DOS Device Drivers

A Debugging Utility and Definition

by
Stephen A. Rodgers

History

Version 1.0 of DOS had no provision for installable device drivers. All device-dependent code was contained within a module in DOS. These "pre-installed" device drivers made it a formidable task to install add-on devices such as external hard disks. Users were stuck with the device complement supplied by the manufacturer of the computer.

Installable device driver support started with DOS 2.0. Installable device drivers were to allow users and third party hardware vendors to write special purpose device drivers for hardware not supported by DOS, or to replace a pre-installed device driver with an enhanced version (such as an interrupt-driven COM driver).

Later releases of DOS enhanced the device driver concept by adding more functions to the device driver interface. DOS 3.0 added open/close/removable media bit, and DOS 3.2 added the generic I/O request that made it possible to format hard disks through the device driver interface. Tables 1 and 2 show the history of device level function codes and IOCTL subfunction calls.

Debugging a Device Driver

One of the most difficult tasks a developer can undertake is debugging a device driver. Unless you want to add a large amount of debug code to a driver, you don't know where DOS is going to load the driver into memory. A utility that would display this information would be very useful, allowing you to debug a device driver within the DOS environment.

Initially, a device driver should be debugged outside the DOS environment before it's installed in CONFIG.SYS. This is best accomplished by writing a front end for the device driver that builds request headers and issues far calls to the strategy and interrupt procedures. This is especially true when debugging the INIT procedure within the device driver since a successful load depends on the proper operation of this code.

How DOS Installs Device Drivers

DOS loads installable device drivers when it is performing boot-up or SYSINIT. The order in which DOS installs a particular device driver is determined by the block/character attribute in the device header. If the attribute is set to block, DOS installs the device in the chain following the base device drivers in the BIOS module. If the at-

Table 1. Device Level Function Codes

		DOS Version Numbers					
Function		2.0	2.1	3.0	3.1	3.2	3.3
0	INIT	*	*	*	*	*	*
1	MEDIA CHECK	*	*	*	*	*	*
2	BUILD BPB	*	*	*	*	*	*
3	IOCTL INPUT	*	*	*	*	*	*
4	INPUT	*	*	*	*	*	*
5	NONDEST. INPUT	*	*	*	*	*	*
6	INPUT STATUS	*	*	*	*	*	*
7	INPUT FLUSH	*	*	*	*	*	*
8	OUTPUT	*	*	*	*	*	*
9	OUTPUT VERIFY	*	*	*	*	*	*
10	OUTPUT STATUS	*	*	*	*	*	*
11	OUTPUT FLUSH	*	*	*	*	*	*
12	IOCTL OUTPUT	*	*	*	*	*	*
13	DEVICE OPEN			*	*	*	*
14	DEVICE CLOSE			*	*	*	*
15	REMOVABLE MEDIA			*	*	*	*
16	OUTPUT UNTIL BUSY			*	*	*	*
19	GENERIC IOCTL					*	*
23	GET LOGICAL DEV.					*	*

Table 2. IOCTL Subfunction Codes

	DOS Version Numbers					
AL	2.0	2.1	3.0	3.1	3.2	3.3
00H GET DEVICE INFORMATION	*	*	*	*	*	*
01H SET DEVICE INFORMATION	*	*	*	*	*	*
02H CHARACTER DEVICE READ	*	*	*	*	*	*
03H CHARACTER DEVICE WRITE	*	*	*	*	*	*
04H BLOCK DEVICE READ	*	*	*	*	*	*
05H BLOCK DEVICE WRITE	*	*	*	*	*	*
06H GET INPUT STATUS	*	*	*	*	*	*
07H GET OUTPUT STATUS	*	*	*	*	*	*
08H REMOVABLE MEDIA TEST			*	*	*	*
09H REMOTE DEVICE TEST				*	*	*
0AH REMOTE HANDLE TEST				*	*	*
0BH ADJUST SHARING RETRIES				*	*	*
0CH CHARACTER GENERIC IOCTL					*	*
0DH BLOCK GENERIC IOCTL					*	*
0DH GET LOGICAL DEVICE MAP					*	*
0DH SET LOGICAL DEVICE MAP					*	*

Table 3. DDI Device Attribute Bits

BIT	DESCRIPTION	DOS Version Numbers 2.0	2.1	3.0	3.1	3.2	3.3
15	CHAR	*	*	*	*	*	*
14	IOCTL	*	*	*	*	*	*
13	NONIBM/OUTUNTILBSY	*	*	*	*	*	*
11	OPNCLS			*	*	*	*
6	LOGDEV/GENERICIOCTL					*	*
3	STDCLK	*	*	*	*	*	*
2	NULDEV	*	*	*	*	*	*
1	STDOUT	*	*	*	*	*	*
0	STDIN	*	*	*	*	*	*

tribute is set to character, DOS installs the device ahead of all the pre-installed devices so that it may replace a pre-installed device if it has the same name. If character devices are loaded ahead of all other devices in the chain, the last character device specified in CONFIG.SYS will be one away from the head of the device list. The NUL device driver, contained within DOS, is at the head of the device list and therefore can't be replaced.

