Borland Graphics Interface (BGI)

The following functions compose the Borland Graphics Interface and are usually available for 16 bit DOS applications. Use them to create onscreen graphics with text. They are defined in graphics.h.

Using BGI with Windows
The BGI graphics functions may also be used with Windows programs created by the Borland 5.0 compiler or the free GNU C++ compiler. These extra functions are described in www.cs.colorado.edu/~main/bgi/doc/bgi.html. In this listing, the extra functions are indicated by WIN. Also, any of the functions that use colors can use RGB colors in addition to the 16-color BGI palette.

Functions:

void arc (int x, int y, int stangle, int endangle, int radius);

void bar (int left, int top, int right, int bottom);
void bar3d (int left, int top, int right, int bottom, int depth, int topflag);

void circle (int x, int y, int radius);
void cleardevice (void);

void clearmouseclick(int kind); WIN
void clearviewport (void);
void closegraph (void);

void delay (int millisec); WIN
void detectgraph (int *graphdriver, int *graphmode);
void drawpoly (int numpoints, int *polypoints);

void ellipse (int x, int y, int stangle, int endangle, int xradius, int yradius);
void fillellipse (int x, int y, int xradius, int yradius);
void fillpoly (int numpoints, int *polypoints);
void floodfill (int x, int y, int border);

int getactivepage (void); WIN
void getarccoords (struct arccoordstype *arccoords);
void getaspectratio (int *xasp, int *yasp);
int getbkcolor (void);
int getch (void);
int getcolor (void);
struct palettetype* getdefaultpalette (void);
char* getdrivername (void);
void getfillpattern (char *pattern);
void getfillsettings (struct fillsettingstype *fillinfo);
int getgraphmode (void);
void getimage (int left, int top, int right, int bottom, void *bitmap);
void getlinesettings (struct linesettingstype *lineinfo);
int getmaxcolor (void);
int getmaxmode (void);
int getmaxx (void);
int getmaxy (void);
char* getmodename (int mode_number);
void getmoderange (int graphdriver, int *lomode, int *himode);
void getmouseclick(int kind, int& x, int& y);
void getpalette (struct palettetype *palette);
int getpalettesize (void);
unsigned getpixel (int x, int y);
void gettextsettings (struct textsettingstype *texttypeinfo);
void getviewsettings (struct viewporttype *viewport);
int getvisualpage (void);
int getx (void);
int gety (void);
void graphdefaults (void);
char* grapherrormsg (int errorcode);
int graphresult (void);
unsigned imagesize (int left, int top, int right, int bottom);

void initgraph (int *graphdriver, int *graphmode, char *pathodriver);

void initwindow (int width, int height);

int installuserdriver (char *name, int huge (*detect)(void));

int installuserfont (char *name);

bool ismouseclick(int kind);

int kbhit (void);

void line (int x1, int y1, int x2, int y2);

void linerel (int dx, int dy);

void lineto (int x, int y);

int mousex (void);

int mousey (void);

void moverel (int dx, int dy);

void moveto (int x, int y);

void outtext (char *textstring);

void outtextxy (int x, int y, char *textstring);

void pieslice (int x, int y, int stangle, int endangle, int radius);

void putimage (int left, int top, void *bitmap, int op);

void putpixel (int x, int y, int color);

void rectangle (int left, int top, int right, int bottom);

int registerbgidriver (void (*driver)(void));

int registerbgifont (void (*font)(void));

void registermousehandler (int kind, void h(int, int));

void restorecrtmode (void);
RGB macros: WIN
COLOR(r,g,b),
RED_VALUE(v), GREEN_VALUE(v), BLUE_VALUE(v),
IS_BGI_COLOR(v), IS_RGB_COLOR(v)

void sector (int x, int y, int stangle, int endangle, int xradius, int yradius);
void setactivepage (int page);
void setallpalette (struct palettetype *palette);
void setaspectratio (int xasp, int yasp);
void setbkcolor (int color);
void setcolor (int color);
void setfillpattern (char *upattern, int color);
void setfillstyle (int pattern, int color);
unsigned setgraphbufsize (unsigned bufsize);
void setgraphmode (int mode);
void setlinestyle (int linestyle, unsigned upattern, int thickness);
void setpalette (int colornum, int color);
void setrgbpalette (int colornum, int red, int green, int blue);
void settextjustify (int horiz, int vert);
void settextstyle (int font, int direction, int charsize);
void setusercharsize (int multx, int divx, int multy, int divy);
void setviewport (int left, int top, int right, int bottom, int clip);
void setvisualpage (int page);
void setwritemodesetwritemode(int mode);
int textheight (char *textstring);
int textwidth (char *textstring);
**arc**

**Syntax**

```c
#include <graphics.h>
void arc(int x, int y, int stangle, int endangle, int radius);
```

**Description**

arc draws a circular arc in the current drawing color centered at (x,y) with a radius given by radius. The arc travels from stangle to endangle. If stangle equals 0 and endangle equals 360, the call to arc draws a complete circle. The angle for arc is reckoned counterclockwise, with 0 degrees at 3 o'clock, 90 degrees at 12 o'clock, and so on.

The linestyle parameter does not affect arcs, circles, ellipses, or pie slices. Only the thickness parameter is used.

If you are using a CGA in high resolution mode or a monochrome graphics adapter, the examples in online Help that show how to use graphics functions might not produce the expected results. If your system runs on a CGA or monochrome adapter, pass the value 1 to those functions that alter the fill or drawing color (setcolor, setfillstyle, and setlinestyle, for example), instead of a symbolic color constant (defined in graphics.h).

**Return Value**

None.

**See also**

circle  
elipse  
fillellipse  
getarcoors  
getaspectratio  
pieslice  
sector

**Example**

```c
/* arc example */

#include <graphics.h>
#include <stdlib.h>
#include <stdio.h>
#include <conio.h>

int main(void)
{
    /* request autodetection */
    int gdriver = DETECT, gmode, errorcode;
    int midx, midy;
    int stangle = 45, endangle = 135;
    int radius = 100;

    /* initialize graphics and local variables */
    initgraph(&gdriver, &gmode, "");

    /* read result of initialization */
    errorcode = graphresult();
    if (errorcode != grOk) { /* an error occurred */
        return 1;
    }

    // Draw the arc
    arc(midx, midy, stangle, endangle, radius);

    // Additional code to render the arc

    getch();
    closegraph();
    return 0;
}
```
bar

Syntax
#include <graphics.h>
void bar(int left, int top, int right, int bottom);

Description
bar draws a filled-in, rectangular, two-dimensional bar. The bar is filled using the current fill pattern and fill color. bar does not outline the bar; to draw an outlined two-dimensional bar, use bar3d with depth equal to 0. The upper left and lower right corners of the rectangle are given by (left, top) and (right, bottom), respectively. The coordinates refer to pixels.

Return Value
None.

See also
bar3d
rectangle
setcolor
setfillstyle
setlinestyle

Example
/* bar example */

#include <graphics.h>
#include <stdlib.h>
#include <stdio.h>
#include <conio.h>

int main(void)
{
    /* request autodetection */
    int gdriver = DETECT, gmode, errorcode;
    int midx, midy, i;

    /* initialize graphics and local variables */
initgraph(&gdriver, &gmode, "");

/* read result of initialization */
errorcode = graphresult();
if (errorcode != grOk) { /* an error occurred */
    printf("Graphics error: %s\n", grapherrormsg(errorcode));
    printf("Press any key to halt: ");
    getch();
    exit(1);               /* terminate with an error code */
}

midx = getmaxx() / 2;
midy = getmaxy() / 2;

/* loop through the fill patterns */
for (i=SOLID_FILL; i<USER_FILL; i++) {
    /* set the fill style */
    setfillstyle(i, getmaxcolor());
    /* draw the bar */
    bar(midx-50, midy-50, midx+50, midy+50);
    getch();
}
/* clean up */
closegraph();
return 0;

bar3d

Syntax
#include <graphics.h>
void bar3d(int left, int top, int right, int bottom, int depth, int topflag);

Description
bar3d draws a three-dimensional rectangular bar, then fills it using the current fill pattern and fill color. The three-dimensional outline of the bar is drawn in the current line style and color. The bar's depth in pixels is given by depth. The topflag parameter governs whether a three-dimensional top is put on the bar. If topflag is nonzero, a top is put on; otherwise, no top is put on the bar (making it possible to stack several bars on top of one another). The upper left and lower right corners of the rectangle are given by (left, top) and (right, bottom), respectively.

To calculate a typical depth for bar3d, take 25% of the width of the bar, like this:

bar3d(left, top, right, bottom, (right-left)/4, 1);

Return Value
None.

See also
bar
rectangle
setcolor
setfillstyle
setlinestyle
Example

/* bar3d example */

#include <graphics.h>
#include <stdlib.h>
#include <stdio.h>
#include <conio.h>

int main(void)
{
    /* request autodetection */
    int gdriver = DETECT, gmode, errorcode;
    int midx, midy, i;

    /* initialize graphics and local variables */
    initgraph(&gdriver, &gmode, "");

    /* read result of initialization */
    errorcode = graphresult();
    if (errorcode != grOk) {   /* an error occurred */
        printf("Graphics error: %s\n", grapherrormsg(errorcode));

        printf("Press any key to halt:");
        getch();
        exit(1);               /* terminate with an error code */
    }

    midx = getmaxx() / 2;
    midy = getmaxy() / 2;

    /* loop through the fill patterns */
    for (i=EMPTY_FILL; i<USER_FILL; i++) {
        /* set the fill style */
        setfillstyle(i, getmaxcolor());

        /* draw the 3-d bar */
        bar3d(midx-50, midy-50, midx+50, midy+50, 10, 1);
        getch();
    }

    /* clean up */
    closegraph();

    return 0;
}

---

circle

description

circle draws a circle in the current drawing color with its center at (x,y) and the radius given by radius.
The linestyle parameter does not affect arcs, circles, ellipses, or pie slices. Only the thickness parameter is used.
If your circles are not perfectly round, adjust the aspect ratio.

**Return Value**
None.

**See also**
arc
ellipse
drawellipse
gdrawaspectratio
sector
drawaspectratio

**Example**

```c
/* circle example */

#include <graphics.h>
#include <stdlib.h>
#include <stdio.h>
#include <conio.h>

int main(void)
{
    /* request autodetection */
    int gdriver = DETECT, gmode, errorcode;
    int midx, midy, radius = 100;

    /* initialize graphics and local variables */
    initgraph(&gdriver, &gmode, "");

    /* read result of initialization */
    errorcode = graphresult();
    if (errorcode != grOk) { /* an error occurred */
        printf("Graphics error: %s\n", grapherrormsg(errorcode));
        printf("Press any key to halt:");
        getch();
        exit(1); /* terminate with an error code */
    }

    midx = getmaxx() / 2;
    midy = getmaxy() / 2;
    setcolor(getmaxcolor());

    /* draw the circle */
    circle(midx, midy, radius);
    /* clean up */
    getch();
    closegraph();
    return 0;
}
```

**cleardevice**

**Syntax**

```c
#include >graphics.h<
```
void cleardevice(void);

**Description**

cleardevice erases (that is, fills with the current background color) the entire graphics screen and moves the CP (current position) to home (0,0).

**Return Value**

None.

**See also**
clearviewport

**Example**

```c
#include <graphics.h>
#include <stdlib.h>
#include <stdio.h>
#include <conio.h>

int main(void) {
    /* request autodetection */
    int gdriver = DETECT, gmode, errorcode;
    int midx, midy;

    /* initialize graphics and local variables */
    initgraph(&gdriver, &gmode, "");

    /* read result of initialization */
    errorcode = graphresult();
    if (errorcode != grOk) {    /* an error occurred */
        printf("Graphics error: %s\n", grapherrormsg(errorcode));

        printf("Press any key to halt: ");
        getch();
        exit(1);                 /* terminate with an error code */
    }

    midx = getmaxx() / 2;
    midy = getmaxy() / 2;
    setcolor(getmaxcolor());

    /* for centering screen messages */
    settextjustify(CENTER_TEXT, CENTER_TEXT);

    /* output a message to the screen */
    outtextxy(midx, midy, "Press any key to clear the screen:");

    getch();   /* wait for a key */
    cleardevice();   /* clear the screen */

    /* output another message */
    outtextxy(midx, midy, "Press any key to quit:");
    /* clean up */
    getch();
    closegraph();
    return 0;
}
```
clearmouseclick

Syntax
#include "winbgim.h"
void clearmouseclick(int kind);

Description
The clearmouseclick function is available in the winbgim implementation of BGI graphics. This is just like getmouseclick, except it does not provide the x and y coordinates of the event. The value of the argument kind may be any of the constants listed above. After calling getmouseclick, for a particular kind of event, the ismouseclick will return false for that kind of event until another such event occurs. The kind argument to clearmouseclick is one of these constants from the winbgim.h file:
WM_MOUSEMOVE
if you want to detect a mouse movement
WM_LBUTTONDOWN
...detect when the left mouse button is double clicked
WM_LBUTTONDOWN
...detect when the left mouse button is clicked down
WM_LBUTTONUP
...detect when the left mouse button is released up
WM_MBUTTONDOWN
...detect when the middle mouse button is double clicked
WM_MBUTTONDOWN
...detect when the middle mouse button is clicked down
WM_MBUTTONUP
...detect when the middle mouse button is released up
WM_RBUTTONDOWN
...detect when the right mouse button is double clicked
WM_RBUTTONDOWN
...detect when the right mouse button is clicked down
WM_RBUTTONUP
...detect when the right mouse button is released up
The middle mouse button handlers aren't working on my machine. I haven't yet tracked down the reason--it could be a broken mouse or it could be a bug in my programming.

See also
ismouseclick
getmouseclick

Example
/* mouse example */
#include "winbgim.h"

void main(void)
{
    // Put the machine into graphics mode and get the maximum
    // coordinates:
    initwindow(450, 300);
    maxx = getmaxx();
maxy = getmaxy();

// Draw a white circle with red inside and a radius of 50 pixels:
setfillstyle(SOLID_FILL, RED);
setcolor(WHITE);
 fillellipse(maxx/2, maxy/2, 50, 50);

// Print a message and wait for a red pixel to be double clicked:
settextstyle(DEFAULT_FONT, HORIZ_DIR, 2);
outtextxy(20, 20, "Left click " << LIMIT << " times to end.");
setcolor(BLUE);
divisor = 2;
while (count < LIMIT)
{
    triangle(maxx/divisor, maxy/divisor);
    delay(500);
    divisor++;
    if (ismouseclick(WM_LBUTTONDOWN))
    {
        clearmouseclick(WM_LBUTTONDOWN);
        count++;
    }
}

// Switch back to text mode:
closegraph();

---

clearviewport

**Syntax**

```
#include <graphics.h>
void clearviewport(void);
```

**Description**

clearviewport erases the viewport and moves the CP (current position) to home (0,0), relative to the viewport.

**Return Value**

None.

**See also**

clearedevice
getviewsettings
setviewport

**Example**

```
/* clearviewport example */

#include <graphics.h>
#include <stdlib.h>
#include <stdio.h>
#include <conio.h>

#define CLIP_ON 1  /* activates clipping in viewport */

int main(void)
{
    /* request autodetection */
```
int gdriver = DETECT, gmode, errorcode, ht;

/* initialize graphics and local variables */
initgraph(&gdriver, &gmode, "");

/* read result of initialization */
errorcode = graphresult();
if (errorcode != grOk) { /* an error occurred */
    printf("Graphics error: %s\n", grapherrormsg(errorcode));

    printf("Press any key to halt:");
    getch();
    exit(1); /* terminate with an error code */
}

setcolor(getmaxcolor());
ht = textheight("\n");

/* message in default full-screen viewport */
outtextxy(0, 0, "* <-- (0, 0) in default viewport");

/* create a smaller viewport */
setviewport(50, 50, getmaxx()-50, getmaxy()-50, CLIP_ON);

/* display some messages */
outtextxy(0, 0, "* <-- (0, 0) in smaller viewport");

outtextxy(0, 2*ht, "Press any key to clear viewport:");

getch(); /* wait for a key */
clearviewport(); /* clear the viewport */
/* output another message */
outtextxy(0, 0, "Press any key to quit:");

/* clean up */
getch();
closegraph();
return 0;

---

closegraph

Syntax

```
#include <graphics.h>
void closegraph(void);
```

Description

closegraph deallocates all memory allocated by the graphics system, then restores the
screen to the mode it was in before you called initgraph. (The graphics system
deallocates memory, such as the drivers, fonts, and an internal buffer, through a call to
_graphfreemem.)

Return Value

None.

See also

initgraph
setgraphbufsize
Example
/* closegraph example */

#include <graphics.h>
#include <stdlib.h>
#include <stdio.h>
#include <conio.h>

int main(void)
{
    /* request autodetection */
    int gdriver = DETECT, gmode, errorcode, x, y;

    /* initialize graphics mode */
    initgraph(&gdriver, &gmode, "");

    /* read result of initialization */
    errorcode = graphresult();

    if (errorcode != grOk) {   /* an error occurred */
        printf("Graphics error: %s\n", grapherrormsg(errorcode));
        printf("Press any key to halt: ");
        getch();
        exit(1);   /* terminate with an error code */
    }

    x = getmaxx() / 2;
    y = getmaxy() / 2;

    /* output a message */
    settextjustify(CENTER_TEXT, CENTER_TEXT);
    outtextxy(x, y, "Press a key to close the graphics system:");

    getch();   /* wait for a key */
    /* closes down the graphics system */
    closegraph();
    printf("We're now back in text mode.\n");
    printf("Press any key to halt:");
    getch();
    return 0;
}

delay

Syntax
#include "winbgim.h"
void delay(int millisec);

Description
The delay function is available in the winbgim implementation of BGI graphics. You do not need to include conio.h; just include winbgim.h. The function pauses the computation for the specified number of milliseconds.

Return Value
None.
See also None.

Example

```c
/* delay example */

#include "winbgim.h"

int main(void)
{
    int midx, midy, i;

    /* initialize the window size */
    initwindow(100, 100);
    midx = getmaxx() / 2;
    midy = getmaxy() / 2;

    /* loop through the fill patterns with 4 second delays */
    for (i=SOLID_FILL; i<USER_FILL; i++) {
        /* set the fill style */
        setfillstyle(i, getmaxcolor());

        /* draw the bar */
        bar(midx-50, midy-50, midx+50, midy+50);
        delay(4000);
    }

    /* clean up */
    closegraph();
    return 0;
}
```

detectgraph

Syntax

```c
#include <graphics.h>
void detectgraph(int *graphdriver, int *graphmode);
```

Description

detectgraph detects your system's graphics adapter and chooses the mode that provides the highest resolution for that adapter. If no graphics hardware is detected, *graphdriver is set to grNotDetected (-2), and graphresult returns grNotDetected (-2). *graphdriver is an integer that specifies the graphics driver to be used. You can give it a value using a constant of the graphics_drivers enumeration type defined in graphics.h and listed as follows:

<table>
<thead>
<tr>
<th>graphics_drivers constant</th>
<th>Numeric value</th>
</tr>
</thead>
<tbody>
<tr>
<td>DETECT</td>
<td>0 (requests autodetect)</td>
</tr>
<tr>
<td>CGA</td>
<td>1</td>
</tr>
<tr>
<td>MCGA</td>
<td>2</td>
</tr>
<tr>
<td>EGA</td>
<td>3</td>
</tr>
<tr>
<td>EGA64</td>
<td>4</td>
</tr>
</tbody>
</table>
*graphmode is an integer that specifies the initial graphics mode (unless *graphdriver equals DETECT; in which case, *graphmode is set to the highest resolution available for the detected driver). You can give *graphmode a value using a constant of the graphics_modes enumeration type defined in graphics.h and listed as follows.

<table>
<thead>
<tr>
<th>Graphics</th>
<th>Columns</th>
</tr>
</thead>
<tbody>
<tr>
<td>Driver</td>
<td>graphics_mode</td>
</tr>
<tr>
<td>CGA</td>
<td>CGAC0</td>
</tr>
<tr>
<td></td>
<td>CGAC1</td>
</tr>
<tr>
<td></td>
<td>CGAC2</td>
</tr>
<tr>
<td></td>
<td>CGAC3</td>
</tr>
<tr>
<td></td>
<td>CGAH1</td>
</tr>
<tr>
<td>MCGA</td>
<td>MCGAC0</td>
</tr>
<tr>
<td></td>
<td>MCGAC1</td>
</tr>
<tr>
<td></td>
<td>MCGAC2</td>
</tr>
<tr>
<td></td>
<td>MCGAC3</td>
</tr>
<tr>
<td></td>
<td>MCGAMED</td>
</tr>
<tr>
<td></td>
<td>MCGAH1</td>
</tr>
<tr>
<td>EGA</td>
<td>EGALO</td>
</tr>
<tr>
<td></td>
<td>EGAHI</td>
</tr>
<tr>
<td>EGA64</td>
<td>EGA64LO</td>
</tr>
<tr>
<td></td>
<td>EGA64HI</td>
</tr>
<tr>
<td>EGA-MONO</td>
<td>EGAMONOHI</td>
</tr>
<tr>
<td></td>
<td>EGAMONOHI</td>
</tr>
<tr>
<td>HERC</td>
<td>HERCMONOHI</td>
</tr>
<tr>
<td>ATT400</td>
<td>ATT400C0</td>
</tr>
<tr>
<td></td>
<td>ATT400C1</td>
</tr>
</tbody>
</table>
Note: The main reason to call detectgraph directly is to override the graphics mode that detectgraph recommends to initgraph.

Return Value
None.

Windows Notes
The winbgim version of detectgraph returns VGA for the graphdriver and VGAHI for the graphmode, regardless of the machine's hardware. However, the screen is not necessarily 640 x 480.

See also
graphresult
initgraph

Example
/* detectgraph example */

#include <graphics.h>
#include <stdlib.h>
#include <stdio.h>
#include <conio.h>

/* the names of the various cards supported */
char *dname[] = { "requests detection",
    "a CGA",
    "an MCGA",
    "an EGA",
    "a 64K EGA",
    "a monochrome EGA",
    "an IBM 8514",
    "a Hercules monochrome",
    "an AT&T 6300 PC",
    "a VGA",
    "an IBM 3270 PC"
};

int main(void)
{
    /* used to return detected hardware info. */
    int gdriver, gmode, errorcode;
/* detect the graphics hardware available */
detectgraph(&gdriver, &gmode);

/* read result of detectgraph call */
errorcode = graphresult();
if (errorcode != grOk) { /* an error occurred */
    printf("Graphics error: %s\n", grapherrormsg(errorcode));
    printf("Press any key to halt:");
    getch();
    exit(1);               /* terminate with an error code */
}

/* display the information detected */
clrscr();
printf("You have %s video display card.\n", dname[gdriver]);
printf("Press any key to halt:");
getch();
return 0;

---

drawpoly

**Syntax**
```c
#include <graphics.h>
void drawpoly(int numpoints, int *polypoints);
```

**Description**
drawpoly draws a polygon with numpoints points, using the current line style and color.
*polypoints points to a sequence of (numpoints * 2) integers. Each pair of integers gives the x- and y-coordinates of a point on the polygon.
In order to draw a closed figure with n vertices, you must pass n + 1 coordinates to drawpoly where the nth coordinate is equal to the 0th.

