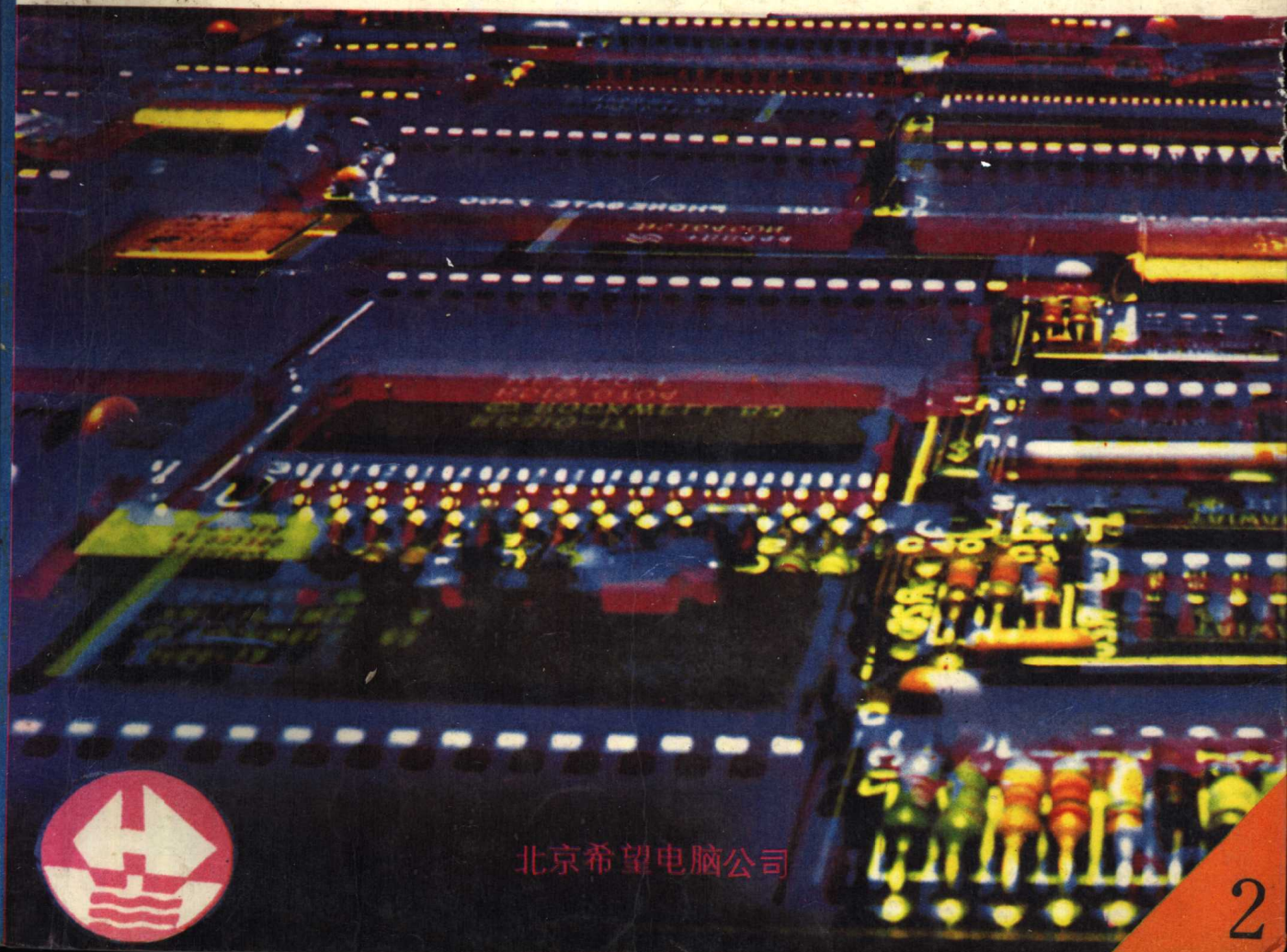
HOPE

6.0 版

Turbo C TOOLS 源程序剖析

Turbo C 高级程序设计实例

(共二册)



北京希望电脑公司

2

Turbo C TOOLS 6.0 源程序剖析

Turbo C 高级程序设计实例(下)

李　文　编译

- 域编辑编程扩展
- 扩展的文件操作
- 帮助系统程序设计
- 用 C 进行中断服务子程序的设计
- 内存驻留程序设计的插入码
- 高级键盘管理程序设计
- 内存管理高级程序设计
- 选单程序设计
- 鼠标器编程
- 打印机编程
- 屏幕(视频)编程
- 字符串处理
- 实用函数和宏
- 窗口系统高级程序设计
- 创建扩展函数库

中国科学院希望高级电脑技术公司

一九九一年七月

第十二章　屏幕(视频)编程

Turbo C TOOLS 的屏幕操作函数提供了一个调用 BIOS 视频服务的高级接口，因而也提供了一个与 IBM 显示硬件的接口。用户可以选取许多 I/O 函数，将它们用于标准的显示方式或显示页。当需要最快的速度时，用户还可直接访问显示内存。

