

console

COLLABORATORS

	<i>TITLE :</i> console		
<i>ACTION</i>	<i>NAME</i>	<i>DATE</i>	<i>SIGNATURE</i>
WRITTEN BY		March 28, 2025	

REVISION HISTORY

NUMBER	DATE	DESCRIPTION	NAME

Contents

1	console	1
1.1	console.doc	1
1.2	console.device/CD_ASKDEFAULTKEYMAP	1
1.3	console.device/CD_ASKKEYMAP	2
1.4	console.device/CD_SETDEFAULTKEYMAP	2
1.5	console.device/CD_SETKEYMAP	3
1.6	console.device/CDInputHandler	3
1.7	console.device/CMD_CLEAR	4
1.8	console.device/CMD_READ	5
1.9	console.device/CMD_WRITE	5
1.10	console.device/OpenDevice	9
1.11	console.device/RawKeyConvert	11

Chapter 1

console

1.1 console.doc

CD_ASKDEFAULTKEYMAP	CDInputHandler()	OpenDevice()
CD_ASKKEYMAP	CMD_CLEAR	RawKeyConvert()
CD_SETDEFAULTKEYMAP	CMD_READ	
CD_SETKEYMAP	CMD_WRITE	

1.2 console.device/CD_ASKDEFAULTKEYMAP

NAME

CD_ASKDEFAULTKEYMAP -- get the current default keymap

FUNCTION

Fill the `io_Data` buffer with the current console device default keymap, which is used to initialize console unit keymaps when opened, and by `RawKeyConvert` with a null `keyMap` parameter.

IO REQUEST

<code>io_Message</code>	<code>mn_ReplyPort</code> set if quick I/O is not possible
<code>io_Device</code>	preset by the call to <code>OpenDevice</code>
<code>io_Unit</code>	preset by the call to <code>OpenDevice</code>
<code>io_Command</code>	<code>CD_ASKDEFAULTKEYMAP</code>
<code>io_Flags</code>	<code>IOF_QUICK</code> if quick I/O possible, else zero
<code>io_Length</code>	<code>sizeof(*keyMap)</code>
<code>io_Data</code>	<code>struct KeyMap *keyMap</code> pointer to a structure that describes the raw keycode to byte stream conversion.

RESULTS

This function sets the `io_Error` field in the `IOStdReq`, and fills the structure pointed to by `io_Data` with the current device default key map.

BUGS

SEE ALSO

exec/io.h, devices/keymap.h, devices/console.h

1.3 console.device/CD_ASKKEYMAP

NAME

CD_ASKKEYMAP -- Get the current key map structure for this console.

FUNCTION

Fill the io_Data buffer with the current KeyMap structure in use by this console unit.

IO REQUEST INPUT

io_Message	mn_ReplyPort set if quick I/O is not possible
io_Device	preset by the call to OpenDevice
io_Unit	preset by the call to OpenDevice
io_Command	CD_ASKKEYMAP
io_Flags	IOF_QUICK if quick I/O possible, else zero
io_Length	sizeof(*keyMap)
io_Data	struct KeyMap *keyMap pointer to a structure that describes the raw keycode to byte stream conversion.

IO REQUEST RESULT

This function sets the io_Error field in the IOStdReq, and fills the structure the structure pointed to by io_Data with the current key map.

SEE ALSO

exec/io.h, devices/keymap.h, devices/console.h

1.4 console.device/CD_SETDEFAULTKEYMAP

NAME

CD_SETDEFAULTKEYMAP -- set the current default keymap

FUNCTION

This console command copies/uses the keyMap structure pointed to by io_Data to the console device default keymap, which is used to initialize console units when opened, and by RawKeyConvert with a null keyMap parameter.

IO REQUEST

io_Message	mn_ReplyPort set if quick I/O is not possible
io_Device	preset by the call to OpenDevice
io_Unit	preset by the call to OpenDevice
io_Command	CD_SETDEFAULTKEYMAP
io_Flags	IOF_QUICK if quick I/O possible, else zero
io_Length	sizeof(*keyMap)
io_Data	struct KeyMap *keyMap pointer to a structure that describes the raw keycode to byte stream conversion.

RESULTS

This function sets the `io_Error` field in the `IOStdReq`, and fills the current device default key map from the structure pointed to by `io_Data`.

BUGS

As of V36, this command no longer copies the keymap structure, and the keymap must remain in memory until the default key map is changed. In general there is no reason for applications to use this command. The default key map will generally be set by the user using a system provided command/tool.

