Open Transport Serial Developer Note

PRELIMINARY
Revision 1.1b14
01/18/96

Table of Contents

Revision History 3

Related Documents 3

Async Serial for Open Transport 4

Technical Specifications 5
- Naming 5
- Addresses 5
- Options 5
  - The SRL_OPT_BAUDRATE Option 5
  - The SRL_OPT_DATABITS Option 5
  - The SRL_OPT_STOPBITS Option 6
  - The SRL_OPT_PARITY Option 6
  - The SRL_OPT_STATUS Option 6
  - The SRL_OPT_HANDSHAKE Option 6
  - The SRL_OPT_RCVTIMEOUT Option 7
  - The SRL_OPT_PECHAR Option 7
  - The SRL_OPT_EXTCLOCK Option 7
  - The SRL_OPT_BURSTMODE Option 7
- IOCTL Commands 8
  - The I_SetSerialDTR 8
  - The I_SetSerialBreak 8
  - The I_SetSerialXOffState 8
  - The I_SetSerialXOn 8
  - The I_SetSerialXOff 8
  - The I_OTSetFramingType 9

Using Serial endpoints 10
- Listening for incoming data 10
- Initiating outgoing data 10

Index 11

Revision History

08/16/95	Update for version 1.1b1 of Open Transport release
04/20/95	Updated for version 10b2 of Open Transport release
07/18/94	Update for version 1.0d13 of Open Transport release
04/10/94	first written

Related Documents

Inside AppleTalk®, Second Edition, Gushuran S. Sidhu, et. at., Addison-Wesley Publishing, Inc.

Apple Shared Library Manager Developer's Guide, by ESD Publications, October 4, 1993, Apple Computer, Inc.

Open Transport Client Developer Note

Async Serial for Open Transport

This document describes how the Open Transport implementation of asynchronous Serial communications can be used by client applications. This document should be used along with the *Open Transport Client Developer Note*. That document describes general information about Open Transport endpoint libraries and Open Transport mapper libraries. However, it does not not provide any information specific to Serial communications.

This document describes specific Serial options and implementation details.

Technical Specifications

This section describes Serial endpoint names, and address and option formats that are used in the Open Transport Endpoint functions (like *Snd(), Bind(),* etc.)

The file OpenTptSerial.h contains the declarations of the necessary constants and data structures needed. This file may be included by both ‘C’ and ‘C++’ source files.

Naming

In order to open serial endpoints, you need to supply a name to the OTOpenEndpoint function. The following names are defined by equates in the OpenTptSerial.h header file:

kSerialName	"serial"	Default serial port
kSerialPortAName	"serialA"	Port A serial port
kSerialPortBName	"serialB"	Port B serial port

If there are additional serial cards with drivers .CIN, .COUT, .DIN, .DOUT, etc. they will be available by using the names "serialC", etc. Other serial drivers registered with the Communications Resource Manager will use the names "serial1", "serial2", etc.

For example:

```
OSErr err = OTOpenEndpoint(OTCreateConfiguration(kSerialPortAName));
```

Addresses

Serial communications is point-to-point. As such, no addressing information is possible. Serial endpoints do not support Mappers, and do not support addressing, All of the structures which describe addresses should have a zero length when dealing with Serial endpoints.

Options

Serial endpoints currently support severak options. These options are defined by the XTI Level COM_SERIAL (whose value is 'SERL') , and have the names indicated below.

The SRL_OPT_BAUDRATE Option;

This option sets the Serial baud rate. The value of the option is a 4-byte unsigned integer corresponding to the desired baud rate. The Serial module will choose the closes baud rate supported that matches the requested rate. The default value is 9600 baud. If the baud rate is negative, the serial driver will go into "burst" mode using the indicated baud rate. "Burst mode" is really only useful at higher baud rates.

The SRL_OPT_DATABITS Option

This option selects the number of data bits to be used. The value of the option is a 4-byte unsigned integer corresponding to the number for data bits. Legal values are 5, 6, 7, and 8. The default value is 8 data bits.

The SRL_OPT_STOPBITS Option

This option selects the number of stop bits to be used. The value of the option is a 4-byte unsigned integer corresponding to ten (10) times the number of stop bits. Legal values are 10, 15, and 20. The default value is 10 data bits.

The SRL_OPT_PARITY Option

This option selects the parity to be used. The value of the option is a 4-byte unsigned integer corresponding to an enumeration. Legal values are kOTNoParity (0), kOTOddParity (1), and kOTEvenParity. (2) The default value is kOTNoParity.