An installable device driver contains a device header at the beginning of the executable image. This device header contains several pointers, one of which is a pointer to the next device in the chain. Initially, this pointer is equal to –1, or the offset portion points to the next driver in the image. During SYSINIT, DOS adjusts each pointer to point to the next device in the chain. If it is at the last device, DOS leaves the next device pointer initialized to –1 (FFFF:FFFF).

An Information Utility—DDI

Using this information, a linear search can be initiated starting at address 0060:0000. A device name is always 8 characters in length and padded with trailing blanks. If we check for a match with the first character (in this case with an "N") and then check for subsequent character matches, the NUL device will be found somewhere within segment 0060. The position of the NUL device in the DOS image is dependent upon the version of DOS and upon the OEM who customized it.

The information contained in the device header of the NUL device provides a means to "trace" through the device chain since it is essentially at the head of a linked list. If we follow the pointers in all of the device headers, we will be able to trace through the entire device chain, allowing us to locate any block or character device. The end of the device chain is denoted by a pointer having the value FFFF:FFFF.

For block devices, a different approach is taken. Undocumented DOS function call 32H will return the address of the device driver responsible for a particular drive. This function is used to obtain the address of the desired block device.

Using DDI

As mentioned above, DDI has two modes of operation. The first mode displays all loaded device drivers as they are actually chained in the system. The second mode displays detailed information about a specific device. The second mode decodes all attribute bits (see Table 3) and displays the strategy and interrupt entry points. For example, to get information on the CON device, type

```
DDI CON
```

Block devices may be displayed by using the appropriate drive letter. For example, to get the device driver information for the C: drive, type

```
DDI C:
```

♦

Stephen A. Rodgers is Chief Engineer for Innovative Data Technology, a manufacturer of 9 track tape drives and 9 track PC tape subsystems.

Code follows.

Did You Know . . .

The size of the punched card was based on the size of the old-style U.S. paper currency (it was supposed to be a good size for convenient handling.)?—*W.G. Madison* ♦

All code in this issue is available on 360K, MS-DOS disk.
ORDER BY PHONE — VISA/MC orders call toll free 800/234-0386.
Mail order information on pages 8 and 88.

Code from "DOS Device Drivers" by Stephen A. Rodgers

1

```
/* DDI - Device information program
*
* Written By:   Steve Rodgers
* Conception:   5-4-86
*
* Copyright (C) 1987 Steve Rodgers
*
* Functional Description
* ----------------------
*
* This program displays information about installable device
* drivers loaded under DOS 2.0 or later. If the command is typed
* without any arguments, a summary will be printed that shows
* the order of the devices in the chain. If a device name is
* typed following the command, detailed information pertinent to
* that device will be displayed. This program works with either
* character or block devices as arguments.
*
* Use the Microsoft C compiler Version 3.00 or later.
* Compile with the -Zp switch.
*
* Use the Microsoft linker Version 3.00 or later.
*/
#include <stdio.h>
#include <dos.h>
#include <string.h>

/* Device attribute definitions */

#define CHARACTER_DEVICE        0x8000
#define IOCTL_SUPPORT           0x4000
#define NON_IBM_FORMAT          0x2000
#define OPEN_CLOSE_SUPPORT      0x800
#define IS_CLOCK                8
#define IS_NUL                  4
#define IS_STDIN                2
#define IS_STDOUT               1

/* Structure definitions--Device header */
struct DEV_HDR
{
    struct DEV_HDR far      *next_hdr ;     /* pointer next device header */
    unsigned int            attribute ;     /* attribute bits */
    unsigned int            strategy ;      /* strategy offset */
    unsigned int            interrupt ;     /* interrupt offset */
    unsigned char           name[8] ;       /* device name */
} ;

/* Disk Parameter Block - Returned by DOS function 32H */

struct DPB
{
    unsigned char           ad ;            /* current assigned disk */
    unsigned char           alt_ad ;        /* alternate assigned disk */
    unsigned int            bps ;           /* bytes per sector */
    unsigned char           last_sic ;      /* max sector addr in a cluster */
    unsigned char           last_head ;     /* max useable head address */
    unsigned int            rs ;            /* total reserved sectors */
    unsigned char           cf ;            /* copies of the FAT */
    unsigned int            d ;             /* max root directory entries */
    unsigned int            fus ;           /* first useable sector */
    unsigned int            tcoplusone ;    /* total clusters available + 1 */
    unsigned char           spf ;           /* sectors per FAT */
    unsigned int            fds ;           /* first directory sector */
    struct DEV_HDR far      *dda ;          /* device driver address */
    unsigned char           md ;            /* media discriptor */
    struct DISK_DPB far     *next_dpb ;     /* address of next dpb in chain */
    unsigned int            cwd_cluster ;   /* cluster of CWD (DOS 2.x only) */
    unsigned char           cwd_string[64] ; /* CWD string (DOS 2.x only) */
```

