APPENDIX 2

RS-232-C PORT ERROR CONDITIONS, CAUSES AND CORRECTIONS

This section contains descriptions of the errors you might encounter while using the 850 Interface Module. Many of these errors also occur with other ATARI peripherals; they are listed here so you can see what they mean when using the Interface Module.

There are a number of NEW errors which you can get from the Interface Module which no other peripheral produce. These new error codes are underlined.

"ERROR" 1 -- Success. This is the status which successful completion of an I/O operation produces. BASIC does not report this to you except by continuing in normal fashion.

ERROR 128 -- Break abort. This means you pressed the BREAK key while I/O was proceeding.

ERROR 129 -- IOCB already OPEN. Your choice of channel number (#n) was that of a channel (IOCB) which was already OPEN. This can happen if you restart a program in a manner other than RUN (RUN closes files). Be careful not to put your OPEN statement inside a programmed loop. The second time OPEN is encountered it will produce ERROR 129.

ERROR 130 -- Nonexistent device. You specified something other than R:, R1:, R2:, R3:, or R4: . Perhaps you were trying to access a file on disk whose name starts with "R" and forgot the D: . THIS ERROR WILL OCCUR IF YOU ATTEMPT TO USE AN RS-232-C PORT AND THE RS232 HANDLER HAS NOT BEEN "BOOTED" WHEN THE SYSTEM WAS TURNED ON. In that case, you should save your program and start a new session, allowing the RS-232-C handler to boot. See the section on automatic bootstrap.

ERROR 131 -- Write only. You tried to read (GET, INPUT) from a port you OPENed as write only.

ERROR 132 -- Invalid command. You specified something incorrectly in an XIO command to the Interface Module.

ERROR 133 -- Channel not OPEN. You neglected to OPEN the channel (IOCB) to the I/O device you are trying to access.

ERROR 135 -- Read only. You tried to write (PUT, PRINT) to a port you opened for read access only.

ERROR 138 -- Device timeout. The Interface Module did not respond to a command. Check the cables. Make sure the Interface Module is powered on.

ERROR 139 -- NAK. The Interface Module refused to perform some command. You may issue a STATUS request to find out what was wrong. Most common causes are: attempts to perform 5-, 6-, or 7-bit input at too high a Baud rate; automatic readiness checking was enabled and the connected device was not ready.

ERROR 150 -- Port already OPEN. You attempted to OPEN an RS-232-C port but is was already OPEN through another channel (IOCB). You can access an RS-232-C port through only one channel at a time.

ERROR 151 -- Concurrent mode I/O not enabled. You attempted to start concurrent mode I/O (XIO 40) but the port was not opened with an odd number specified for Aux1 (Aux1 bit 0 not set).

ERROR 152 -- Illegal User-supplied Buffer. In the START CONCURRENT MODE I/O command with the user-supplied buffer, the buffer address and/or the buffer length were inconsistent.

ERROR 153 -- Active Concurrent Mode I/O Error. You attempted to perform I/O to an RS-232-C port while Concurrent Mode I/O was active to some other RS-232-C port. Only input, output, CLOSE and STATUS commands to the active concurrent mode port are allowed while concurrent mode I/O is active. This error message is not always produced -- attempts to do disallowed I/O while concurrent mode I/O is active may result in the computer "crashing".

ERROR 154 -- Concurrent mode I/O not active. Concurrent mode I/O must be activated in order to perform input (GET, INPUT).

APPENDIX 3

PRINTER PORT ERROR CONDITIONS, CAUSES AND CORRECTIONS

This section describes error conditions which could occur when using the printer port. There are no new error codes associated with the Interface Module's printer port; however, the meaning of some of the errors is slightly different between the Interface Module and other ATARI printers.

If an error occurs which is not listed here, consult the BASIC reference manual. Errors are listed here only if they have some new meaning when reported by the Interface Module.

ERROR 138 -- Timeout. The Interface Module did not respond to a request from the Computer. Check the cables. Make sure the Interface Module and attached printer are powered on. The Interface Module will NOT respond to printer control commands from the computer if the FAULT wire to the printer is low (caused by loose cable or printer off).

ERROR 139 -- NAK. The Interface Module refused an illegal printer command. Make sure that Aux1 and Aux2 are specified as zero (0) in your OPEN command for the printer. This error also occurs when the printer appears active (FAULT line is high) but the printer fails to respond to characters sent to it within four seconds. Check the switches on the printer (online?). If the printer is not performing within 4 seconds, change your PRINT statements to break down transmissions into smaller chunks.

NOTE: Attempts to operate more than one printer at a time will result in unpredictable operation. While one printer may "win" most of the time, errors are always possible, and exactly which error occurs is a matter of chance. If you have more than one ATARI printer attached to your computer, turn on only one of them at a time.

APPENDIX 4

MEANING OF (ERROR) BITS IN LOCATION 746

TABLE I--Decimal Representation of the Error Bits in Location 746

Decimal Equivalent	Error
128	Received data framing error
64	Received data byte overrun error
32	Received data parity error
16	Received data buffer overflow error
8	Illegal option combination attempted
4	External device not fully ready flag
2	Error on block data transfer out
1	Error on command to Interface Module

Below are the descriptions of these error status bits in location 746 after STATUS command.