**Return Value**
None.

**See also**
- **fillpoly**
- **floodfill**
- **graphresult**
- **setwritemode**

**Example**
```c
/* drawpoly example */

#include <graphics.h>
#include <stdlib.h>
#include <stdio.h>
#include <conio.h>

int main(void)
{
    /* request autodetection */
    int gdriver = DETECT, gmode, errorcode;
    int maxx, maxy;
```
```c
int poly[10]; /* our polygon array */

/* initialize graphics and local variables */
initgraph(&gdriver, &gmode, "");

/* read result of initialization */
errorcode = graphresult();
if (errorcode != grOk) { /* an error occurred */
    printf("Graphics error: %s\n", grapherrormsg(errorcode));
    printf("Press any key to halt: ");
    getch();
    exit(1); /* terminate with an error code */
}

maxx = getmaxx();
maxy = getmaxy();
poly[0] = 20; /* first vertex */
poly[1] = maxy / 2;
poly[2] = maxx - 20; /* second vertex */
poly[3] = 20;
poly[4] = maxx - 50; /* third vertex */
poly[5] = maxy - 20;
poly[6] = maxx / 2; /* fourth vertex */
poly[7] = maxy / 2;
poly[8] = poly[0]; /* drawpoly doesn't automatically close */
poly[9] = poly[1]; /* the polygon, so we close it */

drawpoly(5, poly); /* draw the polygon */

/* clean up */
getch();
closegraph();
return 0;
```

eellipse

**Syntax**
```c
#include <graphics.h>
void ellipse(int x, int y, int stangle, int endangle, int xradius, int yradius);
```

**Description**
ellipse draws an elliptical arc in the current drawing color with its center at (x,y) and the horizontal and vertical axes given by xradius and yradius, respectively. The ellipse travels from stangle to endangle. If stangle equals 0 and endangle equals 360, the call to ellipse draws a complete ellipse.

The angle for ellipse is reckoned counterclockwise, with 0 degrees at 3 o'clock, 90 degrees at 12 o'clock, and so on.

The linestyle parameter does not affect arcs, circles, ellipses, or pie slices. Only the thickness parameter is used.

**Return Value**
None.
**See also**

- arc
- circle
- fillellipse
- sector

**Example**

```c
/* ellipse example */

#include <graphics.h>
#include <stdlib.h>
#include <stdio.h>
#include <conio.h>

int main(void)
{
    /* request autodetection */
    int gdriver = DETECT, gmode, errorcode;
    int midx, midy;
    int stangle = 0, endangle = 360;
    int xradius = 100, yradius = 50;

    /* initialize graphics and local variables */
    initgraph(&gdriver, &gmode, "");

    /* read result of initialization */
    errorcode = graphresult();
    if (errorcode != grOk) { /* an error occurred */
        printf("Graphics error: %s\n", grapherrormsg(errorcode));
        printf("Press any key to halt:");
        getch();
        exit(1); /* terminate with an error code */
    }

    midx = getmaxx() / 2;
    midy = getmaxy() / 2;
    setcolor(getmaxcolor());

    /* draw ellipse */
    ellipse(midx, midy, stangle, endangle, xradius, yradius);

    /* clean up */
    getch();
    closegraph();
    return 0;
}
```

---

**fillellipse**

**Syntax**

```c
#include <graphics.h>
void fillellipse(int x, int y, int xradius, int yradius);
```

**Description**

Draws an ellipse using (x,y) as a center point and xradius and yradius as the horizontal and vertical axes, and fills it with the current fill color and fill pattern.
Return Value
None.

See also
arc
circle
ellipse
pieslice

Example

/* fillellipse example */

#include <graphics.h>
#include <stdlib.h>
#include <stdio.h>
#include <conio.h>

int main(void)
{
    /* request autodetection */
    int gdriver = DETECT, gmode, errorcode;
    int midx, midy, i;
    int xradius = 100, yradius = 50;

    /* initialize graphics and local variables */
    initgraph(&gdriver, &gmode, "");

    /* read result of initialization */
    errorcode = graphresult();
    if (errorcode != grOk) { /* an error occurred */
        printf("Graphics error: %s\n", grapherrormsg(errorcode));

        printf("Press any key to halt:");
        getch();
        exit(1); /* terminate with an error code */
    }

    midx = getmaxx() / 2;
    midy = getmaxy() / 2;

    /* loop through the fill patterns */
    for (i = EMPTY_FILL; i < USER_FILL; i++) {
        /* set fill pattern */
        setfillstyle(i, getmaxcolor());

        /* draw a filled ellipse */
        fillellipse(midx, midy, xradius, yradius);
        getch();
    }

    /* clean up */
    closegraph();

    return 0;
}

fillpoly
# Syntax
```c
#include <graphics.h>
void fillpoly(int numpoints, int *polypoints);
```

## Description
`fillpoly` draws the outline of a polygon with `numpoints` points in the current line style and color (just as `drawpoly` does), then fills the polygon using the current fill pattern and fill color. 

`polypoints` points to a sequence of `(numpoints * 2)` integers. Each pair of integers gives the x- and y-coordinates of a point on the polygon.

## Return Value
None.

## See also
- `drawpoly`
- `floodfill`
- `graphresult`
- `setfillstyle`

## Example
```c
/* fillpoly example */

#include <graphics.h>
#include <stdlib.h>
#include <stdio.h>
#include <conio.h>

int main(void)
{
    /* request autodetection */
    int gd = DETECT, gm, errorcode;
    int i, maxx, maxy;

    /* our polygon array */
    int poly[8];

    /* initialize graphics and local variables */
    initgraph(&gd, &gm, "");

    /* read result of initialization */
    errorcode = graphresult();
    if (errorcode != grOk) { /* an error occurred */
        printf("Graphics error: %s\n", grapherrormsg(errorcode));
        printf("Press any key to halt:");
        getch();
        exit(1);               /* terminate with an error code */
    }

    maxx = getmaxx();
    maxy = getmaxy();

    poly[0] = 20;             /* first vertex */
    poly[1] = maxy / 2;
    poly[2] = maxx - 20;      /* second vertex */
    poly[3] = 20;
    poly[4] = maxx - 50;      /* third vertex */
    poly[5] = maxy - 20;
    poly[6] = maxy / 2;       /* fourth, fillpoly automatically */
    poly[7] = maxy / 2;       /* closes the polygon */
}
```
/* loop through the fill patterns */
for (i=EMPTY_FILL; i<USER_FILL; i++) {
    /* set fill pattern */
    setfillstyle(i, getmaxcolor());

    /* draw a filled polygon */
    fillpoly(4, poly);
    getch();
}

/* clean up */
closegraph();
return 0;
}

---

**floodfill**

**Syntax**
```c
#include <graphics.h>
void floodfill(int x, int y, int border);
```

**Description**

floodfill fills an enclosed area on bitmap devices. (x,y) is a "seed point" within the enclosed area to be filled. The area bounded by the color border is flooded with the current fill pattern and fill color. If the seed point is within an enclosed area, the inside will be filled. If the seed is outside the enclosed area, the exterior will be filled. Use fillpoly instead of floodfill whenever possible so that you can maintain code compatibility with future versions.

floodfill does not work with the IBM-8514 driver.

**Return Value**

If an error occurs while flooding a region, graphresult returns a value of -7.

**Windows Notes**

The **winbgim** version allows the border argument to be an ordinary BGI color (from 0 to 15) or an RGB color.

**See also**

- drawpoly
- fillpoly
- graphresult
- setcolor
- setfillstyle

**Example**
```c
/* floodfill example */

#include <graphics.h>
#include <stdlib.h>
#include <stdio.h>
#include <conio.h>

int main(void)
{
    /* request autodetection */
    int gdriver = DETECT, gmode, errorcode;
```
int maxx, maxy;

/* initialize graphics and local variables */
initgraph(&gdriver, &gmode, "");

/* read result of initialization */
errorcode = graphresult();
if (errorcode != grOk) { /* an error occurred */
    printf("Graphics error: %s\n", grapherrormsg(errorcode));
    printf("Press any key to halt:");
    getch();
    exit(1);  /* terminate with an error code */
}

maxx = getmaxx();
maxy = getmaxy();

/* select drawing color */
setcolor(getmaxcolor());

/* select fill color */
setfillstyle(SOLID_FILL, getmaxcolor());

/* draw a border around the screen */
rectangle(0, 0, maxx, maxy);

/* draw some circles */
circle(maxx / 3, maxy /2, 50);
circle(maxx / 2, 20, 100);
circle(maxx-20, maxy-50, 75);
circle(20, maxy-20, 25);

/* wait for a key */
getch();

/* fill in bounded region */
floodfill(2, 2, getmaxcolor());

/* clean up */
getch();
closegraph();
return 0;


getactivepage

Syntax
#include "winbgim.h"
    int getactivepage(void);

Description
WIN
The getactivepage function is available in the winbgim implementation of BGI graphics. getactivepage gets the page number of the currently active page (where drawing takes place).

The active graphics page might not be the one you see onscreen, depending on how many graphics pages are available on your system.

The original winbgim was designed to support up to 16 pages, but I have only used pages 1 and 2 myself. NOTE: Using page number 0 might mess up the colors. I use pages 1-2 for double buffering.

**Return Value**
The page number of the currently active page.

---

### getarccoords

**Syntax**
```c
#include <graphics.h>
void getarccoords(struct arccoordstype *arccoords);
```

**Description**
getarccoords fills in the arccoordstype structure pointed to by arccoords with information about the last call to arc. The arccoordstype structure is defined in graphics.h as follows:

```c
struct arccoordstype {
    int x, y;
    int xstart, ystart, xend, yend;
};
```

The members of this structure are used to specify the center point (x,y), the starting position (xstart, ystart), and the ending position (xend, yend) of the arc. These values are useful if you need to make a line meet at the end of an arc.

**Return Value**
None.

**See also**
Arc
fillellipse
sector

**Example**
```c
/* getarccoords example */

#include <graphics.h>
#include <stdlib.h>
#include <stdio.h>
#include <conio.h>

int main(void)
{
    /* request autodetection */
    int gdriver = DETECT, gmode, errorcode;
    struct arccoordstype arcinfo;
    int midx, midy;
    int stangle = 45, endangle = 270;
    char ssstr[80], estrstr[80];
```
/* initialize graphics and local variables */
initgraph(&gdriver, &gmode, "");

/* read result of initialization */
errorcode = graphresult();
if (errorcode != grOk) {  /* an error occurred */
    printf("Graphics error: %s\n", grapherrormsg(errorcode));
    printf("Press any key to halt:");
    getch();
    exit(1);               /* terminate with an error code */
}

midx = getmaxx() / 2;
midy = getmaxy() / 2;

/* draw arc and get coordinates */
setcolor(getmaxcolor());
arc(midx, midy, stangle, endangle, 100);
getarccoords(&arcinfo);

/* convert arc information into strings */
sprintf(sstr, "*- (%d, %d)\n", arcinfo.xstart, arcinfo.ystart);
sprintf(estr, "*- (%d, %d)\n", arcinfo.xend, arcinfo.yend);

/* output the arc information */
outtextxy(arcinfo.xstart, arcinfo.ystart, sstr);
outtextxy(arcinfo.xend, arcinfo.yend, estr);

/* clean up */
getch();
closegraph();
return 0;
}

getaspectratio

Syntax
#include <graphics.h>
void getaspectratio(int *xasp, int *yasp);

Description
The y aspect factor, *yasp, is normalized to 10,000. On all graphics adapters except
the VGA, *xasp (the x aspect factor) is less than *yasp because the pixels are taller
than they are wide. On the VGA, which has "square" pixels, *xasp equals *yasp. In
general, the relationship between *yasp and *xasp can be stated as
*yasp = 10,000
*xasp <= 10,000
getaspectratio gets the values in *xasp and *yasp.

Return Value
None.

See also
None.
Example

/* getaspectratio example */

#include <graphics.h>
#include <stdlib.h>
#include <stdio.h>
#include <conio.h>

main()
{
    /* request autodetection */
    int gdriver = DETECT, gmode, errorcode;
    int xasp, yasp, midx, midy;

    /* initialize graphics and local variables */
    initgraph(&gdriver, &gmode, "");

    /* read result of initialization */
    errorcode = graphresult();
    if (errorcode != grOk) { /* an error occurred */
        printf("Graphics error: %s\n", grapherrormsg(errorcode));
        printf("Press any key to halt:");
        getch();
        exit(1); /* terminate with an error code */
    }

    midx = getmaxx() / 2;
    midy = getmaxy() / 2;
    setcolor(getmaxcolor());

    /* get current aspect ratio settings */
    getaspectratio(&xasp, &yasp);

    /* draw normal circle */
    circle(midx, midy, 100);
    getch();

    /* draw wide circle */
    cleardevice();
    setaspectratio(xasp/2, yasp);
    circle(midx, midy, 100);
    getch();

    /* draw narrow circle */
    cleardevice();
    setaspectratio(xasp, yasp/2);
    circle(midx, midy, 100);

    /* clean up */
    getch();
    closegraph();
    return 0;
}

getbkcolor
#include <graphics.h>
int getbkcolor(void);

getbkcolor returns the current background color. (See the table in setbkcolor for details.)

Return Value
getbkcolor returns the current background color.

Windows Notes [WIN]
In the winbgim version, the user might set the background color to an RGB color. Therefore, the return value from getbkcolor might be an ordinary BGI color (integer from 0 to 15) or an RGB color.

See also
getcolor
getmaxcolor
getpalette
setbkcolor

Example
/* getbkcolor example */
#include <graphics.h>
#include <stdlib.h>
#include <string.h>
#include <stdio.h>
#include <conio.h>

int main(void)
{
    /* request autodetection */
    int gdriver = DETECT, gmode, errorcode;
    int bkcolor, midx, midy;
    char bkname[35];

    /* initialize graphics and local variables */
    initgraph(&gdriver, &gmode, "");

    /* read result of initialization */
    errorcode = graphresult();
    if (errorcode != grOk) { /* an error occurred */
        printf("Graphics error: %s\n", grapherrormsg(errorcode));
        printf("Press any key to halt:");
        getch();
        exit(1); /* terminate with an error code */
    }

    midx = getmaxx() / 2;
    midy = getmaxy() / 2;
    setcolor(getmaxcolor());

    /* for centering text on the display */
    settextjustify(CENTER_TEXT, CENTER_TEXT);

    /* get the current background color */
    bkcolor = getbkcolor();

    /* convert color value into a string */
getch

**Syntax**
```c
#include "winbgim.h"
int getch(void);
```

**Description**
The `getch` function is available in the `winbgim` implementation of BGI graphics. You do not need to include `conio.h`; just include `winbgim.h`. The function reads one character from the keyboard and returns its ASCII value (without waiting for a return key). In order to work, the user must click in the graphics window (i.e., the Windows focus must be in the graphics window). For special keys, the `getch` function first returns ASCII 0. The next call will then return one of these special keys:

```c
#define KEY_HOME       71
#define KEY_UP         72
#define KEY_PGUP       73
#define KEY_LEFT       75
#define KEY_CENTER     76
#define KEY_RIGHT      77
#define KEY_END        79
#define KEY_DOWN       80
#define KEY_PGDN       81
#define KEY_INSERT     82
#define KEY_DELETE     83
#define KEY_F1         59
#define KEY_F2         60
#define KEY_F3         61
#define KEY_F4         62
#define KEY_F5         63
#define KEY_F6         64
#define KEY_F7         65
#define KEY_F8         66
#define KEY_F9         67
```

**Return Value**
The ASCII value of a key that has been pressed.

**See also**
`kbhit`

**Example**
```c
#include "winbgim.h"
#include <stdio.h>      // Provides sprintf
#include <iostream.h>   // Provides cout
```
void outintxy(int x, int y, int value);

int main()
{
    int i;
    char c;

    // Initialize the graphics window.
    init_window(400, 300);

    // Convert some numbers to strings and draw them in graphics window:
    outtextxy(10, 10, "Here are some numbers:");
    for (i = 10; i <= 100; i += 10)
        outintxy(20, i+10, i);

    // Get some characters from the keyboard until an X is typed:
    outtextxy(20, 130, "Click in this graphics window,");
    outtextxy(20, 140, "and then press arrow keys.");
    outtextxy(20, 150, "Watch the console window while pressing.");
    outtextxy(20, 160, "Press X to exit.");
    do
    {
        c = (char) getch( );
        if (c != 0)
            cout << "That is ASCII value: " << (int) c << endl;
        else
        {
            // Process one of the special keys:
            c = (char) getch( );
            switch (c)
            {
            case KEY_HOME:    cout << "Home key."   << endl;
            case KEY_UP:      cout << "Up key."     << endl;
            case KEY_PGUP:    cout << "PgUp key."   << endl;
            case KEY_LEFT:    cout << "Left key."   << endl;
            case KEY_CENTER:  cout << "Center key." << endl;
            case KEY_RIGHT:   cout << "Right key."  << endl;
            case KEY_END:     cout << "End key."    << endl;
            case KEY_DOWN:    cout << "Down key."   << endl;
            case KEY_PGDN:    cout << "PgDn key."   << endl;
            case KEY_INSERT:  cout << "Insert key." << endl;
            case KEY_DELETE:  cout << "Delete key." << endl;
            case KEY_F1:      cout << "F1 key."     << endl;
            case KEY_F2:      cout << "F2 key."     << endl;
            case KEY_F3:      cout << "F3 key."     << endl;
            break;
            }
        }
    }
    while (c != 'X');
case KEY_F4:      cout << "F4 key."     << endl;
break;
case KEY_F5:      cout << "F5 key."     << endl;
break;
case KEY_F6:      cout << "F6 key."     << endl;
break;
case KEY_F7:      cout << "F7 key."     << endl;
break;
case KEY_F8:      cout << "F8 key."     << endl;
break;
case KEY_F9:      cout << "F9 key."     << endl;
break;
        default: cout << "Unknown extended key." << endl;
    }
}
while ((c != 'x') && (c != 'X'));

closegraph( );

}   void outintxy(int x, int y, int value)
{
    char digit_string[20];
sprintf(digit_string, "%d", value);
    outtextxy(x, y, digit_string);
}

getcolor

Syntax
#include <graphics.h>
int getcolor(void);

Description
getcolor returns the current drawing color. The drawing color is the value to which pixels are set when lines and so on are drawn. For example, in CGAC0 mode, the palette contains four colors: the background color, light green, light red, and yellow. In this mode, if getcolor returns 1, the current drawing color is light green.

Return Value
getcolor returns the current drawing color.

Windows Notes [WIN]
In the winbgim version, the user might set the drawing color to an RGB color. Therefore, the return value from getcolor might be an ordinary BGI color (integer from 0 to 15) or an RGB color.

See also
getbgcolor
getmaxcolor
getpalette
setcolor

Example
/* getcolor example */
#include <graphics.h>
#include <stdlib.h>
#include <string.h>
#include <stdio.h>
#include <conio.h>

int main(void)
{
    /* request autodetection */
    int gdriver = DETECT, gmode, errorcode;
    int color, midx, midy;
    char colname[35];

    /* initialize graphics and local variables */
    initgraph(&gdriver, &gmode, "");

    /* read result of initialization */
    errorcode = graphresult();
    if (errorcode != grOk) {  /* an error occurred */
        printf("Graphics error: %s\n", grapherrormsg(errorcode));
        printf("Press any key to halt:");
        getch();
        exit(1);               /* terminate with an error code */
    }

    midx = getmaxx() / 2;
    midy = getmaxy() / 2;
    setcolor(getmaxcolor());

    /* for centering text on the display */
    settextjustify(CENTER_TEXT, CENTER_TEXT);

    /* get the current drawing color */
    color = getcolor();

    /* convert color value into a string */
    itoa(color, colname, 10);
    strcat(colname, " is the current drawing color.");

    /* display a message */
    outtextxy(midx, midy, colname);

    /* clean up */
    getch();
    closegraph();
    return 0;
}

getdefaultpalette

### Syntax
```
#include <graphics.h>
struct palettetype *getdefaultpalette(void);
```

### Description
getdefaultpalette finds the palettetype structure that contains the palette initialized by the driver during initgraph.
Return Value
getdefaultpalette returns a pointer to the default palette set up by the current driver when that driver was initialized.

See also
getpalette
initgraph

Example
/* getdefaultpalette example */

#include <graphics.h>
#include <stdlib.h>
#include <stdio.h>
#include <conio.h>

int main(void)
{
    /* request autodetection */
    int gdriver = DETECT, gmode, errorcode;

    /* pointer to palette structure */
    struct palettetype *pal = NULL;
    int i;

    /* initialize graphics and local variables */
    initgraph(&gdriver, &gmode, "");

    /* read result of initialization */
    errorcode = graphresult();
    if (errorcode != grOk) { /* an error occurred */
        printf("Graphics error: %s\n", grapherrormsg(errorcode));
        printf("Press any key to halt: ");
        getch();
        exit(1); /* terminate with an error code */
    }

    /* return a pointer to the default palette */
    pal = getdefaultpalette();
    for (i=0; i<pal->size; i++) {
        printf("colors[%d] = %d\n", i, pal->colors[i]);
        getch();
    }

    /* clean up */
    getch();
    closegraph();
    return 0;
}

getdrivername

Syntax
#include <graphics.h>
char *getdrivername(void);

**Description**

After a call to initgraph, getdrivername returns the name of the driver that is currently loaded.

**Return Value**

getdrivername returns a pointer to a string with the name of the currently loaded graphics driver.

**Windows Notes**

The winbgim version of getdrivername returns "EGAVGA" for the driver name, regardless of how initgraph was called.

**See also**

initgraph

**Example**

```c
/* getdrivername example */

#include <graphics.h>
#include <stdlib.h>
#include <stdio.h>
#include <conio.h>

int main()
{
    /* request autodetection */
    int gdriver = DETECT, gmode, errorcode;

    /* stores the device driver name */
    char *drivername;

    /* initialize graphics and local variables */
    initgraph(&gdriver, &gmode, "");

    /* read result of initialization */
    errorcode = graphresult();
    if (errorcode != grOk) { /* an error occurred */
        printf("Graphics error: %s\n", grapherrormsg(errorcode));
        printf("Press any key to halt: ");
        getch();
        exit(1);               /* terminate with an error code */
    }

    setcolor(getmaxcolor());

    /* get the name of the device driver in use */
    drivername = getdrivername();

    /* for centering text onscreen */
    settextjustify(CENTER_TEXT, CENTER_TEXT);

    /* output the name of the driver */
    outtextxy(getmaxx() / 2, getmaxy() / 2, drivername);

    /* clean up */
    getch();
    closegraph();

    return 0;
}
```
getfillpattern

Syntax

```c
#include <graphics.h>
void getfillpattern(char *pattern);
```

Description

getfillpattern copies the user-defined fill pattern, as set by setfillpattern, into the 8-byte area pointed to by pattern.

pattern is a pointer to a sequence of 8 bytes, with each byte corresponding to 8 pixels in the pattern. Whenever a bit in a pattern byte is set to 1, the corresponding pixel will be plotted. For example, the following user-defined fill pattern represents a checkerboard:

```c
char checkboard[8] = {
    0xAA, 0x55, 0xAA, 0x55, 0xAA, 0x55, 0xAA, 0x55
};
```

Return Value

None.