2

```
/* Function declarations */

struct DPB far *disk_dpb() ;
struct DEV_HDR far *device_chain() ;

/* Static Declarations */

union REGS ir,or ;
struct SREGS sr ;

int is_disk = 0 ;

/* Attribute bit table */

int attr_bit_masks[] = {CHARACTER_DEVICE,IOCTL_SUPPORT,NON_IBM_FORMAT,
                        OPEN_CLOSE_SUPPORT,IS_CLOCK,IS_NUL,IS_STDIN,IS_STDOUT};

/* Global pointer variables */

struct DEV_HDR far *current_device, far *previous_device ;
struct DPB far *current_dpb ;

/* Global device type counters */

int block_drivers = 0,character_drivers = 0 ;

/* Function: main() */

main(argc,argv)
int argc ;
char *argv[] ;
{
    int i ;
    char *device_name ;

    printf("\nDDI    -              Device Driver Information Program\n") ;
    printf("Version 1.0 (2-18-88) Copyright (C) 1988 Steve Rodgers\n\n") ;

    if(argc < 2) /* if no argument, print the device list */
    {
        list_devices() ;
        exit(0) ;
    }

    if((device_name = strupr(strdup(argv[1]))) == NULL)
        abort() ;
    print_single_device(device_name) ;
}

/* Print information for a single device. */

print_single_device(device_name)
char device_name[] ;
{
    int i ;

/* Get pointer for a specific device.
* If it doesn't exist print an error message. */

    get_device(device_name) ;
    if(current_device == NULL)
    {
        printf("No such device: %s\n",device_name) ;
        exit(1) ;
    }

/* Print the device address, type, and name. */

    print_device_columns() ;
    print_device_info(current_device) ;

/* Print the detailed information for the device. */
```

Code continues . . .

Programmer's Journal
The Resource Journal for IBM Micro Programmers

P. O. Box 30160 • Eugene, OR 97403 • (503) 747-0800

January 5, 1989

Dear *PJ* Reader:

As you've probably noticed by now, page 38 of the January/February 1989 *Programmer's Journal* is missing a column of text. Mistakes like this are easily and often blamed on "printer error." Well, this time it *wasn't* our fault: Our printer really did goof, and they are working with us to make it up to you.

Although SLR Systems is no doubt thrilled to have their ad appear twice in one issue, we're sure you would rather have the material from Jake Richter's article. Therefore, we are having the printer send you a copy of the corrected page on the back of this letter. (If it's not there, they are *really* in trouble.)

We apologize for any inconvenience this blunder may have caused you. If we're all lucky, however, you may have a collector's item on your hands: The issue may contain the last mistake you ever find in *PJ*. (I know . . . and pigs may learn to fly in 1989, too.)

Sincerely yours,

Robert E. Wanetick

Robert E. Wanetick
Managing Editor

A GP command sequence is like a program, but in a very simple language. You load a string of GP commands into the 82786 memory, terminate it with an NOP that has its ECL (End of Command Line) bit set, and then tell the GP to start executing from a specific point in memory. Since the chip does not have any conditional branch instructions, its programmability is severely limited. It does have CALL and RETURN commands, though; so you could set up a generic command list to, for example, draw a box with an *X* in it at the current *x,y* position and make it a subroutine that you could call in from your 82786 "program."

Table 1. Logical Writing/Pixel Processing Mode on the 34010 (The first 16 are also supported by the 82786.)

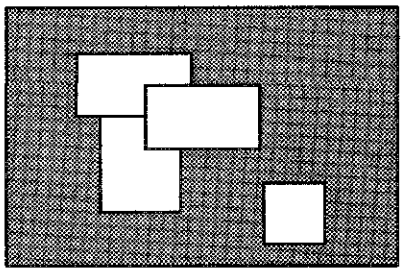
Definitions:

0s	All bits in the pixel are set to 0
1s	All bits in the pixel are set to 1
DST	The pixel at the destination
SRC	The source pixel
AND	Bitwise AND operation
OR	Bitwise OR operation
XOR	Bitwise XOR operation
NOT	Bitwise NOT (one's complement) operation
ADD	Unsigned Addition
SUB	Unsigned Subtraction
ADDS	ADD with Saturation (no wrap, max's out at max pixel value)
SUBS	SUB with Saturation (no wrap, stops at a pixel value of 0)
MAX	Larger of the two operands
MIN	Smaller of the two operands

The result of a pixel processing operation is placed in DST.

