clear To Send). The terminal will only send when CTS is activated. Thus the modem can control the flow rate using CTS.

When the modem considers that the data which it is about to send is suitable, it activates pin 8 - the DCD pin (Data Carrier Detect). When the terminal is ready to receive input it activates pin 20 - the DTR pin (Data Terminal Ready). The modem will only transmit when DTR is activated. Thus the terminal can control the flow rate using DTR.

There are two further signals which must be introduced here. One is on pin 22 - the Ring Indicator, which simply allows the modem to tell the terminal that the 'phone is ringing! (at which point software in the terminal might be expected to wake up). The other signal is on pin 6 - DSR (Data Set Ready). This signal is ignored by the receiving side of the RS232C; the modem will activate this signal at much the same time as it activates DCD, and therefore no functionality is lost by ignoring DSR.

CONNECTIONS TO A MODEM

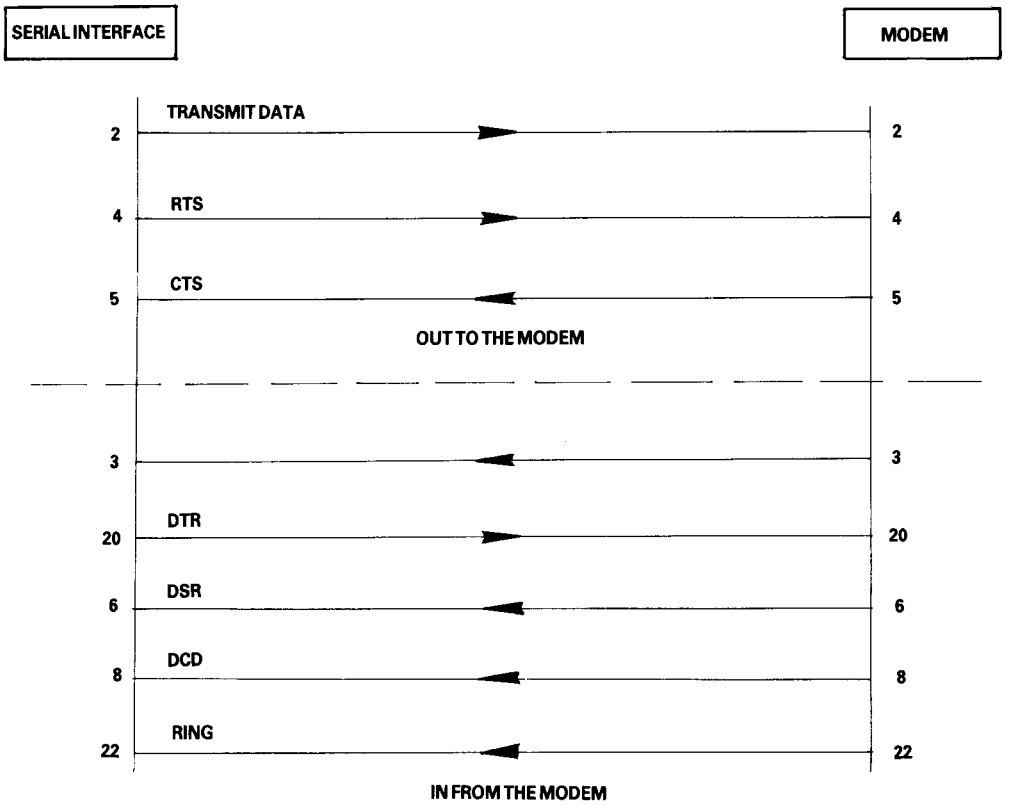



Fig 6

In the case of terminal-to-terminal connections, the Null-modem cable must be used with the additional connections to pins 2, 3, and 7 as already discussed. The full Null-modem cable swaps pins 4 and 8 - the RTS/DCD 'I am happy to send' signals, and pins 20 and 5 - the DTR/CTS 'Busy' signals. To be on the safe side, pin 6 (DSR) is connected to pin 8 (DCD) in case *that* end of the cable is ever connected to a terminal which is fussy and requires DSR as well as DCD.

