The Helios RS232 Device

Perihelion Software Technical Report No. 6

Bart Veer

December 1988

Perihelion Software Limited
The Maltings
Charlton Road
Shepton Mallet
Somerset
BA4 5QE
England
Telephone +44 749 4203
Fax. +44 749 4977

Contents

1 Introduction

It should be relatively simple to take some bytes of data, send them down a bit of wire, and turn them back into the same bytes of data at the receiving end. Unfortunately an RS232 device is anything but simple, and under Helios they tend to be even worse than normal. This is because under Helios the device may be connected directly to a transputer, it may be inside an IBM PC (or compatible) which is being used as an I/O processor, or inside another type of I/O processor, all of which will be using different chips and hence will behave in a slightly different way. To get around this, Helios defines a single interface between clients and all RS232 servers, irrespective of the underlying hardware, and the server has to support this interface to the best of its ability.

Typically an RS232 server needs to cope with more than one port. For example, a common add-on card for a transputer system would have eight RS232 lines to connect to dumb terminals. Hence the /rs232 server is implemented as a directory containing a number of ports, which can be viewed with the Is command just like any other directory. There will always be one entry in the directory, called default. Hence an application program can always open /rs232/default, without needing to know what the ports are actually called. If the server supports more than one port (for example, com1 and com2), then there will be three entries in the directory called default, com1 and com2. There will be some way of configuring the RS232 server so that default maps onto either com1 or com2. Also, it is possible to rename either com1 or com2 to default to change the default port.

For example, a user runs the public domain communications utility kermit to connect to a remote machine. Kermit opens the device /rs232/default unless instructed otherwise, and the RS232 server has been configured to map default onto com2: hence the user attempts to connect via com2. However, if it is necessary to connect through com1 occasionally then the user can write a shell script which renames com1 to default, runs kermit, and then renames com2 to default again to restore the system.

Once an application has opened a stream to an RS232 port, the port can be reconfigured to suit particular needs. There are two parts to configuring a port. First there is the event mechanism, which will ensure that the application is woken up asynchronously when particular events occur. Second there is the attributes mechanism used to control the behaviour of the port. The event mechanism is used with the system call `EnableEvents()`, and the attribute mechanism is used with the system calls `GetAttributes()` and `SetAttributes()` or the posix library termios calls. Full details of these mechanisms may be found in the Helios manual, together with examples of their use.

It should be noted that the terminology used to describe the Helios RS232 device can be rather confusing. Briefly a client program or application interacts with an RS232 server, which is the only program that has access to the physical RS232 hardware: the application program usually runs on a separate processor. The RS232 server can send data down a bit of wire and receive data from a bit of wire. At the other end of this wire there may be another Helios RS232 server, or there may be some non-transputer hardware such as a dumb terminal.

The remainder of this document describes the attributes and events used for the RS232 device.

2 Incoming and Outgoing Data

Data is sent and received in units of characters which may vary in size from 5 to 8 bits. This size excludes the optional parity bit, described below. The attributes for controlling the size are: RS232_Csize_5, RS232_Csize_6, RS232_Csize_7, and RS232_Csize_8. Should you find it necessary to change the current size, you must first `RemoveAttribute()` the current size and then `AddAttribute()` the new size. To determine the current size may involve up to four calls to `IsAnAttribute()`. The usual character sizes are 8 bits without parity or 7 bits with parity.

Data is transmitted as a single start bit, followed by the character, an optional parity bit, and either one or two stop bits. To choose between one and two stop bits, you should use the RS232_Cstopb attribute: if set, the server will use two stop bits; otherwise it will use only one.

3 Baud rates

RS232 lines can operate at a number of different speeds or baud rates. In theory an RS232 line should cope with different baud rates for sending and receiving data, but not all hardware can support this. For example, the 8250 UART chip used in the IBM PC and compatibles can cope with only a single baud rate for both input and output. In such a case only the input speed is used, and the output speed is ignored.