The SRL_OPT_STATUS Option

This option is a read-only option which returns status information on the serial port. It is a 4-byte unsigned integer that contains a bitmap which returns the following information:

```
enum
{
        kOTSrlOverrRun          = 0x01,
        kOTSrlBreakOn           = 0x08,
        kOTSrlParityErr         = 0x10,
        kOTSrlOverrunErr        = 0x20,
        kOTSrlFramingErr        = 0x40,
        kOTSrlXOffSent          = 0x0010000,
        kOTSrlDTRNegated        = 0x0020000,
        kOTSrlCTLHold           = 0x0040000,
        kOTSrlXOffHold          = 0x0080000,
        kOTSrlOutputBreakOn     = 0x1000000
}
```

The SRL_OPT_HANDSHAKE Option

This option selects the handshaking to be used by the serial line. The value of the option is a 4-byte unsigned integer interpreted as follows:

The high word (16 bits) of the integer is a bitmap with 1 or more of the following bits set:

kOTXOnOffInputHandshake = 1
kOTXOnOffOutputHandshake = 2
kOTCTSInputHandshake = 4
kOTDTROutputHandshake = 8

The 2nd lowest byte is the XOn character value, and the lowest byte is the XOff character value. If these values are 0, and XOnOff handshaking was requested, the default values of control-S for XOff and control-Q for XOn will be used.

xxxxxxxxxxxxxxxx xxxxxxxx xxxxxxxx
handshake bitmap XOn char XOff Char

The inline function (or #define for C users) SerialHandshakeData(type, onChar, offChar) in OpenTptSerial.h can be used to create this 4-byte value.

The default value of this option is no handshaking.

The SRL_OPT_RCVTIMEOUT Option

This option selects the receive timeout options for incoming serial characters. It is a 32-bit value representing the number of milliseconds that the receiver should wait before delivering less than the RcvLoWat number of characters. If RcvLoWai is 0, then the value is the number of milliseconds of quiet time (no characters being received) that must elapse before characters are delivered to the client. In all cases, this option is advisory in nature, and serial drivers are free to deliver data whenver they deem it convenient. For instance, many serial drivers will deliver data whenver 64 bytes have been received, since 64 bytes is the smalles STREAMS buffer size.

Examples:

RcvTimeout	RcvLoWat	Action
0	0	Data is delivered immediately after it arrivers
x	0	Data is delivered after "x" milliseconds of no incoming characters on the line.
x	y	Data is delivered after "y" characters are received, or "x" milliseconds after the first character is received, whichever comes first.

The SRL_OPT_PECHAR Option

This option defines how characters that arrive with parity errors are handled. It is a 32-bit value. A 0 value will disable all replacment. A single character in the low byte disignates the replacement character. When caracters are received with a parity error, they are replaced by this specified chracter. If a valid incoming character matches the replacement character, then the received character's most-significant-bit is cleared. For this situation, an alternate replacement character may be specified in bits 8 through 15 of the 32-bit value, with 0xff being placed in bits 16 through 23 (the macros OTSrlSetPEChar(rep) and OTSrlSetPECharWithAlternate(rep, alternate) may be used to get the bit placement correct). In this case, whenever a vlid character is received that matches the first replacement character, it is replaced with this alternate character (which may be 0).

The SRL_OPT_EXTCLOCK Option

This option requests an external clock. It is a 32-bit value. A 0-value turns off external clocking (the default). Any other value is a requested divisor for the external clock. Be aware that not all serial implementations support an external clock, and that not all requested divisors will be supported if it does support an external clock.

The SRL_OPT_BURSTMODE Option

This option requests burst-mode operation. It is a 32-bit value, where 0 requests burst-mode off (the default), and a 1 requests burst mode to be turned on.In burst mode, the serial driver continues looping, reading incoming characters, rather than waiting for an interrupt for each character. This option may not be supported by all Serial drivers. Note that burst mode may adversely impact performance of the Macintosh system, since interrupts may be held of for long periods of time.

IOCTL Commands

Serial endpoints currently supports several IOCTL commands. These commands are defined below.

The I_SetSerialDTR Command

This commands sets the DTR line on the serial port. Use a 0 to turn DTR off, and a 1 to turn DTR on.

```
OTIOCtl(theSerialEndpoint, I_SetSerialDTR, 1);     // Turn on DTR
```

The I_SetSerialBreak Command

This option is used to control a "break" on the serial line. It is a 4-byte unsigned integer. It's value is kOTBreakOff (0) to unconditionally turn "break" off, kOTBreakOn(0xffffffff) to unconditionally turn "break" on, and any other value to turn "break" on for the specified number of milliseconds.

```
OTIoctl(theSerialEndpoint, I_SetSerialBreak, kOTBreakOn);        // Turn on BREAK
```

The I_SetSerialXOffState Command

This commands sets the XOff state of the serial port. A value of 0 will unconditionally clear the XOFF state, while a value of 1 will unconditionally set it.

```
OTIoctl(theSerialEndpoint, I_SetSerialXOffState, 1);      // Set XOFF state to ON
```