See also

getfillsettings
setfillpattern

Example

```c
/* getfillpattern example */
#include <graphics.h>
#include <stdlib.h>
#include <stdio.h>
#include <conio.h>
int main(void) {
    /* request autodetection */
    int gdriver = DETECT, gmode, errorcode;
    int maxx, maxy;
    char pattern[8] = {0x00, 0x70, 0x20, 0x27, 0x25, 0x27, 0x04, 0x04};
    /* initialize graphics and local variables */
    initgraph(&gdriver, &gmode, "");
    /* read result of initialization */
    errorcode = graphresult();
    if (errorcode != grOk) {
        /* an error occurred */
        printf("Graphics error: %s\n", grapherrormsg(errorcode));
        printf("Press any key to halt:";)
        getch();
        exit(1); /* terminate with an error code */
    }
    maxx = getmaxx();
    maxy = getmaxy();
    setcolor(getmaxcolor());
    /* select a user-defined fill pattern */
    setfillpattern(pattern, getmaxcolor());
    /* fill the screen with the pattern */
    bar(0, 0, maxx, maxy);
    /* get the current user-defined fill pattern */
    getfillpattern(pattern); /* fill the screen with the new pattern */
    bar(0, 0, maxx, maxy); /* clean up */
    getch();
    closegraph();
    return 0;
}
```

getfillsettings

Syntax

```c
#include <graphics.h>
void getfillsettings(struct fillsettingstype *fillinfo);
```

Description


getfillsettings fills in the fillsettingstype structure pointed to by fillinfo with information about the current fill pattern and fill color. The fillsettingstype structure is defined in graphics.h as follows:

```c
struct fillsettingstype {
    int pattern;            /* current fill pattern */
    int color;              /* current fill color */
};
```

The functions bar, bar3d, fillpoly, floodfill, and pieslice all fill an area with the current fill pattern in the current fill color. There are 11 predefined fill pattern styles (such as solid, crosshatch, dotted, and so on). Symbolic names for the predefined patterns are provided by the enumerated type fill_patterns in graphics.h, as shown here:

<table>
<thead>
<tr>
<th>Name</th>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>EMPTY_FILL</td>
<td>0</td>
<td>Fill with background color</td>
</tr>
<tr>
<td>SOLID_FILL</td>
<td>1</td>
<td>Solid fill</td>
</tr>
<tr>
<td>LINE_FILL</td>
<td>2</td>
<td>Fill with ---</td>
</tr>
<tr>
<td>LTSLASH_FILL</td>
<td>3</td>
<td>Fill with ///</td>
</tr>
<tr>
<td>SLASH_FILL</td>
<td>4</td>
<td>Fill with ///, thick lines</td>
</tr>
<tr>
<td>BKSLASH_FILL</td>
<td>5</td>
<td>Fill with \, thick lines</td>
</tr>
<tr>
<td>LTBKSLASH_FILL</td>
<td>6</td>
<td>Fill with \.</td>
</tr>
<tr>
<td>HATCH_FILL</td>
<td>7</td>
<td>Light hatch fill</td>
</tr>
<tr>
<td>XHATCH_FILL</td>
<td>8</td>
<td>Heavy crosshatch fill</td>
</tr>
<tr>
<td>INTERLEAVE_FILL</td>
<td>9</td>
<td>Interleaving line fill</td>
</tr>
<tr>
<td>WIDE_DOT_FILL</td>
<td>10</td>
<td>Widely spaced dot fill</td>
</tr>
<tr>
<td>CLOSE_DOT_FILL</td>
<td>11</td>
<td>Closely spaced dot fill</td>
</tr>
<tr>
<td>USER_FILL</td>
<td>12</td>
<td>User-defined fill pattern</td>
</tr>
</tbody>
</table>

Note: All but EMPTY_FILL fill with the current fill color; EMPTY_FILL uses the current background color. In addition, you can define your own fill pattern. If pattern equals 12 (USER_FILL), then a user-defined fill pattern is being used; otherwise, pattern gives the number of a predefined pattern.

**Return Value**
None.

**Windows Notes**
In the winbgim version, the user might set the fill color to an RGB color. Therefore, the color in the fillsettingstype struct might be an ordinary BGI color (integer from 0 to 15) or an RGB color.

**See also**
getfillpattern
setfillpattern
setfillstyle

**Example**

```c
/* getfillsettings example */

#include
#include
#include
#include

/* the names of the fill styles supported */
char *fname[] = { "EMPTY_FILL", "SOLID_FILL", "LINE_FILL", "LTSLASH_FILL", "SLASH_FILL", "BKSLASH_FILL", "LTBKSLASH_FILL", "HATCH_FILL", "XHATCH_FILL", "INTERLEAVE_FILL", "WIDE_DOT_FILL", "CLOSE_DOT_FILL", "USER_FILL";
int main(void)
{
    /* request autodetection */
    int gdriver = DETECT, gmode, errorcode;
    struct fillsettingstype fillinfo;

    int midx, midy;
    char patstr[40], colstr[40];

    /* initialize graphics and local variables */
    initgraph(&gdriver, &gmode, "");

    /* read result of initialization */
    errorcode = graphresult();
    if (errorcode != grOk) { /* an error occurred */
        printf("Graphics error: %s\n", grapherrormsg(errorcode));
        printf("Press any key to halt:");
        getch();
        exit(1); /* terminate with an error code */
    }

    midx = getmaxx() / 2;
    midy = getmaxy() / 2;

    /* get info about current fill pattern and color */
    getfillsettings(&fillinfo);

    /* convert fill information into strings */
    sprintf(patstr, "%s is the fill style.", fname[fillinfo.pattern]);
    sprintf(colstr, "%d is the fill color.", fillinfo.color);

    /* display the information */
    settextjustify(CENTER_TEXT, CENTER_TEXT);
    outtextxy(midx, midy, patstr);
    outtextxy(midx, midy + 2 * textheight("W"), colstr);

    /* clean up */
    getch();
    closegraph();
    return 0;
}
The enumeration graphics_mode, defined in graphics.h, gives names for the predefined graphics modes. For a table listing these enumeration values, refer to the description for initgraph.

**Return Value**

getgraphmode returns the graphics mode set by initgraph or setgraphmode.

**Windows Notes**

The winbgim version of getgraphmode returns VGAHI for the graphmode, regardless of how initgraph was called. However, the screen is not necessarily 640 x 480.

See also

getmoderange
initgraph
restorecrtmode
setgraphmode

**Example**

```c
/* getgraphmode example */

#include <graphics.h>
#include <stdlib.h>
#include <stdio.h>
#include <conio.h>

int main(void)
{
    /* request autodetection */
    int gdriver = DETECT, gmode, errorcode;
    int midx, midy, mode;
    char numname[80], modename[80];

    /* initialize graphics and local variables */
    initgraph(&gdriver, &gmode, "");

    /* read result of initialization */
    errorcode = graphresult();
    if (errorcode != grOk) { /* an error occurred */
        printf("Graphics error: %s\n", grapherrormsg( errorcode));
        printf("Press any key to halt:");
        getch();
        exit(1); /* terminate with an error code */
    }

    midx = maxx() / 2;
    midy = maxy() / 2;

    /* get mode number and name strings */
    mode = getgraphmode();
    sprintf(numname, "%d is the current mode number.\n", mode);
    sprintf(modename, "%s is the current graphics mode.\n", getmodename(mode));

    /* display the information */
    settextjustify(CENTER_TEXT, CENTER_TEXT);
    outtextxy(midx, midy, numname);
    outtextxy(midx, midy + 2*textheight("W"), modename);

    /* clean up */
    getch();
}
```
getimage

Syntax

#include <graphics.h>
void getimage(int left, int top, int right, int bottom, void *bitmap);

Description

getimage copies an image from the screen to memory.
left, top, right, and bottom define the screen area to which the rectangle is copied.
bitmap points to the area in memory where the bit image is stored. The first two words of this area are used for the width and height of the rectangle; the remainder holds the image itself.

Return Value

None.

See also

getpixel
imagesize
putimage

Example

/* getimage example */

#include <graphics.h>
#include <stdlib.h>
#include <stdio.h>
#include <conio.h>
#include <alloc.h>

void save_screen(void *buf[4]);
void restore_screen(void *buf[4]);

int maxx, maxy;
int main(void)
{
    int gdriver=DETECT, gmode, errorcode;
    void *ptr[4];

    /* autodetect the graphics driver and mode */
    initgraph(&gdriver, &gmode, "");
    errorcode = graphresult(); /* check for any errors */
    if (errorcode != grOk) {
        printf("Graphics error: %s\n", grapherrormsg(errorcode));

        printf("Press any key to halt:");
        getch();
        exit(1);
    }

    closegraph();
    return 0;
}

maxx = getmaxx();
maxy = getmaxy();

/* draw an image on the screen */
rectangle(0, 0, maxx, maxy);
line(0, 0, maxx, maxy);
line(0, maxy, maxx, 0);
save_screen(ptr); /* save the current screen */
getch(); /* pause screen */
cleardevice(); /* clear screen */
restore_screen(ptr); /* restore the screen */
getch(); /* pause screen */

closegraph();
return 0;
}

void save_screen(void *buf[4])
{
    unsigned size;
    int ystart=0, yend, yincr, block;
    yincr = (maxy+1) / 4;
yend = yincr;

    /* get byte size of image */
    size = imagesize(0, ystart, maxx, yend);
    for (block=0; block<=3; block++) {
        if ((buf[block] = farmalloc(size)) == NULL) {
            closegraph();
            printf("Error: not enough heap space in save_screen().\n");
            exit(1);
        }
        getimage(0, ystart, maxx, yend, buf[block]);

        ystart = yend + 1;
yend += yincr + 1;
    }
}

void restore_screen(void *buf[4])
{
    int ystart=0, yend, yincr, block;
    yincr = (maxy+1) / 4;
yend = yincr;
    for (block=0; block<=3; block++) {
        putimage(0, ystart, buf[block], COPY_PUT);
        farfree(buf[block]);
        ystart = yend + 1;
yend += yincr + 1;
    }
}

getlinesettings

Syntax
#include <graphics.h>
void getlinesettings(struct linesettingstype *lineinfo);

**Description**

callinesettings fills a linesettingstype structure pointed to by lineinfo with information about the current line style, pattern, and thickness.

The linesettingstype structure is defined in graphics.h as follows:

```c
struct linesettingstype {
    int linestyle;
    unsigned upattern;
    int thickness;
};
```

linestyle specifies in which style subsequent lines will be drawn (such as solid, dotted, centered, dashed). The enumeration line_styles, defined in graphics.h, gives names to these operators:

<table>
<thead>
<tr>
<th>Name</th>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>SOLID_LINE</td>
<td>0</td>
<td>Solid line</td>
</tr>
<tr>
<td>DOTTED_LINE</td>
<td>1</td>
<td>Dotted line</td>
</tr>
<tr>
<td>CENTER_LINE</td>
<td>2</td>
<td>Centered line</td>
</tr>
<tr>
<td>DASHED_LINE</td>
<td>3</td>
<td>Dashed line</td>
</tr>
<tr>
<td>USERBIT_LINE</td>
<td>4</td>
<td>User-defined line</td>
</tr>
</tbody>
</table>

thickness specifies whether the width of subsequent lines drawn will be normal or thick.

<table>
<thead>
<tr>
<th>Name</th>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>NORM_WIDTH</td>
<td>1</td>
<td>1 pixel wide</td>
</tr>
<tr>
<td>THICK_WIDTH</td>
<td>3</td>
<td>3 pixels wide</td>
</tr>
</tbody>
</table>

**Return Value**
None.

**See also**
setlinestyle

**Example**

```c
/* getlinesettings example */

#include <graphics.h>
#include <stdlib.h>
#include <stdio.h>
#include <conio.h>

/* the names of the line styles supported */
char *lname[] = {"SOLID_LINE", "DOTTED_LINE", "CENTER_LINE", "DASHED_LINE", "USERBIT_LINE"};

int main(void)
{
    /* request autodetection */
    int gdriver = DETECT, gmode, errorcode;
    struct linesettingstype lineinfo;
    int midx, midy;
    char lstyle[80], lpattern[80], lwidth[80];

    /* initialize graphics and local variables */

    initgraph(&gdriver, &gmode, ",");
```
/* read result of initialization */
errorcode = graphresult();
if (errorcode != grOk) { /* an error occurred */
    printf("Graphics error: %s\n", grapherrormsg(errorcode));
    printf("Press any key to halt:");
    getch();
    exit(1);               /* terminate with an error code */
}

midx = getmaxx() / 2;
midy = getmaxy() / 2;

/* get information about current line settings */
getlinesettings(&lineinfo);

/* convert line information into strings */
sprintf(lstyle, "%s is the line style.",
        lname[lineinfo.linestyle]);
sprintf(lpattern, "0x%X is the user-defined line pattern.",
        lineinfo.upattern);
sprintf(lwidth, "%d is the line thickness.", lineinfo.thickness);

/* display the information */
settextjustify(CENTER_TEXT, CENTER_TEXT);
outtextxy(midx, midy, lstyle);
outtextxy(midx, midy+2*textheight("W"), lpattern);
outtextxy(midx, midy+4*textheight("W"), lwidth);

/* clean up */
getch();
closegraph();
return 0;
}

getmaxcolor

Syntax
#include <graphics.h>
int getmaxcolor(void);

Description
getmaxcolor returns the highest valid color value for the current graphics driver and
mode that can be passed to setcolor.
For example, on a 256K EGA, getmaxcolor always returns 15, which means that any
call to setcolor with a value from 0 to 15 is valid. On a CGA in high-resolution mode
or on a Hercules monochrome adapter, getmaxcolor returns a value of 1.

Windows Notes [WIN]
The winbgi version of getmaxcolor returns 15 for the maximum color. However, in
addition to the usual BGI colors (0 through 15), the programmer may also use RGB
colors.

Return Value
getmaxcolor returns the highest available color value.

See also
   getbkcolor
   getcolor
   getpalette
   getpalettesize
   setcolor

Example
   /* getmaxcolor example */

#include <graphics.h>
#include <stdlib.h>
#include <stdio.h>
#include <conio.h>

int main(void)
{
   /* request autodetection */
   int gdriver = DETECT, gmode, errorcode;
   int midx, midy;
   char colstr[80];

   /* initialize graphics and local variables */
   initgraph(&gdriver, &gmode, "");

   /* read result of initialization */
   errorcode = graphresult();
   if (errorcode != grOk) { /* an error occurred */
      printf("Graphics error: %s
", grapherrormsg(errorcode));

      printf("Press any key to halt: ");
      getch();
      exit(1);              /* terminate with an error code */
   }

   midx = getmaxx() / 2;
   midy = getmaxy() / 2;

   /* grab the color info. and convert it to a string */
   sprintf(colstr, "This mode supports colors 0..%d", getmaxcolor());

   /* display the information */
   settextjustify(CENTER_TEXT, CENTER_TEXT);
   outtextxy(midx, midy, colstr);

   /* clean up */
   getch();
   closegraph();

   return 0;
}

getmaxmode
Syntax
#include <graphics.h>
int getmaxmode(void);

Description
getmaxmode lets you find out the maximum mode number for the currently loaded
driver, directly from the driver. This gives it an advantage over getmoderange, which
works for Borland drivers only. The minimum mode is 0.

Return Value
getmaxmode returns the maximum mode number for the current driver.

See also
getmodename
getmiderange

guide

guide

guide

guide

guide

guide

guide

guide

Example
/* getmaxmode example */

#include <graphics.h>
#include <stdlib.h>
#include <stdio.h>
#include <conio.h>

int main(void)
{
    /* request autodetection */
    int gdriver = DETECT, gmode, errorcode;
    int midx, midy;
    char modestr[80];

    /* initialize graphics and local variables */
    initgraph(&gdriver, &gmode, "");

    /* read result of initialization */
    errorcode = graphresult();
    if (errorcode != grOk) { /* an error occurred */
        printf("Graphics error: %s\n", grapherrormsg(errorcode));
        printf("Press any key to halt:");
        getch();
        exit(1); /* terminate with an error code */
    }

    midx = getmaxx() / 2;
    midy = getmaxy() / 2;

    /* grab the mode info. and convert it to a string */
    sprintf(modestr, "This driver supports modes 0..%d",
            getmaxmode());

    /* display the information */
    settextjustify(CENTER_TEXT, CENTER_TEXT);
    outtextxy(midx, midy, modestr);

    /* clean up */
    getch();
    closegraph();

    return 0;
}
getmaxx

Syntax

```c
#include <graphics.h>
int getmaxx(void);
```

Description

getmaxx returns the maximum (screen-relative) x value for the current graphics driver and mode.
For example, on a CGA in 320*200 mode, getmaxx returns 319. getmaxx is invaluable for centering, determining the boundaries of a region onscreen, and so on.

Return Value

getmaxx returns the maximum x screen coordinate.

See also

getmax
getx

Example

```c
/* getmaxx example */

#include <graphics.h>
#include <stdlib.h>
#include <stdio.h>
#include <conio.h>

int main(void)
{
    /* request autodetection */
    int gdriver = DETECT, gmode, errorcode;
    int midx, midy;
    char xrange[80], yrange[80];

    /* initialize graphics and local variables */
    initgraph(&gdriver, &gmode, "");

    /* read result of initialization */
    errorcode = graphresult();
    if (errorcode != grOk) {  /* an error occurred */
        printf("Graphics error: %s\n", grapherrormsg(errorcode));

        printf("Press any key to halt:");
        getch();
        exit(1);               /* terminate with an error code */
    }

    midx = getmaxx() / 2;
    midy = getmaxy() / 2;

    /* convert max resolution values to strings */
    sprintf(xrange, "X values range from 0..%d", getmaxx());
    sprintf(yrange, "Y values range from 0..%d", getmaxy());

    /* display the information */
    settextjustify(CENTER_TEXT, CENTER_TEXT);
    outtextxy(midx, midy, xrange);
    outtextxy(midx, midy + textheight("W"), yrange);
```
getmaxy

Syntax
#include <graphics.h>
int getmaxy(void);

Description
getmaxy returns the maximum (screen-relative) y value for the current graphics driver and mode.
For example, on a CGA in 320*200 mode, getmaxy returns 199. getmaxy is invaluable for centering, determining the boundaries of a region onscreen, and so on.

Return Value
getmaxy returns the maximum y screen coordinate.

See also
getmaxx
gety

Example
/* getmaxy example */

#include <graphics.h>
#include <stdlib.h>
#include <stdio.h>
#include <conio.h>

int main(void)
{
    /* request autodetection */
    int gdriver = DETECT, gmode, errorcode;
    int midx, midy;
    char xrange[80], yrange[80];

    /* initialize graphics and local variables */
    initgraph(&gdriver, &gmode, "");

    /* read result of initialization */
    errorcode = graphresult();
    if (errorcode != grOk) {  /* an error occurred */
        printf("Graphics error: %s\n", grapherrormsg(errorcode));
        printf("Press any key to halt:");
        getch();
        exit(1);  /* terminate with an error code */
    }

    midx = getmaxx() / 2;
    midy = getmaxy() / 2;

    /* convert max resolution values into strings */
getmodename

Syntax

```c
#include <graphics.h>
char *getmodename(int mode_number);
```

Description

getmodename accepts a graphics mode number as input and returns a string containing the name of the corresponding graphics mode. The mode names are embedded in each driver. The return values ("320*200 CGA P1," "640*200 CGA", and so on) are useful for building menus or displaying status.

Return Value

getmodename returns a pointer to a string with the name of the graphics mode.

See also

getmaxmode
getmoderange

Example

```c
/* getmodename example */

#include <graphics.h>
#include <stdlib.h>
#include <stdio.h>
#include <conio.h>

int main(void)
{
    /* request autodetection */
    int gdriver = DETECT, gmode, errorcode;
    int midx, midy, mode;
    char numname[80], modename[80];

    /* initialize graphics and local variables */
    initgraph(&gdriver, &gmode, "");

    /* read result of initialization */
    errorcode = graphresult();
    if (errorcode != grOk) { /* an error occurred */
        printf("Graphics error: %s\n", grapherrormsg(errorcode));
        printf("Press any key to halt:");
        getch();
        closegraph();
        return 0;
    }
```
getch(); exit(1);               /* terminate with an error code */
}
midx = getmaxx() / 2;
midy = getmaxy() / 2;

/* get mode number and name strings */
mode = getgraphmode();
sprintf(numname, "%d is the current mode number.", mode);
sprintf(modename, "%s is the current graphics mode.", getmodename(mode));

/* display the information */
settextjustify(CENTER_TEXT, CENTER_TEXT);
outtextxy(midx, midy, numname);
outtextxy(midx, midy+2*textheight("W"), modename);

/* clean up */
getch();
closegraph();
return 0;
}

getmoderange

Syntax
#include <graphics.h>
void getmoderange(int graphdriver, int *lomode, int *himode);

Description
getmoderange gets the range of valid graphics modes for the given graphics driver, graphdriver. The lowest permissible mode value is returned in *lomode, and the highest permissible value is *himode. If graphdriver specifies an invalid graphics driver, both *lomode and *himode are set to -1. If the value of graphdriver is -1, the currently loaded driver modes are given.

Return Value
None.

See also
getgraphmode
getmaxmode
getmodename
initgraph
setgraphmode

Example
/* getmoderange example */

#include <graphics.h>
#include <stdlib.h>
#include <stdio.h>
#include <conio.h>

int main(void)
{ /* request autodetection */
int gdriver = DETECT, gmode, errorcode;
int midx, midy;
int low, high;
char mrange[80];

/* initialize graphics and local variables */
initgraph(&gdriver, &gmode, "");

/* read result of initialization */
errorcode = graphresult();
if (errorcode != grOk) { /* an error occurred */
    printf("Graphics error: %s\n", grapherrormsg(errorcode));

    printf("Press any key to halt:");
    getch();
    exit(1); /* terminate with an error code */
}

midx = getmaxx() / 2;
midy = getmaxy() / 2;

/* get the mode range for this driver */
getmoderange(gdriver, &low, &high);

/* convert mode range info. into strings */
sprintf(mrange, "This driver supports modes %d..%d", low, high);

/* display the information */
settextjustify(CENTER_TEXT, CENTER_TEXT);
outtextxy(midx, midy, mrange);

/* clean up */
getch();
closegraph();
return 0;
}

getmouseclick

Syntax
#include "winbgim.h"
void getmouseclick(int kind, int x, int y);

Description
The getmouseclick function is available in the winbgim implementation of BGI graphics. This function sets x and y to the pixel coordinates of an unprocessed event of the specified kind. If there is no such event, then the function sets both x and y to -1. The value of the argument kind may be any of the constants listed above. After calling getmouseclick, for a particular kind of event, the ismouseclick will return false for that kind of event until another such event occurs.
The kind argument to `getmouseclick` is one of these constants from the `winbgim.h` file:

- `WM_MOUSEMOVE` if you want to detect a mouse movement
- `WM_LBUTTONDOWN`...detect when the left mouse button is clicked down
- `WM_LBUTTONUP`...detect when the left mouse button is released up
- `WM_MBUTTONDOWN`...detect when the middle mouse button is clicked down
- `WM_MBUTTONUP`...detect when the middle mouse button is released up
- `WM_RBUTTONDOWN`...detect when the right mouse button is clicked down
- `WM_RBUTTONUP`...detect when the right mouse button is released up

The middle mouse button handlers aren't working on my machine. I haven't yet tracked down the reason--it could be a broken mouse or it could be a bug in my programming.

See also
- `ismouseclick`
- `clearmouseclick`

Example