RECEIVED DATA FRAMING ERROR (bit 7, decimal value 128) This error bit indicates a framing error was encountered in the data coming from the external RS-232-C-compatible device: the 10th bit of some character was not a STOP bit (9th, 8th or 7th in the cases of 7-, 6-, or 5-bit received sords). This error can be caused either by garbled data (e.g., noise on the phone lines) or by improper configuration to receive the data (e.g., wrong Baud rate). This condition is monitored in one of two places: in the 400/800 computer, or in the Interface Module. The computer watches for this error in the case of 8-bit data. The Interface Module catches this error if you are receiving 7-, 6-, or 5-bit data. In both cases, the error status is set at the time the erroneous character is received (not the time you read it out of the holding buffer). In the 8-bit data case, where the computer monitors the error, you may find out about the error any time after it occurs by issuing STATUS while the concurrent mode input is active. The error bit will be cleared when you issue the STATUS command. This error bit will be cleared also when you CLOSE the concurrent mode channel. In the 7-, 6-, and 5-bit cases, the error is monitored by the Interface Module and cannot be interrogated while the concurrent mode input operation is active. In this case, you must CLOSE and re-OPEN the concurrent mode channel and then issue STATUS to determine if the error occurred. The error bit in the Interface Module is cleared by STATUS when concurrent mode I/O is not active; it is also cleared by most of the configuring and control XIO's (but not all), and [it may be cleared] by CLOSE when concurrent mode I/O is not active.

In general, the error bits read from location 746 after a STATUS request apply only to the most recent I/O operation to the RS-232-C port; that is, they are cleared as the I/O operation is started and

then set if the error occurred. You can see that the previous error is an exception to this rule. Other exceptions will be noted.

RECEIVED DATA BYTE OVERRUN ERROR (bit 6, decimal value 64) This error bit is maintained by the computer and indicates that the computer got too busy to read all the data as it was arriving (due to overly heavy interrupt loading, or perhaps interrupts being masked off totally). This error is flagged when the first character of data following the error is read from the port and placed in the holding buffer. The error should not occur at all under normal conditions.

RECEIVED DATA PARITY ERROR (bit 5, decimal value 32) This error bit is maintained by the computer and indicates that a received character had the wrong parity. The bit will not be set if no parity checking has been enabled. This error occurs during the translation from the external (received) form of the character to the internal (INPUT, GET) form. The error flag bit is cleared by the STATUS command.

RECEIVED DATA BUFFER OVERFLOW ERROR (bit 4, decimal value 16) This error flag indicates that more data has arrived than can be held in the input buffer--data has not been read from the buffer (INPUT, GET) soon enough. This error is maintained by the computer, and it occurs when the overflowing character arrives from the RS-232-C compatible device. The new character replaces the oldest one in the buffer. This error bit is cleared by the STATUS command.

ILLEGAL OPTION COMBINATION ATTEMPTED (bit 3, decimal value 8) This error flag is kept in the Interface Module and may be read by STATUS only if concurrent mode I/O is not active. It is set by an attempt to start concurrent mode input with short words (7-, 6-, or 5-bit) with the port open for both input and output (short words are allowed one direction at a time only) or too high a Baud rate (short words are allowed for input at a maximum rate of 300 Baud). This error may be checked immediately after the Interface Module produces a NAK for the refused command. The bit is cleared by the STATUS request. Error bit (command error, decimal value 1) will always be set when this bit is set.

EXTERNAL DEVICE NOT FULLY READY (bit 2, decimal value 4) This bit is kept in the Interface Module and may be read by STATUS only when concurrent mode I/O is not active. It is set whenever a START CONCURRENT MODE I/O or block output command is refused by the Interface Module because one or more of the external status lines being monitored is not ON. Any of the external status lines not being monitored (as set by the SET BAUD RATE command) is ignored, and if none is being monitored this bit will not be set and the I/O operation will proceed normally. Read this flag bit with a STATUS request immediately after the Interface Module refuses the operation with NAK. This flag is cleared by the STATUS command.

DATA BLOCK ERROR (bit 1, decimal value 2) This error bit is maintained in the Interface Module and may be read by STATUS immediately after a command is refused by NAK. In a block output, the data block was unsuccessfully received from the computer by the Interface Module.

This error should not occur in normal operation; it indicates problems in communication between the computer and Interface Module.

COMMAND ERROR TO INTERFACE MODULE (bit 0, decimal value 1) This error bit is maintained in the Interface Module and may be read by STATUS immediately after a command is refused by a NAK from the Interface Module. This bit indicates that the Interface Module did not recognize a command sent to it from the computer, or that the Interface Module is not configured properly to perform the command (see ILLEGAL OPTION COMBINATION ERROR).

During active concurrent mode I/O, the STATUS command will return the number of characters in the input buffer in locations 747 and 748, and the number of characters in the output buffer in location 749. To find the number of characters in the input buffer in BASIC:

```
LET BUFFERUSE = PEEK(747) + 256*PEEK(748)
```

If you want to find out only whether or not any characters are in the input buffer, you do not need to multiply by 256:

```
IF PEEK(747)+PEEK(748)=0 THEN input buffer empty...
```

or:

```
IF PEEK(747)+PEEK(748) <> 0 THEN input buffer not empty...
```

If you are using the built-in buffer, or if your supplied buffer has fewer than 256 bytes, then location 748 will always be zero and you need to look only at location 747. The output buffer holds only 32 characters; location 749 will never exceed 32.

