serial

COLLABORATORS

	TITLE : serial		
ACTION	NAME	DATE	SIGNATURE
WRITTEN BY		July 19, 2024	

REVISION HISTORY

NUMBER	DATE	DESCRIPTION	NAME

Contents

Chapter 1

serial

1.1 serial.doc

```
AbortIO()
BeginIO()
CloseDevice()
CMD_CLEAR
CMD_FLUSH
CMD_READ
CMD_RESET
CMD_START
CMD_STOP
CMD_WRITE
OpenDevice()
SDCMD_BREAK
SDCMD_QUERY
SDCMD_SETPARAMS
```

1.2 serial.device/AbortIO

```
NAME
    AbortIO(ioRequest) -- abort an I/O request
          A1

FUNCTION
   This is an exec.library call.

   This function attempts to aborts a specified read or write request.
   If the request is active, it is stopped immediately. If the request is
   queued, it is painlessly removed.  The request will be returned
   in the same way any completed request it.

   After AbortIO(), you must generally do a WaitIO().

INPUTS
   iORequest  -- pointer to the IORequest Block that is to be aborted.

RESULTS
```

io_Error -- if the Abort succeded, then io_Error will be #IOERR_ABORTED

(-2) and the request will be flagged as aborted (bit 5 of
io_Flags is set). If the Abort failed, then the Error will be zero.

BUGS
    Previous to version 34, the serial.device would often hang when
    aborting CTS/RTS handshake requests. This was the cause of the
    incorrect assumption that AbortIO() does not need to be followed
    by a wait for a reply (or a WaitIO()).

1.3 serial.device/BeginIO

NAME
    BeginIO(ioRequest),deviceNode -- start up an I/O process
             A1   A6
FUNCTION
    This is a direct function call to the device. It is intended for
    more advanced programmers. See exec's DoIO() and SendIO() for
    the normal method of calling devices.

    This function initiates a I/O request made to the serial
    device. Other than read or write, the functions are performed
    synchronously, and do not depend on any interrupt handling
    logic (or it's associated discontinuities), and hence should
    be performed as IO_QUICK.
    With some exceptions, reads and writes are merely initiated by
    BeginIO, and thusly return to the caller as begun, not completed.
    Completion is signalled via the standard ReplyMsg routine.
    Multiple requests are handled via FIFO queueing.
    One exception to this non-QUICK handling of reads and writes
    is for READS when:
      - IO_QUICK bit is set
      - There are no pending read requests
      - There is already enough data in the input buffer to satisfy
        this I/O Request immediately.
    In this case, the IO_QUICK flag is not cleared, and the request
    is completed by the time it returns to the caller. There is no
    ReplyMsg or signal bit activity in this case.

INPUTS
    ioRequest -- pointer to an I/O Request Block of size
        io_ExtSerSize (see serial.i for size/definition),
        containing a valid command in io_Command to process,
        as well as the command's other required parameters.
    deviceNode -- pointer to the "serial.device", as found in
        the IO_DEVICE of the ioRequest.

RESULTS
    io_Error -- if the BeginIO succeded, then Error will be null.
        If the BeginIO failed, then the Error will be non-zero.
        I/O errors won't be reported until the io completes.

SEE ALSO

```
devices/serial.h
```

1.4 serial.device/CloseDevice

```
NAME
    CloseDevice -- close the serial port

SYNOPSIS
    CloseDevice(deviceNode)
                  A1
FUNCTION
    This is an exec call that terminates communication with the
    serial device.  Upon closing, the device's input buffer is freed.

    Note that all IORequests MUST be complete before closing.
 If any are pending, your program must AbortIO() then WaitIO()
    to complete them.

 INPUTS
    deviceNode - pointer to the device node, set by Open

 SEE ALSO
    serial.device/OpenDevice
```

1.5 serial.device/CMD_CLEAR

```
NAME
    Clear -- clear the serial port buffers

FUNCTION
    This command resets the serial port's read buffer pointers.

 IO REQUEST
    io_Message      mn_ReplyPort initialized
    io_Device       set by OpenDevice
    io_Unit         set by OpenDevice
    io_Command      CMD_CLEAR

 RESULTS
    Error -- If the Clear succeded, then io_Error will be null.
             If the Clear failed, then the io_Error will be non-zero.
```

1.6 serial.device/CMD_FLUSH

```
NAME
    Flush -- clear all queued I/O requests for the serial port

 FUNCTION
    This command purges the read and write request queues for the
    serial device. Flush will not affect active requests.
```

```
IO REQUEST
    io_Message      mn_ReplyPort initialized
    io_Device       set by OpenDevice
    io_Unit         set by OpenDevice
    io_Command      CMD_FLUSH

RESULTS
    Error -- if the Flush succeded, then io_Error will be null.
          If the Flush failed, then the io_Error will be non-zero.
```

