Devices

COLLABORATORS			
	TITLE : Devices		
ACTION	*NAME*	*DATE*	*SIGNATURE*
WRITTEN BY		March 28, 2025	

REVISION HISTORY			
NUMBER	DATE	DESCRIPTION	NAME

Contents

Chapter 1

Devices

1.1 Amiga® RKM Devices: 9 / Parallel Device

The parallel device provides a hardware-independent interface to the Amiga's Centronics-compatible parallel port. The primary use of the Amiga parallel port is for output to printers, but with its extensions for bi-directional I/O, it can also be used for communication with digitizers and high-speed links with other computers. The parallel device is based on the conventions of Exec device I/O, with extensions for parameter setting and control.

1.2 9 Parallel Device / Parallel Device Commands and Functions

Command	Operation
CMD_FLUSH	Purge all queued requests for the parallel device. Does not affect active requests.
CMD_READ	Read a stream of characters from the parallel port. The number of characters can be specified or a termination character(s) can be used.
CMD_RESET	Reset the parallel port to its initialized state. All active and queued I/O requests will be aborted.
CMD_START	Restart all paused I/O over the parallel port. Reactivates the handshaking sequence.
CMD_STOP	Pause all active I/O over the parallel port. Deactivates the handshaking sequence.

```
CMD_WRITE        Write out a stream of characters to the parallel port.
                 The number of characters can be specified or a
                 NULL-terminated string can be sent.

PDCMD_QUERY      Return the status of the parallel port lines and
                 registers.

PDCMD_SETPARAMS  Set the parameters of the parallel port.


Exec Functions as Used in This Chapter
--------------------------------------
AbortIO()        Abort a command to the parallel device. If the command is
                 in progress, it is stopped immediately.  If it is queued,
                 it is removed from the queue.

BeginIO()        Initiate a command and return immediately (asynchronous
                 request).  This is used to minimize the amount of system
                 overhead.

CheckIO()        Determine the current state of an I/O request.

CloseDevice()    Relinquish use of the parallel device.  All requests must
                 be complete.

DoIO()           Initiate a command and wait for completion (synchronous
                 request).

OpenDevice()     Obtain use of the parallel device.

SendIO()         Initiate a command and return immediately (asynchronous
                 request).

WaitIO()         Wait for the completion of an asynchronous request.  When
                 the request is complete the message will be removed from
                 your reply port.


Exec Support Functions as Used in This Chapter
----------------------------------------------
CreateExtIO()    Create an extended I/O request structure of type
                 IOExtPar.  This structure will be used to communicate
                 commands to the parallel device.

CreatePort()     Create a signal message port for reply messages from the
                 parallel device.  Exec will signal a task when a message
                 arrives at the port.

DeleteExtIO()    Delete an extended I/O request structure created by
                 CreateExtIO().

DeletePort()     Delete the message port created by CreatePort().
```

1.3 9 Parallel Device / Device Interface

The parallel device operates like the other Amiga devices. To use it, you must first open the parallel device, then send I/O requests to it, and then close it when finished. See "Introduction to Amiga System Devices" chapter for general information on device usage.

The I/O request used by the parallel device is called IOExtPar.

```
    struct   IOExtPar
        {
        struct  IOStdReq IOPar;
        ULONG   io_PExtFlags;   /* additional parallel flags */
        UBYTE   io_Status;      /* status of parallel port and registers */
        UBYTE   io_ParFlags;    /* parallel device flags */
        struct  IOPArray io_PTermArray; /* termination character array */
        };
```

See the include file devices/parallel.h for the complete structure definition.

Opening The Parallel Device Writing To The Parallel Device
Reading From The Parallel Device Closing The Parallel Device

1.4 9 Device Interface / Opening The Parallel Device

Three primary steps are required to open the parallel device:

* Create a message port using CreatePort(). Reply messages from the device must be directed to a message port.

* Create an extended I/O request structure of type IOExtPar using CreateExtIO(). CreateExtIO() will initialize the I/O request to point to your reply port.

* Open the parallel device. Call OpenDevice(),passing the I/O request.

```
struct MsgPort  *ParallelMP;         /* Pointer to reply port */
struct IOExtPar *ParallelIO;         /* Pointer to I/O request */

if (ParallelMP=CreatePort(0,0) )
   if (ParallelIO=(struct IOExtPar *)
           CreateExtIO(ParallelMP,sizeof(struct IOExtPar)) )
      if (OpenDevice(PARALLELNAME,0L,(struct IORequest *)ParallelIO,0) )
            printf("%s did not open\n",PARALLELNAME);
```

During the open, the parallel device pays attention to just one flag; PARF_SHARED. For consistency, the other flag bits should also be properly set. Full descriptions of all flags will be given later. When the parallel device is opened, it fills the latest default parameter settings into the IOExtPar block.