屏幕操作函数的种类

读取屏幕方式信息

SCEQUIP　识别所安装的各种不同的显示适配器以及安装时设置的可选参数及开关。

SCMODE　报告当前的显示设备、当前方式、列数和活动的(即当前可见的)显示页。SCROWS 报告当前显示设备和方式所支持的正文行的数目。

SCPAGES　报告当前显示设备在当前方式下所支持的显示页的数目。

SCGETVID　报告当前显示设备的全部状态：方式、当前页、活动页、行、列及光标。

选择显示设备和方式

SCNEWDEV　选择一个显示方式，基于所需方式重置合适的设备(于是清除屏幕)。它可以选择 25-、30-、43-或 50-行方式。

SCCHGDEV　切换至给定的显示适配器。(请注意下面提到的重要限制。)

SCSETVID　将显示适配器恢复成原先由 SCGETVID 所记录的状态，包括方式、显示页和光标。

在显示页之间切换

SCAPAGE　显示(激活)当前显示设备上的所需页。它不改变光标的尺寸。

SCPAGE　选择给定的当前设备上的一个显示页，将该页作为 Turbo C TOOLS 屏幕 I/O 函数使用的页。它不显示(激活)任何显示页。

控制/读取光标形状和位置

SCCURST　报告当前显示页上的光标位置和尺寸。

SCCURSET　移动当前显示页上的光标。

SCPGCUR　在当前显示页是活动(即可见)的情况下设置光标的尺寸。

清除和滚动

SCPCLR　清除当前显示页。

SCCLRMSG　清除当前显示页上的一条信息。

VISCROLL　使当前显示页上的一个矩形区域上滚一页或下滚一页。

VIHORIZ　使当前显示页上的一个矩形区域左移或右移。

常规的屏幕写入

SCATTRIB　从当前显示页上的光标位置开始写入一个字符和属性 9 对的拷贝。

SCWRITE　在当前显示页上的光标位置写入一个字符的多个拷贝，原有的属性不变。

SCTTYWRT　向当前显示页(如果是活动的)写入一个字符并移动光标。回车、换行、响铃和退格以电传(TTY)方式处理。必要时屏幕发生滚动。

VIDSPMSG　用给定属性在当前显示页上的指定位置显示一条消息。

SCCLRMSG　　清除当前显示页上的一条消息。

写入一个矩形区域

VIWRRECT　　用一个缓冲区的内容填充当前显示页上的一个矩形区域，该函数的任选项可以对属性进行控制。

VIWRSECT　　类似于 VIWRRECT，但 VIWRSECT 可以显示更大的矩形缓冲区中的一段。

VIATRECT　　改变一个矩表区域的属性，其中的字符不变。

SCTTYWIN　　以 TTY 方式(正如 SCTTYWRT)写一个字符，但所有输出及滚动均限定在一个矩形区域内。

SCWRAP　　以 TTY 方式写入一个字符串(正象 SCTTYWIN)，但它可以“整字换行”，这样能够避免在区域的行之间把一个字折断。

SCBOX　　利用特殊的字符方式下的画框字符用单线或双线画一个方框。

SCRESTPG　　利用 SCSAVEPG 保存的一个压缩拷贝恢复一个显示页。

屏幕读取

SCREAD　　报告当前显示页光标位置的字符及其属性。

VIRDRECT　　读取显示在一个矩形区域中的字符，可带有或不带有属性。

VIRDSECT　　类似 VIRDRECT，但 VIRDSECT 可以将数据读入更大矩形区域的一段中。

SCSAVEPG　　以压缩格式保存当前显示页的内容。

调色板支持

SCPALETT　　定义由 EGA、VGA 或 MCGA 显示的整个颜色调色板。

SCPAL1　　定义由 EGA、VGA 或 MCGA 显示的一个颜色。

SCMODE4　　选择用于显示方式 4 (320X200 四色图形)的色板及背景颜色。

SCBORDER　　设置边界颜色。

SCBLINK　　选择在 Enhanced Graphics Adapter 下属性位 7 的作用：前景闪 烁或背景亮度。

对直接视频访问的支持

VIPTR　　返回一个 far 指针，该指针指向当前显示页上的一个给定位置(在显示内存中)。

保存和恢复整个显示状态

让应用程序在结束之后不干扰显示环境，这是一种良好的行为，有时甚至是必须的。这就需要保存显示状态，在程序退出前小心地恢复状态。

Turbo C TOOLS 提供了一些函数来完成这些任务。SCGETVID 和 SCSETVID 读取和恢复显示方式及光标尺寸而 SCSAVEPG 和 SCRESTPG 记录并恢复所显示的正文数据。