1.7 serial.device/CMD_READ

```
NAME
    Read -- read input from serial port

FUNCTION
This command causes a stream of characters to be read in from
the serial port buffer.  The number of characters is specified
in io_Length.

The Query function can be used to check how many characters
are currently waiting in the serial port buffer.  If more characters
are requested than are currently available, the ioRequest
will be queued until it can be satisfied.


The best way to handle reads is to first Query to get the number
of characters currently in the buffer.  Then post a read request
for that number of characters (or the maximum size of your buffer).

If zero characters are in the buffer, post a request
for 1 character.  When at least one is ready, the device will return
it.  Now start over with another Query.

Before the program exits, it must be sure to AbortIO() then WaitIO()
any outstanding ioRequests.

IO REQUEST
    io_Message      A mn_ReplyPort is required
    io_Device       set by OpenDevice
    io_Unit         set by OpenDevice
    io_Command      CMD_READ
    io_Flags        If the IOB_QUICK bit is set, read will try
  to complete the IO quickly
    io_Length       number of characters to receive.
    io_Data         pointer to buffer

RESULTS
    Error -- if the Read succeded, then io_Error will be null.
        If the Read failed, then io_Error will be non-zero.
    io_Error will indicate problems such as parity mismatch,
    break, and buffer overrun.

SEE ALSO
```

```
serial.device/SDCMD_QUERY
serial.device/SDCMD_SETPARAMS
```

 BUGS
Having multiple outstanding read IORequests at any one time will
probably fail.

Old documentation mentioned a mode where io_Length was set to -1.
If you want a NULL terminated read, use the io_TermArray instead.

1.8   serial.device/CMD_RESET

    NAME
        Reset -- reinitializes the serial port

    FUNCTION
        This command resets the serial port to its freshly initialized
        condition. It aborts all I/O requests both queued and current,
        relinquishes the current buffer, obtains a new default sized
        buffer, and sets the port's flags and parameters to their
        boot-up time default values. The functions places the reset
        parameter values in the ioRequest block.

    IO REQUEST
        io_Message   mn_ReplyPort initialized
        io_Device   set by OpenDevice
        io_Unit   set by OpenDevice
        io_Command   CMD_RESET

    RESULTS
        Error -- if the Reset succeded, then Error will be null.
            If the Reset failed, then the Error will be non-zero.

1.9   serial.device/CMD_START

    NAME
        Start -- restart paused I/O over the serial port

    FUNCTION
        This function restarts all current I/O on the serial port by
        sending an xON to the "other side", and submitting a "logical
        xON" to "our side", if/when appropriate to current activity.

    IO REQUEST
        io_Message   mn_ReplyPort initialized
        io_Device   set by OpenDevice
        io_Unit   set by OpenDevice
        io_Command   CMD_START

    RESULTS

    SEE ALSO

```
        serial.device/CMD_STOP
```

1.10   serial.device/CMD_STOP

```
NAME
    Stop -- pause all current I/O over the serial port

FUNCTION
    This command halts all current I/O on the serial port by
    sending an xOFF to the "other side", and submitting a "logical
    xOFF" to "our side", if/when appropriate to current activity.

IO REQUEST
    io_Message      mn_ReplyPort initialized
    io_Device       set by OpenDevice
    io_Unit         set by OpenDevice
    io_Command      CMD_STOP

RESULTS

SEE ALSO
    serial.device/CMD_START
```

1.11   serial.device/CMD_WRITE

```
NAME
    Write -- send output to serial port

FUNCTION
    This command causes a stream of characters to be written out
    the serial port. The number of characters is specified in
    io_Length, unless -1 is used, in which case output is sent until
    a null(0x00) is encountered.

IO REQUEST
    io_Message      must have mn_ReplyPort initialized
    io_Device       set by OpenDevice
    io_Unit         set by OpenDevice
    io_Command      CMD_WRITE
    io_Flags        Set IOF_QUICK to try quick I/O
    io_Length       number of characters to transmit, or if set
                    to -1 transmit until null encountered in buffer
    io_Data         pointer to block of data to transmit

RESULTS
    Error -- if the Write succeded, then io_Error will be null.
        If the Write failed, then the io_Error will be non-zero.

 SEE ALSO
serial.device/SDCMD_SETPARAMS
```