When concurrent mode I/O is not active, location 747 will contain information about the monitored readiness lines (DSR, CTS and CRX) and the data receive line (RCV) of the specified port after a STATUS request. Locations 748 and 749 will not contain anything useful after a STATUS request when there is no active concurrent I/O.

Location 747 will contain the sum of four numbers, shown in table II. The current and past status of DSR, CTS, and CRX as well as the current status of RCV are included. The past status of DSR, CTS and CRX applies back to the time the Interface Module was booted, or to the most recent STATUS command to the specified port which was made while concurrent mode I/O was not active (i.e., the last time that DSR, CTS and CRX were supplied to a STATUS request). No other operations affect the past status of these lines.

Ports 2 and 3 will always show CTS and CRX as being ON. Port 4 will show CTS, CRX, and DSR as being ON.

A quick way to check whether or not a port is ready is this:

```
STATUS #n, XXX
IF PEEK(747) < 128 THEN not ready ...
```

or to check if it has stayed ready since the last check:

```
IF PEEK(747) >= 192 THEN always ready ...
```

In other words, the DSR status bits are the most significant bits in the sense byte, and you can check them this way without having to worry about the states of the other bits in the byte.

TABLE II--SENSE VALUES ADDED INTO LOCATION 747

DATA SET READY (DSR)

192	Ready now (ON); on since previous STATUS
128	Ready now (ON); not always on since last STATUS
64	Not ready now (OFF); not always off since last STATUS
0	Not ready now (OFF); always off since last STATUS

CLEAR TO SEND (CTS)

48	Clear now (ON); on since previous STATUS
32	Clear now (ON); not always on since last STATUS
16	Not clear now (OFF); not always off since last STATUS
0	Not clear now (OFF); always off since last STATUS

CARRIER DETECT (CRX)

12	Carrier now (ON); on since previous STATUS
8	Carrier now (ON); not always on since last STATUS
4	No carrier now (OFF); not always off since last STATUS
0	No carrier now (OFF); always off since last STATUS

DATA RECEIVE (RCV) *

1	MARK (1) now
0	SPACE (0) now

* No information is supplied about the past status of RCV.

APPENDIX 5

SETTING THE BAUD, WORD SIZE, STOP BITS, AND READY MONITORING

The CONFIGURE BAUD RATE command allows you to set the Baud rate, "word" size, number of stop bits to transmit, and enable or disable checking of DSR, CTS and CRX. The command may be issued through an open channel to the RS-232-C port, or may be issued through a channel which isn't being used. If you have opened a channel to the port you are configuring, you must use that channel. You cannot configure any port if a concurrent mode I/O operation is active.

The CONFIGURE BAUD RATE command looks like this in BASIC:

```
XIO 36, #channel, Aux1, Aux2, "Rn:"
```

The 36 makes this a CONFIGURE BAUD RATE command.

Channel is the number of the channel that BASIC should use to execute the command. The channel should either be open to the port you are configuring, or should not be open at all. No concurrent mode I/O should be active when you issue this command.

Aux1 is a number or expression that specifies the Baud rate, "word" size, and number of stop bits to send with each "word." For each of these, pick a number from tables I, II, and III, and then add the numbers together to form Aux1. You may add them together yourself or you can let BASIC add them for you. For example: XIO 36, #1, 128+0+10, 0, "R:" and XIO 36, 138, 0, "R:" both specify the same thing.

Aux2 is a number or expression that specifies whether or not the Interface Module should check Data Set Ready (DSR), Clear to Send (CTS), and/or Carrier Detect (CRX) when a block mode output or START CONCURRENT MODE I/O operation is performed. If you ask to have the Interface Module check one or more of these, then the Interface Module will return error status if the line(s) checked is not ON. You may TRAP the error and program BASIC to take the action you desire. See table IV for values of Aux2.

The last XIO parameter, "Rn:", specifies which serial port of the Interface Module you are configuring. For n put 1, 2, 3, or 4, just as you would in the OPEN command.

{text continues after tables...}

TABLE I: BAUD RATE SPECIFIERS TO ADD TO AUX1

ADD	BAUD RATE
0	300 bps (bits per second)
1	45.5 bps*
2	50 bps*
3	56.875 bps*
4	75 bps**
5	110 bps
6	134.5 bps***
7	150 bps
8	300 bps
9	600 bps
10	1200 bps
11	1800 bps
12	2400 bps
13	4800 bps
14	9600 bps
15	9600 bps

* These Baud rates are useful for communications with Baudot teletypes, for RTTY (radioteletype) applications. They are more commonly referred to as 60, 67, and 75 words per minute.
** This Baud rate is sometimes used for ASCII communications, and may also be used for 5-bit Baudot RTTY. The latter is commonly referred to as 100 wpm.
*** This Baud rate is used by IBM systems.

TABLE II: WORD SIZE SPECIFIERS TO ADD TO AUX1

ADD	WORD SIZE
0	8 bits
16	7 bits
32	6 bits
48	5 bits

See text for discussion of word sizes.