下面是函数的列表，这些函数保存和恢复显示环境的特定参数。

保存和恢复显示状态的函数

	读取		恢复
当前显示设备	SCMODE		SCNEWDEV
方式	SCMODE		SCNEWDEV
正文行数	SCROWS		SCNEWDEV
活动显示页	SCMODE		SCAPAGE
光标位置	SCCURST		SCCURSET
光标尺寸	SCCURST		SCPGCUR
屏幕上的正文	SCREAD、VIRDRECT 或 SCSAVEPG		SCATTRIB、SCWRITE、VIWRRECT 或 SCRESTPG

屏幕上的属性　　SCREAD、　　SCATTRIB、
VIRDRECT　或　VIWRRECT、
SCSAVEPG　　VIATRECT 或
SCRESTPG

由于 BIOS 未记录某些显示状态，所以有时不可能获取这些参数，例如色板。

强制快速屏幕访问

直接显示(VI)函数检测 Color/Graphics Adapter 的存在，计算操作的时间，防止视频干扰(“下雪”)。然而在某些情况下用户有时希望避免出现延迟。例如，有些显示适配器可以模拟 CGA 而不会出现雪花，它们并不需要特殊的计时工作。有时雪花不是一个问题。

要想使 VI 函数工作得尽可能快而不考虑干扰问题，可使用下面的语句：

```
#include <bvideo.h>
b_vifast = 1;
```

下面的语句将 VI 函数重置为一般的方式，避免 CGA 上的干扰。

```
b_vifast = 0;
```

屏幕(视频)控制函数源程序

SCAPAGE　　显示(激活)一个显示页

```
#include<bscreens.h>
int scapage(int page);
```

page　　待显示(激活)的页号。

(返回)　　实际显示的显示页。

SCAPAGE 设置活动的显示页，即显示该页。它不改变光标尺寸，也不打开或关闭光标。

SCAPAGE 检查当前的显示方式。如果 page 超出范围，则该值被强置为合适的值：负页号置成 0，过大的正页号置为可用的最大页号，实际显示的页号作为函数值返回。

用 SCPAGES 可以查出当前设备在当前方式下能够支持多少页。SCPAGE 指示 Turbo C TOOLS 将屏幕输出送到一个给定页，不论它是活动的(显示的)还是不活动的。

源程序(SCAPAGE.C)：

```
#include <dos.h>
#include <bscreens.h>
int scapage(page)
int page;
{
    int         max_page;
    union REGS inregs,outregs;
    max_page = scpages() - 1;          /* Maximum page value          */
    utbound(page,0,max_page)
    if (max_page)                      /* Switch active page only if  */
    {                                   /* this device supports more   */
       inregs.h.ah = 5;                /* than one.                   */
       inregs.h.al = (unsigned char) page;
       int86(16,&inregs,&outregs);
    }
    return(page);
}
```

SCATTRIB　　用指定的显示属性显示一个字符的拷贝

```
#include<bscreens.h>
int scattrib(  int fore,
               int back,
               char ch,
               unsigned cnt);
```

fore　　前景显示属性。
back　　背景显示属性。
ch　　待显示的字符。
cnt　　待显示的 ch 拷贝的数量。
(返回)　　返回值总为 0。

SCATTRIB 从当前光标位置开始显示字符 ch 的 cnt 份拷贝，光标保持不动。拷贝在当前页(b_curpage)上以属性(fore,back)显示，字符以一行接一行的方式写入。

如果写入时超出屏幕的末端，或(在图形方式下)写入时超出当前的正文行，这时有可能出现未预料到的结果。SCATTRIB 不滚动屏幕。

源程序(SCATTIB.C)：

```
#include <dos.h>
#include <bscreens.h>
int scattrib(fore,back,ch,cnt)
int       fore;
int       back;
char      ch;
unsigned cnt;
{
    union REGS inregs,outregs;
    if (cnt)
    {
        inregs.h.ah = 9;
        inregs.h.al = ch;
        inregs.h.bh = (unsigned char) b_curpage;
        inregs.h.bl = (unsigned char) utnybbyt(back,fore);
        inregs.x.cx = cnt;
        int86(16,&inregs,&outregs);
    }
    return(0);
}
```

SCBLINK　　选择前景闪烁或背景亮度

```
#include<bscreens.h>
int scblink(int option);
```

option　　SC_BLINK (1)　使属性位 7 控制前景闪烁，SC_INTENSITY(0)　使属性位 7 控制背景亮度。
(返回)　　返回的错误代码。可能值包括：
SC_NO_ERROR　　(0)　成功。
SC-BAD_OPT　　(1)　当前显示适配器不支持这个操作。