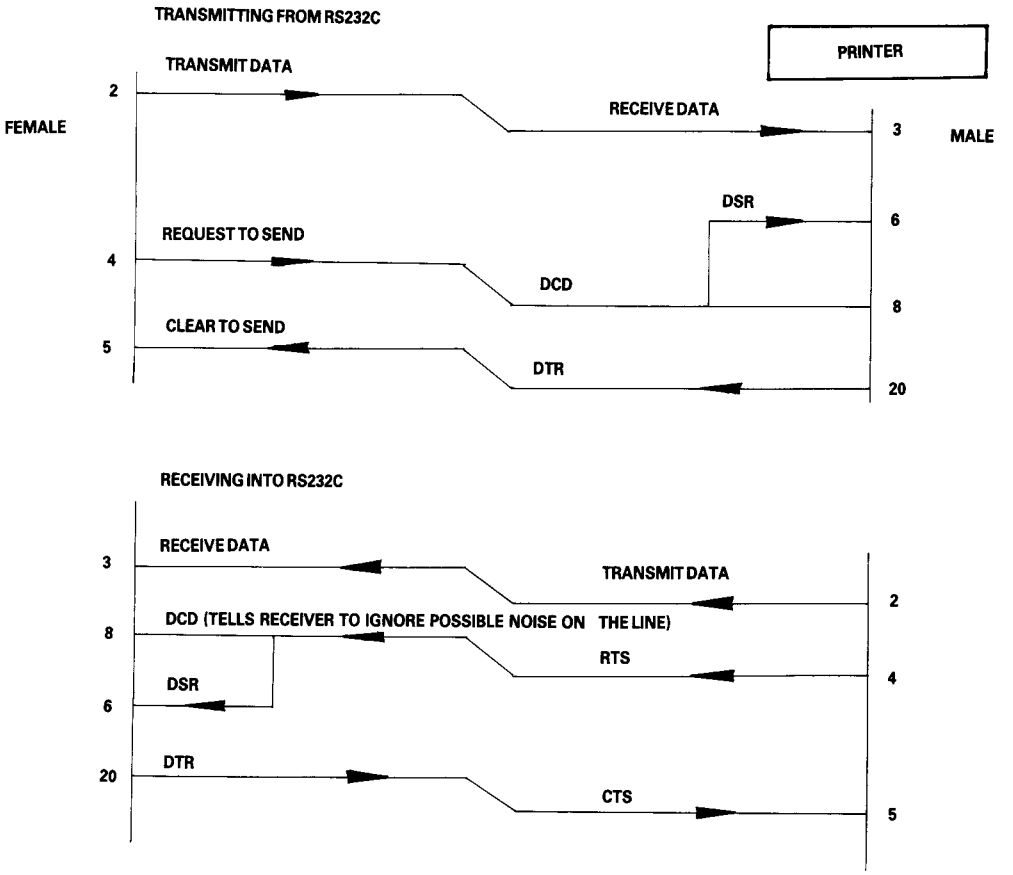
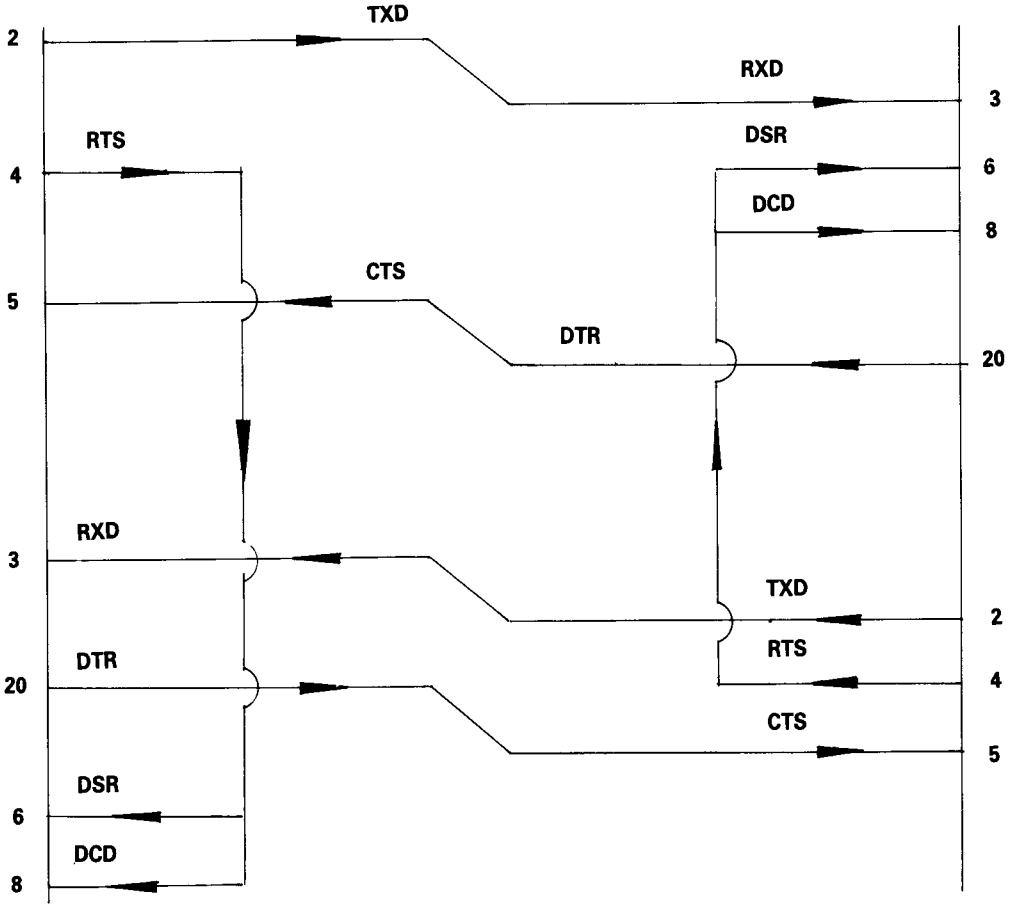



Fig 7

There is a school of thought which says that a Null-modem cable, unlike the pair of modems it replaces, is ALWAYS 'happy to send'. Therefore, it is quite in order to generate DCD (and DSR) permanently. This is achieved by connecting them to the RTS at the same end of the cable, rather than to the RTS at the other end of the cable.




THE RECOMMENDED NMC

Fig 8

Finally, if the transmission rate from one of the two terminals is known to be unstoppable (e.g. a person typing at a keyboard), or is so slow and infrequent (e.g. the software handshake characters 'XON, XOFF' sent by a printer) that there is no danger of over-running the receiving end, then it is permissible to permanently enable the transmission by linking pin 5 (CTS) to pin 4 (RTS), i.e. to always send if ready (at the transmitting end of the cable). It may well be facilitated in any case, for the transmitting terminals to ignore the state of CTS under these circumstances.