TABLE III: SPECIFIER FOR 2 STOP BITS TO ADD TO AUX1

ADD	STOP BITS SEND WITH EACH WORD
0	1
128	2

TABLE IV: AUX2 SPECIFICATION TO MONITOR DSR, CTS, CRX

ADD	TO MONITOR
0	None
1	CRX
2	CTS
3	CTS, CRX
4	DSR
5	DSR, CRX
6	DSR, CTS
7	DSR, CTS, CRX

{...continuation of text}

Note that the default (pre-set) values of Aux1 and Aux2 for all four ports are zero, corresponding to 300 Baud, 8-bit words, one stop bit transmitted, and no checking of DSR, CTS or CRX.

You should know the following things about this command:

The configured parameters will stay as you set them until you either reset them or until you reboot the system (turn the power off and back on). The SYSTEM RESET key will NOT reset any of these parameters. You may configure each RS-232-C port independently.

If you specify 8-bit words, there are no restrictions on operationn of the port. However, the following restrictions apply to 7-, 6-, and 5-bit words: half-duplex operation only; some limitations on baud rates. Specifically, all output baud rates are allowed in Block Output mode. In Concurrent Mode, either in or out, you are limited to operation at 300 Baud and below. If you specify 7-, 6-, or 5-bit words, there is no restriction on the number of stop bits you may specify. Note that most applications of these word sizes will probably be to devices that require more than 1 stop bit--you should specify two.

If you specify 7-, 6-, or 5-bit words, each word sent or recived will be converted from or to an 8-bit byte within the computer by ignoring the most significant bit(s). This will very likely interact with the translation operation, and in particular there may be no way you can receive an EOL. If this is the case, you cannot use the BASIC INPUT statement to read the port and you must retrieve characters one at a time using GET. More details will be found in the section on translation modes. (APPENDIX 6)

If you specify that you want the Interface Module to check DSR, CTS and/or CRX, it will check them whenever you try to start concurrent mode I/O and whenever you try to send a block of data in block output mode. If any of the lines you asked to be checked is not ready (OFF), then the concurrent mode I/O will not be started or the block of data will not be sent. The Interface Module will then return an error to BASIC, and you may TRAP the error and take corrective action. Following the TRAP, you may perform a STATUS request from the Interface Module to find out what the problem was.

Note that CTX and CRX are not supported on ports 2, 3, and 4, and that DSR is not on port 4. The Interface Module behaves as if they were really there, however, and acts as if they were always ready (ON).

You may look at the states of DSR, CTS and CRX at any time that concurrent mode I/O is not active (if you have a channel open to the port) by issuing a STATUS request for the port. Thus, enabling this automatic checking of these lines is not the only option available to you, and you may prefer checking them directly with STATUS.

APPENDIX 6

TRANSLATION AND PARITY HANDLING

The Interface Module handler can be configured to perform certain types of code conversions (translations) and do parity generating and checking for you. These two operations interact with each other. For this reason, they will be described together in this section. The various options you may select for each are even specified by executing the same command--CONFIGURE TRANSLATION AND PARITY.

There are three factors you need to keep in mind when you are setting up your code translations. Translation, of course, is one of them, since it results in (possibly) changing one code into another. Parity generation and checking also may result in changing one code into another. The third factor you need to keep in mind is the the word size you are transmitting/receiving. Inside the computer, all words are the same as bytes; that is, all words are 8 bits. If you are sending/receiving 7-, 6-, or 5-bit words, these shorter words have to come from 8-bit computer words by chopping out some bits, or expanded into 8-bit computer words by adding some bits. These operations obviously are the same as changing one code into another.

Each of these three possible code changes takes place separately from the others, one at a time. For output, translation comes first, followed by parity generation, and finally truncation (shortening by leaving bits off). Of course, at each stage a change may not occur, depending on what selection of options you have configured and depending on which character (code) the computer is sending. For example, if you have configured 8-bit words, the trunction operation does nothing. For input, the order of code changing is expansion (from short words to 8-bit words), followed by parity checking, and finally translation.

At each of the three stages, a code change may occur. If a change DOES occur, then it is the CHANGED code which will be operated on in the next stage. For example, (in a particular configuration of translation and parity options) if you output an ATASCII EOL, it would first be translated to an ASCII CR and then parity would be generated for the CR. This is because the parity step operates on the result of the translation step, in this case the CR.

There is one other translation option which is very specific, namely, the option to have an ASCII LF (line feed) sent after each transmitted CR (Carriage Return). This code change occurs at the translation step. Consequently, the generated LF will go through the parity and truncation (small word) phases just like the CR.

TYPES OF CODE TRANSLATION

You have three options to choose from: no translation at all, "light" translation, or "heavy" translation. Whichever option you choose will

apply both to incoming and outgoing characters. The "no translation" option is just what it says--no change is made to the characters, whether being received or sent. This statement applies only to the translation step, of course--you can still get changes from parity and small words. The no-translate option is useful if you are going to do your own special processing on the characters you are sending and receiving. This can be particularly useful in the small-word situations, since many of the cases where small words are used do not (or cannot) involve ASCII. You may also want to use the no-translation option if the RS-232-C-compatible device you are communicating with understands ATASCII.

No matter which translation option you choose, if you use a BASIC INPUT statement to read data the data must have an ATASCII EOL (End of Line) character at the end of each line. This requirement applies AFTER all translation. Thus, if you select the no-translation option, your incoming data must either contain EOL's or you should use GET instead of INPUT. Remember, also, that using short words and that checking parity also affect data coming in, so you may still need to use GET.