1.12   serial.device/OpenDevice

```
  NAME
      OpenDevice -- Request an opening of the serial device.

  SYNOPSIS
      error = OpenDevice("serial.device", unit, ioRequest, flags)
      D0                     A0              D0    A1          D1

BYTE OpenDevice(STRPTR, ULONG, struct IOExtSer *, ULONG);

  FUNCTION
      This is an exec call.  Exec will search for the serial.device, and
      if found, will pass this call on to the device.

      Unless the shared-access bit (bit 5 of io_SerFlags) is set,
      exclusive use is granted and no other access to that unit is
      allowed until the owner closes it.  All the serial-specific fields
      in the ioRequest are initialized to their most recent values (or
      the Preferences default, for the first time open).

      If support of 7-wire handshaking (i.e. RS232-C CTS/RTS protocol)
      is required, use the serial.device/SDCMD_SETPARAMS command.
This feature should also be specified at inital OpenDevice() time.

  INPUTS
      "serial.device"  - pointer to literal string "serial.device"
      unit        - Must be zero, or a user setable unit number.
       (This field is used by multiple port controllers)
                        Zero specifies the default serial port.
      ioRequest  - pointer to an ioRequest block of size io_ExtSerSize
                        to be initialized by the serial.device.
                    (see devices/serial.h for the definition)
                      NOTE use of io_SerFlags (see FUNCTION above)
   IMPORTANT: The ioRequest block MUST be of size io_ExtSerSize,
        and zeroed (with the exeptions as noted)!
      flags       - Must be zero for future compatibility

  RESULTS
      D0      - same as io_Error
      io_Error  - If the Open succeded, then io_Error will be null.
            If the Open failed, then io_Error will be non-zero.
      io_Device - A pointer to whatever device will handle the calls
                for this unit.  This pointer may be different depending
                on what unit is requested.

  BUGS
If 7-wire handshaking is specified, a timeout "feature" is enabled.
If the device holds off the computer for more than about 30-60
seconds, the device will return the write request with the error
SerErr_TimerErr.  Don't depend on this, however.  If you want a timeout,
set up the timer.device and wait for either timer, or serial IO to
complete.

On open, the serial.device allocates the misc.resource for the
serial port.  It does not return it until the serial.device is
```

```
expunged from memory.  It should return it when no more openers
exist.   This code can force a specified device to try and
expunge.  Of course, if the device is in use nothing will happen:

#include "exec/types.h"
#include "exec/execbase.h"
#include "proto/exec.h"

void FlushDevice(char *);
extern struct ExecBase *SysBase;

void main()
{
  FlushDevice("serial.device"); /* or parallel.device */
}

/*
 * Attempts to flush the named device out of memory.
 * If it fails, no status is returned; examination of
 * the problem will reveal that information has no
 * valid use after the Permit().
 */
void FlushDevice(name)
char  *name;
{
struct Device *result;

    Forbid();
    if( result=(struct Device *)FindName(&SysBase->DeviceList,name) )
  RemDevice(result);
    Permit();
}
```

```
 SEE ALSO
serial.device/CloseDevice
serial.device/SDCMD_SETPARAMS
devices/serial.h
```

1.13 serial.device/SDCMD_BREAK

```
 NAME
     Break -- send a break signal over the serial line

 FUNCTION
     This command sends a break signal (serial line held low for an
     extended period) out the serial port. For the built-in port,
     This is accomplished by setting the UARTBRK bit of regisrer ADKCON.

     After a duration (user specifiable via setparams, default 250000
     microseconds) the bit is reset and the signal discontinued.
     If the QUEUEDBRK bit of io_SerFlags is set in the io_Request
     block, the request is placed at the back of the write-request
     queue and executed in turn. If the QUEUEDBRK bit is not set,
     the break is started immediately, control returns to the
```

      caller, and the timer discontinues the signal after the
      duration is completed. Be aware that calling BREAK may
 affect other commands such as ABORT, FLUSH, STOP, START, etc...

  IO REQUEST
      io_Message   mn_ReplyPort initialized
      io_Device   set by OpenDevice
      io_Unit   set by OpenDevice
      io_Command   SDCMD_BREAK
      io_Flags   set/reset IO_QUICK per above description

  RESULTS
      Error -- if the Break succeded, then Error will be null.
          If the Break failed, then the Error will be non-zero.

1.14   serial.device/SDCMD_QUERY

  NAME
      Query -- query serial port/line status

  FUNCTION
      This command return the status of the serial port lines and
      registers. The number of unread bytes in the serial device's
      read buffer is shown in io_Actual.

 The break send & received flags are cleared by a query, and
 whenever a read IORequest is returned with a error
 in io_Error.

  IO REQUEST
      io_Message   mn_ReplyPort initialized
      io_Device   preset by OpenDevice
      io_Unit   preset by OpenDevice
      io_Command   SDCMD_QUERY

  RESULTS
      io_Status   BIT ACTIVE FUNCTION

            LSB   0   ---   reserved
                       1   ---   reserved
                       2   high   parallel "sel" on the A1000
                                    On the A500 & A2000, "sel" is also
                                    connected to the serial port's
                                    "Ring Indicator". Be cautious when
                                    making cables.
                       3   low   Data Set Ready
                       4   low   Clear To Send
                       5   low   Carrier Detect
                       6   low   Ready To Send
                       7   low   Data Terminal Ready
            MSB   8   high   hardware overrun
                       9   high   break sent (most recent output)
                       10   high   break received (as latest input)
                       11   high   transmit x-OFFed
                       12   high   receive x-OFFed

```
              13-15    ---    reserved
```

io_Actual   set to count of unread input characters

io_Error -- Query will always succeded.