1.5 9 / Device Interface / Reading From The Parallel Device

You read from the parallel device by passing an IOExtPar to the device with CMD_READ set in io_Command, the number of bytes to be read set in io_Length and the address of the read buffer set in io_Data.

```
#define READ_BUFFER_SIZE 256
/* Reserve SIZE bytes of storage */
char ParallelReadBuffer[READ_BUFFER_SIZE];

ParallelIO->IOPar.io_Length   = READ_BUFFER_SIZE;
ParallelIO->IOPar.io_Data     = (APTR)&ParallelReadBuffer[0];
ParallelIO->IOPar.io_Command  = CMD_READ;
DoIO((struct IORequest *)ParallelIO);
```

If you use this example, your task will be put to sleep waiting until the parallel device reads 256 bytes (or terminates early). Early termination can be caused by error conditions.

1.6 9 / Device Interface / Writing To The Parallel Device

You write to the parallel device by passing an IOExtPar to the device with CMD_WRITE set in io_Command, the number of bytes to be written set in io_Length and the address of the write buffer set in io_Data.

To write a NULL-terminated string, set the length to -1; the device will output from your buffer until it encounters and transmits a value of zero (0x00).

```
ParallelIO->IOPar.io_Length   = -1;
ParallelIO->IOPar.io_Data     = (APTR)"Parallel lines cross 7 times...";
ParallelIO->IOPar.io_Command  = CMD_WRITE;
DoIO((struct IORequest *)ParallelIO);             /* execute write */
```

The length of the request is -1, meaning we are writing a NULL-terminated string. The number of characters sent can be found in io_Actual.

1.7 9 / Device Interface / Closing The Parallel Device

Each OpenDevice() must eventually be matched by a call to CloseDevice(). When the last close is performed, the device will deallocate all resources and buffers. The latest parameter settings will be saved for the next open.

All I/O requests must be complete before CloseDevice(). If any requests are still pending, abort them with AbortIO():

```
if (!(CheckIO(ParallelIO)))
    {
    AbortIO(ParallelIO);  /* Ask device to abort request, if pending */
    }
WaitIO(ParallelIO);       /* Wait for abort, then clean up */
CloseDevice((struct IORequest *)ParallelIO);
```

1.8 9 Parallel Device / Ending A Read or Write with Termination Characters

Reads and writes from the parallel device may terminate early if an error occurs or if an end-of-file is sensed. For example, if a break is detected on the line, any current read request will be returned with the error ParErr_DetectedBreak. The count of characters read to that point will be in the io_Actual field of the request.

You can specify a set of possible end-of-file characters that the parallel device is to look for in the input or output stream using the PDCMD_SETPARAMS command. These are contained in an io_PTermArray that you provide. io_PTermArray is used only when the PARF_EOFMODE flag is selected (see Parallel Flags below).

If EOF mode is selected, each input data character read into or written from the user's data block is compared against those in io_PTermArray. If a match is found, the IOExtPar is terminated as complete, and the count of characters transferred (including the termination character) is stored in io_Actual.

To keep this search overhead as efficient as possible, the parallel device requires that the array of characters be in descending order. The array has eight bytes and all must be valid (that is, do not pad with zeros unless zero is a valid EOF character). Fill to the end of the array with the lowest value termination character. When making an arbitrary choice of EOF character(s), you will get the quickest response from the lowest value(s) available.

```
     Terminate_Parallel.c
```

The read will terminate before the io_Length number of characters is read if a "Q", "E", or "A" is detected.

```
   It's Usually For Output.
   ------------------------
   Most applications for the parallel device use the device for output,
   hence the termination feature is usually done on the output stream.
```

1.9 9 Parallel Device / Setting Parallel Parameters

You can control the parallel parameters shown in the following table. The parameter name within the parallel IOExtPar data structure is shown below. All of the fields described in this section are filled with defaults when you call OpenDevice(). Thus, you need not worry about any parameter that you do not need to change. The parameters are defined in the include file devices/parallel.h.

```
                  PARALLEL PARAMETERS (IOExtPar)
   IOExtPar
   Field Name     Parallel Device Parameter It Controls
   ----------     -------------------------------------
   io_PExtFlags   Reserved for future use.

   io_PTermArray  A byte-array of eight termination characters, must
```

	be in descending order. If EOFMODE is set in the parallel flags, this array specifies eight possible choices of characters to use as an end-of-file mark. See the section above titled "Ending A Read Or Write with Termination Characters" and the PDCMD_SETPARAMS summary page in the Autodocs.
io_Status	Contains status information. It is filled in by the PDCMD_QUERY command.
io_ParFlags	See "Parallel Flags" below.