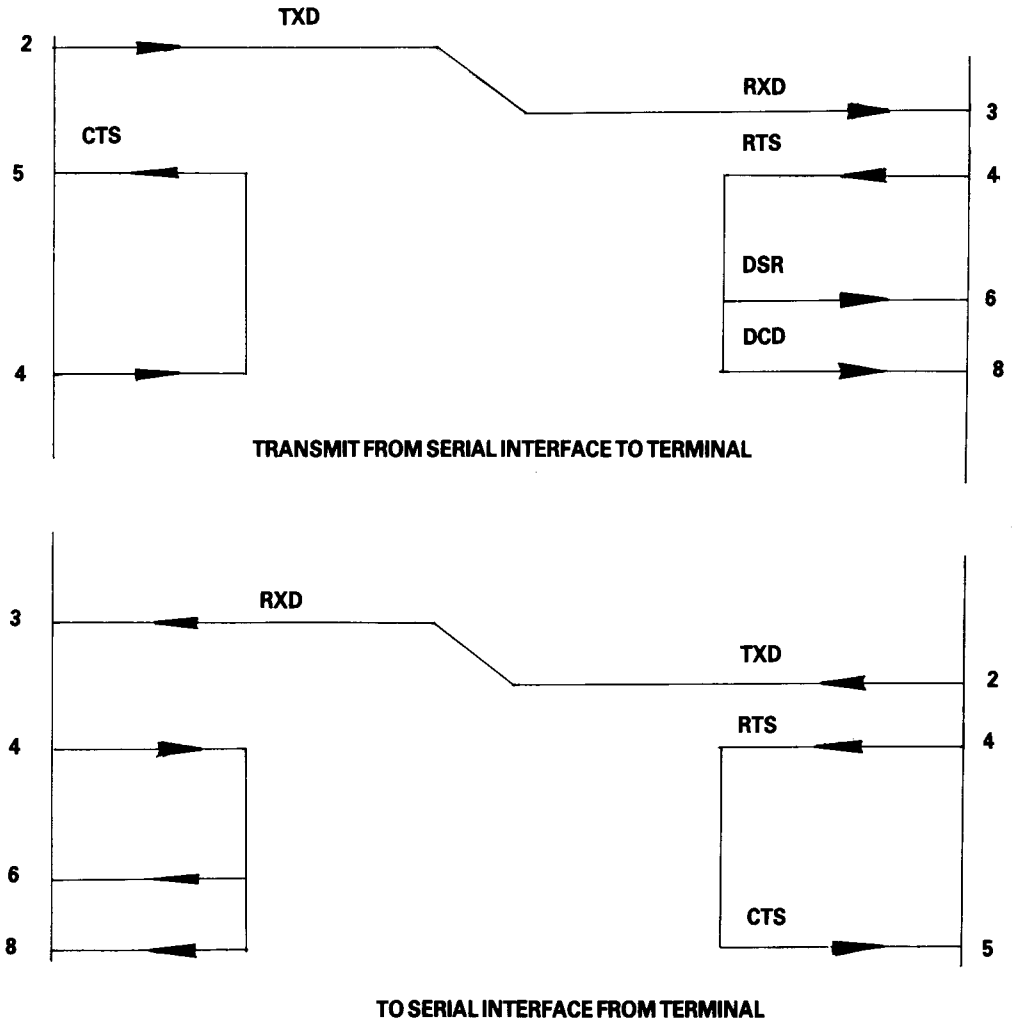

TXD
2
RXD
3
CTS
RTS
5
4
DSR
6
DCD
4
8
TRANSMIT FROM SERIAL INTERFACE TO TERMINAL
3
RXD
TXD
2
4
RTS
4
6
CTS
8
5
TO SERIAL INTERFACE FROM TERMINAL

Fig 9

# Appendix 2: Using the RS232C in CP/M

The ROM program supplied with the RS232C is not used by CP/M. It is possible, but unlikely, that a CP/M program will access the RS232C ROM by using firmware. The facilities described below are supplied as standard with the CP/M for the DDI-1, the CPC664, or the CPC6128.

**NOTE** - Users of CP/M Plus version 1.0 for the CPC6128 must incorporate a small change described (in a box) at the end of this appendix.

Full details of the facilities and programs described below may be obtained by reference to the various firmware and CP/M guides for your computer or disc system.

## CP/M 2.2 (as supplied with the DDI-1, the CPC664 and, for compatibility purposes, with the CPC6128):

All parameters relating to the RS232C are set up when CP/M 2.2 is loaded. Each system disc includes a 'configuration sector' which contains the particular parameters required. The utility program `SETUP` provides a means to edit the configuration sector.

The AMSTRAD BIOS 2.2 implements the RS232C as the physical device `TTY:`. Both `SETUP` and `STAT` allow you to assign this physical device to various logical devices and thus permit you to refer to the RS232C when operating the utility program `PIP`, or when writing assembler programs which access the BDOS (`CALL 5`) interface. The BIOS also contains an extended jumpblock which provides direct RS232C support to the assembly language programmer.

`STAT` provides the facility of re-assigning the physical devices `TTY:`, `CRT:`, and `LPT:` to the logical devices `CON:`, `RDR:`, `PUN:`, and `LST:`.

The console (`CON:`) is normally assigned to the keyboard/screen (`CRT:`).

The printer (`LST:`) is normally assigned to the parallel printer port (`LPT:`).

The input device (`RDR:`) and the output device (`PUN:`) are normally assigned to the RS232C (`TTY:`).

The current assignments may be interrogated with the command:

```
STAT DEV:
```

....and may be changed by commands of the form:

`STAT` ‹logical device› = ‹physical device›

e.g. `STAT LST:=TTY:`

....which assigns the RS232C to send printer output.

If an output assignment is made to a non-existent or permanently busy physical device, then the computer may require a **[CTRL]C** to be typed at the keyboard in order to prevent a 'lock-up' condition.

PIP is a file-copying program which regards input and output logical devices as if they were files. Thus....

```
PIP PUN:=FILE.TYP   will send a file to the RS232C
PIP FILE.TYP=RDR:   will receive a file from the RS232C
PIP PUN:=CON:       sends keyboard input to the RS232C
PIP CON:=RDR:       sends RS232C input to the screen
```

....if the default assignments are in force. Transfers from the keyboard or the RS232C are generally terminated by an end-of-file **[CTRL]Z** character.

SETUP allows you to change the RS232C parameters (amongst others). The configuration includes support for a two-channel serial interface - the RS232C is channel A. The defaults are:

9600 baud
8 data bits
no parity
1 stop bit
RDR: and PUN: assigned to RS232C

Insert a copy of the CP/M 2.2 system disc, start up CP/M, and at the A> prompt, type:

```
SETUP
```

Step through the irrelevant questions by typing Y (for Yes) until you reach the sixth section (the default IO byte settings).

This section offers a change to the power-up assignment of physical and logical devices (see the previous discussion of STAT). The IO byte is a special marker stored at address 0003 which indicates the physical to logical device allocations. The new power-on assignments are indicated by instructions such as LST:=TTY: which would assign the printer output to the RS232C.