The I_SetSerialXOn Command

This commands causes the serial port to send an XON character. A value of 0 will only cause it to be sent if we're in the XOFF state, while a value of 1 will unconditionally send the character.

```
OTIoctl(theSerialEndpoint, I_SetSerialXOn, 1);     // Unconditionally send an XON
```

The I_SetSerialXOff Command

This commands causes the serial port to send an XOFF character. A value of 0 will only cause it to be sent if we're in the XONstate, while a value of 1 will unconditionally send the character.

```
OTIoctl(theSerialEndpoint, I_SetSerialXOff, 1);    // Unconditionally send an XOFF
```

The I_OTSetFramingType Command

Currently, serial ports can support four different framing types. These types are enumerated in the `fCapabilities` field of the `OTPortRecord`. These are `kOTSrlFramingAsync`, `kOTSrlFramingHDLC`, `kOTSrlFramingSDLC`, and `kOTSrlFramingAsyncPackets`. The normal mode of operation is kOTSrlFramingAsync. A client may change the mode of operation to one of the other modes by making this IOCTL command:

```
OTIoctl(theSerialEndpoint, I_OTSetFramingType, kOTSrlFamingAsyncPackets);
```

The kOTSrlFramingAsyncPackets type is a special version of async serial, where the underlying serial provider assumes that each individual message that arrives is a separate packet, and should be sent as such. It also means that the underlying provider will insure that if data is flushed, all data will be flushed EXCEPT any "packet" which happens to be in process at the time of the flush. This behavior is important to technologies like Apple Remote Access (ARA) or Point-to-Point Protocol (PPP) implementations (which are using the Serial port for delivery of discrete packets) because: 1) stopping a packet in the middle of transfer causes some thrashing of the upper protocols to resynchronize, which causes performance degradation; and 2) Both protocols want to insure that if they have to flush the queue of waiting messages that all waiting messages are flushed, even if they are queued up in the protocol module.

Using Serial endpoints

Serial endpoints are connection-oriented streams. They do not support any of the connectionless datagram or transaction endpoint calls. Because of the point-to-point nature of serial communications, there are a few differences between using a serial endpoint and using other networking-oriented endpoints.

One of the key differences is that there are no addresses for serial endpoints. Those functions that expect an incoming address should be given an address length of zero (0), and those functions that return an address will return an address with a length of zero (0).

The other key difference is that only one serial endpoint can own the hardware at a given time. There is no sharing of the serial hardware between endpoints. This sharing is typically done by a higher level protocol (for instance, PPP).

Serial endpoints are created using the names described in the *Naming* section in the preceding section. Any number of serial endpoints may be created, but only one may own the hardware. The first endpoint to either *Bind()* with a qlen of 1, or *Connect()* with a qlen of 0 will own the hardware. All other endpoints that attempt to execute either of these functions will receive a kOTAddressBusyErr.

Listening for incoming data

Use the *Bind()* function to bind the endpoint, using a qlen of 1 (a qlen greater than 1 is not allowed). When an incoming character is detected on the serial port, you will receive a connect indication. You may accept the indication on the current endpoint, or you may accept it on another serial endpoint, which was bound with a qlen of 0. In either case, once the accepting endpoint returns to the T_IDLE state, the original endpoint will once again get a connect indication if another incoming character is detected. Executing an *Unbind()* will release the hardware for other endpoints to use.

Initiating outgoing data

As with all endpoints, you must issue the *Bind()* function before you can use the endpoint to send or receive data. For serial endpoints, you may bind with a qlen of 0, or a qlen of 1. If you wish to initiate data transfer, you must issue the *Connect()* function. This will place the endpoint in the data transfer state and allow you to issue *Snd()* and *Rcv()* commands. Executing a *Disconnect()* (qlen = 0), or an *Unbind()* (qlen = 1) will release the hardware for other endpoints to use.

Index

Addresses 5
baud rate 5
break 8
COM_SERIAL 5
data bits 5
handshaking 6
Index 11
Initiating outgoing data 10
IOCTL Commands 8
I_OTSetFramingType 9
I_SetSerialBreak 8
I_SetSerialDTR 8
I_SetSerialXOff 8
I_SetSerialXOffState 8
I_SetSerialXOn 8
Listening for incoming data 10
Naming 5
Options 5
OTSrlSetPEChar 7
OTSrlSetPECharWithAlternate 7
parity 6
RcvLoWat 7
SRL_OPT_BAUDRATE Option 5
SRL_OPT_BURSTMODE Option 7
SRL_OPT_DATABITS Option 5
SRL_OPT_EXTCLOCK Option 7
SRL_OPT_HANDSHAKE Option 6
SRL_OPT_PARITY Option 6
SRL_OPT_PECHAR Option 7
SRL_OPT_RCVTIMEOUT Option 7
SRL_OPT_STATUS Option 6
SRL_OPT_STOPBITS Option 6
stop bits 6