Heavy and light translation are two ways to convert between ASCII and ATASCII. In either translation mode, the ATASCII EOL (9B in hexadecimal, 155 in decimal) is converted to and from the ASCII CR (0D in hex, 13 in decimal). In the case of output, EOL is changed to CR; if you also selected the Append LF option, EOL is changed to CR followed by LF, that is, the translation function produces two characters out for one in. On input, a CR will be translated to EOL. Both Heavy and Light translation modes assume ASCII in the outside world and they assume ATASCII in the computer. ASCII is treated as a 7-bit code; that is, the 8th (most significant) bit is always treated as if it is zero. On input, then, if you select Heavy or Light translation, the 8th bit of each word is cleared to zero. On output, the translation step will set this bit to zero.

Light translation performs the fewest changes between ASCII and ATASCII. The assumption is that you wish to work with ATASCII within the computer but treat it as if it were really ASCII. Note, for example, that the ATASCII graphics codes are the same (numerically, and for the most part the way you type them, too) as the ASCII control codes (1-26). So for input, the character has its high bit stripped (set to zero), and that's all--except if the code is found to be a CR, it is changed to an EOL. For output, if the character being sent is EOL it is changed to CR; then, no matter what the character is, the high bit is set to zero. Light translation is the pre-set default mode.

Heavy translation is a more thorough translation mode. Here the assumption is that if there is no direct correspondence between the character in ASCII and ATASCII, then the code should not be translated. So for input, after the high bit is cleared to zero, if the character is CR it is changed to EOL; otherwise, the character is checked to see if it is the same in ATASCII as in ASCII. If it is not, it is translated to the WON'T TRANSLATE character. Specifically,

if the code for the ASCII character is less than 32 decimal (i.e., the character is a control character) or greater than 124 decimal (7C hex) it will be translated to the "won't translate" character. Thus, heavily translated ASCII corresponds to the printable characters from blank through vertical bar. The "won't translate" character is specified by you in the CONFIGURE TRANSLATION command. If you do not specify it, the pre-set default value for it is zero (ATASCII graphic heart).

On output, heavy translation converts EOL to CR, and will output any character whose ASCII meaning is the same as it is in ATASCII. That is, characters whose values range from 0-31 decimal (ASCII control values) or whose values are above 124 decimal (7C hex) will not be sent. Note that characters whose high bit is one will be translated to nothing, that is, characters which would show on the TV screen as INVERSE VIDEO WILL NOT BE SENT in heavy translation mode. Note also the difference between input and output in the heavy trans- lation mode: untranslatable characters in the input are converted to the "won't translate" value, where untranslatable output simply is not sent out.

The (optional) sending of LF after CR is produced in the translation step. If you specify no translation, the option of adding LF to CR is not available. If you specify light translation, LF will follow EOL (which of course becomes CR). Note that sending the 13 decimal code (CR) by itself EOL will be turned into a CR/LF pair (with the append LF feature turned on). Each character in the CR/LF pair is independently sent through the parity and word-shortening steps on its way out. The pre-set default setting of the append LF feature is OFF, that is, the default is to NOT append the LF.

PARITY

You may select input and output parity handling separately. Thus, you may choose to send, for example, even parity while you ignore the parity of what you are receiving. The parity is always the most significant bit of each 8-bit byte (bit number 7). Thus, this parity operation is not applicable if you are working with 7-, 6-, or 5-bit words.

In the default parity condition, the parity bit of neither input nor output is altered. Note, however, that the parity bit of out- going messages may have been changed during the translation step.

For output, you may select even parity, odd parity, set parity bit or no parity.

For input, your choices are "don't touch", check even, check odd, and "don't check". Each of these last three options will clear the top bit to zero, whether or not a parity check is made. If an input parity error is found, the character will still be input as if it were all

right; the parity error flag will be turned on in the status bytes (see STATUS REQUEST).

SHORT-WORD CONVERSION

The third operation which affects your code translation is the short-word conversion (if you are using 8-bit words, this is a "no-effect" operation). Short words sent out are made from 8-bit computer characters by omitting the most significant bits. That is, a 7-bit word is bits 0-6 of the character, a 6-bit word is bits 0-5, and a five-bit word is bits 0-4. Thus the parity, if generated, is lost. ASCII is a 7-bit code; you can send ASCII in 7-bit form without parity (this is not common practice, though--usually 8 bits are sent even if the 8th bit is not used for parity). With 6-bit and 5-bit codes, you will not be using ASCII, so you will have to concern yourself with the codes you want to be sending. With these word sizes, you should turn translation off so the translation performed by the Interface Module handler will not affect the codes you are working with.

On input, small words are converted to 8-bit computer characters by adding high-order bits. These added bits are always set to 1. Thus, if you are receiving 7-bit ASCII, the parity and translation steps will be getting ASCII with the 8th bit set high. If you are receiving 6- or 5-bit codes, there is no way you can receive the 13 decimal (OD hex) code (ASCII CR)--after all, you cannot receive ASCII in 6 or 5 bits anyway. This means that in BASIC you will have to use the GET statement, not INPUT. Of course, you will be doing your own code conversion, so you should turn off the conversions of the Interface Module handler.