```c
/* mouse example */
#include "winbgim.h"

void main(void)
{
    int maxx, maxy;  // Maximum x and y pixel coordinates
    int x, y;        // Coordinates of the mouse click
    int divisor;     // Divisor for the length of a triangle side

    // Put the machine into graphics mode and get the maximum coordinates:
    initwindow(450, 300);
    maxx = getmaxx();
    maxy = getmaxy();

    // Draw a white circle with red inside and a radius of 50 pixels:
    setfillstyle(SOLID_FILL, RED);
    setcolor(WHITE);
    fillellipse(maxx/2, maxy/2, 50, 50);

    // Print a message and wait for a red pixel to be double clicked:
    settextstyle(DEFAULT_FONT, HORIZ_DIR, 2);
    outtextxy(20, 20, "Left click in to end.");
    setcolor(BLUE);
    divisor = 2;
    while (!ismouseclick(WM_LBUTTONDOWN))
    {
        triangle(maxx/divisor, maxy/divisor);
    }
```
```cpp
delay(500);
divisor++;
}

getmouseclick(WM_LBUTTONDOWN, x, y);
cout << "The mouse was clicked at: ";
cout << "x=" << x;
cout << " y=" << y << endl;

// Switch back to text mode:
closegraph( );
```

---

**getpalette**

**Syntax**

```cpp
#include <graphics.h>
void getpalette(struct palettetype *palette);
```

**Description**

getpalette fills the palettetype structure pointed to by palette with information about the current palette's size and colors.

The MAXCOLORS constant and the palettetype structure used by getpalette are defined in graphics.h as follows:

```cpp
#define MAXCOLORS 15
```

```cpp
struct palettetype {
    unsigned char size;
    signed char colors[MAXCOLORS + 1];
};
```

size gives the number of colors in the palette for the current graphics driver in the current mode.

colors is an array of size bytes containing the actual raw color numbers for each entry in the palette.

getpalette cannot be used with the IBM-8514 driver.

**Return Value**

None.

**Windows Notes**

The winbgim version of getpalette returns a palettetype object of 16 colors. Each color is either one of the 16 BGI color numbers (0 through 15) or it is -1 to indicate that the location of the palette has been set to an RGB color.

See also

- `getbkcolor`
- `getcolor`
- `getdefaultpalette`
- `getmaxcolor`
- `setallpalette`
- `setpalette`

**Example**

```cpp
/* getpalette example */

#include <graphics.h>
```
```c
#include <stdlib.h>
#include <stdio.h>
#include <conio.h>

int main()
{
    /* request autodetection */
    int gdriver = DETECT, gmode, errorcode;
    struct palettetype pal;
    char psize[80], pval[20];
    int i, ht;
    int y = 10;

    /* initialize graphics and local variables */
    initgraph(&gdriver, &gmode, "");

    /* read result of initialization */
    errorcode = graphresult();
    if (errorcode != grOk) {  /* an error occurred */
        printf("Graphics error: %s\n", grapherrormsg(errorcode));
        printf("Press any key to halt:");
        getch();
        exit(1);               /* terminate with an error code */
    }

    /* grab a copy of the palette */
    getpalette(&pal);

    /* convert palette info into strings */
    sprintf(psize, "The palette has %d modifiable entries.",
            pal.size);

    /* display the information */
    outtextxy(0, y, psize);
    if (pal.size != 0) {
        ht = textheight("W");
        y += 2*ht;
        outtextxy(0, y, "Here are the current values:");
        y += 2*ht;
        for (i=0; i<pal.size; i++, y+=ht) {
            sprintf(pval, "palette[%02d]: 0x%02X", i, pal.colors[i]);
            outtextxy(0, y, pval);
        }
    }

    /* clean up */
    getch();
    closegraph();
    return 0;
}
```

---

getpalettesize

**Syntax**
#include <graphics.h>
int getpalettesize(void);

Description
getpalettesize is used to determine how many palette entries can be set for the current graphics mode. For example, the EGA in color mode returns 16.

Return Value
getpalettesize returns the number of palette entries in the current palette.

Windows Notes
The winbgim version of getpalettesize returns 16 for the palette color. However, in addition to the palette colors, the programmer may also use RGB colors.

See also
setallpalette
setpalette

Example
/* getpalettesize example */

#include <graphics.h>
#include <stdlib.h>
#include <stdio.h>
#include <conio.h>

int main()
{
    /* request autodetection */
    int gdriver = DETECT, gmode, errorcode;
    int midx, midy;
    char psize[80];

    /* initialize graphics and local variables */
    initgraph(&gdriver, &gmode, "");

    /* read result of initialization */
    errorcode = graphresult();
    if (errorcode != grOk) { /* an error occurred */
        printf("Graphics error: %s\n", grapherrormsg(errorcode));
        printf("Press any key to halt:");
        getch();
        exit(1); /* terminate with an error code */
    }

    midx = getmaxx() / 2;
    midy = getmaxy() / 2;

    /* convert palette size info into string */
    sprintf(psize, "The palette has %d modifiable entries.", getpalettesize());

    /* display the information */
    settextjustify(CENTER_TEXT, CENTER_TEXT);
    outtextxy(midx, midy, psize);

    /* clean up */
    getch();
    closegraph();

    return 0;
}
getpixel

Syntax

```c
#include <graphics.h>
unsigned getpixel(int x, int y);
```

Description

getpixel gets the color of the pixel located at (x,y).

Return Value

getpixel returns the color of the given pixel.

Windows Notes

In the winbgim version, the user might set a pixel color to an RGB color. Therefore, the return value from getpixel might be an ordinary BGI color (integer from 0 to 15) or an RGB color.

See also

getimage
putpixel

Example

```c
/* getpixel example */

#include <graphics.h>
#include <stdlib.h>
#include <stdio.h>
#include <conio.h>
#include <dos.h>

#define PIXEL_COUNT 1000
#define DELAY_TIME 100 /* in milliseconds */

int main(void)
{
    /* request autodetection */
    int gdriver = DETECT, gmode, errorcode;
    int i, x, y, color, maxx, maxy, maxcolor, seed;

    /* initialize graphics and local variables */
    initgraph(&gdriver, &gmode, "");

    /* read result of initialization */
    errorcode = graphresult();

    if (errorcode != grOk) { /* an error occurred */
        printf("Graphics error: %s\n", grapherrormsg(errorcode));
        printf("Press any key to halt:");
        getch();
        exit(1); /* terminate with an error code */
    }

    maxx = getmaxx() + 1;
    maxy = getmaxy() + 1;
    maxcolor = getmaxcolor() + 1;
    while (!kbhit()) {
        seed = random(32767); /* seed the random number generator */
        srand(seed);
```
for (i=0; i<PIXEL_COUNT; i++) {
    x = random(maxx);
    y = random(maxy);
    color = random(maxcolor);
    putpixel(x, y, color);
}
delay(DELAY_TIME);
srand(seed);
for (i=0; i<PIXEL_COUNT; i++) {
    x = random(maxx);
    y = random(maxy);
    color = random(maxcolor);
    if (color == getpixel(x, y))
        putpixel(x, y, 0);
}

/* clean up */
getch();
closegraph();
return 0;
}

getviewsettings

Syntax
#include <graphics.h>
void getviewsettings(struct viewporttype *viewport);

Description
getviewsettings fills the viewporttype structure pointed to by viewport with
information about the current viewport.
The viewporttype structure used by getviewport is defined in graphics.h as follows:
struct viewporttype {
    int left, top, right, bottom;
    int clip;
};

Return Value
None.

See also
clearviewport
getch
gety
setviewport

Example
/* getviewsettings example */

#include <graphics.h>
#include <stdlib.h>
#include <stdio.h>
#include <conio.h>

char *clip[] = { "OFF", "ON" };
int main(void)
{
    /* request autodetection */
    int gdriver = DETECT, gmode, errorcode;
    struct viewporttype viewinfo;
    int midx, midy, ht;
    char topstr[80], botstr[80], clipstr[80];
    /* initialize graphics and local variables */
    initgraph(&gdriver, &gmode, "");
    /* read result of initialization */
    errorcode = graphresult();
    if (errorcode != grOk) {  /* an error occurred */
        printf("Graphics error: %s\n", grapherrormsg(errorcode));
        printf("Press any key to halt:");
        getch();
        exit(1);               /* terminate with an error code */
    }
    midx = getmaxx() / 2;
    midy = getmaxy() / 2;
    /* get information about current viewport */
    getviewsettings(&viewinfo);
    /* convert text information into strings */
    sprintf(topstr, "(%d, %d) is the upper left viewport
corner.", viewinfo.left, viewinfo.top);
    sprintf(botstr, "(%d, %d) is the lower right viewport
corner.", viewinfo.right, viewinfo.bottom);
    sprintf(clipstr, "Clipping is turned %s.", clip[viewinfo.clip]);
    /* display the information */
    settextjustify(CENTER_TEXT, CENTER_TEXT);
    ht = textheight("W");
    outtextxy(midx, midy, topstr);
    outtextxy(midx, midy+2*ht, botstr);
    outtextxy(midx, midy+4*ht, clipstr);
    /* clean up */
    getch();
    closegraph();
    return 0;
}

getvisualpage

Syntax
#include "winbgim.h"
int getvisualpage(void);

Description
WIN
The `getvisualpage` function is available in the `winbgim` implementation of BGI graphics. `getvisualpage` gets the page number of the currently visable page (which is visible on the screen).

The visual graphics page might not be the one where drawing currently takes place. The original `winbgi` was designed to support up to 16 pages, but I have only used pages 1 and 2 myself. **NOTE:** Using page number 0 might mess up the colors. I use pages 1-2 for double buffering.

**Return Value**
The page number of the currently visible page.

**See also**
- `getactivepage`
- `setvisualpage`

### getx

**Syntax**
```
#include <graphics.h>
int getx(void);
```

**Description**
`getx` finds the current graphics position's x-coordinate. The value is viewport-relative.

**Return Value**
`getx` returns the x-coordinate of the current position.

**See also**
- `getmaxx`
- `getviewsettings`
- `gety`
- `moveto`

**Example**
```
/* getx example */

#include <graphics.h>
#include <stdlib.h>
#include <stdio.h>
#include <conio.h>

int main(void)
{
    /* request autodetection */
    int gdriver = DETECT, gmode, errorcode;
    char msg[80];

    /* initialize graphics and local variables */
    initgraph(&gdriver, &gmode, "");

    /* read result of initialization */
    errorcode = graphresult();
    if (errorcode != grOk) { /* an error occurred */
        printf("Graphics error: %s\n", grapherrormsg(errorcode));
        printf("Press any key to halt: ");
        getch();
    }

    /* draw something */
    /* do something */

    /* output current graphics position's x-coordinate */
    printf("Current x-coordinate: %d\n", getx());

    /* do more */

    /* free all allocated resources */
    closegraph();
}
```
getch();
exit(1);              /* terminate with an error code */
}
/* move to the screen center point */
moveto(getmaxx() / 2, getmaxy() / 2);
/* create a message string */
sprintf(msg, "<-(%d, %d) is the here.", getx(), gety());
/* display the message */
outtext(msg);
/* clean up */
getch();
closegraph();
return 0;
}

gety

Syntax
#include <graphics.h>
int gety(void);

Description
gety returns the current graphics position's y-coordinate. The value is viewport-relative.

Return Value
gety returns the y-coordinate of the current position.

See also
getmaxy
getviewsettings
gtx
moveto

Example
/* gety example */

#include <graphics.h>
#include <stdlib.h>
#include <stdio.h>
#include <conio.h>

int main(void)
{
    /* request autodetection */
    int gdriver = DETECT, gmode, errorcode;
    char msg[80];

    /* initialize graphics and local variables */
    inittgraph(&gdriver, &gmode, "");

    /* read result of initialization */
    errorcode = graphresult();
    if (errorcode != grOk) { /* an error occurred */
printf("Graphics error: %s\n", grapherrormsg(errorcode));

printf("Press any key to halt:");
getch();
exit(1); /* terminate with an error code */
}

/* move to the screen center point */
moveto(getmaxx() / 2, getmaxy() / 2);

/* create a message string */
sprintf(msg, "<-(%d, %d) is the here.", getx(), gety());

/* display the message */
outtext(msg);

/* clean up */
getch();
closegraph();
return 0;
}

graphdefaults

Syntax
#include <graphics.h>
void graphdefaults(void);

Description
graphdefaults resets all graphics settings to their defaults:

- sets the viewport to the entire screen.
- moves the current position to (0,0).
- sets the default palette colors, background color, and drawing color.
- sets the default fill style and pattern.
- sets the default text font and justification.

Return Value
None.

See also
initgraph
setgraphmode

Example
/* graphdefaults example */

#include <graphics.h>
#include <stdlib.h>
#include <stdio.h>
#include <conio.h>

int main(void)
{
    /* request autodetection */
    int gdriver = DETECT, gmode, errorcode;
int maxx, maxy;
/* initialize graphics and local variables */
initgraph(&gdriver, &gmode, "");

/* read result of initialization */
errorcode = graphresult();
if (errorcode != grOk) { /* an error occurred */
    printf("Graphics error: %s\n", grapherrormsg(errorcode));
    printf("Press any key to halt:");
    getch();
    exit(1);              /* terminate with an error code */
}
maxx = getmaxx();
maxy = getmaxy();

/* output line with nondefault settings */
setlinestyle(DOTTED_LINE, 0, 3);  
line(0, 0, maxx, maxy);
outtextxy(maxx/2, maxy/3, "Before default values are restored.");
getch();

/* restore default values for everything */
graphdefaults();

/* clear the screen */
cleardevice();

/* output line with default settings */
line(0, 0, maxx, maxy);
outtextxy(maxx/2, maxy/3, "After restoring default values.");

/* clean up */
getch();
closegraph();
return 0;
}

grapherrormsg

Syntax
#include <graphics.h>
char * grapherrormsg(int errorcode);

Description
grapherrormsg returns a pointer to the error message string associated with errorcode, the value returned by graphresult. Refer to the entry for errno in the Library Reference, Chapter 4, for a list of error messages and mnemonics.

Return Value
grapherrormsg returns a pointer to an error message string.
See also

graphresult

Example

/* grapherrormsg example */

#include <graphics.h>
#include <stdlib.h>
#include <stdio.h>
#include <conio.h>

#define NONSENSE -50

int main(void)
{
    /* force an error to occur */
    int gdriver = NONSENSE, gmode, errorcode;

    /* initialize graphics mode */
    initgraph(&gdriver, &gmode, "");

    /* read result of initialization */
    errorcode = graphresult();

    /* if an error occurred, then output descriptive error message*/
    if (errorcode != grOk) {
        printf("Graphics error: %s\n", grapherrormsg(errorcode));
        printf("Press any key to halt: ");
        getch();
        exit(1); /* terminate with an error code */
    }

    /* draw a line */
    line(0, 0, getmaxx(), getmaxy());

    /* clean up */
    getch();
    closegraph();
    return 0;
}

graphresult

Syntax

#include <graphics.h>
int graphresult(void);

Description
graphresult returns the error code for the last graphics operation that reported an error
and resets the error level to grOk.
The following table lists the error codes returned by graphresult. The enumerated type
graph_errors defines the errors in this table. graph_errors is declared in graphics.h.

code  constant  Corresponding error message string
<table>
<thead>
<tr>
<th>Code</th>
<th>Error Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>grOk</td>
</tr>
<tr>
<td>-1</td>
<td>grNoInitGraph</td>
</tr>
<tr>
<td>-2</td>
<td>grNotDetected</td>
</tr>
<tr>
<td>-3</td>
<td>grFileNotFoundException</td>
</tr>
<tr>
<td>-4</td>
<td>grInvalidDriver</td>
</tr>
<tr>
<td>-5</td>
<td>grNoLoadMem</td>
</tr>
<tr>
<td>-6</td>
<td>grNoScanMem</td>
</tr>
<tr>
<td>-7</td>
<td>grNoFloodMem</td>
</tr>
<tr>
<td>-8</td>
<td>grFontNotFound</td>
</tr>
<tr>
<td>-9</td>
<td>grNoFontMem</td>
</tr>
<tr>
<td>-10</td>
<td>grInvalidMode</td>
</tr>
<tr>
<td>-11</td>
<td>grError</td>
</tr>
<tr>
<td>-12</td>
<td>grIOerror</td>
</tr>
<tr>
<td>-13</td>
<td>grInvalidFont</td>
</tr>
<tr>
<td>-14</td>
<td>grInvalidFontNum</td>
</tr>
<tr>
<td>-15</td>
<td>grInvalidDeviceNum</td>
</tr>
<tr>
<td>-18</td>
<td>grInvalidVersion</td>
</tr>
</tbody>
</table>

Note: The variable maintained by graphresult is reset to 0 after graphresult has been called. Therefore, you should store the value of graphresult into a temporary variable and then test it.

**Return Value**

graphresult returns the current graphics error number, an integer in the range -15 to 0; grapherrormsg returns a pointer to a string associated with the value returned by graphresult.

**See also**

detectgraph
drawpoly
fillpoly
floodfill
grapherrormsg
initgraph
pieslice
registerbgidriver
registerbgifont
setallpalette
setcolor
setfillstyle
setgraphmode
setlinestyle
setpalette
settextjustify
settextstyle
setusercharsize
setviewport
setvisualpage

**Example**

/* graphresult example */
#include <graphics.h>
#include <stdlib.h>
#include <stdio.h>
#include <conio.h>

int main(void)
{
    /* request autodetection */
    int gd = DETECT, gm, er;

    /* initialize graphics and local variables */
    initgraph(&gd, &gm, "");

    /* read result of initialization */
    er = graphresult();

    if (er != grOk) { /* an error occurred */
        printf("Graphics error: %s\n", grapherrormsg(er));
        printf("Press any key to halt: ");
        getch();
        exit(1); /* terminate with an error code */
    }

    /* draw a line */
    line(0, 0, getmaxx(), getmaxy());

    /* clean up */
    getch();
    closegraph();
    return 0;
}

imagesize

Syntax
#include <graphics.h>
unsigned imagesize(int left, int top, int right, int bottom);

Description
imagesize determines the size of the memory area required to store a bit image. If the
size required for the selected image is greater than or equal to 64K - 1 bytes, imagesize
returns 0xFFFF (-1).

Return Value
imagesize returns the size of the required memory area in bytes.

See also
   getimage
   putimage

Example
/* imagesize example */

#include <graphics.h>
#include <stdlib.h>
#include <stdio.h>
#include <conio.h>

#define ARROW_SIZE 10

void draw_arrow(int x, int y);

int main(void)
{
    /* request autodetection */
    int gdriver = DETECT, gmode, errorcode;
    void *arrow;
    int x, y, maxx;
    unsigned int size;

    /* initialize graphics and local variables */
    initgraph(&gdriver, &gmode, "");

    /* read result of initialization */
    errorcode = graphresult();
    if (errorcode != grOk) {  /* an error occurred */
        printf("Graphics error: %s\n", grapherrormsg(errorcode));
        printf("Press any key to halt:");
        getch();
        exit(1);               /* terminate with an error code */
    }

    maxx = getmaxx();
    x = 0;
    y = getmaxy() / 2;

    /* draw the image to be grabbed */
    draw_arrow(x, y);

    /* calculate the size of the image */
    size = imagesize(x, y-ARROW_SIZE, x+(4*ARROW_SIZE), y+ARROW_SIZE);

    /* allocate memory to hold the image */
    arrow = malloc(size);

    /* grab the image */
    getimage(x, y-ARROW_SIZE, x+(4*ARROW_SIZE), y+ARROW_SIZE, arrow);

    /* repeat until a key is pressed */
    while (!kbhit()) {
        /* erase old image */
        putimage(x, y-ARROW_SIZE, arrow, XOR_PUT);
        x += ARROW_SIZE;
        if (x >= maxx)
            x = 0;

        /* plot new image */
        putimage(x, y-ARROW_SIZE, arrow, XOR_PUT);
    }

    /* clean up */
    free(arrow);
    closegraph();
    return 0;
}
void draw_arrow(int x, int y)
{
    /* draw an arrow on the screen */
    moveto(x, y);
    linerel(4*ARROW_SIZE, 0);
    linerel(-2*ARROW_SIZE, -1*ARROW_SIZE);
    linerel(0, 2*ARROW_SIZE);
    linerel(2*ARROW_SIZE, -1*ARROW_SIZE);
}

initgraph

Syntax
#include <graphics.h>
void initgraph(int *graphdriver, int *graphmode, char *pathtodriver);

Description
initgraph initializes the graphics system by loading a graphics driver from disk (or validating a registered driver), and putting the system into graphics mode.
To start the graphics system, first call the initgraph function. initgraph loads the graphics driver and puts the system into graphics mode. You can tell initgraph to use a particular graphics driver and mode, or to autodetect the attached video adapter at runtime and pick the corresponding driver.
If you tell initgraph to autodetect, it calls detectgraph to select a graphics driver and mode. initgraph also resets all graphics settings to their defaults (current position, palette, color, viewport, and so on) and resets graphicsresult to 0.
Normally, initgraph loads a graphics driver by allocating memory for the driver (through _graphgetmem), then loading the appropriate .BGI file from disk. As an alternative to this dynamic loading scheme, you can link a graphics driver file (or several of them) directly into your executable program file.
pathtodriver specifies the directory path where initgraph looks for graphics drivers. initgraph first looks in the path specified in pathtodriver, then (if they are not there) in the current directory. Accordingly, if pathtodriver is null, the driver files (*.BGI) must be in the current directory. This is also the path settextstyle searches for the stroked character font files (*.CHR).
*graphdriver is an integer that specifies the graphics driver to be used. You can give it a value using a constant of the graphics_drivers enumeration type, which is defined in graphics.h and listed below.

<table>
<thead>
<tr>
<th>graphics_drivers constant</th>
<th>Numeric value</th>
</tr>
</thead>
<tbody>
<tr>
<td>DETECT</td>
<td>0 (requests autodetect)</td>
</tr>
<tr>
<td>CGA</td>
<td>1</td>
</tr>
<tr>
<td>MCGA</td>
<td>2</td>
</tr>
<tr>
<td>EGA</td>
<td>3</td>
</tr>
<tr>
<td>EGA64</td>
<td>4</td>
</tr>
<tr>
<td>EGAMONO</td>
<td>5</td>
</tr>
<tr>
<td>IBM8514</td>
<td>6</td>
</tr>
</tbody>
</table>
*graphmode is an integer that specifies the initial graphics mode (unless *graphdriver equals DETECT; in which case, *graphmode is set by initgraph to the highest resolution available for the detected driver). You can give *graphmode a value using a constant of the graphics_modes enumeration type, which is defined in graphics.h and listed below.

Graphdriver and graphmode must be set to valid values from the following tables, or you will get unpredictable results. The exception is graphdriver = DETECT.

Palette listings C0, C1, C2, and C3 refer to the four predefined four-color palettes available on CGA (and compatible) systems. You can select the background color (entry #0) in each of these palettes, but the other colors are fixed.

<table>
<thead>
<tr>
<th>Palette Number</th>
<th>Three Colors</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>LIGHTGREEN</td>
</tr>
<tr>
<td>1</td>
<td>LIGHTCYAN</td>
</tr>
<tr>
<td>2</td>
<td>GREEN</td>
</tr>
<tr>
<td>3</td>
<td>CYAN</td>
</tr>
</tbody>
</table>

After a call to initgraph, *graphdriver is set to the current graphics driver, and *graphmode is set to the current graphics mode.

<table>
<thead>
<tr>
<th>Graphics</th>
<th>Columns</th>
</tr>
</thead>
<tbody>
<tr>
<td>Driver</td>
<td></td>
</tr>
<tr>
<td>CGA</td>
<td></td>
</tr>
<tr>
<td>CGAC0</td>
<td>0</td>
</tr>
<tr>
<td>CGAC1</td>
<td>1</td>
</tr>
<tr>
<td>CGAC2</td>
<td>2</td>
</tr>
<tr>
<td>CGAC3</td>
<td>3</td>
</tr>
<tr>
<td>CGAHI</td>
<td>4</td>
</tr>
<tr>
<td>MCGA</td>
<td></td>
</tr>
<tr>
<td>MCGAC0</td>
<td>0</td>
</tr>
<tr>
<td>MCGAC1</td>
<td>1</td>
</tr>
<tr>
<td>MCGAC2</td>
<td>2</td>
</tr>
<tr>
<td>MCGAC3</td>
<td>3</td>
</tr>
<tr>
<td>MCGAMED</td>
<td>4</td>
</tr>
<tr>
<td>MCGAHI</td>
<td>5</td>
</tr>
<tr>
<td>EGA</td>
<td></td>
</tr>
<tr>
<td>EGA64LO</td>
<td>0</td>
</tr>
<tr>
<td>EGAHI</td>
<td>1</td>
</tr>
<tr>
<td>EGA64</td>
<td></td>
</tr>
</tbody>
</table>
EGA64HI  1  640 x 350  4 color  1
EGA-MONO EGAMONOH  3  640 x 350  2 color  1  w/64K
EGAMONOH  3  640 x 350  2 color  2  w/256K
HERC HERCMONOH  0  720 x 348  2 color  2
ATT400 ATT400C0  0  320 x 200  C0  1
ATT400C1  1  320 x 200  C1  1
ATT400C2  2  320 x 200  C2  1
ATT400C3  3  320 x 200  C3  1
ATT400MED  4  640 x 200  2 color  1
ATT400HI  5  640 x 400  2 color  1
VGA VGALO  0  640 x 200  16 color  2
VGAMED  1  640 x 350  16 color  2
VGAHI  2  640 x 480  16 color  1
PC3270 PC3270HI  0  720 x 350  2 color  1
IBM8514 IBM8514HI  0  640 x 480  256 color ?
IBM8514LO  0  1024 x 768  256 color ?

Return Value
initgraph always sets the internal error code; on success, it sets the code to 0. If an error occurred, *graphdriver is set to -2, -3, -4, or -5, and graphresult returns the same value as listed below:

<table>
<thead>
<tr>
<th>Constant Name</th>
<th>Number</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>grNotDetected</td>
<td>-2</td>
<td>Cannot detect a graphics card</td>
</tr>
<tr>
<td>grFileNotFound</td>
<td>-3</td>
<td>Cannot find driver file</td>
</tr>
<tr>
<td>grInvalidDriver</td>
<td>-4</td>
<td>Invalid driver</td>
</tr>
<tr>
<td>grNoLoadMem</td>
<td>-5</td>
<td>Insufficient memory to load driver</td>
</tr>
</tbody>
</table>

Windows Notes
The winbgim version of initgraph uses its parameters only to determine the size of the window that will be created. For example, initgraph(CGA, CGAC3) will create a 320 x 200 window. As an alternative, the user may call initwindow(width, height) instead of initgraph.

See also
closegraph
detectgraph
getdefaultpalette
getdrivername
getgraphmode
getmoderange
Example

/* initgraph example */

#include <graphics.h>
#include <stdlib.h>
#include <stdio.h>
#include <conio.h>

int main(void)
{
    /* request autodetection */
    int gdriver = DETECT, gmode, errorcode;

    /* initialize graphics mode */
    initgraph(&gdriver, &gmode, "");

    /* read result of initialization */
    errorcode = graphresult();

    if (errorcode != grOk)    /* an error occurred */
    {
        printf("Graphics error: %s\n", grapherrormsg(errorcode));
        printf("Press any key to halt:");

        getch();
        exit(1);               /* return with error code */
    }

    /* draw a line */
    line(0, 0, getmaxx(), getmaxy());

    /* clean up */
    getch();
    closegraph();
    return 0;
}

initwindow

Syntax

#include "winbgim.h"

void initwindow(int width, int height);

Description

WIN
The initwindow function is available in the winbgim implementation of BGI graphics. You do not need to include conio.h; just include winbgim.h. The function initializes the graphics system by opening a graphics window of the specified size.

**Return Value**
None.

**See also**
- closegraph
- initgraph

**Example**