The `GetInputSpeed()` and `GetOutputSpeed()` routines may be used to determine the current baud rate. This should be one of RS232_B50, RS232_B75, RS232_B110, RS232_B134, RS232_B150, RS232_B200, RS232_B300, RS232_B600, RS232_B1200, RS232_B1800, RS232_B2400, RS232_B4800, RS232_B9600, RS232_B19200, or RS232_B38400. `SetInputSpeed()` and `SetOutputSpeed()`

may be used to modify the baud rates. Not all hardware can cope with all the baud rates, particularly 19200 baud and 38400 baud, and if an attempt is made to set the baud rate to an illegal value then the server will use some default instead. To detect this, the application should perform another `GetAttributes()` after reconfiguring the port.

4 Flow Control

Flow control has to be used between the sending and receiving ends of an RS232 line to control the rate at which data is sent. The recommended approach to flow control is XON/XOFF. When the sender is transmitting data too quickly and the receiver is unable to process it quickly enough, usually because its client is not reading the data, the receiver must suspend the sender for a while to avoid overflowing its buffer. This is done by sending a single XOFF character. When the receiver is ready for more data, it should send an XON character. After sending the XOFF character, there may be some delay before the sender gets a chance to process it so the receiver must be able to buffer at least another 128 characters.

To control XON/XOFF flow control there are two attributes. First, RS232_IXON controls XON/XOFF on output. If this attribute is set and the application is writing down the serial line, then the other side can suspend the write by sending an XOFF character. If the attribute is not set then the XON/XOFF characters may be read by the application. Note that there may be some considerable delay between the server receiving the XON/XOFF character and the application reading it, so applications should not normally perform their own XON/XOFF handling. The second attribute is RS232_IXOFF and controls XON/XOFF on input. If this attribute is set and the server is receiving data faster than the application is reading it, then the server can send XOFF characters to the other side. If the attribute is not set then the server may overflow its buffers, and data will be lost irretrievably. Data may also be lost irretrievably if the other side is ill-behaved and continues to send data after an XOFF.

The alternative approach to handshaking is to use the modem status lines. If the attribute RS232_Clocal is set then these lines are ignored. If the attribute is not set, then the server will use the handshake lines to the best of its ability. However, the exact operation of these handshake lines is not well-defined and different pieces of hardware are likely to disagree about their interpretation. Under Helios the Data Terminal Ready (DTR) line should be kept high whilst there is an outstanding read to be satisfied, allowing the other side to continue to send data; also, while there is an outstanding write the Request To Send (RTS) line should be kept high, and

the server will send data when the other side asserts the Clear To Send (CTS) line. However, RS232 servers are free to interpret these handshaking lines in different ways or ignore them completely, if this is appropriate for the hardware. Hence using hardware handshaking rather than XON/XOFF for flow control is strongly discouraged.

5 Parity

No less than six attributes control the parity behaviour of an RS232 line. The first attribute is RS232_ParEnb: if this attribute is set then the server will use either odd or even parity on both input and output. Note that it is not possible to use parity on input but not on output or vice versa, because little if' any hardware supports it. If the attribute is not set then no parity is used, which means that the application does not have to worry about the various parity errors.

The next attribute is RS232_Istrip. If this attribute is set then all data received from the RS232 line is stripped to the bottom seven bits. This is very useful if the application is unsure whether the other side is using seven bits with parity or eight bits without parity, because it ensures that at least seven bits are correct assuming no transmission errors. If the attribute is not set then the top bit is not stripped.

The RS232_ParOdd attribute is useful only if parity has been enabled: if set, odd parity will be used; otherwise even parity is used.