You set the parallel parameters by passing an IOExtPar to the device with PDCMD_SETPARAMS set in io_Command and with the flags and parameters set to the values you want.

```
ParallelIO->io_ParFlags     &= ~PARF_EOFMODE;   /* Set EOF mode */
ParallelIO->IOPar.io_Command = PDCMD_SETPARAMS; /* Set params command */
if (DoIO(ParallelIO);
    printf("Error setting parameters!\n");
```

The above code fragment modifies one bit in io_ParFlags, then sends the command.

Proper Time for Parameter Changes.

A parameter change should not be performed while an I/O request is actually being processed, because it might invalidate already active request handling. Therefore you should use PDCMD_SETPARAMS only when you have no parallel I/O requests pending.

Parallel Flags (Bit Definitions For Io_parflags)

1.10 9 / Setting Parallel Parameters / Parallel Flags (Bits for io_ParFlags)

The flags shown in the following table can be set to affect the operation of the parallel device. Note that the default state of all of these flags is zero. The flags are defined in the include file devices/parallel.h.

PARALLEL FLAGS (io_ParFlags)

Flag Name	Effect on Device Operation
PARF_EOFMODE	Set this bit if you want the parallel device to check I/O characters against io_TermArray and terminate the I/O request immediately if an end-of-file character has been encountered. Note: This bit can be set and reset directly in the user's IOExtPar block without a call to PDCMD_SETPARAMS.
PARF_ACKMODE	Set this bit if you want to use ACK handshaking.
PARF_FASTMODE	Set this bit if you want to use

high-speed mode for transfers to high-speed printers. This mode will send out data as long as the BUSY signal is low. The printer must be able to raise the BUSY signal within three microseconds or data will be lost. Should only be used when the device has been opened for exclusive -access.

PARF_SLOWMODE — Set this bit if you want to use slow -speed mode for transfers to very slow printers. Should not be used with high-speed printers.

PARF_SHARED — Set this bit if you want to allow other tasks to simultaneously access the parallel port. The default is exclusive access. If someone already has the port, whether for exclusive or shared access, and you ask for exclusive access, your OpenDevice() call will fail (must be modified before OpenDevice()).

1.11 9 Parallel Device / Querying the Parallel Device

You query the parallel device by passing an IOExtPar to the device with PDCMD_QUERY set in io_Command. The parallel device will respond with the status of the parallel port lines and registers.

```
UWORD Parallel_Status;

ParallelIO->IOPar.io_Command  = PDCMD_QUERY; /* indicate query */
DoIO((struct IORequest *)ParallelIO);

Parallel_Status = ParallelIO->io_Status;     /* store returned status */
```

The 8 status bits of the parallel device are returned in io_Status.

PARALLEL DEVICE STATUS BITS

Bit	Active	Function
0	high	Printer busy toggle (offline)
1	high	Paper out
2	high	Printer Select on the A1000. On theA500 and A2000, select is also connected to to the parallel port's Ring Indicator. Be cautious when making cables.
3	-	read=0; write=1
4-7	-	(reserved)

The parallel device also returns error codes whenever an operation is attempted.

```
struct IOPArray Terminators =
{
0x51454141,   /* Q E A A */
0x41414141    /* fill to end with lowest value,must be in desc. order */
};
```

```
   ParallellIO->io_ParFlags != PARF_EOFMODE;        /* Set EOF mode flag */
   ParallelIO->io_PTermArray = Terminators;/* Set termination characters */
   ParallellIO->IOPar.io_Command  = PDCMD_SETPARAMS;   /* Set parameters */
   if (DoIO((struct IORequest *)ParallelIO))
       printf("Set Params failed. Error: %ld ",ParallelIO->IOPar.io_Error);
```

The error is returned in the io_Error field of the IOExtPar structure.

PARALLEL DEVICE ERROR CODES

Error	Value	Explanation
ParErr_DevBusy	1	Device in use
ParErr_BufToBig	2	Out of memory
ParErr_InvParam	3	Invalid parameter
ParErr_LineErr	4	Parallel line error
ParErr_NotOpen	5	Device not open
ParErr_PortReset	6	Port Reset
ParErr_InitErr	7	Initialization Error

Parallel.c

1.12 9 Parallel Device / Additional Information on the Parallel Device

Additional programming information on the parallel device can be found in the include files and the Autodocs for the parallel device. Both are contained in the Amiga ROM Kernel Reference Manual: Includes and Autodocs.

Parallel Device Information height	
INCLUDES	devices/parallel.h devices/parallel.i
AUTODOCS	parallel.doc