SEE ALSO

`exec/io.h`, `devices/keymap.h`, `devices/console.h`

1.5 console.device/CD_SETKEYMAP

NAME

`CD_SETKEYMAP` -- set the current key map structure for this console

FUNCTION

Set the current `KeyMap` structure used by this console unit to the structure pointed to by `io_Data`.

IO REQUEST

<code>io_Message</code>	<code>mn_ReplyPort</code> set if quick I/O is not possible
<code>io_Device</code>	preset by the call to <code>OpenDevice</code>
<code>io_Unit</code>	preset by the call to <code>OpenDevice</code>
<code>io_Command</code>	<code>CD_SETKEYMAP</code>
<code>io_Flags</code>	<code>IOF_QUICK</code> if quick I/O possible, else zero
<code>io_Length</code>	<code>sizeof(*keyMap)</code>
<code>io_Data</code>	<code>struct KeyMap *keyMap</code> pointer to a structure that describes the raw keycode to byte stream conversion.

RESULTS

This function sets the `io_Error` field in the `IOStdReq`, and fills the current key map from the structure pointed to by `io_Data`.

BUGS

SEE ALSO

`exec/io.h`, `devices/keymap.h`, `devices/console.h`

1.6 console.device/CDInputHandler

NAME

`CDInputHandler` -- handle an input event for the console device

SYNOPSIS

```
events = CDInputHandler(events, consoleDevice)
                a0          a1
```

FUNCTION

Accept input events from the producer, which is usually the
rom input.task.

INPUTS

events - a pointer to a list of input events.
consoleDevice - a pointer to the library base address of the
console device. This has the same value as ConsoleDevice
described below.

RESULTS

events - a pointer to a list of input events not used by this
handler.

NOTES

This function is available for historical reasons. It is
preferred that input events be fed to the system via the
WriteEvent command of the input.device.

This function is different from standard device commands in
that it is a function in the console device library vectors.
In order to obtain a valid library base pointer for the
console device (a.k.a. ConsoleDevice) call
OpenDevice("console.device", -1, IOStdReq, 0),
and then grab the io_Device pointer field out of the IOStdReq
and use as ConsoleDevice.

BUGS**SEE ALSO**

input.device

1.7 console.device/CMD_CLEAR

NAME

CMD_CLEAR -- Clear console input buffer.

FUNCTION

Remove from the console input buffer any reports waiting to
satisfy read requests.

IO REQUEST INPUT

io_Message	mn_ReplyPort set if quick I/O is not possible
io_Device	preset by the call to OpenDevice
io_Unit	preset by the call to OpenDevice
io_Command	CMD_CLEAR
io_Flags	IOB_QUICK set if quick I/O is possible, else 0

SEE ALSO

exec/io.h, devices/console.h

1.8 console.device/CMD_READ

NAME

CMD_READ -- return the next input from the keyboard

FUNCTION

Read the next input, generally from the keyboard. The form of this input is as an ANSI byte stream: i.e. either ASCII text or control sequences. Raw input events received by the console device can be selectively filtered via the aSRE and aRRE control sequences (see the write command). Keys are converted via the keymap associated with the unit, which is modified with AskKeyMap and SetKeyMap

If, for example, raw keycodes had been enabled by writing <CSI>1{ to the console (where <CSI> is \$9B or Esc[]), keys would return raw keycode reports with the information from the input event itself, in the form:
<CSI>1;0;<keycode>;<modifiers>;0;0;<seconds>;<microseconds>q

If there is no pending input, this command will not be satisfied, but if there is some input, but not as much as can fill io_Length, the request will be satisfied with the input currently available.

IO REQUEST

io_Message	mn_ReplyPort set if quick I/O is not possible
io_Device	preset by the call to OpenDevice
io_Unit	preset by the call to OpenDevice
io_Command	CMD_READ
io_Flags	IOF_QUICK if quick I/O possible, else zero
io_Length	sizeof(*buffer)
io_Data	char buffer[] a pointer to the destination for the characters to read from the keyboard.

RESULTS

This function sets the error field in the IOStdReq, and fills in the io_Data area with the next input, and io_Actual with the number of bytes read.

BUGS

SEE ALSO

exec/io.h, devices/console.h

1.9 console.device/CMD_WRITE

NAME

CMD_WRITE -- Write ANSI text to the console display.

FUNCTION

Write a text record to the display. Interpret the ANSI control characters in the data as described below. Note

that the RPort of the console window is in use while this write command is pending.