```c
/* initwindow example */
#include "winbgim.h"

int main(void)
{
    /* initialize graphics window at 400 x 300 */
    initwindow(400, 300);

    /* draw a line */
    line(0, 0, getmaxx(), getmaxy());

    /* clean up */
    getch();
    closegraph();
    return 0;
}
```

**installuserdriver**

**Syntax**

```c
#include <graphics.h>
int installuserdriver(char *name, int huge (*detect)(void));
```

**Description**

installuserdriver lets you add a vendor-added device driver to the BGI internal table. The name parameter is the name of the new device-driver file (.BGI), and the detect parameter is a pointer to an optional autodetect function that can accompany the new driver. This autodetect function takes no parameters and returns an integer value.

There are two ways to use this vendor-supplied driver. Suppose you have a new video card called the Spiffy Graphics Array (SGA) and that the SGA manufacturer provided you with a BGI device driver (SGA.BGI). The easiest way to use this driver is to install it by calling installuserdriver and then passing the return value (the assigned driver number) directly to initgraph.

The other, more general way to use this driver is to link in an autodetect function that will be called by initgraph as part of its hardware-detection logic (presumably, the manufacturer of the SGA gave you this autodetect function). When you install the driver (by calling installuserdriver), you pass the address of this function, along with the device driver's file name.

After you install the device-driver file name and the SGA autodetect function, call initgraph and let it go through its normal autodetection process. Before initgraph calls its built-in autodetection function (detectgraph), it first calls the SGA autodetect
function. If the SGA autodetect function doesn't find the SGA hardware, it returns a value of -11 (grError), and initgraph proceeds with its normal hardware detection logic (which can include calling any other vendor-supplied autodetection functions in the order in which they were "installed"). If, however, the autodetect function determines that an SGA is present, it returns a nonnegative mode number; then initgraph locates and loads SGA.BGI, puts the hardware into the default graphics mode recommended by the autodetect function, and finally returns control to your program. You can install up to ten drivers at one time.

**Return Value**
The value returned by installuserdriver is the driver number parameter you would pass to initgraph in order to select the newly installed driver manually.

**Windows Notes**
installuserdriver is not available in the winbgim implementation.

See also
initgraph
registerbgidriver

**Example**

```c
/* installuserdriver example */

#include
#include
#include
#include

/* function prototypes */
int huge detectEGA(void);
void checkerrors(void);
int main(void)
{
    int gdriver, gmode;

    /* install a user written device driver */
gdriver = installuserdriver("EGA", detectEGA);

    /* must force use of detection routine */
gdriver = DETECT;

    /* check for any installation errors */
    checkerrors();

    /* initialize graphics and local variables */
    initgraph(&gdriver, &gmode, "");

    /* check for any initialization errors */
    checkerrors();

    /* draw a line */
    line(0, 0, getmaxx(), getmaxy());

    /* clean up */
    getch();
    closegraph();
    return 0;
}

/* detects EGA or VGA cards */
int huge detectEGA(void)
```
installuserfont

Syntax
#include <graphics.h>
int installuserfont(char *name);

Description
name is a file name in the current directory (pathname is not supported) of a font file containing a stroked font. Up to twenty fonts can be installed at one time.

Return Value
installuserfont returns a font ID number that can then be passed to settextstyle to select the corresponding font. If the internal font table is full, a value of -11 (grError) is returned.

Windows Notes
installuserfont is not available in the winbgim implementation.

See also
installuserfont
settextstyle

Example
/* installuserfont example */
#include <graphics.h>
#include <stdlib.h>
#include <stdio.h>
#include <conio.h>

/* function prototype */
void checkerrors(void);
int main(void)
{
    /* request autodetection */
    int gdriver = DETECT, gmode;
    int userfont;
    int midx, midy;

    /* initialize graphics and local variables */
    initgraph(&gdriver, &gmode, "");

    midx = getmaxx() / 2;
    midy = getmaxy() / 2;

    /* check for any initialization errors */
    checkerrors();

    /* install a user-defined font file */
    userfont = installuserfont("USER.CHR");

    /* check for any installation errors */
    checkerrors();

    /* select the user font */
    settextstyle(userfont, HORIZ_DIR, 4);

    /* output some text */
    outtextxy(midx, midy, "Testing!");

    /* clean up */
    getch();
    closegraph();
    return 0;
}

/* check for and report any graphics errors */
void checkerrors(void)
{
    int errorcode;

    /* read result of last graphics operation */
    errorcode = graphresult();

    if (errorcode != grOk) {
        printf("Graphics error: %s\n", grapherrormsg(errorcode));
        printf("Press any key to halt:");
        getch();
        exit(1);
    }
}

ismouseclick

Syntax

#include "winbgim.h"
bool ismouseclick(int kind);

**Description**

The ismouseclick function is available in the `winbgim` implementation of BGI graphics. This function returns true if there is an unprocessed mouse event of the specified kind. The argument to ismouseclick is one of these constants from the `winbgim.h` file:

- `WM_MOUSEMOVE` if you want to detect a mouse movement
- `WM_LBUTTONDOWN`...detect when the left mouse button is clicked down
- `WM_LBUTTONUP`...detect when the left mouse button is released up
- `WM_MBUTTONDOWN`...detect when the middle mouse button is clicked down
- `WM_MBUTTONUP`...detect when the middle mouse button is released up
- `WM_RBUTTONDOWN`...detect when the right mouse button is clicked down
- `WM_RBUTTONUP`...detect when the right mouse button is released up

The middle mouse button handlers aren't working on my machine. I haven't yet tracked down the reason--it could be a broken mouse or it could be a bug in my programming.

A mouse event can be processed by calling `getmouseclick` (which gets the coordinates of the event), or by calling `clearmouseclick` (which processes the event without providing its coordinates).

**Return Value**

True if there is an unprocessed mouse event of the specified kind; otherwise false.

**See also**

- `getmouseclick`
- `clearmouseclick`

**Example**

/* mouse example */
#include "winbgim.h"

void main(void)
{
    int maxx, maxy; // Maximum x and y pixel coordinates
    int x, y; // Coordinates of the mouse click
    int divisor; // Divisor for the length of a triangle side

    // Put the machine into graphics mode and get the maximum coordinates:
    initwindow(450, 300);
    maxx = getmaxx();
    maxy = getmaxy();

    // Draw a white circle with red inside and a radius of 50 pixels:
    setfillstyle(SOLID_FILL, RED);
setcolor(WHITE);
fillellipse(maxx/2, maxy/2, 50, 50);

// Print a message and wait for a red pixel to be double clicked:
settextstyle(DEFAULT_FONT, HORIZ_DIR, 2);
outtextxy(20, 20, "Left click in to end.");
setcolor(BLUE);
divisor = 2;
while (!ismouseclick(WM_LBUTTONDOWN))
{
    triangle(maxx/divisor, maxy/divisor);
delay(500);
divisor++;
}

getmouseclick(WM_LBUTTONDOWN, x, y);
cout << "The mouse was clicked at: ";
cout << "x= " << x;
cout << " y= " << y << endl;

// Switch back to text mode:
closegraph( );

---

**kbhit**

**Syntax**

```
#include "winbgim.h"
int kbhit(void);
```

**Description**

The `kbhit` function is available in the [winbgim](#) implementation of BGI graphics. You do not need to include conio.h; just include winbgim.h. The function returns true (non-zero) if there is a character in the input buffer ready to read. Otherwise it returns false.

In order to work, the user must click in the graphics window (i.e., the Windows focus must be in the graphics window).

**Return Value**

True (non-zero) if there is a character in the input buffer, otherwise false.

**See also**

`kbhit`

**Example**

```c
#include "winbgim.h"
#include <stdio.h>    // Provides sprintf

void outintxy(int x, int y, int value);

int main( )
{
    int i;

    // Initialize the graphics window.
    init_window(400, 300);

    // Convert some numbers to strings and draw them in graphics window:
```
Click in this graphics window,
and then press a key to stop.
Here are some numbers:
for (i = 0; !kbhit( ); i++)
    outintxy(20 + (i/10)*40 , (i % 10)*+10, i);
delay(4000);
}
closegraph();

void outintxy(int x, int y, int value)
{
    char digit_string[20];
    sprintf(digit_string, "%d", value);
    outtextxy(x, y, digit_string);
}

line

Syntax
#include <graphics.h>
void line(int x1, int y1, int x2, int y2);

Description
line draws a line in the current color, using the current line style and thickness between the two points specified, (x1,y1) and (x2,y2), without updating the current position (CP).

Return Value
None.

See also
getlinesettings
linerel
lineto
setcolor
setlinestyle
setwritemode

Example
/* line example */
#include <graphics.h>
#include <stdlib.h>
#include <stdio.h>
#include <conio.h>

int main(void)
{
    /* request autodetection */
    int gd = DETECT, gm, errorcode;
    int x, y;
    /* initialize graphics and local variables */
    initgraph(&gd, &gm, "");

/* read result of initialization */
errorcode = graphresult();

if (errorcode != grOk) { /* an error occurred */
    printf("Graphics error: %s\n", grapherrmsg(errorcode));

    printf("Press any key to halt: ");
    getch();
    exit(1);
}

setcolor(getmaxcolor());
xmax = getmaxx();
ymax = getmaxy();

/* draw a diagonal line */
line(0, 0, xmax, ymax);

/* clean up */
getch();
closegraph();
return 0;

linerel

Syntax
#include <graphics.h>
void linerel(int dx, int dy);

Description
linerel draws a line from the CP (current position) to a point that is a relative distance (dx,dy) from the CP. The CP is advanced by (dx,dy).

Return Value
None.

See also
getlinestings
line
lineto
setcolor
setlinestyle
setwritemode

Example
/* linerel example */

#include <graphics.h>
#include <stdlib.h>
#include <stdio.h>
#include <conio.h>

int main(void)
{
    /* request autodetection */
    int gdriver = DETECT, gmode, errorcode;
    char msg[80];
/* initialize graphics and local variables */
initgraph(&gdriver, &gmode, "");

/* read result of initialization */
errorcode = graphresult();
if (errorcode != grOk) {
    printf("Graphics error: %s\n", grapherrormsg(errorcode));
    printf("Press any key to halt: ");
    getch();
    exit(1);
}

/* move the CP to location (20,30) */
moveto(20,30);

/* create and output a message at (20,30) */
sprintf(msg, " (%d, %d)", getx(), gety());
outtextxy(20,30, msg);

/* draw line to a point a relative distance away from current CP */
linerel(100, 100);

/* create and output a message at CP */
sprintf(msg, " (%d, %d)", getx(), gety());
outtext(msg);

/* clean up */
getch();
closegraph();
return 0;
}

lineto

**Syntax**
```c
#include <graphics.h>
void lineto(int x, int y);
```

**Description**
lineto draws a line from the CP (current position) to (x,y), then moves the CP to (x,y).

**Return Value**
None.

**See also**
getlinesettings
line
linerel
setcolor
setlinestyle
setwritemode

**Example**
```c
/* lineto example */

#include <graphics.h>
```
#include <stdlib.h>
#include <stdio.h>
#include <conio.h>

int main(void)
{
    /* request autodetection */
    int gdriver = DETECT, gmode, errorcode;
    char msg[80];

    /* initialize graphics and local variables */
    initgraph(&gdriver, &gmode, "");

    /* read result of initialization */
    errorcode = graphresult();
    if (errorcode != grOk) {
        printf("Graphics error: %s\n", grapherrormsg(errorcode));
        printf("Press any key to halt: ");
        getch();
        exit(1);
    }

    /* move the CP to location (20,30) */
    moveto(20, 30);

    /* create and output a message at (20,30) */
    sprintf(msg, " (%d, %d)", getx(), gety());
    outtextxy(20, 30, msg);

    /* draw a line to (100,100) */
    lineto(100, 100);

    /* create and output a message at CP */
    sprintf(msg, " (%d, %d)", getx(), gety());
    outtext(msg);

    /* clean up */
    getch();
    closegraph();
    return 0;
}

mousex

Syntax
#include "winbgim.h"
int mousex(void);

Description
The mousex function is available in the winbgim implementation of BGI graphics. It returns the most recent x coordinate of the mouse within the graphics window. X-coordinates start with 0 at the left edge of the window and increase to getmaxx() at the right edge of the window.

Return Value
Most recent x coordinate of the mouse within the graphics window

See also
Example

/* mouse example */
#include "winbgim.h"

// The click_handler will be called whenever the left mouse button is
// clicked. It checks copies the x,y coordinates of the click to
// see if the click was on a red pixel. If so, then the boolean
// variable red_clicked is set to true. Note that in general
// all handlers should be quick. If they need to do more than a little
// work, they should set a variable that will trigger the work going,
// and then return.
bool red_clicked = false;
void click_handler(int x, int y)
{
    if (getpixel(x,y) == RED)
        red_clicked = true;
}

// Call this function to draw an isosoles triangle with the given
// base and
// height. The triangle will be drawn just above the bottom of the
// screen.
void triangle(int base, int height)
{
    int maxx = getmaxx();
    int maxy = getmaxy();

    line(maxx/2 - base/2, maxy - 10, maxx/2 + base/2, maxy - 10);
    line(maxx/2 - base/2, maxy - 10, maxx/2, maxy - 10 - height);
    line(maxx/2 + base/2, maxy - 10, maxx/2, maxy - 10 - height);
}

void main(void)
{
    int maxx, maxy; // Maximum x and y pixel coordinates
    int divisor; // Divisor for the length of a triangle side

    // Put the machine into graphics mode and get the maximum
    // coordinates:
    initwindow(450, 300);
    maxx = getmaxx();
    maxy = getmaxy();

    // Register the function that handles a left mouse click
    registermousehandler(WM_LBUTTONDOWN, click_handler);

    // Draw a white circle with red inside and a radius of 50 pixels:
    setfillstyle(SOLID_FILL, RED);
    setcolor(WHITE);
    fillellipse(maxx/2, maxy/2, 50, 50);

    // Print a message and wait for a red pixel to be double clicked:
    settextstyle(DEFAULT_FONT, HORIZ_DIR, 2);
    outtextxy(20, 20, "Left click in RED to end.");
    setcolor(BLUE);
    red_clicked = false;
divisor = 2;
while (!red_clicked)
{
    triangle(maxx/divisor, maxy/divisor);
    delay(500);
    divisor++;
}

cout << "The mouse was clicked at: ";
cout << "x=\n" << mousex() ;
cout << " y=\n" << mousey() << endl;
// Switch back to text mode:
closegraph();

mousey

Syntax
#include "winbgim.h"
int mousey(void);

Description
The mousey function is available in the winbgim implementation of BGI graphics. It returns the most recent y coordinate of the mouse within the graphics window. Y-coordinates start with 0 at the top edge of the window and increase to getmaxy( ) at the bottom edge of the window.

Return Value
Most recent y coordinate of the mouse within the graphics window

See also
getmaxy
mousey
registermousehandler

Example
/* mouse example */
#include "winbgim.h"

// The click_handler will be called whenever the left mouse button is
// clicked. It checks copies the x,y coordinates of the click to
// see if the click was on a red pixel. If so, then the boolean
// variable red_clicked is set to true. Note that in general
// all handlers should be quick. If they need to do more than a
// little
// work, they should set a variable that will trigger the work going,
// and then return.
bool red_clicked = false;
void click_handler(int x, int y)
{
    if (getpixel(x,y) == RED)
        red_clicked = true;
}

// Call this function to draw an isosoles triangle with the given
base and
void triangle(int base, int height) {
    int maxx = getmaxx();
    int maxy = getmaxy();
    line(maxx/2 - base/2, maxy - 10, maxx/2 + base/2, maxy - 10);
    line(maxx/2 - base/2, maxy - 10, maxx/2, maxy - 10 - height);
    line(maxx/2 + base/2, maxy - 10, maxx/2, maxy - 10 - height);
}

void main(void) {
    int maxx, maxy; // Maximum x and y pixel coordinates
    int divisor; // Divisor for the length of a triangle side

    // Put the machine into graphics mode and get the maximum coordinates:
    inittwindow(450, 300);
    maxx = getmaxx();
    maxy = getmaxy();

    // Register the function that handles a left mouse click
    registermousehandler(WM_LBUTTONDOWN, click_handler);

    // Draw a white circle with red inside and a radius of 50 pixels:
    setfillstyle(SOLID_FILL, RED);
    setcolor(WHITE);
    fillellipse(maxx/2, maxy/2, 50, 50);

    // Print a message and wait for a red pixel to be double clicked:
    settextstyle(DEFAULT_FONT, HORIZ_DIR, 2);
    outtextxy(20, 20, "Left click in RED to end.");
    setcolor(BLUE);
    red_clicked = false;
    divisor = 2;
    while (!red_clicked) {
        triangle(maxx/divisor, maxy/divisor);
        delay(500);
        divisor++;
    }

    cout << "The mouse was clicked at: ";
    cout << "x=" << mousex( ) << endl;
    cout << " y=" << mousey( ) << endl;

    // Switch back to text mode:
    closegraph();
}

moverel

Syntax
#include <graphics.h>
void moverel(int dx, int dy);
Description
moverel moves the current position (CP) dx pixels in the x direction and dy pixels in the y direction.

Return Value
None.

See also
moveto

Example
/* moverel example */

#include <graphics.h>
#include <stdlib.h>
#include <stdio.h>
#include <conio.h>

int main(void)
{
    /* request autodetection */
    int gdriver = DETECT, gmode, errorcode;
    char msg[80];

    /* initialize graphics and local variables */
    initgraph(&gdriver, &gmode, "");

    /* read result of initialization */
    errorcode = graphresult();
    if (errorcode != grOk) { /* an error occurred */
        printf("Graphics error: %s\n", grapherrormsg(errorcode));

        printf("Press any key to halt:");
        getch();
        exit(1); /* terminate with an error code */
    }

    /* move the CP to location (20,30) */
    moveto(20, 30);

    /* plot a pixel at the CP */
    putpixel(getx(), gety(), getmaxcolor());

    /* create and output a message at (20,30) */
    sprintf(msg, " (%d, %d)", getx(), gety());
    outtextxy(20, 30, msg);

    /* move to a point a relative distance away from the current CP */
    moverel(100, 100);

    /* plot a pixel at the CP */
    putpixel(getx(), gety(), getmaxcolor());

    /* create and output a message at CP */
    sprintf(msg, " (%d, %d)", getx(), gety());
    outtext(msg);

    /* clean up */
    getch();
    closegraph();
    return 0;
}
moveto

Syntax