The thirteenth section (Z80 SIO Channel A settings) offers a change to the power-up parameters, which must be input in the order displayed, i.e:

Transmit (tx) baud rate (must be one of the numbers 19200, 9600, 4800, 3600, 2400, 2000, 1800, 1200, 600, 300, 200, 150, 110, 75, or 50)

Receive (rx) baud rate (must be one of the numbers 19200, 9600, 4800, 3600, 2400, 2000, 1800, 1200, 600, 300, 200, 150, 110, 75, or 50)

Data bits (must be one of the numbers 5, 6, 7, or 8)

Parity (must be one of the words ODD, EVEN, or NONE)

Stop bits (must be one of the numbers 1, 1.5, or 2)

Example:

```
9600 9600 8 NONE 1
```

....which corresponds to the default settings.

Finally, the option is offered to update the system disc with the new configuration. At this point, you will be given the opportunity of restarting CP/M (which will invoke any new RS232C settings just set up in the configuration sector).

# CP/M Plus (as supplied with the CPC6128):

The RS232C parameters are set to default values when CP/M Plus is started. The various parameters can be altered at any time by using the program SETSIO. It is intended that SETSIO will normally be invoked as part of the operation of the startup file PROFILE.SUB.

The AMSTRAD BIOS for CP/M Plus implements the RS232C as the physical device SIO. The utility program DEVICE allows you to re-assign this physical device (which defaults to the logical device AUX:) and thus permits you to refer to the RS232C when operating the utility program PIP or when writing assembler programs which access the BDOS (CALL 5) interface. The BIOS also contains an extended jumpblock which provides direct RS232C support to the assembly language programmer.

DEVICE provides the facility of re-assigning the physical devices SIO, CRT, and LPT to the logical devices CON: (which can be sub-divided into CONIN: and CONOUT:), AUX: (which can be sub-divided into AUXIN: and AUXOUT:), and LST:.

The console (CON:) is normally assigned to the keyboard/screen (CRT).

The printer (LST:) is normally assigned to the parallel printer port (LPT).

The auxiliary device (AUX:) is normally assigned to the RS232C (SIO).

The current assignments may be interrogated with the command:

```
DEVICE
```

....and may be changed by commands of the form:

```
DEVICE ‹logical device› = ‹physical device›
```

e.g. `DEVICE LST:=SIO`

....which assigns the RS232C to send printer output, or:

```
DEVICE CON:=SIO
```

....which allows operation of the computer from a terminal attached to the RS232C (`DEVICE CON:=CRT` typed on the remote terminal will restore operation to the local keyboard and screen).

If an output assignment is made to a non-existent or permanently busy physical device, then the computer will issue a warning message in order to prevent a 'lock-up' condition.

`PIP` is a file-copying program which regards input and output logical devices as if they were files. Thus....

```
PIP AUX:=FILE.TYP   will send a file to the RS232C
PIP FILE.TYP=AUX:   will receive a file from the RS232C
```

....if the default assignments are in force. Transfers from keyboard or RS232C are generally terminated by an end-of-file **[CTRL]Z** character.

`SETSIO` displays and changes the parameters of the RS232C. Any number of the parameters may be changed - simply include the required clauses in any order:

e.g. `SETSIO TX 75 RX 1200 BITS 7 PARITY ODD`

Only the initial letter of the first word in each clause is required; the rest of that word is optional. An illegal clause will produce an error message, and if a cause is specified twice, then the latter clause is used.

If the baud rate name (TX or RX) is omitted, then both TX and RX are set. If the baud rate is changed but the number of stop bits is not, then the number of stop bits will be set to 1 (if the baud rate is greater than 110), or will otherwise be set to 2.

Clauses available are:

`TX` ‹baud rate› - default 9600 - sets transmitter baud rate

`RX` ‹baud rate› - default 9600 - sets receiver baud rate

....where ‹baud rate› must be one of the numbers `50`, `75`, `110`, `134.5`, `150`, `300`, `600`, `1200`, `1800`, `2400`, `3600`, `4800`, `7200`, `9600`, or `19200`.

`BITS` ‹n› - default 8 - sets the number of data bits

....where ‹n› must be one of the numbers `5`, `6`, `7`, or `8`.

`PARITY` ‹p› - default NONE - sets the type of parity

....where ‹p› must be one of the words `EVEN`, `ODD`, or `NONE`.

`STOP` ‹n› - default 1 - sets the number of stop bits

....where ‹n› must be one of the numbers 1, 1.5, or 2.

XON ‹x› - default OFF - invokes or cancels the software flow control

....where ‹x› must be one of the words ON or OFF.

HANDSHAKE ‹x› - default ON - invokes or cancels the hardware flow control

....where ‹x› must be one of the words ON or OFF.

## CP/M Plus - version 1.0

The following 'patch' should be applied to version 1.0 of CP/M Plus if the RS232C is to be used from CP/M Plus. Two discs are required as follows:

The disc inserted in Drive B: should be Side 2 of the master system discs package provided with the CPC6128.

The disc inserted in Drive A: should be a write-enabled COPY of Side 1 of the master system discs package (or of whatever other disc you use to load CP/M Plus).

Now start up CP/M Plus, and type in ONLY the normally printed characters of the following, checking that the computer responds as indicated by REVERSE BLOCK characters

```
A>B:SID C10CPM3.EMS
CP/M 3 SID - Version 3.0
NEXT MSZE PC END
6500 6500 0100DAFF
#S4EF
04EF B7 57
04F0 28 ED
04F1 08 78
04F2 ED 1F
04F3 78 38
04F4 1F 0A
04F5 38 7A
04F6 08 B7
04F7 CD C4
04F8 8B .
#WC10CPM3.EMS
00C8h record(s) written.
#[CTRL]C
A>REN P11CPM3.EMS=C10CPM3.EMS
A>
```

# Appendix 3:
# RS232C ROM in BASIC and Machine Code

The ROM supplied in your RS232C contains a program which automatically adds a number of 'external' commands to BASIC. These commands are identified by the bar | symbol (obtained by pressing **[SHIFT]@** ). The commands can be accessed in machine code programs by using the firmware facilities of KL_FIND_COMMAND and preparing suitable parameter blocks. Refer to the Concise Firmware Specification for your particular AMSTRAD computer.

The ROM program automatically reserves an allocation of RAM, thus reducing the amount left for your BASIC programs. Computers fitted with an RS232C will have approximately 40K free, whilst those also having a disc drive will have approximately 39K free. (There is no erosion of available memory when using CP/M in conjunction with the RS232C.)