IO REQUEST INPUT

io_Message	mn_ReplyPort set if quick I/O is not possible
io_Device	preset by the call to OpenDevice
io_Unit	preset by the call to OpenDevice
io_Command	CMD_WRITE
io_Flags	IOF_QUICK if quick I/O possible, else zero
io_Length	sizeof(*buffer), or -1 if io_Data is null terminated
io_Data	a pointer to a buffer containing the ANSI text to write to the console device.

IO REQUEST RESULTS

io_Error	the error result (no errors are reported as of V36)
io_Actual	the number of bytes written from io_Data
io_Length	zero
io_Data	original io_Data plus io_Actual

ANSI CODES SUPPORTED

Codes are specified in the standard fashion for ANSI documents, as the two 4 bit nibbles that comprise the character code, high nibble first, separated by a slash. Thus 01/11 (ESC) is a character with the hex value 1B (or the decimal value 27).

A character on the Amiga falls into one of the following four ranges:

00/ 0-01/15	C0: ASCII control characters. See below.
02/ 0-07/15	G0: ASCII graphic characters. These characters have an image that is displayed. Note that the DEL character is displayed by the Console Device: it is not treated as control character here.
08/ 0-09/15	C1: ANSI 3.41 control characters. See below.
10/ 0-15/15	G1: ECMA 94 Latin 1 graphic characters.

Independent Control Functions (no introducer) --

Code	Name	Definition
00/ 7	BEL	BELL: actually an Intuition DisplayBeep()
00/ 8	BS	BACKSPACE
00/ 9	HT	HORIZONTAL TAB
00/10	LF	LINE FEED
00/11	VT	VERTICAL TAB
00/12	FF	FORM FEED
00/13	CR	CARRIAGE RETURN
00/14	SO	SHIFT OUT: causes all subsequent G0 (ASCII) characters to be shifted to G1 (ECMA 94/1) characters.
00/15	SI	SHIFT IN: cancels the effect of SHIFT OUT.
01/11	ESC	ESCAPE

Code or Esc Name Definition

Code	or Esc	Name	Definition
08/ 4	D	IND	INDEX: move the active position down one line.
08/ 5	E	NEL	NEXT LINE

```

08/ 8   H   HTS   HORIZONTAL TABULATION SET
08/13  M   RI   REVERSE INDEX
09/11  [   CSI   CONTROL SEQUENCE INTRODUCER: see next list

```

ISO Compatible Escape Sequences (introduced by Esc) --

Esc Name Definition

```

-----
c      RIS   RESET TO INITIAL STATE: reset the console display.

```

Control Sequences, with the number of indicated parameters.
i.e. <CSI><parameters><control sequence letter(s)>. Note the
last entries consist of a space and a letter. CSI is either
9B or Esc[. A minus after the number of parameters (#p)
indicates less is valid. Parameters are separated by
semicolons, e.g. Esc[14;80H sets the cursor position to row
14, column 80.

CSI #p Name Definition

```

-----
@   1-  ICH   INSERT CHARACTER
A   1-  CUU   CURSOR UP
B   1-  CUD   CURSOR DOWN
C   1-  CUF   CURSOR FORWARD
D   1-  CUB   CURSOR BACKWARD
E   1-  CNL   CURSOR NEXT LINE
F   1-  CPL   CURSOR PRECEDING LINE
H   2-  CUP   CURSOR POSITION
I   1-  CHT   CURSOR HORIZONTAL TABULATION
J   1-  ED    ERASE IN DISPLAY (only to end of display)
K   1-  EL    ERASE IN LINE (only to end of line)
L   1-  IL    INSERT LINE
M   1-  DL    DELETE LINE
P   1-  DCH   DELETE CHARACTER
R   2   CPR   CURSOR POSITION REPORT (in Read stream only)
S   1-  SU    SCROLL UP
T   1-  SD    SCROLL DOWN
W   n   CTC   CURSOR TABULATION CONTROL
Z   1-  CBT   CURSOR BACKWARD TABULATION
f   2-  HVP   HORIZONTAL AND VERTICAL POSITION
g   1-  TBC   TABULATION CLEAR
h   n   SM    SET MODE: see modes below.
l   n   RM    RESET MODE: see modes below.
m   n   SGR   SELECT GRAPHIC RENDITION
n   1-  DSR   DEVICE STATUS REPORT
t   1-  aSLPP SET PAGE LENGTH (private Amiga sequence)
u   1-  aSLL  SET LINE LENGTH (private Amiga sequence)
x   1-  aSLO  SET LEFT OFFSET (private Amiga sequence)
y   1-  aSTO  SET TOP OFFSET (private Amiga sequence)
{   n   aSRE  SET RAW EVENTS (private Amiga sequence)
|   8   aIER  INPUT EVENT REPORT (private Amiga Read sequence)
}   n   aRRE  RESET RAW EVENTS (private Amiga sequence)
~   1   aSKR  SPECIAL KEY REPORT (private Amiga Read sequence)
p   1-  aSCR  SET CURSOR RENDITION (private Amiga sequence)
q   0   aWSR  WINDOW STATUS REQUEST (private Amiga sequence)
r   4   aWBR  WINDOW BOUNDS REPORT (private Amiga Read sequence)
v   1   aRAV  RIGHT AMIGA V PRESS (private Amiga Read sequence-V37)