The CONFIGURE TRANSLATION MODE command is specified in BASIC this way:

```
XIO 38, #channel, Aux1, Aux2, "Rn:"
```

38 specifies the CONFIGURE TRANSLATION MODE command.

#Channel specifies the channel numoer (IOCB number from 1 to 7) you wish to use to configure the translation mode. You may use an unopen channel if you have no channel open to the port you are configuring; otherwise you must use the channel you have opened to that port. You cannot issue the CONFIGURE TRANSLATION MODE command if any concurrent mode I/O is active.

Aux1 specifies the translation mode, the input parity mode, the output parity mode, and the Append LF option. You specify these options by adding numbers taken from tables I, II, III and IV. You may add the numbers yourself and put the sum in your program for Aux1, or you may let BASIC add them for you (e.g., you can say either 2+8+32 or 42 to

mean even parity in, even parity out, and no translation). Do not add in more than one value from each table.

Aux2 is the numeric representation of the "won't translate" character for heavy translation. Remember that the BASIC function ASC will give you the numeric representation of a character. For example, 41 and ASC("A") mean the same number. The number you specify should be from 0 through 255.

"Rn:" specifies the port you are configuring. For n, you put 1, 2, 3, or 4. You may omit n, which will mean you are configuring port 1.

The default configuration is Aux1=0 and Aux2=0. If you execute the CONFIGURE TRANSLATION MODE command for one of the RS-232-C ports, that configuration will remain in effect until you do another CONFIGURE TRANSLATION MODE for that port. SYSTEM RESET will not change the translation mode for any port. Of course, you can configure each port a different way.

TABLE I--TRANSLATION MODE OPTIONS ADDED TO AUX1

Add	To Get
0	Light ATASCII/ASCII translation
16	Heavy ATASCII/ASCII translation
32	No translation

TABLE II--INPUT PARITY MODE OPTIONS ADDED TO AUX1

Add	To get
0	Ignore and do not change parity bit
4	Check for odd parity, clear parity bit
8	Check for even parity, clear parity bit
12	Do not check parity but clear parity bit

TABLE III--OUTPUT PARITY MODE OPTIONS ADDED TO AUX1

Add	To get
0	Do not change parity bit
1	Set output parity odd
2	Set output parity even
3	Set parity bit to 1

TABLE IV--APPEND LINE FEED OPTIONS ADDED TO AUX1

Add	To get
0	Do not append LF
64	Append LF after CR (translated from EOL)

APPENDIX 7

CONTROLLING THE OUTGOING LINES--DTR, RTS & XMT

There are up to three outgoing RS-232-C signals on each of the RS232 ports of the Interface Module: Data Terminal Ready (DTR), Request to Send (RTS), and Data Transmit (XMT). Each of these lines can be turned ON or OFF with the CONTROL command.

Port 1 supports all three outputs. Ports 2 and 3 have DTR and XMT; port 4 has only XMT. You may use this command the same way with any port--it is not an error to try to control a line that does not exist. Your attempt will simply have no effect.

You may control any or all of these lines on a single RS-232-C port with the CONTROL command (controlling lines on other ports requires one CONTROL command for each port). The CONTROL command may be issued to a port which is not OPEN through an I/O channel by specifying any unopen channel number in the CONTROL command. If the port has been opened through a channel, you must use that channel in the CONTROL command. You may not issue a CONTROL command if any concurrent mode I/O is active.

Controlling XMT line has very limited use and few users will be concerned with it. In its normal state XMT is passive. If you change XMT you are likely to interfere with the normal transmission of data. In the serial communication world the only practical use of control of the XMT line is to send a BREAK signal. The BREAK is simply a period of holding the XMT line out of its normal resting state. Specifically, the normal resting state is called MARK, which corresponds to the binary "1" state. A BREAK is a period of the state called SPACE, which corresponds to binary "0". (Actually, since MARK and SPACE are the only legal states of any RS-232-C signal, all data consists of alternating MARKS and SPACES. What distinguishes BREAK from other uses of SPACE is that a BREAK is a SPACE which is a lot longer in duration than the time that a transmitted word would be. This is so because any transmitted word ALWAYS has one or more MARK bits in it--in particular, each word ends with one or more stop bits represented by MARK.) Thus to send a BREAK, first issue a CONTROL command to set the XMT line to SAPCE (0), then a little while later issue a control to set it back to MARK (1).

The uses of the other lines will depend on your application. For some guidelines, see APPENDIX 1.

The pre-set default state of the DTR and RTS lines is OFF. The pre-set default state of the XMT line is MARK. Once you change any of them with the CONTROL command, the new setting will remain until you either turn the computer off or issue another CONTROL command to change things. The SYSTEM RESET key has no effect on these lines.

The form of the CONTROL command in BASIC is:

```
XIO 34, #channel, Aux1, Aux2, "Rn:"
```

34 specifies the CONTROL command.

#channel specifies the IOCB or channel number (1-7) you wish to use for the command. If no channel is open to the RS-232-C port, specify an unused channel. If the port is open through a channel, use that channel.

Aux1 is the sum of three numbers chosen from tables I, II, and III to control DTR, RTS, and XMT. Choose only one number from each table. You may add the numbers together yourself and put the resulting sum in your program for Aux1, or you may put an expression for the sum and let BASIC do the arithmetic for you.

Aux2 is not used by this command; the best value to specify is zero.