The RS232C external commands are grouped into categories for easy reference. The following conventions are followed in the descriptions of the commands:

1. Parameters in [square brackets] are optional.

2. ‹status› means the address of an integer variable (which must have previously been assigned a dummy value) into which a code, representing the success or reason for failure of the command, is placed. The address of an integer variable is calculated by BASIC if the variable name is preceded by an @ symbol.

   e.g. `S%=0: |RINGWAIT,@S%,60`

3. ‹input string› means the address of a string variable, which must have previously been assigned a dummy string containing at least as many characters as you are expecting to receive during execution of the command. The address of a string variable is calculated by BASIC if the string name is preceded by an @ symbol.

   e.g. `S%=0: RCVD$=STRING$(100," "): |INBLOCK,@S%,@RCVD$`

4. ‹output string› means the address of a string to be sent by the command. Version 1 of BASIC (CPC464) requires that this address is the address of a string variable, which is indicated by preceding a string variable with an @ symbol. In later versions of BASIC (CPC664 and CPC6128), the @ symbol is optional and constant strings may be specified.

   e.g.

   ```
   S%=0: TRANS$="Any version of BASIC": |OUTBLOCK,@S%,@TRANS$
   S%=0: TRANS$="Version 2 or later": |OUTBLOCK,@S%,TRANS$
   ```

   ....or....

   ```
   S%=0: |OUTBLOCK,@S%,"Version 2 or later"
   ```

5. ‹input character› means the address of an integer into which the ASCII value of a single input character will be placed. The integer must have previously been assigned a dummy value. The address of an integer variable is calculated by BASIC if the variable name is preceded by an @ symbol.

   e.g. `S%=Ø: C%=Ø: |INCHAR,@S%,@C%`

6. ‹output character› means an integer or real variable, constant or numeric expression, evaluating to the ASCII value of a character (in the range 0 to 255).

   e.g. `|SETBLOCKEND,13 ' carriage return`

7. ‹filename› means a legal cassette or disc (as appropriate) filename. Version 1 BASIC (CPC464) requires that the filename is specified as the address of a string variable, which is indicated by preceding the string variable with an @ symbol. In later versions of BASIC (CPC664 and CPC6128), the @ symbol is optional and constant strings may be specified.

8. ‹*parameters other than those mentioned above*› means an integer or real variable, constant or numeric expression, evaluating to the parameter described.

# Commands for ROM housekeeping

## |ROMOFF

```
|ROMOFF
```

COMMAND: Re-initialises BASIC (losing any currently loaded program and the values of all variables) after disabling the mechanism which automatically logs-on external ROM commands. This will restore the free RAM to the absolute maximum, making it possible to load very large programs from cassette (The disc ROM will also be disabled) which would otherwise experience a RAM clash with the disc, the RS232C, or other ROMs.

## |ROMCAT

```
|ROMCAT[,‹ROM Number›]
```

COMMAND: Displays a 'catalog' (directory) of up to 16 currently logged-in external ROMs. If the optional parameter is specified, then the directory shows the external commands available from that ROM. Note how all empty slots appear to be occupied by BASIC.

# Commands for RS232C housekeeping

## !SETSIO

!SETSIO,‹baud rate›[,‹receive baud rate›[,‹hardware flowcontrol›
[,‹data bits›[,‹parity›[,‹stop bits›]]]]]

COMMAND: Alters the fundamental operating parameters of the RS232C. After the first parameter, all subsequent parameters are optional. If an invalid parameter is specified, then all subsequent parameters are ignored.

The parameters ‹data bits›,‹parity›, and ‹stop bits› are referred to collectively as the 'framing bits'.

The ‹baud rate› parameter (default 9600 baud) is an integer specifying one of the allowed baud rate values. If a ‹receive baud rate› is specified then the first parameter will only set the transmit baud rate.

The ‹receive baud rate› parameter (default 9600 baud) is an integer specifying one of the allowed baud rates.

The ‹hardware flow control› parameter (default ENABLED) is an integer which enables or disables hardware flow control. Zero disables flow control; non-zero enables flow control.

If hardware flow control is disabled, then the RS232C will transmit regardless of the condition of the hardware flow control CTS on pin 5, and will receive regardless of the condition of the hardware flow control DCD on pin 8. If hardware flow control is disabled, then the RS232C will permanently activate the flow control signal DTR on pin 20 until a !CLOSESIO command is issued or hardware flow control is re-enabled.

Disabling flow control can be useful if the cable or equipment to which the RS232C is connected, is unable to play its part in the operation of the flow control signals. A major disadvantage of disabling hardware flow control is that there is a danger of characters being lost if the receiving device is operating in a start/stop manner, slower than the transmitting device.

The ‹data bits› parameter (default 8) is an integer in the range 5 to 8, specifying the number of data bits in transmitted and received characters.

The ‹parity› parameter (default NO parity) is an integer in the range 0 to 2, specifying the form of parity to be generated and checked. 0 specifies NO parity, 1 specifies ODD parity, and 2 specifies EVEN parity.

The ‹stop bits› parameter (default one stop bit) is an integer in the range 0 to 2, specifying the number of stop bits in transmitted and received characters. 0 specifies one stop bit, 1 specifies one and a half stop bits, and 2 specifies two stop bits.

Allowed baud rates are: 19200, 9600, 4800, 3600, 2400, 2000, 1800, 1200, 600, 300, 200, 150, 110, 75, 50.

```
e.g. ISETSIO,300 '  300 baud
     ISETSIO,9600,9600,0 ' disable flow control, 9600 baud
     ISETSIO,75,1200,1,7,1,0 ' TX75 baud, RX1200 baud,
     enable hardware flow control, 7 data bits, odd parity,
     1 stop bit.
```

## ISETTIMEOUT

ISETTIMEOUT,‹timeout period›

COMMAND: Alters the timeout period after which a break, or a character or block transfer command will 'resign' if it has not yet been completed.