```

Modes, set with <CSI><mode-list>h, and cleared with

<CSI><mode-list>l, where the mode-list is one or more of the following parameters, separated by semicolons --

Mode Name Definition

```
-----
20      LNM  LINEFEED NEWLINE MODE: if a linefeed is a newline
>1      ASM  AUTO SCROLL MODE: if scroll at bottom of window
?7      AWM  AUTO WRAP MODE: if wrap at right edge of window
```

NOTES

The console.device recognizes these SGR sequences.

Note that some of these are new to V36.

SGR (SELECT GRAPHICS RENDITION)

Selects colors, and other display characteristics for text.

Syntax:

<ESC>[graphic-rendition...m

Example:

<ESC>[1;7m (sets bold, and reversed text)

Parameters:

```

0          - Normal colors, and attributes
1          - Set bold
2          - Set faint (secondary color)
3          - Set italic
4          - Set underscore
7          - Set reversed character/cell colors
8          - Set concealed mode.
22         - Set normal color, not bold      (V36)
23         - Italic off                      (V36)
24         - Underscore off                  (V36)
27         - Reversed off                    (V36)
28         - Concealed off                   (V36)

30-37      - Set character color
39         - Reset to default character color

40-47      - Set character cell color
49         - Reset to default character cell color

>0-7       - Set background color            (V36)
            Used to set the background color before
            any text is written. The numeric parameter
            is prefixed by ">". This also means that if
            you issue an SGR command with more than one
            parameter, you must issue the digit only
            parameters first, followed by any prefixed
            parameters.
```

BUGS

Does not correctly display cursor in SuperBitMap layers for versions prior to V36.

SEE ALSO

ROM Kernal Manual (Volume 1), exec/io.h

1.10 console.device/OpenDevice

NAME

OpenDevice -- a request to open a Console device

SYNOPSIS

```
error = OpenDevice("console.device", unit, IOStdReq, flags )
d0                a0                d0    a1        d1
```

FUNCTION

The open routine grants access to a device. There are two fields in the IOStdReq block that will be filled in: the io_Device field and possibly the io_Unit field.

As of (V37) the flags field may also be filled in with a value described below (see conunit.h or conunit.i).

This open command differs from most other device open commands in that it requires some information to be supplied in the io_Data field of the IOStdReq block. This initialization information supplies the window that is used by the console device for output.

The unit number that is a standard parameter for an open call is used specially by this device. See conunit.h, or conunit.i for defined valid unit numbers.

unit number: -1 (CONU_LIBRARY)

Used to get a pointer to the device library vector which is returned in the io_Device field of the IOStdReq block. No actual console is opened. You must still close the device when you are done with it.

unit number: 0 (CONU_STANDARD)

A unit number of zero binds the supplied window to a unique console. Sharing a console must be done at a level higher than the device.

unit number: 1 (CONU_CHARMAP) (V36)

A unit number of one is similar to a unit number of zero, but a console map is also created, and maintained by the console.device. The character map is used by the console device to restore obscured portions of windows which are revealed, and to redraw a window after a resize. Character mapped console.device windows must be opened as SIMPLE REFRESH windows.