"Rn:" specifies the RS-232-C port you are acting on. For n you put 1, 2, 3, or 4. If you omit n, the Interface Module handler will assume you mean port 1.

APPENDIX 8

STARTING CONCURRENT I/O MODE

Use the command START CONCURRENT I/O (XIO 40) to start concurrent I/O mode. This mode may be used for output and must be used for input or full duplex. The port must be open before you can start concurrent I/O. Once concurrent I/O is in effect no other I/O operations which use the computer I/O connector can be performed. I/O operation to another serial port, for example, can not be performed. I/O to the keyboard, the screen, the Editor and the controller jacks can still be performed.

The concurrent mode I/O operation may be terminated by SYSTEM RESET, BREAK, or by closing the port.

Operations which are allowed while concurrent mode I/O is active are input and output operations to the active port (GET, INPUT, PUT, PRINT), and STATUS commands to that port.

There are two different forms of the START CONCURRENT MODE I/O command. The main difference between them is that one specifies the use of a small input buffer built into the Interface Module handler (in the computer), and the other allows you to give your own buffer to the handler so it can be any size you wish. (NOTE: in Assembly Language these two options are really just different forms of the same command.)

The form of the START CONCURRENT MODE I/O command which allows you to specify your own I/O buffer has two disadvantages: the command is complicated to specify in this form, and the BASIC array you use as the buffer may be moved by the BASIC interpreter. Once created, BASIC arrays are NOT moved while a program is being run, but arrays are moved whenever you add or delete a BASIC statement, even in immediate mode. The handler for the Interface Module is told of the location of the buffer only when you start the comcurrent I/O; thus, if you allow BASIC to move the array, data will be inserted in unpredictable locations, possibly destroying even the BASIC program itself. Ongoing concurrent input could wind up in other arrays or variables, or even in your BASIC program! SO REMEMBER: IF A PROGRAM IS USING CONCURRENT MODE INPUT, ALWAYS MAKE SURE THE CONCURRENT MODE OPERATION IS STOPPED WHEN YOUR PROGRAM STOPS. This will be done for you if you stop by using BREAK key, SYSTEM RESET key, or end your program with END or letting the program stop by "running off the end."

STOP does not terminate the concurrent input, and neither will it be stopped if an ERROR happens. IN THESE CASES, THE WAY TO STOP THE CONCURRENT I/O IS TO PRESS THE BREAK KEY.

None of these problems occur if you use the buffer which is built into the Interface Module handler, since that buffer does not move! On the

other hand, that buffer is quite small (32 bytes) and this may not be adequate for all programs.

With a small input buffer you need to GET or INPUT the data from the buffer before the buffer fills up with data that you have not yet read. Of course, if in the long-range average you read the data out of the buffer more slowly than it is arriving, you will eventually lose data anyway. If this is the case, you will either have to put up with losing it (which is not all that bad in some cases), or you will have to figure out a way to slow down the device that is sending the data to you (such as setting a lower Baud rate). Even if your program processes the data fast enough in the long run, a small buffer puts demands on your program to get data quickly and often. Here are some things to consider.

The BASIC interpreter is quite slow relative to incoming data, if you want to do some processing on each and every character that comes in. In that case, 300 Baud would be fast. On the other hand, the system is more than fast enough to read in a line of data (terminated by CR) at 9600 Baud (960 cps)--as long as there is enough time between lines for your program to do its processing. It pays to read a whole line of input at a time (use INPUT wherever possible instead of GET), and it's really helpful if the inputting device will pause for you after each line. Even if the inputting device will not pause, reading a line at a time may buy you the processing time you need. The best thing to do is try it.

NOTE: In order to perform line-oriented input using the BASIC INPUT statement, the input must either have an ATASCII EOL at the end of each input line, or must have an ASCII CR terminate each line. In the latter case, you must configure the translation mode of the Interface Module port to convert the CR into EOL. This is discussed more fully in the section on configuring translation mode.

A large input buffer will be needed if you can read the data from the buffer only in large, occasional bursts. For example, if you do not know how long it will take to process a line of input because some lines require a lot of work, you will want to allow lines to "back up" in the input buffer. This will work fine as long as you do not get too many of these "slow" lines at once. You will probably have to determine the needed size of your input buffer by trial.

The number of characters that can come in every second depends on the Baud rate--the higher the Baud rate the faster characters can arrive. Thirty characters may arrive each second at 300 Baud, 480 may arrive in the same time at 4800 Baud. Of course, if the sending device does not run at the maximum possible speed-- if there are "gaps" between characters anywhere--then the speed of characters will not be ???. Thus the Baud rate can control the MAXIMUM data transfer rate, but the actual or EFFECTIVE data transfer rate may be smaller.

What things boil down to is that your program in BASIC must INPUT data from the input buffer faster than the Interface Module puts them there

from your RS-232-C compatible device; that is, your BASIC program must read the data faster than your device's effective data transmission rate (on average). You can control that rate by setting the Baud rate, and possibly there are other ways to control the transfer rate (that depends on the device itself). Be prepared to experiment to find the best mode of operation.

In BASIC, the START CONCURRENT MODE I/O operation which uses the built-in input buffer looks like this:

```
XIO 40, #channel, 0, 0, "Rn:"
```

Specify the appropriate open channel, and specify 1, 2, 3, or 4 for n in "Rn:". If you leave n out (i.e., "R:"), then port 1 is assumed. You MUST specify zero for both Aux1 and Aux1, since this is the way you tell the RS-232-C handler to use its own input buffer.