1.15 serial.device/SDCMD_SETPARAMS

```
NAME
    SetParams -- change parameters for the serial port

FUNCTION
    This command allows the caller to change parameters for the
    serial device. Except for xON-xOFF enable/disable, it will
    reject a setparams call if any reads or writes are active
    or pending.
        Note specifically:

 1. Valid input for io_Baud is between 112 and 292000 baud inclusive;
    asynchronous i/o above 32KB (especially on a busy system) may
    be ambitious.
 2. The EOFMODE and QUEUEDBRK bits of io_SerFlags can be set/reset
    in the io_Rqst block without a call to SetParams. The SHARED
    and 7WIRE bits of io_SerFlags can be used in OpenDevice calls.
    ALL OTHER PARAMETERS CAN ONLY BE CHANGED BY THE SetParams
    COMMAND.
 3. RBufLen must be at least 64.  The buffer may be any multiple of
64 bytes.
 4. If not used, io_ExtFlags MUST be set to zero.
 5. xON-xOFF is by default enabled. The XDISABLED bit is the only
    parameter that can be changed via a SetParams call while the
    device is active. Note that this will return the value
    SerErr_DevBusy in the io_Error field.

xON/xOFF handshaking is inappropriate for certain binary transfer
protocalls, such as Xmodem.  The binary data might contain the
xON (ASCII 17) and xOFF (ASCII 19) characters.

 6. If trying to run MIDI, you should set the RAD_BOOGIE bit of
    io_SerFlags to eliminate unneeded overhead. Specifically, this skips
    checks for parity, x-OFF handling, character lengths other than
    8 bits, and testing for a break signal. Setting RAD_BOOGIE will
    also set the XDISABLED bit.
    Note that writing data (that's already in MIDI format) at MIDI rates
    is easily accomplished. Using this driver alone for MIDI reads may,
    however, may not be reliable, due to MIDI timestamping requirements,
    and possibility of overruns in a busy multitasking and/or display
    intensive environment.
 7. If you select mark or space parity (see io_ExtFlags in serial.h),
    this will cause the SERB_PARTY_ON bit to be set, and the setting
    of SERB_PARTY_ODD to be ignored.
 8. For best results, set the RAD_BOOGIE flag whenever possible.  See
#6 for details.
 9. Note that at this time parity is *not* calculated for the xON-xOFF
characters.  If you have a system that is picky about the parity of
```

these, you must set your own xON-xOFF characters in io_CtlChar.
 10. 7WIRE (CTS/RTS) handshake is bi-directional. The external side
is expected to drop CTS several character times before the external
buffer is full. The Amiga will drop RTS several character times
before the Amiga's buffer is full.

  IO REQUEST
      io_Message   mn_ReplyPort initialized
      io_Device   preset by OpenDevice
      io_Unit   preset by OpenDevice
      io_Command   SDCMD_SETPARAMS (0x0B)
       NOTE that the following fields are filled in by Open
                      to reflect the serial device's current configuration.
      io_CtlChar   a longword containing byte values for the
                      xON,xOFF,INQ,ACK fields (respectively)
                      (INQ/ACK not used at this time)
      io_RBufLen   length in bytes of input buffer
       NOTE that any change in buffer size causes the
                      current buffer to be deallocated and a new,
                      correctly sized one to be allocated. Thusly,
                      the CONTENTS OF THE OLD BUFFER ARE LOST.
      io_ExtFlags   additional serial flags (bitdefs in devices/serial.h)
      mark & space parity may be specified here.
      io_Baud   baud rate for reads AND writes. (See 1 above)
      io_BrkTime   duration of break signal in MICROseconds
      io_TermArray   ASCII descending-ordered 8-byte array of
                      termination characters. If less than 8 chars
                      used, fill out array w/lowest valid value.
                      Terminators are checked only if EOFMODE bit of
                      io_Serflags is set. (e.g. x512F040303030303)
      io_ReadLen   number of bits in read word (1-8) not including parity
      io_WriteLen   number of bits in write word (1-8) "   "   "
      io_StopBits   number of stop bits (0, 1 or 2)
      io_SerFlags   see devices/serial.h for bit equates, NOTE that x00
                      yields exclusive access, xON/OFF-enabled, no
                      parity checking, 3-wire protocol and TermArray
                      inactive.

  RESULTS
      Error -- if the SetParams succeded, then Error will be null.
         If the SetParams failed, then the Error will be non-zero.

  SEE ALSO
exec/OpenDevice