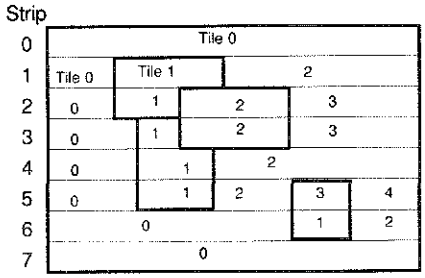
Processing Modes:

```
SRC
SRC AND DST
SRC AND (NOT DST)
0s
SRC OR (NOT DST)
(NOT SRC) XOR DST
NOT DST
(NOT SRC) OR (NOT DST)
SRC OR DST
DST
SRC XOR DST
(NOT SRC) AND DST
1s
(NOT SRC) OR DST
(NOT SRC) AND (NOT DST)
NOT SRC
SRC ADD DST
SRC ADDS DST
DST SUB SRC
DST SUBS SRC
MAX(SRC, DST)
MIN(SRC, DST)
```



Shaded area is field, i.e., not data.

What you see on screen, 4 hardware windows



8 strips, with 1 to 5 tiles each

Strip and tile breakdown of an 82786 screen

The Display Processor is probably the most underutilized feature of the chip. Most graphics-board companies using the 82786 use the DP for driving the monitor output and not much else. But the DP provides two other features: a hardware cursor and hardware windowing.

Hardware windows are similar in concept to the overlapping software windows utilized in such packages as Microsoft Windows. The big difference is that hardware windows do not require a redraw of screen information when the window is moved. This means that you can have virtually simultaneous drawing going on in a number of overlapping windows without worrying about clipping, because the hardware windows only display what's in memory; they do not alter memory. Hardware windows can also display *field*, a term used to indicate non–memory consuming display information ("dead space"). Because all normal limitations are removed, the long-time tie between memory and display is completely altered, requiring programmers to toss out everything they know about displaying PC-based graphics.

Of course, you can still use the one-to-one correlation between memory and display by creating a full-screen hardware window. Hardware windows are rectangular tiles that display data from memory at 1, 2, 4, or 8 bits per pixel. To top that off, a window can be flagged to indicate to the hardware that it should be zoomed. The zoom factor ranges from 1 to 64 in both the *x* and *y* directions. The screen position of a hardware window as well as the contents are determined by the DP display list. This display list consists of a series of strips, each of which contains a number of tiles. There is a limit of 15 tiles per strip, but no limit on the number of strips, providing each one is at least one scan line high.

Although the 82786 is a coprocessor, it does have three sets of registers that can be accessed by an external processor. One set belongs to the DP and is used for setting display frequencies, display list location, cursor pattern, and so forth. Another set belongs to the GP and contains information regarding the current drawing environment, such as *x,y* position, colors, and start of drawing area in memory. The last set of registers is the set containing the processor control. This set can be either memory

Microsoft C Users

Lattice C Users

Frustrated by Long Development Cycles? Constrained by 640K Limitations?

"It's the best of both worlds for the experienced C programmer."
- PC Magazine, September 13, 1988

"...the best product which has ever been written for next generation PC's."
- David Kuechler, Vice President R&D, Program Technologies

Instant-C/16M™ is your solution.

"Unique and highly responsive C programming environment, ... ideally suited for developing C application prototypes."
- PC Week, August 1, 1988

"...best program development environment available anywhere."
- Karim Lakhani, Vice President Technical Development, Fast Trax

Instant-C/16M is the perfect complement to the Microsoft C and Lattice C compilers. You can develop your Microsoft C or Lattice C program in *Instant-C/16M* and save more than half of your development time. Once your program has been completely debugged with *Instant-C/16M*, compile it with Microsoft C or Lattice C for smallest code size and optimal code performance. You get the best of both worlds: fast development and superb debugging from *Instant-C/16M* and fast code from your current C compiler. Development benchmarks consistently show that *Instant-C/16M* is at least **twice as fast** as either Turbo C 1.5, Microsoft C 5.1, or QuickC 1.0 when comparing the turnaround times for compiling and linking.

Instant-C/16M was designed to give the professional programmer the ability to **make much larger programs -- much faster.** By providing access to the full 16MB of extended memory on 80286/386-based PCs, *Instant-C/16M* lets you create any size program you wish with all the features you want. ***Instant-C/16M* is simply the best and fastest way to develop and debug your C programs.**

Instant-C/16M

Instant-C/16M is constructed around an incremental C compiler and includes: a source-level debugger, editor, run-time checker, and library source. ***Instant-C/16M*** lets you add all the features you want to your program without worrying about the 640K limit. Programs of 75,000 lines of code and more than 1MB of dynamically allocated memory are easily handled.

Instant-C/16M's debugging capabilities are unsurpassed. Programs too large for Codeview™ fit easily within ***Instant-C/16M*** to make debugging large programs much easier. Combine ***Instant-C/16M***'s automatic run-time checking with protected mode hardware's checking and you have a "safety net" for catching those particularly difficult bugs that might normally take you hours, days, or even weeks to find and correct.