SCBLINK 控制在 EGA、VGA 和 MCGA 字符方式下属性位 7 的作用。

如果指定了 SC_BLINK，属性位 7 控制字符的闪烁。这是建立当显示适配器重置的缺省条件，此时背景可用 8 种颜色，由属性位 4-6 指定。

如果指定了 SC_INTENSITY，属性位 7 用于增加背景颜色的这度，使得背景可以象前景一样选择 16 种颜色。

源程序(SCBLINK.C)：

```
#include <dos.h>
#include <bscreens.h>
#define BLINK_BIT          0x20
#define  OK                  0              /* Error codes.              */
#define  INVALID_REQUEST     1
int scblink(option)
int option;
{
    int          result;
    int          mode,act_page,columns;
    union REGS inregs,outregs;
    scequip();
    scmode(&mode,&columns,&act_page);
    if (   b_device   == b_ega
        || b_device   == b_vga
        || b_device   == b_mcga
        || b_pcmodel == IBM_JR)
    {
        inregs.x.ax = 0x1003;          /* Set intensify/blinking bit.  */
        inregs.h.bl = (unsigned char) option;
        int86(0x10,&inregs,&outregs);
        result = OK;
    }
    else
        result = INVALID_REQUEST;
    return result;
}
```

SCBORDER　　设置当前显示屏幕的边界颜色

```
#include<bscreens.h>
int scborder(unsignde color);
```

color　　选择的颜色。

(返回)　　返回的错误代码。可能值包括：

SC_NO_ERROR　　(0)　成功。

SC_BAD_OPT　　(1)　当前显示适配器和方式不支持这个操作。

SCBORDER 设置当前显示设备的边界(“屏幕外”)的颜色。

在方式 7(单色字符)或 Professional Graphics Controller 的 High_Function Graphics 方式下或 Enhanced Color Display 的 350 扫描线方式下不能使用非黑颜色。如果这些条件中的某一个成立，该函数返回 SC_BAD_OPT，不做任何操作。

适配器的重置将边界颜色置为 0(黑色)。

在 EGA、VGA 和 MCGA 下，边界颜色与在色板中选择的其它颜色无关(见 SCPALETT 和 SCPAL1)。

源程序(SCBORDER.C)：

```
#include <dos.h>
#include <bscreens.h>
int scborder(color)
unsigned color;
{
    int         result;
    int         mode,act_page,columns;
    union REGS inregs,outregs;
    scequip();                              /* Sense equipment and switch   */
                                            /* settings.                    */
    scmode(&mode,&columns,&act_page); /* Retrieve current device        */
                                            /* (b_device).                  */
    if (mode == 7)                          /* Rule out monochrome text mode*/
        result = SC_BAD_OPT;
    else if (   b_device  == b_ega
             || b_device  == b_vga
             || b_pcmodel == IBM_JR)
    {                                       /* Rule out Enhanced Color      */
                                            /* Display 350-line modes.      */
        if (   (mode      <= 3 || mode      == 15  || mode == 16)
            && b_device  == b_ega
            && (b_sw_ega == 3 || b_sw_ega ==  9))
            result = SC_BAD_OPT;
        else
        {                                   /* Use BIOS function.           */
            inregs.x.ax = 0x1001;
            inregs.h.bh = (unsigned char) color;
            int86(SC_BIOS_INT,&inregs,&outregs);
            result = SC_NO_ERROR;
        }
    }
    else
    {                                       /* Use the CGA BIOS function.   */
        inregs.h.ah = 0x0b;
        inregs.h.bh = 0;
        inregs.h.bl = (unsigned char) color;
        int86(SC_BIOS_INT,&inregs,&outregs);
        result = SC_NO_ERROR;
    }
    return result;
}
```

SCBOX　　用图形字符在屏幕上画一个方框

```
#include<bscreens.h>
int scbox(  int u_row, int u_col,
            int l_row, int l_col,
            int boxtype,
            char boxchar,
            int attrib);
```

u_row, u_col　框的上行左列。
l_row,l_col　方框的下行右列。
boxtype　方框字符的类型：如果使用 boxchar，为-1；如果使用 IBM 方框字符(见下面)，则为 0 到 15。
boxchar　如果 boxtype 为-1，则 boxchar 是用于绘制方框的字符。
attrib　使用的属性。低字节的高四位描述前景属性，低字节的低四位描述前景属性。
(返回)　错误代码：如果方框尺寸非法，返回 1；如果没有错误，返回 0。

SCBOX 在当前显示页上画一个方框。

如果某个角超出了屏幕、方框的高度或宽度小于两个字符或者左上角(u_row, u_col)在右下角(l_row,l_col)的下方或右方，则方框的尺寸非法。如果出现这种情况，1 作为函数值返回。

如果 boxtype 是-1，则使用 boxchar 指定的字符画方框；如果 boxtype 在范围 0 到 15 内，则使用 IBM PC 字符集的方框字符绘制该方框。根据下表，boxtype 的实际值指明方框的边线是单线是双线。

SCBOX 的方框类型码

方框	类型	下	右	上	左
0	(0000)	单	单	单	单
1	(0001)	单	单	单	单
2	(0010)	单	单	双	单
3	(0011)	单	单	双	双
4	(0100)	单	双	单	单
5	(0101)	单	双	单	双
6	(0110)	单	双	双	单
7	(0111)	单	双	双	双
8	(1000)	双	单	单	单
9	(1001)	双	单	单	双
10	(1010)	双	单	双	单
11	(1011)	双	单	双	双
12	(1100)	双	双	单	单
13	(1101)	双	双	单	双
14	(1110)	双	双	双	单
15	(1111)	双	双	双	双