```c
#include <graphics.h>
void moveto(int x, int y);
```

Description

moveto moves the current position (CP) to viewport position (x,y).

Return Value

None.

See also

moverel

Example

```c
/* moveto example */

#include <graphics.h>
#include <stdlib.h>
#include <stdio.h>
#include <conio.h>

int main(void)
{
    /* request autodetection */
    int gdriver = DETECT, gmode, errorcode;
    char msg[80];

    /* initialize graphics and local variables */
    initgraph(&gdriver, &gmode, "");

    /* read result of initialization */
    errorcode = graphresult();
    if (errorcode != grOk) { /* an error occurred */
        printf("Graphics error: %s\n", grapherrormsg(errorcode));

        printf("Press any key to halt: ");
        getch();
        exit(1);               /* terminate with an error code */
    }

    /* move the CP to location (20,30) */
    moveto(20,30);

    /* plot a pixel at the CP */
    putpixel(getx(), gety(), getmaxcolor());

    /* create and output a message at (20,30) */
    sprintf(msg, " (%d, %d)\n", getx(), gety());
    outtextxy(20,30, msg);

    /* move to (100,100) */
    moveto(100,100);

    /* plot a pixel at the CP */
    putpixel(getx(), gety(), getmaxcolor());

    /* create and output a message at CP */
    sprintf(msg, " (%d, %d)\n", getx(), gety());
```
outtext

**Syntax**

```c
#include <graphics.h>
void outtext(char *textstring);
```

**Description**

outtext displays a text string in the viewport, using the current font, direction, and size. outtext outputs textstring at the current position (CP). If the horizontal text justification is LEFT_TEXT and the text direction is HORIZ_DIR, the CP’s x-coordinate is advanced by textwidth(textstring). Otherwise, the CP remains unchanged.

To maintain code compatibility when using several fonts, use textwidth and textheight to determine the dimensions of the string.

If a string is printed with the default font using outtext, any part of the string that extends outside the current viewport is truncated.

outtext is for use in graphics mode; it will not work in text mode.

**Return Value**

None.

**See also**

- gettextsettings
- outtextxy
- settextjustify
- textheight
- textwidth

**Example**

```c
/* outtext example */

#include <graphics.h>
#include <stdlib.h>
#include <stdio.h>
#include <conio.h>

int main(void)
{
    /* request autodetection */
    int gdriver = DETECT, gmode, errorcode;
    int midx, midy;

    /* initialize graphics and local variables */
    initgraph(&gdriver, &gmode, "");

    /* read result of initialization */
    errorcode = graphresult();
    if (errorcode != grOk) { /* an error occurred */
        printf("Graphics error: %s\n", grapherrormsg(errorcode));
```
```c
#include <graphics.h>

int main(void)
{
    printf("Press any key to halt:");
    getch();
    exit(1); /* terminate with an error code */
}

midx = getmaxx() / 2;
midy = getmaxy() / 2;

/* move the CP to the center of the screen */
moveto(midx, midy);

/* output text starting at the CP */
outtext("This ");
outtext("is ");
outtext("a ");
outtext("test.");

/* clean up */
getch();
closegraph();
return 0;
}
```

---

**outtextxy**

**Syntax**

```c
#include <graphics.h>
void outtextxy(int x, int y, char *textstring);
```

**Description**

`outtextxy` displays a text string in the viewport at the given position `(x, y)`, using the current justification settings and the current font, direction, and size. To maintain code compatibility when using several fonts, use `textwidth` and `textheight` to determine the dimensions of the string. If a string is printed with the default font using `outtext` or `outtextxy`, any part of the string that extends outside the current viewport is truncated. `outtextxy` is for use in graphics mode; it will not work in text mode.

**Return Value**

None.

**See also**

- `gettextsettings`
- `outtext`
- `settextjustify`
- `textheight`
- `textwidth`

**Example**

```c
/* outtextxy example */

#include <graphics.h>
#include <stdlib.h>
#include <stdio.h>
#include <conio.h>

int main(void)
```
/* request autodetection */
int gdriver = DETECT, gmode, errorcode;
int midx, midy;

/* initialize graphics and local variables */
initgraph(&gdriver, &gmode, "");

/* read result of initialization */
errorcode = graphresult();
if (errorcode != grOk) { /* an error occurred */
    printf("Graphics error: %s\n", grapherrormsg(errorcode));
    printf("Press any key to halt:");
    getch();
    exit(1);               /* terminate with an error code */
}

midx = getmaxx() / 2;
midy = getmaxy() / 2;

/* output text at center of the screen; CP doesn't get changed */
outtextxy(midx, midy, "This is a test.");

/* clean up */
getch();
closegraph();
return 0;

pieslice

Syntax
#include <graphics.h>
void pieslice(int x, int y, int stangle, int endangle, int radius);

Description
pieslice draws and fills a pie slice centered at (x,y) with a radius given by radius. The slice travels from stangle to endangle. The slice is outlined in the current drawing color and then filled using the current fill pattern and fill color.
The angles for pieslice are given in degrees. They are measured counterclockwise, with 0 degrees at 3 o'clock, 90 degrees at 12 o'clock, and so on.
If you're using a CGA or monochrome adapter, the examples in online Help that show how to use graphics functions might not produce the expected results. If your system runs on a CGA or monochrome adapter, use the value 1 (one) instead of the symbolic color constant, and see the second example under arc whis shows how to use the pieslice function.

Return Value
None.
See also
fillellipse
graphresult
sector
setfillstyle

Example

/* pieslice example */

#include <graphics.h>
#include <stdlib.h>
#include <stdio.h>
#include <conio.h>

int main(void)
{
    /* request autodetection */
    int gdriver = DETECT, gmode, errorcode;
    int midx, midy;
    int stangle = 45, endangle = 135, radius = 100;

    /* initialize graphics and local variables */
    initgraph(&gdriver, &gmode, "");

    /* read result of initialization */
    errorcode = graphresult();
    if (errorcode != grOk) /* an error occurred */
    {
        printf("Graphics error: %s\n", grapherrormsg(errorcode));

        printf("Press any key to halt:");
        getch();
        exit(1);               /* terminate with an error code */
    }

    midx = getmaxx() / 2;
    midy = getmaxy() / 2;

    /* set fill style and draw a pie slice */
    setfillstyle(EMPTY_FILL, getmaxcolor());
    pieslice(midx, midy, stangle, endangle, radius);

    /* clean up */
    getch();
    closegraph();
    return 0;
}

putimage

Syntax

#include <graphics.h>
void putimage(int left, int top, void *bitmap, int op);

Description

putimage puts the bit image previously saved with getimage back onto the screen, with
the upper left corner of the image placed at (left, top). bitmap points to the area in
memory where the source image is stored.
The op parameter to putimage specifies a combination operator that controls how the
color for each destination pixel onscreen is computed, based on the pixel already
onscreen and the corresponding source pixel in memory. The enumeration putimage_ops, as defined in graphics.h, gives names to these operators.

<table>
<thead>
<tr>
<th>Name</th>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>COPY_PUT</td>
<td>0</td>
<td>Copy</td>
</tr>
<tr>
<td>XOR_PUT</td>
<td>1</td>
<td>Exclusive or</td>
</tr>
<tr>
<td>OR_PUT</td>
<td>2</td>
<td>Inclusive or</td>
</tr>
<tr>
<td>AND_PUT</td>
<td>3</td>
<td>And</td>
</tr>
<tr>
<td>NOT_PUT</td>
<td>4</td>
<td>Copy the inverse of the source</td>
</tr>
</tbody>
</table>

In other words, COPY_PUT copies the source bitmap image onto the screen, XOR_PUT XORs the source image with the image already onscreen, OR_PUT ORs the source image with that onscreen, and so on.

Return Value
None.

See also
getimage
imagesize
putpixel
setvisualpage

Example
/* putimage example */

#include <graphics.h>
#include <stdlib.h>
#include <stdio.h>
#include <conio.h>

#define ARROW_SIZE 10

void draw_arrow(int x, int y);

int main()
{
    /* request autodetection */
    int gd = DETECT, gm, errorcode;
    void *arrow;
    int x, y, maxx;
    unsigned int size;

    /* initialize graphics and local variables */
    initgraph(&gd, &gm, "");

    errorcode = graphresult();
    if (errorcode != grOk) /* an error occurred */
    {
        printf("Graphics error: %s\n", grapherrormsg(errorcode));
        printf("Press any key to halt:");
        getch();
        exit(1); /* terminate with an error code */
    }

    maxx = getmaxx();
    x = 0;
    y = getmaxy() / 2;
    draw_arrow(x, y);
/* calculate the size of the image and allocate space for it */
size = imagesize(x, y-ARROW_SIZE, x+(4*ARROW_SIZE), y+ARROW_SIZE);
arrow = malloc(size);

/* grab the image */
getimage(x, y-ARROW_SIZE, x+(4*ARROW_SIZE), y+ARROW_SIZE, arrow);

/* repeat until a key is pressed */
while (!kbhit()) {
    /* erase old image */
    putimage(x, y-ARROW_SIZE, arrow, XOR_PUT);
    x += ARROW_SIZE;
    if (x >= maxx)
        x = 0;
    /* plot new image */
    putimage(x, y-ARROW_SIZE, arrow, XOR_PUT);
}
free(arrow);
closegraph();
return 0;
}

void draw_arrow(int x, int y) {
    moveto(x, y);
    linerel(4*ARROW_SIZE, 0);
    linerel(-2*ARROW_SIZE, -1*ARROW_SIZE);
    linerel(0, 2*ARROW_SIZE);
    linerel(2*ARROW_SIZE, -1*ARROW_SIZE);
}

cputpixel

---

Syntax
#include <graphics.h>
void putpixel(int x, int y, int color);

Description
putpixel plots a point in the color defined by color at (x,y).

Return Value
None.

Windows Notes
The winbgim version allows the color argument to be an ordinary BGI color (from 0 to 15) or an RGB color.

See also
ggetpixel
putimage

Example
/* putpixel example */

#include <graphics.h>
#include <stdlib.h>
#include <stdio.h>
#include <conio.h>
#include <dos.h>

#define PIXEL_COUNT 1000
#define DELAY_TIME 100 /* in milliseconds */

int main()
{
    /* request autodetection */
    int gdriver = DETECT, gmode, errorcode;
    int i, x, y, color, maxx, maxy, maxcolor, seed;

    /* initialize graphics and local variables */
    initgraph(&gdriver, &gmode, "");

    /* read result of initialization */
    errorcode = graphresult();

    if (errorcode != grOk) { /* an error occurred */
        printf("Graphics error: %s\n", grapherrormsg(errorcode));
        printf("Press any key to halt:");
        getch();
        exit(1);               /* terminate with an error code */
    }

    maxx = getmaxx() + 1;
    maxy = getmaxy() + 1;
    maxcolor = getmaxcolor() + 1;

    while (!kbhit())
    {
        /* seed the random number generator */
        seed = random(32767);
        srand(seed);
        for (i=0; i<PIXEL_COUNT; i++) {
            x = random(maxx);
            y = random(maxy);
            color = random(maxcolor);
            putpixel(x, y, color);
        }
        delay(DELAY_TIME);
        srand(seed);
        for (i=0; i<PIXEL_COUNT; i++) {
            x = random(maxx);
            y = random(maxy);
            color = random(maxcolor);
            if (color == getpixel(x, y))
                putpixel(x, y, 0);
        }
    }

    /* clean up */
    getch();
    closegraph();
    return 0;
}
rectangle

Syntax
```c
#include <graphics.h>
void rectangle(int left, int top, int right, int bottom);
```

Description
rectangle draws a rectangle in the current line style, thickness, and drawing color. 
(left,top) is the upper left corner of the rectangle, and (right,bottom) is its lower right 
corner.

Return Value
None.

See also
- bar
- bar3d
- setcolor
- setlinestyle

Example
```c
/* rectangle example */

#include <graphics.h>
#include <stdlib.h>
#include <stdio.h>
#include <conio.h>

int main(void)
{
  /* request autodetection */
  int gdriver = DETECT, gmode, errorcode;
  int left, top, right, bottom;

  /* initialize graphics and local variables */
  initgraph(&gdriver, &gmode, "");

  /* read result of initialization */
  errorcode = graphresult();
  if (errorcode != grOk) { /* an error occurred */
    printf("Graphics error: %s\n", grapherrormsg(errorcode));

    printf("Press any key to halt: ");
    getch();
    exit(1);                           /* terminate with an error code */
  }

  left = getmaxx() / 2 - 50;
  top = getmaxy() / 2 - 50;
  right = getmaxx() / 2 + 50;
  bottom = getmaxy() / 2 + 50;

  /* draw a rectangle */
  rectangle(left, top, right, bottom);

  /* clean up */
  getch();
  closegraph();
  return 0;
}
```
registerbgidriver

Syntax
#include <graphics.h>
int registerbgidriver(void (*driver)(void));

Description
registerbgidriver enables a user to load a driver file and "register" the driver. Once its
memory location has been passed to registerbgidriver, initgraph uses the registered
driver. A user-registered driver can be loaded from disk onto the heap, or converted to
an .OBJ file (using BGIOBJ.EXE) and linked into the .EXE.
Calling registerbgidriver informs the graphics system that the driver pointed to by
driver was included at link time. This routine checks the linked-in code for the
specified driver: if the code is valid, it registers the code in internal tables.
By using the name of a linked-in driver in a call to registerbgidriver, you also tell the
compiler (and linker) to link in the object file with that public name.

Return Value
registerbgidriver returns a negative graphics error code if the specified driver or font is
invalid. Otherwise, registerbgidriver returns the driver number.
If you register a user-supplied driver, you must pass the result of registerbgidriver to
initgraph as the driver number to be used.

Windows Notes
registerbgidriver is not available in the winbgim implementation.

See also
graphresult
initgraph
installuserdriver
registerbgifont

Example
/* registerbgidriver example */

#include <graphics.h>
#include <stdlib.h>
#include <stdio.h>
#include <conio.h>

int main(void)
{
    /* request autodetection */
    int gdriver = DETECT, gmode, errorcode;

    /* register a driver that was added into GRAPHICS.LIB */
    errorcode = registerbgidriver(EGAVGA_driver);

    /* report any registration errors */
    if (errorcode < 0) {
        printf("Graphics error: %s\n", grapherrormsg(errorcode));
        printf("Press any key to halt: ");
        getch();
        exit(1); /* terminate with an error code */
    }

    /* initialize graphics and local variables */
    initgraph(&gdriver, &gmode, "");
}
/* read result of initialization */
errorcode = graphresult();
if (errorcode != grOk) { /* an error occurred */
    printf("Graphics error: %s\n", grapherrormsg(errorcode));
    printf("Press any key to halt:");
    getch();
    exit(1); /* terminate with an error code */
}
/* draw a line */
line(0, 0, getmaxx(), getmaxy());
/* clean up */
getch();
closegraph();
return 0;

registerbgifont

Syntax
#include <graphics.h>
int registerbgifont(void (*font)(void));

Description
Calling registerbgifont informs the graphics system that the font pointed to by font
was included at link time. This routine checks the linked-in code for the specified font;
if the code is valid, it registers the code in internal tables.
By using the name of a linked-in font in a call to registerbgifont, you also tell the
compiler (and linker) to link in the object file with that public name.
If you register a user-supplied font, you must pass the result of registerbgifont to
settextstyle as the font number to be used.

Return Value
registerbgifont returns a negative graphics error code if the specified font is invalid.
Otherwise, registerbgifont returns the font number of the registered font.

Windows Notes
registerbgifont is not available in the winbgim implementation.

See also
graphresult
initgraph
installuserdriver
registerbgidriver
settextstyle

Example
/* registerbgifont example */

#include <graphics.h>
#include <stdlib.h>
#include <stdio.h>
#include <conio.h>
int main(void)
{
    /* request autodetection */
    int gdriver = DETECT, gmode, errorcode;
    int midx, midy;

    /* register a font file that was added into GRAPHICS.LIB */
    errorcode = registerbgifont(triplex_font);

    /* report any registration errors */
    if (errorcode < 0) {
        printf("Graphics error: %s\n", grapherrormsg(errorcode));
        printf("Press any key to halt:");
        getch();
        exit(1);            /* terminate with an error code */
    }

    /* initialize graphics and local variables */
    initgraph(&gdriver, &gmode, "");

    /* read result of initialization */
    errorcode = graphresult();
    if (errorcode != grOk) { /* an error occurred */
        printf("Graphics error: %s\n", grapherrormsg(errorcode));
        printf("Press any key to halt:");
        getch();
        exit(1);            /* terminate with an error code */
    }

    midx = getmaxx() / 2;
    midy = getmaxy() / 2;

    /* select the registered font */
    settextstyle(TRIPLEX_FONT, HORIZ_DIR, 4);

    /* output some text */
    settextjustify(CENTER_TEXT, CENTER_TEXT);
    outtextxy(midx, midy, "The TRIPLEX FONT");

    /* clean up */
    getch();
    closegraph();
    return 0;
}

registermousehandler

Syntax
#include "winbgim.h"
void registermousehandler(int kind, void h(int, int));

Description
The registermousehandler function is available in the winbgim implementation of BGI graphics. In general, you write a different "handler function" to handle each different
kind of mouse event, and you "register" each of your handlers by calling `registermousehandler`. The first argument to `registermousehandler` is one of these constants from the `winbgim.h` file:

- `WM_MOUSEMOVE` if you want the handler called whenever the mouse moves
- `WM_LBUTTONDOWN`...called whenever the left mouse button is clicked down
- `WM_LBUTTONUP`...called whenever the left mouse button is released up
- `WM_MBUTTONDOWN`...called whenever the middle mouse button is clicked down
- `WM_MBUTTONUP`...called whenever the middle mouse button is released up
- `WM_RBUTTONDOWN`...called whenever the right mouse button is clicked down
- `WM_RBUTTONUP`...called whenever the right mouse button is released up

The second argument to `registermousehandler` must be the name of the handler function that you wrote. This function must be a void function with two int parameters. Whenever the specified mouse event occurs, your handler will be called and the two int parameters will be the x and y positions where the event happened.

The middle mouse button handlers aren't working on my machine. I haven't yet tracked down the reason--it could be a broken mouse or it could be a bug in my programming.

**Return Value**
None.

**See also**
- `mousex`
- `mousey`

**Example**

```c
/* mouse example */
#include "winbgim.h"

// The click_handler will be called whenever the left mouse button is clicked. It checks copies the x,y coordinates of the click to see if the click was on a red pixel. If so, then the boolean variable red_clicked is set to true. Note that in general all handlers should be quick. If they need to do more than a little work, they should set a variable that will trigger the work going, and then return.
bool red_clicked = false;
void click_handler(int x, int y)
{
  if (getpixel(x,y) == RED)
    red_clicked = true;
}
```

// Call this function to draw an isosoles triangle with the given base and
void triangle(int base, int height)
{
    int maxx = getmaxx();
    int maxy = getmaxy();
    line(maxx/2 - base/2, maxy - 10, maxx/2 + base/2, maxy - 10);
    line(maxx/2 - base/2, maxy - 10, maxx/2, maxy - 10 - height);
    line(maxx/2 + base/2, maxy - 10, maxx/2, maxy - 10 - height);
}

void main(void)
{
    int maxx, maxy;  // Maximum x and y pixel coordinates
    int divisor;     // Divisor for the length of a triangle side

    // Put the machine into graphics mode and get the maximum coordinates:
    initwindow(450, 300);
    maxx = getmaxx();
    maxy = getmaxy();

    // Register the function that handles a left mouse click
    registermousehandler(WM_LBUTTONDOWN, click_handler);

    // Draw a white circle with red inside and a radius of 50 pixels:
    setfillstyle(SOLID_FILL, RED);
    setcolor(WHITE);
    fillellipse(maxx/2, maxy/2, 50, 50);

    // Print a message and wait for a red pixel to be double clicked:
    settextstyle(DEFAULT_FONT, HORIZ_DIR, 2);
    outtextxy(20, 20, "Left click in RED to end.");
    setcolor(BLUE);
    red_clicked = false;
    divisor = 2;
    while (!red_clicked)
    {
        triangle(maxx/divisor, maxy/divisor);
        delay(500);
        divisor++;
    }

cout << "The mouse was clicked at: ";
cout << "x=" << mousex() << " y=" << mousey() << endl;

    // Switch back to text mode:
    closegraph();
}

#include <graphics.h>
void restorecrtmode(void);

Description
restorecrtmode restores the original video mode detected by initgraph.
This function can be used in conjunction with setgraphmode to switch back and forth
between text and graphics modes. textmode should not be used for this purpose; use it
only when the screen is in text mode, to change to a different text mode.

Return Value
None.

Windows Notes
restorecrtmode is implemented in winbgim, but it does not do any work. This is
because both the graphics window and the text window are always open during any
Windows program, so there is no need to switch back and forth between the two
modes.

See also
getgraphmode
initgraph
setgraphmode

Example
/* restorecrtmode example */

#include <graphics.h>
#include <stdlib.h>
#include <stdio.h>
#include <conio.h>

int main(void)
{
    /* request autodetection */
    int gdriver = DETECT, gmode, errorcode;
    int x, y;

    /* initialize graphics and local variables */
    initgraph(&gdriver, &gmode, "");

    /* read result of initialization */
    errorcode = graphresult();
    if (errorcode != grOk) { /* an error occurred */
        printf("Graphics error: %s\n", grapherrormsg(errorcode));
        printf("Press any key to halt:");
        getch();
        exit(1); /* terminate with an error code */
    }

    x = getmaxx() / 2;
    y = getmaxy() / 2;

    /* output a message */
    settextjustify(CENTER_TEXT, CENTER_TEXT);
    outtextxy(x, y, "Press any key to exit graphics:");
    getch();

    /* restore system to text mode */
    restorecrtmode();
    printf("We're now in text mode.\n");
    printf("Press any key to return to graphics mode:");
    getch();
Colors for Windows BGI

The winbgim package supports two types of colors that may be used with any of the functions that expect colors as arguments:

1. The sixteen ordinary BGI colors. These are the integers 0 through 15 or you may use the symbolic names:
   2. BLACK          BLUE          GREEN         CYAN
   3. RED            MAGENTA       BROWN         LIGHTGRAY
   4. DARKGRAY       LIGHTBLUE     LIGHTGREEN    LIGHTCYAN
   5. LIGHTRED       LIGHTMAGENTA  YELLOW        WHITE

2. A color may be specified from red, green and blue components using a new macro called COLOR(r,g,b). Each of the r,g,b arguments must be a number in the range 0 to 255. For example, COLOR(255,100,0) is a mostly red color with some green and no blue. If you create one of these colors, it may be used as an argument to any of the BGI functions that expect a color. These colors may also be returned from BGI functions such as getbkcolor.

Three other macros (RED_VALUE, GREEN_VALUE, BLUE_VALUE, IS_BGI_COLOR and IS_RGB_COLOR) are explained in the examples below.

RGB Examples:

```c
#include "winbgim.h"

setcolor(BLUE);             // Change drawing color to BLUE.
setcolor(COLOR(255,100,0)); // Change drawing color to reddish-green.
setpalette(4, BLUE);        // Change palette entry 4 to BLUE.
setpalette(4, COLOR(9,9,9));// Change palette entry 4 to nearly black.

int current = getcolor();  // Set current to current drawing color.

if (IS_BGI_COLOR(current)) // Check whether it is a BGI color.
   cout << "Current BGI drawing color is: " << current << endl;

if (IS_RGB_COLOR(current)) // Check whether it is an RGB color.
   cout << "Current RGB drawing color has these components:\n" << "Red: " << RED_VALUE(current) << '\n'
   << "Green: " << GREEN_VALUE(current) << '\n'
   << "Blue: " << BLUE_VALUE(current) << '\n'
**sector**

**Syntax**
```c
#include <graphics.h>
void sector(int x, int y, int stangle, int endangle, int xradius, int yradius);
```

**Description**
Draws and fills an elliptical pie slice using (x,y) as the center point, xradius and yradius as the horizontal and vertical radii, respectively, and drawing from stangle to endangle. The pie slice is outlined using the current color, and filled using the pattern and color defined by setfillstyle or setfillpattern.