Features and Benefits

* ***Large Program Development*** -- *Instant-C/16M* gives you access to 16MB of extended memory on 80286/386-based machines. Development of multi-megabyte programs under DOS is no longer a problem. -- **Develop and debug very large programs.**

* ***Instant compilation*** -- Whenever you make a change *Instant-C/16M*'s incremental compiler recompiles only the changed portions of your code, not your entire source file. A typical change takes only one half (.5) to 2 seconds. -- **Saves your valuable development time.**

* ***Instant linking*** -- *Instant-C/16M*'s incremental compiler automatically links your code. -- **Zero link time.**

* ***Integrated environment*** -- The compiler, debugger, editor, run-time checker, and source formatter are tied together. You no longer need to quit one tool to start up another one. And all of *Instant-C/16M*'s tools are just a single keystroke away. -- **Everything is in one package.**

* ***Superior debugging*** -- Source-level debugger automatically traps errors in: invalid pointer references, computed function addresses, overstored return addresses, out of bounds array references, invalid pointer differences, conversion errors, invalid shifts, and arithmetic exceptions. The debugger supports the full C language including local symbols and #defines. -- **The best debugger available for C today.**

* ***Supports your existing programs*** -- *Instant-C/16M* supports the Microsoft and Lattice C compilers and all of your current C source code and libraries. -- **Protects your software investment.**

Instant-C

Instant-C has all the same productivity benefits of *Instant-C/16M* and supports 8086 and 8088-based PCs and compatibles.

Instant-C is designed to run your program at speeds comparable to C compilers. In addition, it lets you generate stand-alone programs -- if you don't need the benefits of a traditional optimizing compiler, don't buy one. ***Instant-C*** can be the only development tool you use.

Just like *Instant-C/16M*, ***Instant-C*** offers you full support for Microsoft C 5.1 and Lattice C 3.2 and support for graphics boards and non-PC hardware.

Why waste any more of your development time? ***Instant-C*** lets you develop your code twice as fast.

The Optimal C Development System

Develop and Debug
in
Instant-C/16M
then
Deliver
with
Your Favorite C Compiler

For More Information Call or Write

Rational Systems, Inc.

P.O. Box 480
220 North Main Street
Natick, MA 01760
Tel: **(508) 653-6006**
FAX: **(508) 655-2753**

Unconditional 30 Day Money Back Guarantee!

Rodgers Code, Continued

3

```
    printf("\n") ;
    printf("                A T T R I B U T E S\n") ;
    printf("-------------------------------------------------------\n") ;
    printf("| CHAR   | IOCTL  | NONIBM | OPNCLS | STDCLK | NULDEV | STDOUT | STDIN |\n") ;
    for(i = 0 ; i < 8 ; i++)
    {
         printf("   ") ;
         if((attr_bit_masks[i] & current_device->attribute))
              printf("yes   ") ;
         else
              printf("no    ") ;
    }
    printf("\n\n") ;
    printf("Strategy entry point\t\t: %04X:%04X\n", FP_SEG(current_device),
        current_device->strategy) ;

    printf("Interrupt entry point\t\t: %04X:%04X\n", FP_SEG(current_device),
        current_device->interrupt) ;

    exit(0) ; /* Return to DOS. */
}

/* Print a summary of all devices in the chain. */

list_devices()
{

/* Get the head of the device list.
* If this isn't possible, print an error message and exit. */

    if((current_device = device_chain()) == NULL)
        no_device_chain() ;

/* Print the column information. */

    print_device_columns() ;

/* Follow the chain until a -1 is reached. */

    while(FP_OFF(current_device) != -1)
    {
         print_device_info(current_device) ;
         current_device = current_device->next_hdr ;
    }
    printf("\nSystem has %d character drivers, and %d block drivers\n",
            character_drivers,block_drivers) ;
}

/* Find character or block device driver.  Fill in current device
* pointer, previous device pointer, and current DPB if block device. */

get_device(d)
char d[] ;
{
    int i ;
    char device_string[9] ;

    current_device = NULL ;
    current_dpb = NULL ;
    if(strlen(d) > 8)
         return ;

/* If there is a colon in the second character position, user
* is requesting information about a block device. */

    if(d[1] != ':')
    {

/* The name specified is probably a character device.
* If a colon follows the device name, delete it from the string. */

       if(d[strlen(d) - 1] == ':') /* kill colon at end of device name */
            d[strlen(d) - 1] = 0 ;

/* Attempt to get the pointer to the first device in the chain.  If it
* comes back NULL, the user probably did not install the trace device driver. */
```