The ‹timeout period› parameter (default 0 mS) is an integer in the range −1 to 65534, specifying the length of the timeout period in milliseconds. The value −1 disables the timeout, i.e. sets the timeout to infinity.

```
e.g. ISETTIMEOUT,1000 ' one second timeout
```

## ISIO

ISIO,‹status›

COMMAND: Returns a general purpose status from the serial interface. This command is supplied for specialist use only, as other commands implicitly interrogate the relevant status bits as required during their execution. The status is returned as a bit-significant 16 bit integer:

| | | |
|---|---|---|
| Bit 15 | (MSB) | 0 |
| Bit 14 | | Framing error |
| Bit 13 | | Overrun error |
| Bit 12 | | Parity error |
| Bit 11 | | 0 |
| Bit 10 | | 0 |
| Bit 9 | | 0 |
| Bit 8 | | All sent |
| Bit 7 | | Break received |
| Bit 6 | | 0 |
| Bit 5 | | CTS |
| Bit 4 | | Ring detect |
| Bit 3 | | DCD |
| Bit 2 | | Transmitter buffer empty |
| Bit 1 | | 0 |
| Bit 0 | (LSB) | Received data available |

Example sub-routine:

```
110 ' simulate RINGWAIT,@S%,100
120 S%=0: T=TIME+(100*300)
130 WHILE TIME<T
140 ISIO,@S%
150 IF S% AND 16 THEN S%=0: RETURN
160 WEND
170 S%=1: RETURN
```

## I RINGWAIT

I RINGWAIT, ‹status›, ‹timeout period›

COMMAND: Waits until Ring detect is true, or the timeout expires. This command is intended for use in an auto-answer modem configuration. The status is returned as an integer where 0 indicates that Ring detect is true, and 1 indicates that the timeout occurred.

The ‹timeout period› parameter is an integer in the range 0 to 65535 which sets the timeout period in seconds (approximately).

**NOTE** - This timeout is quite independent of the timeout associated with break, character and block transfer operations.

## I BREAKSEND

I BREAKSEND, ‹status›, ‹break time›

COMMAND: Waits until the transmitter buffer is empty ('all sent' status true) and then sends a 'break' with the line continuously 'marking'. This break can be detected by, and is intended as a signal to, equipment connected to the RS232C.

The status returned is:

0 * 256 - The break was sent OK.
2 * 256 - Timeout, the 'all sent' status did not occur within the previously specified timeout period, so the break was not sent.

The ‹break time› parameter is an integer in the range 0 to 65535 (where 0 means 65536), specifying the number of milliseconds the line is held marking.

## | CLOSESIO

| CLOSESIO,‹status›

COMMAND: Waits until the transmitter buffer is empty ('all sent' status true) and then shuts down the RS232C. Closing down the RS232C includes turning off the hardware flow control signals RTS pin 4 and DTR pin 20.

The status returned is:

0 * 256 - The RS232C was closed OK.
2 * 256 - Timeout, the 'all sent' status did not occur within the previously specified timeout period, so the RS232C was not closed.

# Commands for Character transfer

## | INCHAR

| INCHAR,‹status›,‹input character›

COMMAND: Reads a single character from the RS232C. If hardware flow control is enabled, then the signal DCD pin 8 must be true and the RS232C will activate DTR pin 20 if no data is available.

The status returned is:

0 * 256 - A character was read OK.
1 * 256 - No ‹input character› specified.
2 * 256 - Timeout, no character was read within the previously specified timeout period.
3 * 256 - A line break was received.
4 * 256 - A character was read with a framing error.
5 * 256 - A character was read with an overrun error.
6 * 256 - A character was read with a parity error.

If more than one hardware error condition (break, framing, overrun, and parity) occurs, then the lowest numbered error condition is reported. All error status bits are reset.

## |OUTCHAR

|OUTCHAR,‹status›,‹output character›

COMMAND: Waits for the transmitter buffer to become empty, then sends a single character to the RS232C. If hardware flow control is enabled, then the character will not be sent unless the signal CTS pin 5 is true. On power-up or after calling |SIO, the signal RTS pin 4 is activated until |CLOSESIO is called.

The status returned is:

0*256 - The character was sent OK.
1*256 - No ‹output character› was specified.
2*256 - Timeout, the character could not be sent within the previously specified timeout period.

# Commands for Block transfer

## |SETBLOCKEND

|SETBLOCKEND,‹output character›

COMMAND: Specifies a character as the end of block marker. A value of 256 or greater disables the end of block checking (default setting DISABLED).

## |INBLOCK

|INBLOCK,‹status›,‹input string›

COMMAND: Reads a string from the RS232C. The operation may terminate either when a timeout occurs, when the end of block marker is read, or when the ‹input string› is filled. If hardware flow control is enabled, then the signal DCD pin 8 must be true and the RS232C will activate DTR pin 20 if no data is available.

The status returned is:

0*256+0 - The ‹input string› was filled OK.
0*256+N - N characters, including the end of block marker, have been received.
1*256+0 - No ‹input string› or a zero length ‹input string› specified.
2*256+N - The (N+1)th character was not received during the previously specified timeout period. N characters received OK.
3*256+N - The Nth character received was a line break.
4*256+N - The Nth character was read with a framing error.
5*256+N - The Nth character was read with an overrun error.
6*256+N - The Nth character was read with a parity error.

If more than one hardware error condition (break, framing, overrun, and parity) occurs, then the lowest numbered error condition is reported. All error status bits are reset.

### |OUTBLOCK

|OUTBLOCK,‹status›,‹output string›

COMMAND: Sends a string to the RS232C. If hardware flow control is enabled then the character will not be sent unless the signal CTS pin 5 is true. On power-up or after calling |SIO, the signal RTS pin 4 is activated until |CLOSESIO is called.

The status returned is:

0*256+0 - The string was sent OK.
1*256+0 - No ‹output string› or a zero length ‹output string› was specified.
2*256+N - Timeout, a character from the string could not be sent within the previously specified timeout period.
N characters from the string remain to be sent.

## Commands for 'unintelligent' file transfer

### |SETFILEEND

|SETFILEEND,‹character›

COMMAND: Specifies a character as the end of file marker. The value is taken MOD 256 (default setting 27/**[CTRL]Z**).