如果选择了上表中的一个方框类型，函数自动选择角字符与边线匹配。

源程序(SCBOX.C)：

```
#include <dos.h>
#include <bscreens.h>
    /* Macro to write a single character at a given point on the      */
    /* screen                                                        */
#define  write_one(row,col,ch,fore,back)          \
        {                                          \
           scursset(row,col);                      \
           scattrib(fore,back,ch,1);               \
        }

    /* Macro to write more than one character at a given point on the */
    /* screen                                                        */

#define  write_some(row,col,ch,fore,back,number)  \
        {                                          \
           if (number)                             \
           {                                       \
```

```
                sccurset(row,col);                          \
                scattrib(fore,back,ch,number);              \
            }                                               \
        }

int scbox(u_row,u_col,l_row,l_col,boxtype,boxchar,attrib)
int  u_row,u_col,l_row,l_col,boxtype;
char boxchar;
int  attrib;
{
    int  mode,act_page,columns;        /* Returned by SCMODE          */
    int  save_row,save_col,high,low;
    int  row,width,fore,back;
    char topleft,topright,botleft,botright,
         top,bottom,left,right;        /* Characters to write         */
    /* Tables of box characters                                       */
    static char upleft[]               /* Top left corners            */
                    = {'\332','\326','\325','\311'},
                upright[]              /* Top right corners           */
                    = {'\277','\270','\267','\273'},
                lowleft[]              /* Bottom left corners         */
                    = {'\300','\323','\324','\310'},
                lowright[]             /* Bottom right corners        */
                    = {'\331','\275','\276','\274'};
    static char horiz[]                /* @orizontal lines            */
                    = {'\304','\315'},
                vert[]                 /* Vertical lines              */
                    = {'\263','\272'};
    /* Validate the box position and dimensions                       */
    scmode(&mode,&columns,&act_page);
    if (        0 >  u_row
        || u_row >= l_row
        || l_row >= scrows()
        ||      0 >  u_col
        || u_col >= l_col
        || l_col >= columns     )
        return 1;
    width = l_col - u_col - 1;         /* Distance between the sides  */
    if ((mode > 3) && (mode != 7))     /* Force attribute for graphics */
        attrib = min(3,max(0,attrib));
    fore = utlonyb(attrib);
    back = uthinyb(attrib);
    /* Look up the box characters from the tables                     */
    if (boxtype < 0)
        topleft = topright = botleft = botright
                = left     = right   = top      = bottom   = boxchar;
    else
    {
        topleft  = upleft  [ boxtype       & 3];
        topright = upright [ (boxtype >> 1) & 3];
```

```
        botleft  = lowleft [((boxtype >> 2) & 2) | (boxtype & 1)];
        botright = lowright[ (boxtype >> 2) & 3];

        top      = horiz[(boxtype >> 1) & 1];
        bottom   = horiz[(boxtype >> 3) & 1];
        left     = vert [ boxtype       & 1];
        right    = vert [(boxtype >> 2) & 1];
    }

    /* Draw the box, starting from upper left corner, working          */
    /* horizontally.                                                   */

                                        /* Save cursor location        */
    sccurst(&save_row,&save_col,&high,&low);
    write_one (u_row,u_col,      topleft, fore,back);
    write_some(u_row,(u_col + 1),top,     fore,back,width);
    write_one (u_row,l_col,      topright,fore,back);
    for (row = u_row + 1; row < l_row; row++)
    {
        write_one(row,u_col,left, fore,back);       /* Left side       */
        write_one(row,l_col,right,fore,back);       /* Right side      */
    }
    write_one (l_row,u_col,      botleft, fore,back);
    write_some(l_row,(u_col + 1),bottom,  fore,back,width);
    write_one (l_row,l_col,      botright,fore,back);
    sccurset(save_row,save_col);       /* Restore cursor location      */
    return 0;
}
```

SCCHGDEV　　切换至彩色或单色显示

```
#include<bscreens.h>
int scchgdev(int device);
```

device　　待选择的设备(SC_MONO (0) 或 SC_COLOR (1))。

(返回)　　返回的错误代码：如果设备不存在必须用 SCNEWDEV 重置，返回 1；如果没有错误，返回 0。

SCCHGDEV 不经过重置就能够为以后的 I/O 选择显示适配器，这个过程包括测试所需设备是否存在、设置 BIOS 装备字、恢复 BIOS 显示状态变量、恢复全局变量 b_device 和 b_curpage,从而指明保存在新设备上的当前项。

程序启动时的当前设备可以在任何时候用 SCCHGDEV 或 SCNEWDEV 来选择。然而，其它设备(如果存在)在被 SCCHGDEV 选择之前必须至少要重置一次，否则 SCCHGDEV 报告一个错误。