4

```
       if((current_device = device_chain()) == NULL)
            no_device_chain() ;  /* no device chain available */

/* Trace the device chain until device name matches a device in the chain. */

       while(FP_OFF(current_device) != -1)
       {
           for(i = 0 ; i < 8 ; i++)
               device_string[i] = current_device->name[i] ;
               device_string[8] = 0 ;
               if(!strncmp(device_string,d,strlen(d)))
                   if((strlen(d) == 8) || (current_device->name[strlen(d)] == ' '))
                   return ;
               previous_device = current_device ;
               current_device = current_device->next_hdr ;
       }

/* If no match, set the current device to NULL and return. */

            current_device = NULL ;
            return ;
    }

/* A colon was found in the second character position of the device
* name. Check for a valid block device. */

    else
        is_disk = 1 ; /* set a flag to signify a disk device */

    if(strlen(d) > 2)  /* length must be less than or equal to 2 */
        return ;

/* Get the disk DPB for the drive specified. If the disk doesn't exist, the * current_dpb
pointer will be set to NULL. Print a message and exit if this is the case. */

    if((current_dpb = disk_dpb[d[0] - 0x40]) == NULL)
        no_disk_dpb() ;
    current_device = current_dpb->dda ;
    previous_device = NULL ;
    return ;
}

/* Print the headings and columns when listing all device drivers. */

print_device_columns()
{
    printf("Base Address\t\t\tType\t\t\tName/Units\n") ;
    printf("------------\t\t\t----\t\t\t----------\n") ;
}

/* This function prints detailed information for a specific device. */

print_device_info(device)
struct DEV_HDR far *device ;
{
    char device_string[9] ;
    int i ;
    printf("%04X:%04X\t\t\t",FP_SEG(device),FP_OFF(device)) ; /* print base address */
    if(device->attribute & CHARACTER_DEVICE)
    {
        ++character_drivers ;
        for(i = 0 ; i < 8 ; i++)
            device_string[i] = device->name[i] ; /* copy it into local data segment */
        device_string[8] = 0 ;
        printf("Char\t\t\t%s\n",device_string) ; /* print "Char" and name */
    }
    else
    {
        ++block_drivers ;
        printf("Block\t\t\t%u units\n",(int) device->name[0]) ;/* block dev.*/
    }
}
```

Code Continues . . .

5

```
/* Return the address of the Disk Parameter Block (DPB) for the drive
 * specified. This DOS function is undocumented in current DOS documentation. */

struct DPB far *disk_dpb(drive)
int drive ;
{
    char far *dsk_dpb ;

    ir.x.ax = 0x3200 ; /* Function call 32H */
    ir.h.dl = drive ;
    intdosx(&ir,&or,&sr) ;
    if(++or.h.al == 0)
        return (char far *) NULL ;
    FP_SEG(dsk_dpb) = sr.ds ;
    FP_OFF(dsk_dpb) = or.x.bx ;
    return (struct DPB far *) dsk_dpb ;
}

/* Return the address of the first device in the device chain. */

struct DEV_HDR far *device_chain()
{
    struct DEV_HDR far *dev_chain ;
    auto char far *search_ptr = (char far *) 0x00600000 ; /* start search at 0060:0000 */
    static char *nuldev ="NUL     " ;
    unsigned int x,i ;

    for(x = 0, i = 0 ; x < 65535 ; x++)
    {
        if(*search_ptr == *nuldev)    /* if first character matches, check others*/
        {
            for(i = 0 ; i < 8 ; i++)
            {
                if(*(search_ptr + i) != nuldev[i])
                    break ;
            }
            if(i == 8)      /* if i = 8 then the NUL device has been found */
                break ;
        }
        search_ptr++ ;
    }
    if(x == 0)
        return NULL ;

    search_ptr -= 10 ; /* get to device header  */
    return (struct DEV_HDR far *) search_ptr ;
}

/* Print message "Can't find device chain" and exit. */

no_device_chain()
{
    fprintf(stderr,"Can't find device chain\n") ;
    exit(1) ;
}

/* Print  message "Can't get DPB - Possible invalid drive\n" and exit. */

no_disk_dpb()
{
    fprintf(stderr,"Can't get DPB - Possible invalid drive\n") ;
    exit(1) ;
}
```