### |INFILE

|INFILE,‹filename›

COMMAND: Receives an ASCII file and writes it to cassette or disc. This operation terminates when the end of file marker is read. There is no timeout. Error messages are sent to the screen if a filing system error occurs.

## |OUTFILE

|OUTFILE,‹filename›

COMMAND: Sends an ASCII file (plus an end of file marker character) from cassette or disc. If the end of file marker character exists in the file, it will be sent regardless, followed by the rest of the file. Error messages are sent to the screen if a filing system error occurs.

# Commands for 'intelligent' file transfer

## |BLOW

|BLOW,‹filename›

COMMAND: Sends an ASCII file, from cassette or disc using a special transfer protocol which ensures synchronisation with the receiving program and provides error-detection. The error message 'transfer aborted' is sent to the screen if the receiving program reports an unrecoverable error.

## |SUCK

|SUCK,‹filename›

COMMAND: Receives an ASCII file and writes it to cassette or disc using a special transfer protocol which ensures synchronisation with the sending program and provides error-detection. An appropriate error message is sent to the screen if the receiving program detects an unrecoverable error.

Remember that any CP/M file is treated as ASCII and can therefore be transferred using these commands. BASIC and other program languages should be saved in ASCII before being transferred.

The protocol for |BLOW and |SUCK is as follows:

Transmitting (|BLOW):

(1) Send STX, listen for ACK.

(2) Send 16 byte filename, 2 byte block number, 1 byte block length (0 to 128), data (0 to 128 bytes), 2 byte checksum (sum of all data bytes). Zero block length means end-of-file.

(3) Listen for ETX, ACK, or NAK.

(4) If ETX then abort, else if NAK, goto (2) retrying same block; else if ACK goto (2) sending next block, or finish if last block.

Receiving (`|SUCK`):

(1) Listen for STX, respond with ACK. NOTE - This means that the receiving program should be started before the transmitting program, otherwise the initial STX might be missed.

(2) Receive filename, block number, block length, data, checksum.

(3) Check for same filename as block 1, and consecutive block numbering, if error then send ETX and abort.

(4) Check for hardware error or checksum error; if OK send ACK, else send NAK.

(5) Check block length; if zero, finish, else goto (2).

(For your reference):

STX = **[CTRL]B** = ASCII 2
ETX = **[CTRL]C** = ASCII 3
ACK = **[CTRL]F** = ASCII 6
NAK = **[CTRL]U** = ASCII 21

# Commands for re-directing printer output

## |SERIAL

`|SERIAL`

COMMAND: Sends all stream #8 output to the RS232C instead of to the Centronics parallel printer port.

The RS232C, by default, will respond to a standard XON/XOFF software handshake in addition to any hardware handshaking. This means that if the device connected to the RS232C sends an XOFF character to the RS232C, then the RS232C will cease transmitting until an XON character is received.

(For your reference):

XON = **[CTRL]Q** = ASCII 17
XOFF = **[CTRL]S** = ASCII 19

## |PARALLEL

```
|PARALLEL
```

COMMAND: Restores stream #8 output to the Centronics parallel printer port.

## |NOXON

```
|NOXON
```

COMMAND: Cancels the XON/XOFF software handshake; the RS232C ignores any characters received.

## |XON

```
|XON
```

COMMAND: Restores the XON/XOFF handshake.

# Commands for VDU emulation

## |TERMINAL

```
|TERMINAL
```

COMMAND: Connects the computer keyboard to the RS232C output, and connects the RS232C input to the screen. The VDU emulation mode is terminated by pressing a key set to ASCII &FC. Normally, this will be returned by **[CRTL] [ESC]** (or just **[ESC]** unless the more standard value of decimal 27 has been assigned to it).

By default, there is no local 'echo' of what is typed. This echo will normally be provided by the equipment to which the RS232C is connected.

The keyboard codes should be set up using the `KEY DEF` command if it is required to transmit codes other than those described in your computer's user instruction manual.

By default, the screen will obey any control codes sent to it in the way described in your computer's user instruction manual.

## |HALFDUPLEX

```
|HALFDUPLEX
```

COMMAND: Makes a local connection between the keyboard and the screen in VDU emulation mode, so that you can see what is being typed (if the equipment connected to the RS232C is not echoing the keyboard).

## |FULLDUPLEX

```
|FULLDUPLEX
```

COMMAND: Cuts the keyboard to screen connection which is made by |HALFDUPLEX.

## |CTRLDISPLAY

```
|CTRLDISPLAY
```

COMMAND: Causes the screen to display (rather than obey) control codes it receives whilst in VDU emulation mode. This is often useful for diagnostic purposes. You could use the program below to help identify the various displayed control code symbols.

```
10 MODE 1
20 FOR N=0 TO 30 STEP 3
30  FOR M=0 TO 2
40   IF N+M<32 THEN PRINT CHR$(1);CHR$(N+M);" = [CTRL]";
     CHR$(64+N+M),
50  NEXT M
60  PRINT
70 NEXT N
```

## |CTRLACTION

```
|CTRLACTION
```

COMMAND: Restores the default setting of obeying (rather than displaying) control codes in the VDU emulation mode.

# Commands for PRESTEL emulation

## |PRESTEL

|PRESTEL[,‹character›]

COMMAND: Invokes a PRESTEL emulator. This emulator displays 40 columns by 24 lines in 16 colours (8 steady, 8 flashing) using special characters and graphics in Mode 0. The emulator responds to all the standard codes, including double height, graphics hold, and cursor positioning. It is beyond the scope of this manual to provide a complete description of the full range of PRESTEL facilities.

It is important to remember to set up the correct baud rate, data, and framing bits together with the screen mode, before logging-on to the PRESTEL service. This will normally be 75 TX, 1200 RX, 7 data bits, odd parity, 1 stop bit, and screen mode 0.

i.e. `|SETSIO,75,1200,1,7,1,0: MODE 0`

If the optional ‹character› parameter is specified, then that single character is sent to the PRESTEL emulator, and control returns to BASIC (it is then necessary to issue a |REFRESH,‹line number› command in order to update the screen after being sent the new character).