如果所需设备不是当前设备，则 SCCHGDEV 用保存在一个内部表中的值恢复 BIOS 的内部显示状态变量。

源程序(SCCHGDEC.C)：

```
#include <dos.h>
#include <bscreens.h>
extern int b_vidcpy(int,int);          /* Internal (see SCNEWDEV.C).   */
#define  FROM_BIOS      0              /* Direction flags for copying  */
#define  TO_BIOS        1              /* BIOS video state variables.  */
```

```
#define  VIDEO_FIELD  0x0030          /* The bits to be modified in   */
                                      /* the BIOS equipment word.     */
#define  MONO_FLAG    0x0030
#define  COLOR_80     0x0020
#define  COLOR_40     0x0010
                                /* Address of BIOS equipment word.    */
#define  EQUIP_WORD_LOC  (uttofar(0x0040,0x0010,unsigned int))
int scchgdev(device)
int device;
{
    int old_mode,old_columns,old_act_page;  /* Previous state          */
    int old_dev;
    union REGS regs;
#define our_word     (regs.x.ax)        /* Our copy of equipment word.  */
    int  old_flag,new_flag;             /* Old and new values of video  */
                                        /* device bit field.            */
    if (device != SC_MONO && device != SC_COLOR)
        return 1;                       /* Unknown device.              */
    /* Note current state of BIOS.                                      */
    old_dev = scmode(&old_mode,&old_columns,&old_act_page);
    if (device == old_dev)
        return 0;                       /* Nothing to do.               */
    /* Record current BIOS variables in case we ever need to change    */
    /* back to the old device.                                          */
    if (b_vidcpy(FROM_BIOS,old_dev))
        return 1;
    if (!b_adap_state[device].known)    /* Abort if we've never         */
        return 1;                       /* recorded the BIOS state      */
                                        /* variables for new device.    */
    scequip();                          /* Sense which hardware is      */
                                        /* present.                     */
    switch (device)
    {
        case SC_COLOR:
            if (   b_cga != SC_COLOR  && b_pcmodel != IBM_JR
                && b_ega != SC_COLOR  && b_pgc     != SC_COLOR
                && b_vga == SC_ABSENT && b_mcga    == SC_ABSENT)
                return 1;               /* Color/Graphics Adapter       */
                                        /* functions are not available. */
            new_flag = COLOR_80;
            break;
        case SC_MONO:
            if (   b_mdpa != SC_MONO
                && b_ega  != SC_MONO
                && b_vga  == SC_ABSENT)
                return 1;               /* Monochrome Adapter functions */
                                        /* are not available.           */
            new_flag = MONO_FLAG;
            break;
```

```
    }

    b_device = device;
    /* If we are switching VGA from color to monochrome or vice        */
    /* versa, note the change in our global variables.                  */
    if (   device != b_mdpa && device != b_cga && device != b_ega
        && device != b_pgc)
    {
        if (b_vga == old_dev)
            b_vga = device;
    }
    /* The BIOS keeps track of the color vs. monochrome distinction    */
    /* by two bits in the equipment word, so we must poke those bits   */
    /* to signal the change in device.                                  */
    int86(17,&regs,&regs);              /* Retrieve equipment word.     */

    old_flag = (our_word & VIDEO_FIELD);
    if (new_flag != old_flag            /* There is a change.           */
            && (old_flag != COLOR_40 || new_flag != COLOR_80))
                                        /* (Skip the change if 40-col   */
                                        /*    mode is set and we're     */
                                        /*    setting color.)           */
    {
                                        /* Install modified equipment   */
                                        /* word.                        */
        utpokeb(EQUIP_WORD_LOC,
                (unsigned char) ((our_word & ~VIDEO_FIELD) | new_flag));
    }
    /* Restore BIOS variables that were in effect when we last quit    */
    /* this device.                                                     */
    b_vidcpy(TO_BIOS,b_device);
    scpage(b_adap_state[device].curpage);   /* Restore current page.   */
    return 0;
}
```

SCCLRMSG　　清除屏幕上的消息

```
#include <bscreens.h>
int scclrmsg(  int row,
               int col,
               int len);
```

row, col　　待清除的位置(0,0 是屏 幕的左上角)。
len　　待清除的消息的长度。
(返回)　　返回值总是 0。

这个函数在当前显示页上的指定位置清除一个消息而不改变屏幕的显示属性，光标位置保持不变。

在图形方式下，被清除区域不能超出当前行而发生折转；在字符方式下，被清除区域可以折转超出当前行(但不能超出屏幕的末端)。SCCLRMSG 不滚动 屏幕。

源程序(SCCLRMSG.C):

```
#include <bscreens.h>
```

```
#define BLANK    ' '
int scclrmsg(row,col,len)
int row,col,len;
{
    int c_row,c_col,high,low;

    sccurst(&c_row,&c_col,&high,&low);/* Current position.          */
    sccurset(row,col);
    scwrite((char) BLANK,len);          /* Write len blanks.          */
    sccurset(c_row,c_col);              /* Return the original position.*/
    return(0);
}
```