C CODE FOR THE PC

source code, of course

	MS-DOS File Compatibility Package (create, read, & write MS-DOS file systems on non-MS-DOS computers)	$500
	Bluestreak Plus Communications (two ports, programmer's interface, terminal emulation)	$400
	CQL Query System (SQL retrievals plus windows)	$325
	GraphiC 4.1 (high-resolution, DISSPLA-style scientific plots in color & hardcopy)	$325
	PC Curses (Aspen, Software, System V compatible, extensive documentation)	$250
NEW!	Greenleaf Business Mathlib (exact decimal math, formatting, depreciation, interest, cash flow, statistics)	$250
	Greenleaf Data Windows (windows, menus, data entry, interactive form design)	$220
	Greenleaf Communications Library (interrupt mode, modem control, XON-XOFF)	$175
	TurboTEX (TRIP certified; HP, PS, dot drivers; CM fonts; LaTEX)	$170
NEW!	Sherlock (C debugging aid)	$170
	Essential resident C (TSRify C programs, DOS shared libraries)	$165
	Greenleaf Functions (296 useful C functions, all DOS services)	$160
	Essential C Utility Library (400 useful C functions)	$160
	Essential Communications Library (C functions for RS-232-based communication systems)	$160
	WKS Library Version 2.0 (C program interface to Lotus 1-2-3, dBase, Supercalc 4, Quatro, & Clipper)	$155
	OS/88 (U**x-like operating system, many tools, cross-development from MS-DOS)	$150
	ME Version 2.0 (programmer's editor with C-like macro language by Magma Software; Version 1.31 still $75)	$140
	Turbo G Graphics Library (all popular adapters, hidden line removal)	$135
	Vmem/C (virtual memory manager; least-recently used pager; dynamic expansion of swap file)	$130
	TurboGeometry (library of routines for computational geometry)	$125
	CBTree (B+tree ISAM driver, multiple variable-length keys)	$115
	Minix Operating System (U**x-like operating system, includes manual)	$105
	PC/IP (CMU/MIT TCP/IP implementation for PCs)	$100
	B-Tree Library & ISAM Driver (file system utilities by Softfocus)	$100
NEW!	Tele Operating System (TeleKernel, TeleWindows, TeleFile, & TeleBTree by Ken Berry)	$100
	The Profiler (program execution profile tool)	$100
	QC88 C compiler (ASM output, small model, no longs, floats or bit fields, 80+ function library)	$90
	Wendin Operating System Construction Kit or PCNX, PCVMS O/S Shells	$80
	C Windows Toolkit (pop-up, pull-down, spreadsheet, CGA/EGA/Hercules)	$80
	JATE Async Terminal Emulator (includes file transfer and menu subsystem)	$80
NEW!	Polyglot Lisp-to-C Translator (includes Lisp interpreter, Prolog, and simple calculus prover)	$80
	MultiDOS Plus (DOS-based multitasking, intertask messaging, semaphores)	$80
	XT BIOS Kit (roll your own BIOS with this complete set of basic input/output functions for XTs)	$75
NEW!	TE Editor Developer's Kit (full screen editor, undo command, multiple windows)	$75
	Professional C Windows (lean & mean window and keyboard handler)	$70
	lp (flexible printer driver; most popular printers supported)	$65
	Quincy (interactive C interpreter)	$60
	PTree (parse tree management)	$60
NEW!	Icon-Tools (full-featured icon display and editing system)	$60
NEW!	Polyglot TSR Package (includes reminder, bookmark, virus catcher, cache manager, & speech generator)	$50
	MicroFirm Toolkit (28 Unixesque utilities for MS-DOS)	$50
	HELP! (pop-up help system builder)	$50
	Multi-User BBS (chat, mail, menus, sysop displays; uses Galacticomm modem card)	$50
	Make (macros, all languages, built-in rules)	$50
	Vector-to-Raster Conversion (stroke letters & Tektronix 4010 codes to bitmaps)	$50
	Coder's Prolog (inference engine for use with C programs)	$45
	Virtual Memory System (least recently used swapping)	$40
	C-Notes (pop-up help for C programmers ... add your own notes)	$40
NEW!	Heap I/O (treat all or part of a disk file as heap storage)	$40
	Biggerstaff's System Tools (multi-tasking window manager kit)	$40
	PC-XINU (Comer's XINU operating system for PC)	$35
	CLIPS (rule-based expert system generator, Version 4.2)	$35
	Tiny Curses (Berkeley curses package)	$35
NEW!	Polyglot RAM Disk (change disk size on the fly; includes utilities)	$30
	SP (spelling checker with dictionary and maintenance tools)	$30
	Clisp (Lisp interpreter with extensive internals documentation)	$30
	Translate Rules to C (YACC-like function generator for rule-based systems)	$30
	6-Pack of Editors (six public domain editors for use, study & hacking)	$30
	Crunch Pack (14 file compression & expansion programs)	$30
NEW!	Pascal Compiler & Interpreter (P-codes, standard Pascal)	$25
	ICON (string and list processing language, Version 7)	$25
	FLEX (fast lexical analyzer generator; new, improved LEX)	$25
	LEX (lexical analyzer generator; an oldie but a goodie)	$25
	Bison & PREP (YACC workalike parser generator & attribute grammar preprocessor)	$25
	AutoTrace (program tracer and memory trasher catcher)	$25
NEW!	Data Handling Utilities in C (data entry, validation & display; specify Turbo C or Microsoft)	$25
	Arrays for C (macro package to ease handling of arrays)	$25
NEW!	ANSI Forms (forms manager based on ANSI codes)	$20
	C Compiler Torture Test (checks a C compiler against K & R)	$20
	Benchmark Package (C compiler, PC hardware, and Unix system)	$20
	TN3270 (remote login to IBM VM/CMS as a 3270 terminal on a 3274 controller)	$20
	A68 (68000 cross-assembler)	$20
	List-Pac (C functions for lists, stacks, and queues)	$20
	XLT Macro Processor (general purpose text translator)	$20
NEW!	KeySwap (swaps CapsLock and CRTL key functions on XTs and ATs)	$20
NEW!	OOPS (collection of handy C++ classes by Keith Gorlen of NIH)	$20
	C/reativity (Eliza-based notetaker)	$15
	Data	
	DNA Sequences (GenBank 52.0 including fast similarity search program)	$150
	Protein Sequences (5,415 sequences, 1,302,966 residuals, with similarity search program)	$60
	Dictionary Words (234,932 words in alphabetical order, no definitions)	$60
	U. S. Cities (names & longitude/latitude of 32,000 U.S. cities and 6,000 state boundary points)	$35
	The World Digitized (100,000 longitude/latitude of world country boundaries)	$30
	KST Fonts (13,200 characters in 139 mixed fonts: specify TEX or bitmap format)	$30
	USNO Floppy Almanac (high-precision moon, sun, planet & star positions)	$20
	NBS Hershey Fonts (1,377 stroke characters in 14 fonts)	$15
	U. S. Map (15,701 points of state boundaries)	$15