The angles for sector are given in degrees. They are measured counter-clockwise with 0 degrees at 3 o'clock, 90 degrees at 12 o'clock, and so on.

If an error occurs while the pie slice is filling, graphresult returns a value of -6 (grNoScanMem).

**Return Value**
None.

**See also**
- arc
- circle
- ellipse
- getarccoords
- getaspectratio
- graphresult
- pieslice
- setfillpattern
- setfillstyle
- setgraphbufsize

**Example**
```c
/* sector example */

#include <graphics.h>
#include <stdlib.h>
#include <stdio.h>
#include <conio.h>

int main(void)
{
    /* request autodetection */
    int gdriver = DETECT, gmode, errorcode;
    int midx, midy, i;
    int stangle = 45, endangle = 135;
    int xrad = 100, yrad = 50;

    /* initialize graphics and local variables */
    initgraph(&gdriver, &gmode, "");

    /* read result of initialization */
    errorcode = graphresult();
    if (errorcode != grOk) { /* an error occurred */
        printf("Graphics error: %s\n", grapherrormsg(errorcode));
        printf("Press any key to halt:\n");
        getch();
    }
    return 0;
}
```
exit(1);               /* terminate with an error code */
}

midx = getmaxx() / 2;
midy = getmaxy() / 2;

/* loop through the fill patterns */
for (i=EMPTY_FILL; i<USER_FILL; i++) {
    /* set the fill style */
    setfillstyle(i, getmaxcolor());

    /* draw the sector slice */
    sector(midx, midy, stangle, endangle, xrad, yrad);

    getch();
}

/* clean up */
closegraph();
return 0;
}

setactivepage

Syntax
#include <graphics.h>
void setactivepage(int page);

Description
setactivepage makes page the active graphics page. All subsequent graphics output
will be directed to that graphics page.
The active graphics page might not be the one you see onscreen, depending on how
many graphics pages are available on your system. Only the EGA, VGA, and Hercules
graphics cards support multiple pages.

Return Value
None.

See also
setvisualpage

Example
/* setactivepage example */

#include <graphics.h>
#include <stdlib.h>
#include <stdio.h>
#include <conio.h>

int main(void)
{
    /* select driver and mode that supports multiple pages */
    int gdriver = EGA, gmode = EGAHI, errorcode;
    int x, y, ht;

    /* initialize graphics and local variables */
    initgraph(&gdriver, &gmode, "");
/* read result of initialization */
errorcode = graphresult();
if (errorcode != grOk)    /* an error occurred */
{
    printf("Graphics error: %s\n", graperrorormsg(errorcode));
    printf("Press any key to halt:");
    getch();
    exit(1);               /* terminate with an error code */
}

x = getmaxx() / 2;
y = getmaxy() / 2;
ht = textheight("W");

/* select the off screen page for drawing */
setactivepage(1);

/* draw a line on page #1 */
line(0, 0, getmaxx(), getmaxy());

/* output a message on page #1 */
settextjustify(CENTER_TEXT, CENTER_TEXT);
outtextxy(x, y, "This is page #1: ");
outtextxy(x, y+ht, "Press any key to halt: ");

/* select drawing to page #0 */
setactivepage(0);

/* output a message on page #0 */
outtextxy(x, y, "This is page #0.");
outtextxy(x, y+ht, "Press any key to view page #1.");
getch();

/* select page #1 as the visible page */
setvisualpage(1);

/* clean up */
getch();
closegraph();
return 0;

setallpalette

---

**Syntax**

```c
#include <graphics.h>
void setallpalette(struct palettetype *palette);
```

**Description**

setallpalette sets the current palette to the values given in the palettetype structure pointed to by palette.

You can partially (or completely) change the colors in the EGA/VGA palette with setallpalette.

The MAXCOLORS constant and the palettetype structure used by setallpalette are defined in graphics.h as follows:

```c
#define MAXCOLORS 15
```
struct palettetype {
    unsigned char size;
    signed char colors[MAXCOLORS + 1];
};

size gives the number of colors in the palette for the current graphics driver in the
current mode.

colors is an array of size bytes containing the actual raw color numbers for each entry
in the palette. If an element of colors is -1, the palette color for that entry is not
changed.

The elements in the colors array used by setallpalette can be represented by symbolic
constants which are defined in graphics.h. See Actual Color Table given here:

<table>
<thead>
<tr>
<th>Name</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>BLACK</td>
<td>0</td>
</tr>
<tr>
<td>BLUE</td>
<td>1</td>
</tr>
<tr>
<td>GREEN</td>
<td>2</td>
</tr>
<tr>
<td>CYAN</td>
<td>3</td>
</tr>
<tr>
<td>RED</td>
<td>4</td>
</tr>
<tr>
<td>MAGENTA</td>
<td>5</td>
</tr>
<tr>
<td>BROWN</td>
<td>6</td>
</tr>
<tr>
<td>LIGHTGRAY</td>
<td>7</td>
</tr>
<tr>
<td>DARKGRAY</td>
<td>8</td>
</tr>
<tr>
<td>LIGHTBLUE</td>
<td>9</td>
</tr>
<tr>
<td>LIGHTGREEN</td>
<td>10</td>
</tr>
<tr>
<td>LIGHTCYAN</td>
<td>11</td>
</tr>
<tr>
<td>LIGHTRED</td>
<td>12</td>
</tr>
<tr>
<td>LIGHTMAGENTA</td>
<td>13</td>
</tr>
<tr>
<td>YELLOW</td>
<td>14</td>
</tr>
<tr>
<td>WHITE</td>
<td>15</td>
</tr>
<tr>
<td>EGA_BROWN</td>
<td>20</td>
</tr>
<tr>
<td>EGA_DARKGRAY</td>
<td>56</td>
</tr>
<tr>
<td>EGA_LIGHTBLUE</td>
<td>57</td>
</tr>
<tr>
<td>EGA_LIGHTGREEN</td>
<td>58</td>
</tr>
<tr>
<td>EGA_LIGHTCYAN</td>
<td>59</td>
</tr>
<tr>
<td>EGA_LIGHTRED</td>
<td>60</td>
</tr>
<tr>
<td>EGA_LIGHTMAGENTA</td>
<td>61</td>
</tr>
<tr>
<td>EGA_YELLOW</td>
<td>62</td>
</tr>
<tr>
<td>EGA_WHITE</td>
<td>63</td>
</tr>
</tbody>
</table>

Changes made to the palette are seen immediately onscreen. Each time a palette color
is changed, all occurrences of that color onscreen change to the new color value.

Note: Valid colors depend on the current graphics driver and current graphics mode.
setallpalette cannot be used with the IBM-8514 driver.

Return Value
If invalid input is passed to setallpalette, graphresult returns -11 (grError), and the
current palette remains unchanged.
The `winbgim` version of `setallpalette` expects a palettetype object of up to 16 colors. Each color is one of the 16 BGI color numbers (0 through 15). If you want to set a palette color to an RGB color, then use `setrgbpalette`.

In the windows version, changing the palette affects only future drawing. Currently drawn pixels do not change their color when the palette changes (no "palette animation").

**See also**
- `getpalette`
- `getpalettesize`
- `graphresult`
- `setbkcolor`
- `setcolor`
- `setpalette`

**Example**

```c
/* setallpalette example */

#include <graphics.h>
#include <stdlib.h>
#include <stdio.h>
#include <conio.h>

int main(void)
{
    /* request autodetection */
    int gdriver = DETECT, gmode, errorcode;
    struct palettetype pal;
    int color, maxcolor, ht;
    int y = 10;
    char msg[80];

    /* initialize graphics and local variables */
    initgraph(&gdriver, &gmode, "");

    /* read result of initialization */
    errorcode = graphresult();
    if (errorcode != grOk)    /* an error occurred */
    {
        printf("Graphics error: %s\n", grapherrormsg(errorcode));
        printf("Press any key to halt:");
        getch();
        exit(1);               /* terminate with an error code */
    }

    maxcolor = getmaxcolor();
```
ht = 2 * textheight("W");

/* grab a copy of the palette */
getpalette(&pal);

/* display the default palette colors */
for (color=1; color<=maxcolor; color++) {
    setcolor(color);
    sprintf(msg, "Color: %d", color);
    outtextxy(1, y, msg);
    y += ht;
}

/* wait for a key */
getch();

/* black out the colors one by one */
for (color=1; color<=maxcolor; color++) {
    setpalette(color, BLACK);
    getch();
}

/* restore the palette colors */
setallpalette(&pal);

/* clean up */
getch();
closegraph();
return 0;
}

setaspectratio

Syntax
#include <graphics.h>
void setaspectratio(int xasp, int yasp);

Description
setaspectratio changes the default aspect ratio of the graphics system. The graphics system uses the aspect ratio to make sure that circles are round onscreen. If circles appear elliptical, the monitor is not aligned properly. You could correct this in the hardware by realigning the monitor, but it's easier to change in the software by using setaspectratio to set the aspect ratio. To obtain the current aspect ratio from the system, call getaspectratio.

Return Value
None.

See also
circle
global
getaspectratio

Example
/* setaspectratio example */

#include <graphics.h>
```c
#include <stdlib.h>
#include <stdio.h>
#include <conio.h>

int main(void) {
    /* request autodetection */
    int gdriver = DETECT, gmode, errorcode;
    int xasp, yasp, midx, midy;

    /* initialize graphics and local variables */
    initgraph(&gdriver, &gmode, "");

    /* read result of initialization */
    errorcode = graphresult();
    if (errorcode != grOk) /* an error occurred */
    {
        printf("Graphics error: %s\n", grapherrormsg(errorcode));
        printf("Press any key to halt:");
        getch();
        exit(1); /* terminate with an error code */
    }

    midx = getmaxx() / 2;
    midy = getmaxy() / 2;
    setcolor(getmaxcolor());

    /* get current aspect ratio settings */
    getaspectratio(&xasp, &yasp);

    /* draw normal circle */
    circle(midx, midy, 100);
    getch();

    /* clear the screen */
    cleardevice();

    /* adjust the aspect for a wide circle */
    setaspectratio(xasp/2, yasp);
    circle(midx, midy, 100);
    getch();

    /* adjust the aspect for a narrow circle */
    cleardevice();
    setaspectratio(xasp, yasp/2);
    circle(midx, midy, 100);

    /* clean up */
    getch();
    closegraph();
    return 0;
}
```
Syntax

```c
#include <graphics.h>
void setbkcolor(int color);
```

Description

`setbkcolor` sets the background to the color specified by `color`. The argument `color` can be a name or a number as listed below. (These symbolic names are defined in `graphics.h`.)

<table>
<thead>
<tr>
<th>Name</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>BLACK</td>
<td>0</td>
</tr>
<tr>
<td>BLUE</td>
<td>1</td>
</tr>
<tr>
<td>GREEN</td>
<td>2</td>
</tr>
<tr>
<td>CYAN</td>
<td>3</td>
</tr>
<tr>
<td>RED</td>
<td>4</td>
</tr>
<tr>
<td>MAGENTA</td>
<td>5</td>
</tr>
<tr>
<td>BROWN</td>
<td>6</td>
</tr>
<tr>
<td>LIGHTGRAY</td>
<td>7</td>
</tr>
<tr>
<td>DARKGRAY</td>
<td>8</td>
</tr>
<tr>
<td>LIGHTBLUE</td>
<td>9</td>
</tr>
<tr>
<td>LIGHTGREEN</td>
<td>10</td>
</tr>
<tr>
<td>LIGHTCYAN</td>
<td>11</td>
</tr>
<tr>
<td>LIGHTRED</td>
<td>12</td>
</tr>
<tr>
<td>LIGHTMAGENTA</td>
<td>13</td>
</tr>
<tr>
<td>YELLOW</td>
<td>14</td>
</tr>
<tr>
<td>WHITE</td>
<td>15</td>
</tr>
</tbody>
</table>

For example, if you want to set the background color to blue, you can call

```
setbkcolor(BLUE) /* or */ setbkcolor(1)
```

On CGA and EGA systems, `setbkcolor` changes the background color by changing the first entry in the palette.

If you use an EGA or a VGA, and you change the palette colors with `setpalette` or `setallpalette`, the defined symbolic constants might not give you the correct color. This is because the parameter to `setbkcolor` indicates the entry number in the current palette rather than a specific color (unless the parameter passed is 0, which always sets the background color to black).

Return Value

None.

Windows Notes

The winbgim version allows the color argument to be an ordinary BGI color (from 0 to 15) or an RGB color. Also, only future drawing will use the new background color (anything currently drawn in the old background color will stay in the old color). Calling `setbkcolor(0)` will change the background color to the current color at index [0] of the palette (rather than always changing the background to black).

See also

- `getbkcolor`
- `setallpalette`
- `setcolor`
- `setpalette`
Example
/* setbkcolor example */

#include <graphics.h>
#include <stdlib.h>
#include <stdio.h>
#include <conio.h>

int main(void)
{
    /* _select driver and mode that supports multiple background colors*/
    int gdriver = EGA, gmode = EGAHI, errorcode;
    int bkcol, maxcolor, x, y;
    char msg[80];

    /* initialize graphics and local variables */
    initsgraph(&gdriver, &gmode, "");

    /* read result of initialization */
    errorcode = graphresult();
    if (errorcode != grOk) { /* an error occurred */
        printf("Graphics error: %s\n", grapherrormsg(errorcode));
        printf("Press any key to halt:");
        getch();
        exit(1); /* terminate with an error code */
    }

    /* maximum color index supported */
    maxcolor = getmaxcolor();

    /* for centering text messages */
    settextjustify(CENTER_TEXT, CENTER_TEXT);
    x = getmaxx() / 2;
    y = getmaxy() / 2;

    /* loop through the available colors */
    for (bkcol=0; bkcol<=maxcolor; bkcol++) {

        /* clear the screen */
        cleardevice();

        /* select a new background color */
        setbkcolor(bkcol);

        /* output a message */
        if (bkcol == WHITE)
            setcolor(EGA_BLUE);
        sprintf(msg, "Background color: %d", bkcol);
        outtextxy(x, y, msg);
        getch();
    } /* clean up */
    closegraph();
    return 0;
}
**setcolor**

**Syntax**

```c
#include <graphics.h>
void setcolor(int color);
```

**Description**

setcolor sets the current drawing color to color, which can range from 0 to `getmaxcolor`. The current drawing color is the value to which pixels are set when lines, and so on are drawn. The drawing colors shown below are available for the CGA and EGA, respectively.

<table>
<thead>
<tr>
<th>Palette Number</th>
<th>Three Colors</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>LIGHTGREEN LIGHTRED YELLOW</td>
</tr>
<tr>
<td>1</td>
<td>LIGHTCYAN LIGHTMAGENTA WHITE</td>
</tr>
<tr>
<td>2</td>
<td>GREEN RED BROWN</td>
</tr>
<tr>
<td>3</td>
<td>CYAN MAGENTA LIGHTGRAY</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Name</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>BLACK</td>
<td>0</td>
</tr>
<tr>
<td>BLUE</td>
<td>1</td>
</tr>
<tr>
<td>GREEN</td>
<td>2</td>
</tr>
<tr>
<td>CYAN</td>
<td>3</td>
</tr>
<tr>
<td>RED</td>
<td>4</td>
</tr>
<tr>
<td>MAGENTA</td>
<td>5</td>
</tr>
<tr>
<td>BROWN</td>
<td>6</td>
</tr>
<tr>
<td>LIGHTGRAY</td>
<td>7</td>
</tr>
<tr>
<td>DARKGRAY</td>
<td>8</td>
</tr>
<tr>
<td>LIGHTBLUE</td>
<td>9</td>
</tr>
<tr>
<td>LIGHTGREEN</td>
<td>10</td>
</tr>
<tr>
<td>LIGHTCYAN</td>
<td>11</td>
</tr>
<tr>
<td>LIGHTRED</td>
<td>12</td>
</tr>
<tr>
<td>LIGHTMAGENTA</td>
<td>13</td>
</tr>
<tr>
<td>YELLOW</td>
<td>14</td>
</tr>
<tr>
<td>WHITE</td>
<td>15</td>
</tr>
</tbody>
</table>

You select a drawing color by passing either the color number itself or the equivalent symbolic name to setcolor. For example, in CGA mode, the palette contains four colors: the background color, light green, light red, and yellow. In this mode, either `setcolor(3)` or `setcolor(CG_A_YELLOW)` selects a drawing color of yellow.

**Return Value**

None.

**Windows Notes**

The `winbgim` version allows the color argument to be an ordinary BGI color (from 0 to 15) or an RGB color.

**See also**

getcolor
getmaxcolor
Example

/* setcolor example */

#include <graphics.h>
#include <stdlib.h>
#include <stdio.h>
#include <conio.h>

int main(void)
{
    /* select driver and mode that supports multiple drawing colors */
    int gdriver = EGA, gmode = EGAHI, errorcode;
    int color, maxcolor, x, y;
    char msg[80];

    /* initialize graphics and local variables */
    initgraph(&gdriver, &gmode, "");

    /* read result of initialization */
    errorcode = graphresult();
    if (errorcode != grOk)    /* an error occurred */
    {
        printf("Graphics error: %s\n", grapherrormsg(errorcode));
        printf("Press any key to halt:");
        getch();
        exit(1);               /* terminate with an error code */
    }

    /* maximum color index supported */
    maxcolor = getmaxcolor();

    /* for centering text messages */
    settextjustify(CENTER_TEXT, CENTER_TEXT);
    x = getmaxx() / 2;
    y = getmaxy() / 2;

    /* loop through the available colors */
    for (color=1; color<=maxcolor; color++) {
        cleardevice();         /* clear the screen */
        setcolor(color);       /* select new background color */

        /* output a message */
        sprintf(msg, "Color: %d", color);
        outtextxy(x, y, msg);
        getch();
    }

    /* clean up */
    closegraph();
    return 0;
}
setfillpattern

Syntax

```c
#include <graphics.h>
void setfillpattern(char *upattern, int color);
```

Description

setfillpattern is like setfillstyle, except that you use it to set a user-defined 8x8 pattern rather than a predefined pattern.

`upattern` is a pointer to a sequence of 8 bytes, with each byte corresponding to 8 pixels in the pattern. Whenever a bit in a pattern byte is set to 1, the corresponding pixel is plotted.

Return Value

None.

Windows Notes

The `winbgim` version allows the color argument to be an ordinary BGI color (from 0 to 15) or an RGB color.

See also

- `getfillpattern`
- `getfillsettings`
- `graphresult`
- `sector`
- `setfillstyle`

Example

```c
/* setfillpattern example */

#include <graphics.h>
#include <stdlib.h>
#include <stdio.h>
#include <conio.h>

int main(void)
{
    /* request autodetection */
    int gdriver = DETECT, gmode, errorcode;
    int maxx, maxy;

    /* a user-defined fill pattern */
    char pattern[8] = {0x00, 0x70, 0x20, 0x27, 0x24, 0x24, 0x07, 0x00};

    /* initialize graphics and local variables */
    initgraph(&gdriver, &gmode, "");

    /* read result of initialization */
    errorcode = graphresult();
    if (errorcode != grOk)    /* an error occurred */
    {
        printf("Graphics error: %s\n", grapherrormsg(errorcode));
        printf("Press any key to halt:");
        getch();
        exit(1);               /* terminate with an error code */
    }

    maxx = getmaxx();
```
maxy = getmaxy();
setcolor(getmaxcolor());

/* select a user-defined fill pattern */
setfillpattern(pattern, getmaxcolor());

/* fill the screen with the pattern */
bar(0, 0, maxx, maxy);

/* clean up */
getch();
closegraph();
return 0;
}

setfillstyle

Syntax

#include <graphics.h>
void setfillstyle(int pattern, int color);

Description

setfillstyle sets the current fill pattern and fill color. To set a user-defined fill pattern,
do not give a pattern of 12 (USER_FILL) to setfillstyle; instead, call setfillpattern.
If invalid input is passed to setfillstyle, graphresult returns -1(grError), and the current
fill pattern and fill color remain unchanged.

Return Value

None.

Windows Notes

The winbgim version allows the color argument to be an ordinary BGI color (from 0 to
15) or an RGB color.

See also

bar
bar3d
fillpoly
floodfill
getfillsettings
graphresult
pieslice
sector
setfillpattern

Example

/* setfillstyle example */

#include <graphics.h>
#include <stdlib.h>
#include <string.h>
#include <stdio.h>
#include <conio.h>

/* the names of the fill styles supported */

int main(void)
{
    /* request autodetection */
    int gdriver = DETECT, gmode, errorcode;
    int style, midx, midy;

    char stylestr[40];

    /* initialize graphics and local variables */
    initgraph(&gdriver, &gmode, "");

    /* read result of initialization */
    errorcode = graphresult();
    if (errorcode != grOk) { /* an error occurred */
        printf("Graphics error: %s\n", grapherrormsg(errorcode));
        printf("Press any key to halt: ");
        getch();
        exit(1);               /* terminate with an error code */
    }

    midx = getmaxx() / 2;
    midy = getmaxy() / 2;

    for (style = EMPTY_FILL; style < USER_FILL; style++) {
        /* select the fill style */
        setfillstyle(style, getmaxcolor());

        /* convert style into a string */
        strcpy(stylestr, fname[style]);

        /* fill a bar */
        bar3d(0, 0, midx-10, midy, 0, 0);

        /* output a message */
        outtextxy(midx, midy, stylestr);

        /* wait for a key */
        getch();
        cleardevice();
    }

    /* clean up */
    getch();
    closegraph();
    return 0;
}
Syntax

```c
#include <graphics.h>
unsigned setgraphbufsize(unsigned bufsize);
```

Description

Some of the graphics routines (such as floodfill) use a memory buffer that is allocated when initgraph is called and released when closegraph is called. The default size of this buffer, allocated by _graphgetmem, is 4,096 bytes.

You might want to make this buffer smaller (to save memory space) or bigger (if, for example, a call to floodfill produces error -7: Out of flood memory).

setgraphbufsize tells initgraph how much memory to allocate for this internal graphics buffer when it calls _graphgetmem.

You must call setgraphbufsize before calling initgraph. Once initgraph has been called, all calls to setgraphbufsize are ignored until after the next call to closegraph.