The remaining parity attributes control the detection of parity errors on input. Note that the server cannot take any action if there are parity errors on output because this can be detected only at the other side, and higher levels of protocol have to take recovery action. The attribute RS232_InPck controls whether it is the client or the server that should notice parity errors: if the attribute is not set then any data received with parity errors is sent to the application program as usual (in effect, the server ignores the parity error); otherwise the server takes some recovery action depending on attributes RS232_IgnPar and RS232_ParMrk. If the RS232_IgnPar attribute is set and the server detects a parity error then the data received in error is discarded; otherwise some special characters are placed in the read stream, depending on the remaining attribute RS232_ParMrk. If that attribute is not set then the data received in error will be replaced by a single byte 0x00, and it is upto the application to work out whether this 0x00 is the result of a parity error or some real data. If the RS232_ParMrk attribute is set then a parity' error will generate two bytes, a byte 0xFF followed by a byte 0x00; this can lead to ambiguity if a character 0xFF is received correctly, possible only if the RS232_Istrip attribute is not set, so in that case a character 0xFF

received correctly is placed in the read stream as two bytes 0xFF 0xFF.

Consider the following example: an RS232 port is configured with 7-bit characters, RS232_ParEnb, RS232_ParOdd, RS232_InPck, RS232_ParMrk, not RS232_IgnPar, and not RS232_Istrip; the data received is the byte 00111001. Parity is enabled so the first bit (that is, a 0 is the parity bit). However, odd parity is in use and there are an even number of is in the byte, so a parity error of some sort has occurred. RS232_InPck is enabled, so the parity error should be handled by the RS232 server rather than by the application. Since RS232_IgnPar is not set, the data received should not be thrown away, but instead placed in the read stream as a special sequence. Because RS232_ParMrk is set the data placed in the read stream will be two bytes, 0xFF followed by 0x00. When the application discovers a byte 0xFF in the read stream, and the next byte is an 0x00 rather than an 0xFF, this means that a parity error has occurred and it can take whatever recovery action is appropriate.

There is a related error code: if the RS232 server is kept sufficiently busy that it cannot handle the incoming data as it arrives, and some data is lost before the server had a chance to buffer it, an overrun error has occurred; this is indicated to the application as the sequence 0xFF 0x01.

6 Break Signals

A break signal occurs when the voltage on the RS232 line drops to 0 for a time, and is usually generated when one of the sides wishes to drop the connection. First, the application must be able to generate a break signal. There are two ways to do this: it can set the baud rate to RS232_BO (the output baud rate if separate baud rates arc supported on input and output, otherwise the input baud rate), in which case the server will reset the baud rate to its default value afterwards; alternatively, if the RS232_HupCl attribute is set and the stream to the RS232 port is closed then the server will generate a break signal automatically.

The second problem is detecting when a break signal has occurred. There are two attributes to control this. The first is RS232_IgnoreBreak: if this is set any break signals are ignored completely, otherwise the behaviour is determined by the next attribute. If RS232_BreakInterrupt is not set then a break signal will cause a byte 0x00 to be inserted in the read stream, and it is up to the application to distinguish this byte 0x00 from a transmitted byte 0x00 or a byte 0x00 generated by parity errors given the appropriate attributes. If RS232_BreakInterrupt is set, the server will send a break event to an event handler if the application has installed one;; if the application has not installed an event handler, the break is ignored. To install such

an event handler the application should use an `EnableEvents()` call with Event_RS232Break as one of the arguments. Break events are treated as one-off events, and there is no need to `Acknowledge()` or `NegAcknowledge()` them.

7 Modem Interrupts

One of the lines specified in the RS232 protocol is Ring Indicator (RI). This is used mainly by dial-up modems, to inform an application that somebody is trying to dial in. To detect such events, the application should install an event handler by calling `EnableEvents()` with Event_ModemRing as one of the arguments. Modem rings are one-off events, and there is no need to `Acknowledge()` or `NegAcknowledge()` them.

8 The Default Configuration

The default configuration for a Helios RS232 port is as follows: 9600 baud for both input and output; 8 bits per character, and one stop bit; parity and Istrip disabled; XON/XOFF flow control enabled on both input and output with hardware handshaking disabled; break interrupts enabled, but the client has to install an event handler; HupCl disabled. This configuration should work correctly on all implementations.

IBM is a registered trademark of international Business Machines, Inc.

Posix library refers to a library of calls used by Helios that largely conform to the IEEE Std. 1003.1-1988, IEEE Standard Portable Operating System Interface for Computer Environments.