The Austin Code Works — *Voice: (512) 258-0785*
11100 Leafwood Lane — *acw!info@uunet.uu.net* — *BBS: (512) 258-8891*
Austin, Texas 78750-3409 USA — *FAX: (512) 258-1342*

Free surface shipping on prepaid orders For delivery in Texas add 7% MasterCard/VISA

The Proposed ANSI C Language Standard

C Ware's DeSmet C V3.1

by
Rex Jaeschke

The Standard's almost ready to be voted out, but Rex questions whether conformance is all as he looks at C Ware's latest version of DeSmet C.

Finally, I took delivery of a latest release of a C compiler whose ad doesn't claim that it's ANSI C compatible. It's not that C Ware isn't interested in ANSI conformance (DeSmet C V3.1 shows much evidence to the contrary), but rather that they're more restrained than some vendors. This approach is refreshing. Besides, the DeSmet compiler has a large and loyal band of followers who aren't about to abandon it just because it doesn't have all the latest ANSI goodies. In fact, for the market the compiler sells to it does very nicely, and its speed of compilation and integrated set of tools are among the best in the business. (The SEE editor included in the package has been my only full screen editor on DOS these last five years. It's adequate, powerful, and simple, and I thoroughly recommend it.)

The preprocessor, compiler, and library all come up short in numerous areas of conformance to the proposed ANSI C Standard. However, since C Ware makes no claims to the contrary, this review should not be seen as "negative" simply because some instances of ANSI conformance are missing. Instead, you could look at it from the viewpoint of "here's a long list of new things V3.1 now supports."

The Compiler Options

A couple compiler switches are worth mentioning. **-pt** enables the recognition of ANSI's 3-character trigraph sequences, and **-pw** causes various warning messages to be displayed. Perhaps the most useful of these occurs when you pass an argument to a function and its type is changed by the prototype. No other compiler I have seen does this—I think it's a great idea and shows up lazy programming practices.

No switch exists to make plain **char**s signed.

The Preprocessor

DeSmet C supports the following ANSI C attributes:

- #, ## and **defined** operators
- white space before directives
- **#elif**
- **#include** identifier
- **#error**
- various **pragma**s
- the predefined macros __LINE__, __FILE__, __DATE__ and __TIME__

DeSmet C mimics ANSI's detailed phases of translation quite well, down to the point of allowing any token to be split across lines using the backslash/new-line continuation method.

The predefined macro __STDC__ is indeed defined (and could not be **#undef**ed), and it has the value 0. ANSI C requires this macro to have the value 1 if the compiler conforms to the standard. Since there is not yet a standard, setting the value to 1 would be premature although several compilers already do so. Note, therefore, that you should test __STDC__ specifically against 1 rather than use **#ifdef** __STDC__ since any value other than 1 is permitted for non-conforming implementations.

DeSmet C differs from ANSI C in the following ways:

- The same macro name can be defined differently.
- **#undef**ing a non-existent macro is an error and shouldn't be.
- Non-white space tokens are permitted (and ignored) on directive lines.
- Casts and floating constants are permitted in **#if** expressions, and they shouldn't be.
- The standard headers are not precluded from being **#included** more than once, and they should be.
- While 1024 macros without arguments compiled, only 999 with arguments would, presumably due to the use of the small model.

Continued . . .