If you opened the port for output only, then only concurrent output is enabled. If the port is open for input only, then only concurrent input is started. If the port was opened for both, then concurrent mode input and output are started (full duplex). See the section about the input and output commands for details on how these various modes operate.

In BASIC, the START CONCURRENT MODE I/O operation in which you supply the input buffer for the handler is specified by a series of POKEs followed by calling the Central I/O (CIO) through a USR function. The POKEs specify the type of operation, and specify the buffer address and length. You POKE these values into the I/O Control Block (IOCB) corresponding to the channel you have opened for the RS-232-C port. Here is an example program:

```
10 DIM BUF$(500), RSTART$(7)

20 LET RSTART$ = "hhh#LVd"      NOTE: a bar over a
                                      character here means
                                      inverse video.)

30 LET FILE = 2

40 OPEN #FILE, 13, 0, "R4:"

50 LET IOCB = 16*FILE

60 LET BUF = ADR(BUF$)

65 LET BUFLEN = 500

70 LET RSTART = ADR(RSTART$)

80 POKE 832+IOCB+2, 40
```

```
90 POKE 832+IOCB+4, BUF-(INT(BUF/256)*256)
100 POKE 832+IOCB+5, INT(BUF/256)
110 POKE 832+IOCB+8, BUFLEN-(INT(BUFLEN/256)*256)
120 POKE 832+IOCB+9, INT(BUFLEN/256)
125 POKE 832+IOCB+10, 13
130 DUMMY = USR(RSTART, IOCB)
140 STARTSTATUS = PEEK(832+IOCB+3)
```

In this program, a full duplex file is opened through channel 2 to RS-232-C port number 4 (the 13 in line 40 specifies full duplex). Lines 50 through 70 set up some values which are used by the START CONCURRENT MODE I/O operation. The buffer is setup in lines 80 through 130. Line 140 gets the status value returned by the I/O call. Each POKE statement puts some needed value into the I/O Control Block (IOCB). The address to poke is specified as the sum of the following: the first address of the IOCB's (832), a value specifying which IOCB, and an "offset" into the IOCB for the particular value you are POKEing. The value specifying the IOCB is 16 times the channel number through which you have opened the RS-232-C port (in this case we set the variable IOCB to 32 in line 50, since the channel is 2).

The values poked into the IOCB are: 40 into offset 2; the buffer location (address) into locations 4 and 5; the buffer length into offsets 8 and 9; and 13 into offset 10. Pay special attention to the fact that the buffer address and the buffer length are both 2-byte values, requiring two POKEs to put them into the IOCB. Those complex-looking expressions in lines 90 through 120 are simply splitting the address and length into their low-part and high-part so each part can be POKEd individually.

Line 130 calls the I/O system through a USR function. This USR function has two arguments: the address of the function, and the IOCB specifier (the same as was used in specifying the POKE locations). The address of this USR function was found in line 70, so you see that the function is the character array called RSTART$. The function itself is the odd-looking sequence of characters in line 20. Be sure to type this character sequence carefully when before you call this USR function--any mistakes and your program will probably produce an unrecoverable failure.

Assembler note: This USR function is the following in Assembly Language: PLA; PLA; PLA; TAX; JMP $E456. The first four instructions

get the IOCB number into the X- register, and the return address is on the stack, so the I/O system is "called" by jumping to it!)

Line 140 gets the I/O status after the USR I/O call. You do not need to get the status if you do not want to. To get status PEEK at offset 3 in the IOCB. The status will be 1 if all went well. Other- wise, the status is the same as the error number that BASIC prints after an I/O call fails. (Note that the variable DUMMY in the program above does not get any meaningful value.)

Once this START CONCURRENT MODE I/O operation has been performed, the concurrent I/O is active. The operation may be either in-only, or it may be full-duplex (as specified in the OPEN). If you are running full-duplex, the output buffer is built into the Interface Module handler. The input and output buffers are accessed through normal input and output statements in BASIC; see the section on input and output statements. Once again, take note: BASIC MAY MOVE ARRAYS AROUND IF YOUR PROGRAM STOPS. IF THE CONCURRENT MODE INPUT CONTINUES AFTER YOUR PROGRAM STOPS, THIS MAY RESULT IN OVERWRITING SOMETHING OUTSIDE YOUR BUFFER ARRAY. IF YOU ARE NOT SURE WHETHER OR NOT THE CONCURRENT I/O HAS STOPPED, PRESS THE BREAK KEY TO STOP IT.

NOTE: there is a 256 byte area at address 1536 (decimal) which you may use as an input buffer or anything else. Be sure that area is only being used for the one thing you wish. No ATARI software uses this area except just after you turn the machine on, but you should be careful of non-ATARI software you use with BASIC. 1536 splits nicely into low- and high-parts (so does 256), so you could replace lines 90 through 120 of the above program:

```
90 POKE 832+IOCB+4, 0
100 POKE 832+IOCB+5, 6
110 POKE 832+IOCB+8, 0
120 POKE 832+IOCB+9, 1
```

If you use this area, you do not need to worry about it when your program stops since BASIC will not move it.