SCCURSET　　移动当前显示页上的光标

```
#include <bscreens.h>
int sccurset(   int row,
                int col);
```

row　　行位置(0=屏幕顶行)

col　　列位置(0 = 最左列)

(返回)　　结果位置。行在高八位，列在低八位。如果所需位置超出范围，结果位置可能与所需位置不相同。

SCCURSET 将光标移到由 row 和 col 指定的当前显示页的一个位置。

为使光标移动时可见，当前页必须是活动(即当前显示)。 如果当前页是不活动，则当该页呈活动状态时，新光标位置将会出。

如果所需位置超出范围，则位置将位于屏幕边缘上，尽可能与所需位置接近。

源程序(SCCURSET.C)：

```
#include  <dos.h>
#include  <bscreens.h>
int sccurset(row,col)
int row,col;
{
    union REGS inregs,outregs;          /* Registers for BIOS call      */
    int mode,columns,act_page;
    scmode(&mode,&columns,&act_page);
    utbound(row,0,scrows() - 1)         /* Force reasonable values      */
    utbound(col,0,columns-1)
    inregs.h.ah  =  0x02;               /* Set up call to the BIOS      */
    inregs.h.bh  =  (unsigned char) b_curpage;
    inregs.h.dh  =  (unsigned char) row;
    inregs.h.dl  =  (unsigned char) col;
    int86(16,&inregs,&outregs);
    return (int) inregs.x.dx;
}
```

SCCURST　　返回当前显示页上的光标位置和尺寸

```
#include <bscreens.h>
int sccurst(   int *prow,
```

```
                int *pcol,
                int *phigh,
                int *plow);
```

prow　　指向返回的行值(0=屏幕顶行)。
pcol　　指向返回的列值(0=最左列)。
phigh　　指向返回的高扫描行。
plow　　指向返回的低扫描行。
(返回)　　如果光标关闭，为1；若打开，为0。

SCCURST返回当前显示设备和显示页上的光标位置及光标尺寸，报告光标是打开还是关闭的。

这个函数询问ROM BIOS有关光标状态的信息。有些IBM程序改变了光标尺寸但不通知BIOS，所以BIOS报告的光标尺寸也许不正确。SCCURST也面临着这种不准确性。

Turbo C Tools和BIOS都不记录非活动页的光标尺寸和打开/关闭状态，也就是说，活动页的光标尺寸也适用于非活动页。

源程序(SCCURST.C)：

```
#include <dos.h>
#include <bscreens.h>
int sccurst(prow,pcol,phigh,plow)
int *prow,*pcol,*phigh,*plow;
{
    union REGS inregs,outregs;          /* Registers for BIOS call       */
    inregs.h.ah = 0x03;                 /* Set up the call to the BIOS   */
    inregs.h.bh = (unsigned char) b_curpage;
    int86(SC_BIOS_INT,&inregs,&outregs);
    *prow  = outregs.h.dh;
    *pcol  = outregs.h.dl;
    *phigh = utlonyb(outregs.h.ch);             /* Use actual values. */
    *plow  = utlonyb(outregs.h.cl);

    return ((outregs.h.ch & 0x60) != 0);
}
```

SCEQUIP　　检测显示硬件环境

```
#include<bscreens.h>
char scequip();
```

(返回)　　IBM型号代码，正象UTMODEL返回的那样。

SCEQUIP取得有关安装的显示适配器及其可选设置的一般信息。下面的全局变量将被置位：

变量	头文件	说明
b_know_hw	bscreens.h	设置成1，表示SCEQUIP已经运行。
b_mdpa	bscreens.h	存在单色适配器(MDPA):SC_ABSENT (-2)或SC_MONO(0) (如果是Hercules图形适配器，则为SC)MONO。)
b_cga	bscreens.h	存在Color/Graphics Adapter: SC_ABSENT (-2) 或SC_COLOR (1)。
b_ega	bscreens.h	Enhanced Graphics Adapter (EGA)状态:SC_ABSENT (-2)、SC_MONO (0) 或SC_COLOR (1)。
b_vga	bscreens.h	Video Graphics Array (VGA)状态:SC_ABSENT (-2)、SC_MONO (0) 或SC_COLOR的(1)。
b_mcga	bscreens.h	Multi-Color Graphics Array (MCGA)状态:SC_ABSENT (-2)、SC_MONO (0) 或SC_COLOR (1)。
b_herc	bscreens.h	存在Hercules单色图形适配器:SC_ABSENT (-2) 或SC_MONO (0)

b_pgc　　bscreens.h　　Professional Graphics Controller 状态:SC_ABSENT (-2)、SC_COLOR (1) 或 SC_HIFUNC (2)。