If the ‹character› parameter is not specified, characters are read continuously from the RS232C until a key returning ASCII &FC is pressed on the keyboard. Normally, **[CRTL] [ESC]** can be relied upon to return this value.

In the continuous mode, the PRESTEL emulator receives characters and simultaneously writes them into an internal buffer. Periodically it refreshes the screen display from that buffer, so that the screen appears to be updated in a series of 'bursts'. A cursor is displayed at the current PRESTEL cursor position, and any keystrokes are transmitted immediately.

In the continuous mode, the decimal point key on the numeric keypad is *temporarily* set up to be the special * character and the **[ENTER]** key adjacent to the numeric keypad is *temporarily* set up to be the PRESTEL 'enter' character _

In the continuous mode, the cursor keys are *temporarily* set up to send appropriate codes for screen editors operated via the PRESTEL service.

## |SAVEPRESTEL

|SAVEPRESTEL,‹filename›

COMMAND: Stores the contents of the PRESTEL emulator's internal buffer to cassette or disc.

## |LOADPRESTEL

|LOADPRESTEL,‹filename›

COMMAND: Loads the PRESTEL emulator's internal buffer from a cassette or disc file. It is necessary to issue a |REFRESH command in order to update the screen from the newly loaded buffer.

## |CURSOR

|CURSOR,‹column number›,‹row number›

COMMAND: Overrides the current cursor position as calculated by the PRESTEL emulator, where ‹column number› is an integer in the range (1 to 40), and ‹row number› is an integer in the range (1 to 24).

## |REFRESH

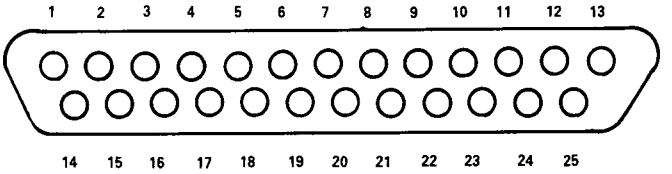
|REFRESH[,‹line number›]

COMMAND: Updates the screen from the PRESTEL emulator's internal buffer. Either the whole screen, or optionally one line (in the range 1 to 24) can be updated. The screen colours are set up to be suitable for PRESTEL.

Because of the way in which character attributes of colour, height, etc., are set up from left to right on a line-by-line basis, it is not appropriate to refresh the screen in elements of less than one whole line.

# Appendix 4: Hardware

The complete pin-out of the RS232C is as follows:




| | |
|---|---|
| PIN 1 NOT USED | PIN 14 NOT USED |
| PIN 2 DATA OUT (TXD) | PIN 15 NOT USED |
| PIN 3 DATA IN (RXD) | PIN 16 NOT USED |
| PIN 4 RTS OUT | PIN 17 NOT USED |
| PIN 5 CTS IN | PIN 18 NOT USED |
| PIN 6 NOT USED | PIN 19 NOT USED |
| PIN 7 GND | PIN 20 DTR OUT |
| PIN 8 DCD IN | PIN 21 NOT USED |
| PIN 9 NOT USED | PIN 22 RING IND.IN |
| PIN 10 NOT USED | PIN 23 NOT USED |
| PIN 11 NOT USED | PIN 24 NOT USED |
| PIN 12 NOT USED | PIN 25 NOT USED |
| PIN 13 NOT USED | |

The RS232C implements Channel A of the AMSOFT recommended serial interface specification. The I/O ports used are as follows:

| Address | Output | Input |
|---|---|---|
| &FADC | DART data | DART data |
| &FADD | DART control | DART control |
| &FADE | *DO NOT USE* | *DO NOT USE* |
| &FADF | *DO NOT USE* | *DO NOT USE* |
| &FBDC | 8253 load counter 0 | 8253 read counter 0 |
| &FBDD | 8253 load counter 1 | 8253 read counter 1 |
| &FBDE | *DO NOT USE* | *DO NOT USE* |
| &FBDF | 8253 write mode word | *NOT USED* |

**NOTE** - The software ROM is set to be ROM number 6.

# AMSTRAD RS232C USER INSTRUCTION MANUAL

## ADDENDUM TO PAGE 28

### HEX DUMP FILE FOR CP/M 'BLOW' ROUTINE

Please note that the HEX dump as printed on page 28 of your RS232C user instruction manual can NOT be used for the transfer of files greater than 16K in length.

If you *do* wish to transfer such files, use the following alternative HEX dump, entering it by the method described in the manual.

Note that this alternative routine can be used for transferring files both greater *and* smaller than 16K.

```
:180100003A5D00FE20CA0502115C000E0FCD0500FEFFCA0E02CD26013A
:18011800E10CD050011A1020E09CD0500C721FFFF22B8021E020E044E
:18013000CD0500CD8101D22C012AB8022322B802CD9C01CDB001CDD02F
:180148001B7C26A01CDB801CDC001CDC801CD810138DECD9C01CDB0C4
:18016000C34D01CD9C01CDB0011E000E04CD0500210000022BA02CDBF
:18017800C801CD8101D26401C90E03CD0500FE03C29601F1F1117D02A8
:18019000E09CD0500C9FE0637C83FC9215C007EF6405F0E04E5CD0541
:1801A80000E123060BC3170221B8020602C3F5011E800E04CD0500C967
:1801C0002180000680C3F50121BA020602C3F501210000022BA020E1488
:1801D800115C00CD0500218000680E516005E2ABA021922BA02E1236F
:1801F00005C2E301C9E5C55E0E04CD0500C1E1230 5C2F501C9112A020F
:18020800E09CD0500C7115502E09CD0500C7CDF50106041E000E0419
:18022000C5CD0500C105C21C02C90A074E6F2066696C652073706563 67
:1802380069666965642E0D0A0A5472616E736657220616 26F7274657C
:18025000640D0A0A240A0746696C65206E6F7420666F756E642E0D0A6A
:18026800 0A5472616E736657220616 26F72746564 0D0A0A240A075484
:18028000 72616E736657220416 26F72746564206279206F7468657257
:18029800 20656E642E0D0A0A240A5472616E736657220636F6D706CFA
:0C02B000 6574652E0D0A0A240000000091
:00000000 00
```