The character map is currently for internal use

only, and is not accessible by the programmer. The character map stores characters, attributes, and style information for each character written with the CMD_WRITE command.

unit number: 3 (CONU_SNIPMAP) (V36)

A unit number of three is similar to a unit number of one, but also gives the user the ability to highlight text with the mouse which can be copied by pressing RIGHT AMIGA C. See NOTES below.

flags: 0 (CONFLAG_DEFAULT)

The flags field should be set to 0 under V34, or less.

flags: 1 (CONFLAG_NODRAW_ON_NEWSIZE) (V37)

The flags field can be set to 0, or 1 as of V37. The flags field is ignored under V36, so can be set, though it will have no effect. When set to 1, it means that you don't want the console.device to redraw the window when the window size is changed (assuming you have opened the console.device with a character map - unit numbers 1, or 3). This flag is ignored if you have opened a console.device with a unit number of 0. Typically you would use this flag when you want to perform your own window refresh on a newsiz, and you want the benefits of a character mapped console.

IO REQUEST

io_Data	struct Window *window
	This is the window that will be used for this console. It must be supplied if the unit in the OpenDevice call is 0 (see above). The RPort of this window is potentially in use by the console whenever there is an outstanding write command.

INPUTS

"console.device" - a pointer to the name of the device to be opened.
unit - the unit number to open on that device.
IOStdReq - a pointer to a standard request block
0 - a flag field of zero (CONFLAG_DEFAULT)
1 - a flag field of one (CONFLAG_NODRAW_ON_NEWSIZE) (V37)

RESULTS

error - zero if successful, else an error is returned.

NOTES

As noted above, opening the console.device with a unit number of 3 allows the user to drag select text, and copy the selection with RIGHT AMIGA C. The snip is copied to a private buffered managed by the console.device (as of V36). The snip can be copied to any console.device window unless you are running a console to clipboard utility such as that provided with V37.

The user pastes text into console.device windows by pressing RIGHT AMIGA V. Both RIGHT AMIGA V, and RIGHT AMIGA C are swallowed

by the `console.device` (unless you have asked for key presses as RAW INPUT EVENTS). Text pasted in this way appears in the console read stream as if the user had typed all of the characters manually. Additional input (e.g., user input, RAW INPUT EVENTS) are queued up after pastes. Pastes can theoretically be quite large, though they are no larger than the amount of text which is visible in a `console.device` window.

When running the console to clipboard utility, text snips are copied to the `clipboard.device`, and RIGHT AMIGA V key presses are broadcast as an escape sequence as part of the `console.device` read stream ("`<CSI>0 v`" - \$9B,\$30,\$20,\$76).

It is left up to the application to decide what to do when this escape sequence is received. Ideally the application will read the contents of the clipboard, and paste the text by using successive writes to the `console.device`.

Because the contents of the `clipboard.device` can be quite large, your program should limit the size of writes to something reasonable (e.g., no more than 1K characters per `CMD_WRITE`, and ideally no more than 256 characters per write). Your program should continue to read events from the `console.device` looking for user input, and possibly RAW INPUT EVENTS. How you decide to deal with these events is left up to the application.

If you are using a character mapped console you should receive Intuition events as RAW INPUT EVENTS from the `console.device`. By doing this you will hear about these events after the console device does. This allows the `console.device` to deal with events such as window resizing, and refresh before your application.

BUGS

SEE ALSO

`exec/io.h`, `intuition/intuition.h`

1.11 console.device/RawKeyConvert

NAME

`RawKeyConvert` -- decode raw input classes

SYNOPSIS

```
actual = RawKeyConvert(event, buffer, length, keyMap)
D0                      A0      A1      D1      A2
```

ConsoleDevice in A6 if called from Assembly Language.

FUNCTION

This console function converts input events of type IECLASS_RAWKEY to ANSI bytes, based on the `keyMap`, and places the result into the buffer.

INPUTS

`event` - an `InputEvent` structure pointer.

buffer - a byte buffer large enough to hold all anticipated characters generated by this conversion.
length - maximum anticipation, i.e. the buffer size in bytes.
keyMap - a KeyMap structure pointer, or null if the default console device key map is to be used.

RESULTS

actual - the number of characters in the buffer, or -1 if a buffer overflow was about to occur.

ERRORS

if actual is -1, a buffer overflow condition was detected.
Not all of the characters in the buffer are valid.

NOTES

This function is different from standard device commands in that it is a function in the console device library vectors. In order to obtain a valid library base pointer for the console device (a.k.a. ConsoleDevice) call `OpenDevice("console.device", -1, IOStdReq, 0)`, and then grab the `io_Device` pointer field out of the `IOStdReq` and use as `ConsoleDevice`.

BUGS

SEE ALSO

`exec/io.h`, `devices/inpustevent.h`, `devices/keymap.h`