b_mem_ega　　bscreens.h　　EGA 上安装的内存，以 1024 字节为单位：64、128 或 256。

b_sw_ega　　bscreens.h　　EGA 开关设置。低位被置位表示 SW1 断开，等。

b_pcmodel　　butil.h　　IBM 型号代码。

b_submod　　butil.h　　IBM 子型号代码。 如果不存在，则为 0。

b_biosre　　butil.h　　IBM BIOS 修订版号。如果不存在，则为 0。

SCMODE、SCROWS 和 SCPAGES 可以取得在软件控制下能够修改的显示信息(如显示方式或 屏幕尺寸)。

源程序(SCEQUIP.C):

```
#include <conio.h>
#include <dos.h>
#include <bscreens.h>
int b_know_hw = 0;              /* Flag stating whether we have yet      */
                                /*      run SCEQUIP:  0 if no, 1 if yes  */
int b_mdpa = SC_DONT_KNOW;/* Monochrome Display & Printer Adapter      */
                          /*   SC_DONT_KNOW, SC_ABSENT, or SC_MONO     */
                          /*   (SC_MONO if Hercules graphics adapter.)*/
                          /*                                           */
int b_cga = SC_DONT_KNOW; /* Color/Graphics Monitor Adapter            */
                          /*   SC_DONT_KNOW, SC_ABSENT, or SC_COLOR    */
                          /*                                           */
int b_ega = SC_DONT_KNOW; /* Enhanced Graphics Adapter: SC_DONT_KNOW, */
                          /*   SC_ABSENT, SC_MONO or SC_COLOR          */
                          /*                                           */
int b_mcga = SC_DONT_KNOW;/* Multicolor Graphics Array: SC_DONT_KNOW, */
                          /*   SC_ABSENT, or SC_COLOR                  */
                          /*                                           */
int b_vga = SC_DONT_KNOW; /* Video Graphics Array:  SC_DONT_KNOW,      */
                          /*  SC_ABSENT, SC_MONO or SC_COLOR           */
                          /*                                           */
int b_herc = SC_DONT_KNOW;/* Hercules monochrome graphics adapter      */
                          /*   SC_DONT_KNOW, SC_ABSENT, or SC_MONO     */
                          /*                                           */
int b_pgc = SC_DONT_KNOW; /* Professional Graphics Controller:         */
                          /*   SC_DONT_KNOW, SC_ABSENT, SC_COLOR, or   */
                          /*   SC_HIFUNC                               */
int b_mem_ega = 0;              /* Amount of memory installed on EGA in */
                                /* 1024-byte units:  64, 128, 192, 256  */
unsigned b_sw_ega = 0x0ffff;  /* Switch settings on EGA                 */
                              /* Low-order bit set means SW1 off, etc.*/
                              /* (Bit value 1 implies that              */
                              /* corresponding switch is off.)          */
static volatile int waiter;   /* Variable to assign to to wait for      */
                              /* CGA and PGC to stabilize when using    */
                              /* fast processors.                       */
#define  PGC_SEG   0xc600     /* Segment of memory-mapped PGC ports.    */
                              /* PGC port for presence test.            */
#define  PGC_PRES_BYTE  (uttofar(PGC_SEG,0x03db,unsigned char))
```

```
#define  PGC_INDEX 0x03d4      /* Offset of PGC/CGA index port          */
#define  PGC_IND_BYTE    (uttofar(PGC_SEG,PGC_INDEX,unsigned char))
#define WAIT                                                            \
    {                                                                   \
        waiter  = 0;                                                    \
        waiter += 1;                                                    \
    }

static void note_adapter(unsigned char,int); /* Internal function      */
static int adap_present(int);          /* Internal function            */
static int pgc_present(void);          /* Internal function            */
static int pgc_emulating(void);        /* Internal function            */
static int here(void);                 /* Internal function            */
    /* Internal variables containing interim knowledge of adapter      */
    /* state and modes.                                                */

#define _DONT_KNOW  (-1)
#define _ABSENT      0
#define _ACTIVE      1
#define _ALTERNATE   2
#define _PRESENT     3
static int pgc_state = _DONT_KNOW;
static int vga_state;

char scequip()
{
    union REGS inregs,outregs;         /* Registers for BIOS calls     */
    unsigned   color_or_mono,mem;
    int        pgc_emul_tested;
    int        mode,columns,act_page;
    if (b_know_hw)
    {
        scmode(&mode,&columns,&act_page);  /* Check VGA state.         */
        return b_pcmodel;      /* No need to do this work again        */
    }

    if (utmodel() == IBM_JR)           /* Fetch & save model code.     */
    {                                  /* This is a PCjr               */
        b_mdpa = SC_ABSENT;
        b_cga  = SC_ABSENT;
        b_ega  = SC_ABSENT;
        b_pgc  = SC_ABSENT;
        b_vga  = SC_ABSENT;
        b_mcga = SC_ABSENT;
        b_herc = SC_ABSENT;
    }
    else if (b_pcmodel == IBM_CV)
    {                                  /* IBM PC Convertible acts like */
        b_mdpa = SC_MONO;                  /* both MDPA & CGA.         */
        b_cga  = SC_COLOR;
```