Return Value

setgraphbufsize returns the previous size of the internal buffer.

Windows Notes

setgraphbufsize is not available in the winbgim implementation.

See also

- closegraph
- initgraph
- sector

Example

```c
/* setgraphbufsize example */

#include <graphics.h>
#include <stdlib.h>
#include <stdio.h>
#include <conio.h>

#define BUFSIZE 1000 /* internal graphics buffer size */

int main(void)
{
    /* request autodetection */
    int gdriver = DETECT, gmode, errorcode;
    int x, y, oldsize;
    char msg[80];

    /* _set size of internal graphics buffer before calling initgraph */
    oldsize = setgraphbufsize(BUFSIZE);

    /* initialize graphics and local variables */
    initgraph(&gdriver, &gmode, "");

    /* read result of initialization */
    errorcode = graphresult();
    if (errorcode != grOk) { /* an error occurred */
        printf("Graphics error: %s\n", grapherrormsg(errorcode));
        printf("Press any key to halt:");
        getch();
        exit(1); /* terminate with an error code */
    }

    x = getmaxx() / 2;
```
y = getmaxy() / 2;

/* output some messages */
sprintf(msg, "Graphics buffer size: %d", BUFSIZE);
settextjustify(CENTER_TEXT, CENTER_TEXT);

outtextxy(x, y, msg);
sprintf(msg, "Old graphics buffer size: %d", oldsize);
outtextxy(x, y + textheight("W"), msg);

/* clean up */
getch();
closegraph();
return 0;
}

setgraphmode

Syntax
#include <graphics.h>
void setgraphmode(int mode);

Description
setgraphmode selects a graphics mode different than the default one set by initgraph.
mode must be a valid mode for the current device driver. setgraphmode clears the
screen and resets all graphics settings to their defaults (current position, palette, color,
viewport, and so on).
You can use setgraphmode in conjunction with restorecrtmode to switch back and
forth between text and graphics modes.

Return Value
If you give setgraphmode an invalid mode for the current device driver, graphresult
returns a value of -10 (grInvalidMode).

Windows Notes WIN
setgraphmode is implemented in winbgim, but it does not do any work.

Windows Notes WIN
setgraphmode is implemented in winbgim, but it does not do any work. This is
because both the graphics window and the text window are always open during any
Windows program, so there is no need to switch back and forth between the two
modes.

See also
getgraphmode
getmoderange
graphresult
initgraph
restorecrtmode

Example
/* setgraphmode example */

#include <graphics.h>
#include <stdlib.h>
#include <stdio.h>
#include <conio.h>
int main(void)
{
    /* request autodetection */
    int gdriver = DETECT, gmode, errorcode;
    int x, y;

    /* initialize graphics and local variables */
    initgraph(&gdriver, &gmode, "");

    /* read result of initialization */
    errorcode = graphresult();
    if (errorcode != grOk) { /* an error occurred */
        printf("Graphics error: %s\n", grapherrormsg(errorcode));
        printf("Press any key to halt: ");
        getch();
        exit(1); /* terminate with an error code */
    }

    x = getmaxx() / 2;
    y = getmaxy() / 2;

    /* output a message */
    settextjustify(CENTER_TEXT, CENTER_TEXT);
    outtextxy(x, y, "Press any key to exit graphics:");
    getch();

    /* restore system to text mode */
    restorecrtmode();
    printf("We're now in text mode.\n");
    printf("Press any key to return to graphics mode: ");
    getch();

    /* return to graphics mode */
    setgraphmode(getgraphmode());

    /* output a message */
    settextjustify(CENTER_TEXT, CENTER_TEXT);
    outtextxy(x, y, "We're back in graphics mode.");
    outtextxy(x, y + textheight("W"), "Press any key to halt:");

    /* clean up */
    getch();
    closegraph();
    return 0;
}

setlinestyle

Syntax
#include <graphics.h>
void setlinestyle(int linestyle, unsigned upattern, int thickness);

Description
setlinestyle sets the style for all lines drawn by line, lineto, rectangle, drawpoly, and so on.
The linesettingstype structure is defined in graphics.h as follows:
```c
struct linesettingstype {
    int linestyle;
    unsigned upattern;
    int thickness;
};
```
lindrome specifies in which of several styles subsequent lines will be drawn (such as solid, dotted, centered, dashed). The enumeration line_styles, which is defined in graphics.h, gives names to these operators:

<table>
<thead>
<tr>
<th>Name</th>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>SOLID_LINE</td>
<td>0</td>
<td>Solid line</td>
</tr>
<tr>
<td>DOTTED_LINE</td>
<td>1</td>
<td>Dotted line</td>
</tr>
<tr>
<td>CENTER_LINE</td>
<td>2</td>
<td>Centered line</td>
</tr>
<tr>
<td>DASHED_LINE</td>
<td>3</td>
<td>Dashed line</td>
</tr>
<tr>
<td>USERBIT_LINE</td>
<td>4</td>
<td>User-defined line style</td>
</tr>
</tbody>
</table>

thickness specifies whether the width of subsequent lines drawn will be normal or thick.

<table>
<thead>
<tr>
<th>Name</th>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>NORM_WIDTH</td>
<td>1</td>
<td>1 pixel wide</td>
</tr>
<tr>
<td>THICK_WIDTH</td>
<td>3</td>
<td>3 pixels wide</td>
</tr>
</tbody>
</table>

upattern is a 16-bit pattern that applies only if linestyle is USERBIT_LINE (4). In that case, whenever a bit in the pattern word is 1, the corresponding pixel in the line is drawn in the current drawing color. For example, a solid line corresponds to a upattern of 0xFFFF (all pixels drawn), and a dashed line can correspond to a upattern of 0x3333 or 0x0F0F. If the linestyle parameter to setlinestyle is not USERBIT_LINE (in other words, if it is not equal to 4), you must still provide the upattern parameter, but it will be ignored.

Note: The linestyle parameter does not affect arcs, circles, ellipses, or pie slices. Only the thickness parameter is used.

**Return Value**

If invalid input is passed to setlinestyle, graphresult returns -11, and the current line style remains unchanged.

**See also**

- arc
- bar3d
- bar
- circle
- drawpoly
- ellipse
- getlinesettings
- graphresult
- line
- linerel
- lineto
- pieslice
- rectangle

**Example**

```c
/* setlinestyle example */

#include <graphics.h>
```
/* the names of the line styles supported */
char *lname[] = { "SOLID_LINE", "DOTTED_LINE", "CENTER_LINE",
"DASHED_LINE", "USERBIT_LINE" };

int main(void)
{
    /* request autodetection */
    int gdriver = DETECT, gmode, errorcode;
    int style, midx, midy, userpat;
    char stylestr[40];

    /* initialize graphics and local variables */
    initgraph(&gdriver, &gmode, "");

    /* read result of initialization */
    errorcode = graphresult();
    if (errorcode != grOk) {  /* an error occurred */
        printf("Graphics error: %s\n", grapherrormsg(errorcode));
        printf("Press any key to halt:");
        getch();
        exit(1);               /* terminate with an error code */
    }

    midx = getmaxx() / 2;
    midy = getmaxy() / 2;

    /* a user-defined line pattern */
    /* binary: "0000000000000001" */
    userpat = 1;
    for (style=SOLID_LINE; style<=USERBIT_LINE; style++)
    {
        /* select the line style */
        setlinestyle(style, userpat, 1);

        /* convert style into a string */
        strcpy(stylestr, lname[style]);

        /* draw a line */
        line(0, 0, midx-10, midy);

        /* draw a rectangle */
        rectangle(0, 0, getmaxx(), getmaxy());

        /* output a message */
        outtextxy(midx, midy, stylestr);

        /* wait for a key */
        getch();
        cleardevice();
    }

    /* clean up */
    closegraph();
setpalette

Syntax
#include <graphics.h>
void setpalette(int colornum, int color);

Description
setpalette changes the colornum entry in the palette to color. For example, setpalette(0,5) changes the first color in the current palette (the background color) to actual color number 5. If size is the number of entries in the current palette, colornum can range between 0 and (size - 1).

You can partially (or completely) change the colors in the EGA/VGA palette with setpalette. On a CGA, you can only change the first entry in the palette (colornum equals 0, the background color) with a call to setpalette.

The color parameter passed to setpalette can be represented by symbolic constants which are defined in graphics.h.

<table>
<thead>
<tr>
<th>Name</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>BLACK</td>
<td>0</td>
</tr>
<tr>
<td>BLUE</td>
<td>1</td>
</tr>
<tr>
<td>GREEN</td>
<td>2</td>
</tr>
<tr>
<td>CYAN</td>
<td>3</td>
</tr>
<tr>
<td>RED</td>
<td>4</td>
</tr>
<tr>
<td>MAGENTA</td>
<td>5</td>
</tr>
<tr>
<td>BROWN</td>
<td>6</td>
</tr>
<tr>
<td>LIGHTGRAY</td>
<td>7</td>
</tr>
<tr>
<td>DARKGRAY</td>
<td>8</td>
</tr>
<tr>
<td>LIGHTBLUE</td>
<td>9</td>
</tr>
<tr>
<td>LIGHTGREEN</td>
<td>10</td>
</tr>
<tr>
<td>LIGHTCYAN</td>
<td>11</td>
</tr>
<tr>
<td>LIGHTRED</td>
<td>12</td>
</tr>
<tr>
<td>LIGHTMAGENTA</td>
<td>13</td>
</tr>
<tr>
<td>YELLOW</td>
<td>14</td>
</tr>
<tr>
<td>WHITE</td>
<td>15</td>
</tr>
</tbody>
</table>

setpalette cannot be used with the IBM-8514 driver; use setrgbpalette instead.

Return Value
If invalid input is passed to setpalette, graphresult returns -11, and the current palette remains unchanged.

Windows Notes
The winbgim version allows the color argument to be an ordinary BGI color (from 0 to 15) or an RGB color. The colornum should be a palette index from 0 to 15.
In the windows version, changing the palette effects only future drawing. Currently drawn pixels do not change their color when the palette changes (no "palette animation").

See also

setrgbpalette

Example

/* setpalette example */

#include <graphics.h>
#include <stdlib.h>
#include <stdio.h>
#include <conio.h>

int main(void)
{
    /* request autodetection */
    int gdriver = DETECT, gmode, errorcode;
    int color, maxcolor, ht;
    int y = 10;
    char msg[80];

    /* initialize graphics and local variables */
    initgraph(&gdriver, &gmode, "");

    /* read result of initialization */
    errorcode = graphresult();
    if (errorcode != grOk) {  /* an error occurred */
        printf("Graphics error: %s\n", grapherrormsg(errorcode));
        printf("Press any key to halt:");
        getch();
        exit(1);               /* terminate with an error code */
    }

    maxcolor = getmaxcolor();
    ht = 2 * textheight("W");

    /* display the default colors */
    for (color=1; color<=maxcolor; color++) {
        setcolor(color);
        sprintf(msg, "Color: %d", color);
        outtextxy(1, y, msg);
        y += ht;
    }

    /* wait for a key */
    getch();

    /* black out the colors one by one */
    for (color=1; color<=maxcolor; color++) {
        setpalette(color, BLACK);
        getch();
    }

    /* clean up */
    closegraph();
    return 0;
}
setrgbpalette

Syntax

```
#include <graphics.h>
void setrgbpalette(int colornum, int red, int green, int blue);
```

Description

setrgbpalette can be used with the IBM 8514 and VGA drivers. colornum defines the palette entry to be loaded, while red, green, and blue define the component colors of the palette entry. For the IBM 8514 display (and the VGA in 256K color mode), colornum is in the range 0 to 255. For the remaining modes of the VGA, colornum is in the range 0 to 15. Only the lower byte of red, green, or blue is used, and out of each byte, only the 6 most significant bits are loaded in the palette. For compatibility with other IBM graphics adapters, the BGI driver defines the first 16 palette entries of the IBM 8514 to the default colors of the EGA/VGA. These values can be used as is, or they can be changed using setrgbpalette.

Return Value

None.

Windows Notes

The winbgim version allows the colornum to be a palette index from 0 to 15. The call
```
setrgbpalette(colornum, r, g, b);
```
is similar to
```
setpalette(colornum, COLOR(r,g,b) );
```
The difference is that setrgbpalette will use only the six most significant bits of the least significant byte of r, g and b. However, COLOR(r,g,b) uses the entire least significant byte of r, g and b. color argument to be an ordinary BGI color (from 0 to 15) or an RGB color.
In the windows version, changing the palette effects only future drawing. Currently drawn pixels do not change their color when the palette changes (no "palette animation").

See also

setpalette

Example

```
/* setrgbpalette example */

#include <graphics.h>
#include <stdlib.h>
#include <stdio.h>
#include <conio.h>

int main(void)
{
    /* select driver and mode that supports use of setrgbpalette */
    int gdriver = VGA, gmode = VGAHI, errorcode;
    struct palettetype pal;
    int i, ht, y, xmax;

    /* initialize graphics and local variables */
    initgraph(&gdriver, &gmode, "");
```
/* read result of initialization */
errorcode = graphresult();
if (errorcode != grOk) { /* an error occurred */
    printf("Graphics error: %s\n", grapherrormsg(errorcode));
    printf("Press any key to halt:\n");
    getch();
    exit(1);               /* terminate with an error code */
}

/* grab a copy of the palette */
getpalette(&pal);

/* create gray scale */
for (i=0; i<pal.size; i++)
    setrgbpalette(pal.colors[i], i*4, i*4, i*4);

/* display the gray scale */
ht = getmaxy() / 16;
xmax = getmaxx();
y = 0;
for (i=0; i<pal.size; i++) {
    setfillstyle(SOLID_FILL, i);
    bar(0, y, xmax, y+ht);
    y += ht;
}

/* clean up */
getch();
closegraph();
return 0;

setrgbpalette

**Syntax**
```
#include <graphics.h>
void setrgbpalette(int colornum, int red, int green, int blue);
```

**Description**
setrgbpalette can be used with the IBM 8514 and VGA drivers.
colornum defines the palette entry to be loaded, while red, green, and blue define the
component colors of the palette entry.
For the IBM 8514 display (and the VGA in 256K color mode), colornum is in the
range 0 to 255. For the remaining modes of the VGA, colornum is in the range 0 to 15.
Only the lower byte of red, green, or blue is used, and out of each byte, only the 6
most significant bits are loaded in the palette.
For compatibility with other IBM graphics adapters, the BGI driver defines the first 16
palette entries of the IBM 8514 to the default colors of the EGA/VGA. These values
can be used as is, or they can be changed using setrgbpalette.

**Return Value**
None.
Windows Notes

The `winbgim` version allows the colornum to be a palette index from 0 to 15. The call

```c
setrgbpalette(colornum, r, g, b);
```

is similar to

```c
setpalette(colornum, COLOR(r,g,b) );
```

The difference is that `setrgbpalette` will use only the six most significant bits of the least significant byte of r, g and b. However, `COLOR(r,g,b)` uses the entire least significant byte of r, g and b. color argument to be an ordinary BGI color (from 0 to 15) or an RGB color.

In the windows version, changing the palette effects only future drawing. Currently drawn pixels do not change their color when the palette changes (no "palette animation").

See also

`setpalette`

Example

```c
/* setrgbpalette example */

#include <graphics.h>
#include <stdlib.h>
#include <stdio.h>
#include <conio.h>

int main(void)
{
    /* select driver and mode that supports use of setrgbpalette */
    int gdriver = VGA, gmode = VGAHI, errorcode;
    struct palettetype pal;
    int i, ht, y, xmax;

    /* initialize graphics and local variables */
    initgraph(&gdriver, &gmode, "");

    /* read result of initialization */
    errorcode = graphresult();
    if (errorcode != grOk) {  /* an error occurred */
        printf("Graphics error: %s\n", grapherrormsg(errorcode));
        printf("Press any key to halt:");
        getch();
        exit(1);               /* terminate with an error code */
    }

    /* grab a copy of the palette */
    getpalette(&pal);

    /* create gray scale */
    for (i=0; i<pal.size; i++)
        setrgbpalette(pal.colors[i], i*4, i*4, i*4);

    /* display the gray scale */
    ht = getmaxy() / 16;
    xmax = getmaxx();
    y = 0;
    for (i=0; i<pal.size; i++) {
        setfillstyle(SOLID_FILL, i);
        ```
settextstyle

Syntax
#include <graphics.h>
void settextstyle(int font, int direction, int charsize);

Description
settextstyle sets the text font, the direction in which text is displayed, and the size of the characters. A call to settextstyle affects all text output by outtext and outtextxy.
The parameters font, direction, and charsize passed to settextstyle are described in the following:
font: One 8x8 bit-mapped font and several "stroked" fonts are available. The 8x8 bit-mapped font is the default. The enumeration font_names, which is defined in graphics.h, provides names for these different font settings:

<table>
<thead>
<tr>
<th>Name</th>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>DEFAULT_FONT</td>
<td>0</td>
<td>8x8 bit-mapped font</td>
</tr>
<tr>
<td>TRIPLEX_FONT</td>
<td>1</td>
<td>Stroked triplex font</td>
</tr>
<tr>
<td>SMALL_FONT</td>
<td>2</td>
<td>Stroked small font</td>
</tr>
<tr>
<td>SANS_SERIF_FONT</td>
<td>3</td>
<td>Stroked sans-serif font</td>
</tr>
<tr>
<td>GOTHIC_FONT</td>
<td>4</td>
<td>Stroked gothic font</td>
</tr>
<tr>
<td>SCRIPT_FONT</td>
<td>5</td>
<td>Stroked script font</td>
</tr>
<tr>
<td>SIMPLEX_FONT</td>
<td>6</td>
<td>Stroked triplex script font</td>
</tr>
<tr>
<td>TRIPLEX_SCR_FONT</td>
<td>7</td>
<td>Stroked triplex script font</td>
</tr>
<tr>
<td>COMPLEX_FONT</td>
<td>8</td>
<td>Stroked complex font</td>
</tr>
<tr>
<td>EUROPEAN_FONT</td>
<td>9</td>
<td>Stroked European font</td>
</tr>
<tr>
<td>BOLD_FONT</td>
<td>10</td>
<td>Stroked bold font</td>
</tr>
</tbody>
</table>

The default bit-mapped font is built into the graphics system. Stroked fonts are stored in *.CHR disk files, and only one at a time is kept in memory. Therefore, when you select a stroked font (different from the last selected stroked font), the corresponding *.CHR file must be loaded from disk.
To avoid this loading when several stroked fonts are used, you can link font files into your program. Do this by converting them into object files with the BGIOBJ utility, then registering them through registerbgifont.
direction: Font directions supported are horizontal text (left to right) and vertical text (rotated 90 degrees counterclockwise). The default direction is HORIZ_DIR. The size of each character can be magnified using the charsize factor. If charsize is nonzero, it can affect bit-mapped or stroked characters. A charsize value of 0 can be used only with stroked fonts.
- If `charsize` equals 1, `outtext` and `outtextxy` displays characters from the 8x8 bit-mapped font in an 8x8 pixel rectangle onscreen.
- If `charsize` equals 2, these output functions display characters from the 8x8 bit-mapped font in a 16*16 pixel rectangle, and so on (up to a limit of ten times the normal size).
- When `charsize` equals 0, the output functions `outtext` and `outtextxy` magnify the stroked font text using either the default character magnification factor (4) or the user-defined character size given by `setusercharsize`.

Always use `textheight` and `textwidth` to determine the actual dimensions of the text.

**Return Value**

**See also**

`gettextsettings`
`graphresults`
`installuserfont`
`settextjustify`
`setusercharsize`
`textheight`
`textwidth`

**Example**

```c
/* settextstyle example */

#include <graphics.h>
#include <stdlib.h>
#include <stdio.h>
#include <conio.h>

/* the names of the text styles supported */
char *fname[] = { "DEFAULT font", "TRIPLEX font",
    "SMALL font", "SANS SERIF font",
    "GOTHIC font", "SCRIPT font",
    "SIMPLEX font", "TRIPLEX SCRIPT font",
    "COMPLEX font", "EUROPEAN font",
    "BOLD font"};

int main(void)
{
    /* request autodetection */
    int gdriver = DETECT, gmode, errorcode;
    int style, midx, midy;
    int size = 1;

    /* initialize graphics and local variables */
    initgraph(&gdriver, &gmode, "");

    /* read result of initialization */
    errorcode = graphresult();
    if (errorcode != grOk) { /* an error occurred */
        printf("Graphics error: %s\n", grapherrormsg(errorcode));
        printf("Press any key to halt:"); 
        getch();
        exit(1); /* terminate with an error code */
    }

    midx = getmaxx() / 2;
```
midy = getmaxy() / 2;
settextjustify(CENTER_TEXT, CENTER_TEXT);

/* loop through the available text styles */
for (style=DEFAULT_FONT; style<=BOLD_FONT; style++) {
    cleardevice();
    if (style == TRIPLEX_FONT)
        size = 4;
    /* select the text style */
    settextstyle(style, HORIZ_DIR, size);

    /* output a message */
    outtextxy(midx, midy, fname[style]);
    getch();
}
/* clean up */
closegraph();
return 0;
}

setusercharsize

Syntax
#include <graphics.h>
void setusercharsize(int multx, int divx, int multy, int divy);

Description
setusercharsize gives you finer control over the size of text from stroked fonts used
with graphics functions. The values set by setusercharsize are active only if charsize
equals 0, as set by a previous call to settextstyle.

With setusercharsize, you specify factors by which the width and height are scaled.
The default width is scaled by multx : divx, and the default height is scaled by multy :
divy. For example, to make text twice as wide and 50% taller than the default, set
multx = 2;  divx = 1;
multy = 3;  divy = 2;

Return Value
None.

See also
gettextsettings
graphresult
settextstyle

Example
/* setusercharsize example */
#include <graphics.h>
#include <stdlib.h>
#include <stdio.h>
#include <conio.h>
int main(void) {
    /* request autodetection */
    int gdriver = DETECT, gmode, errorcode;
    /* initialize graphics and local variables */
    initgraph(&gdriver, &gmode, "");
/* read result of initialization */
errorcode = graphresult();
if (errorcode != grOk) {
    /* an error occurred */
    printf("Graphics error: %s\n", grapherrormsg(errorcode));
    printf("Press any key to halt: ");
    getch();
    exit(1);
    /* terminate with an error code */
}
/* select a text style */
settextstyle(TRIPLEX_FONT, HORIZ_DIR, 4);
/* move to the text starting position */
moveto(0, getmaxy() / 2);
/* output some normal text */
outtext("Norm ");
/* make the text 1/3 the normal width */
setusercharsize(1, 3, 1, 1); outtext("Short ");
/* make the text 3 times normal width */
setusercharsize(3, 1, 1, 1); outtext("Wide ");
/* clean up */
getch();
closegraph();
return 0;
}

setviewport

**Syntax**

```c
#include <graphics.h>
void setviewport(int left, int top, int right, int bottom, int clip);
```

**Description**

`setviewport` establishes a new viewport for graphics output. The viewport corners are given in absolute screen coordinates by (left,top) and (right,bottom). The current position (CP) is moved to (0,0) in the new window. The parameter `clip` determines whether drawings are clipped (truncated) at the current viewport boundaries. If `clip` is nonzero, all drawings will be clipped to the current viewport.

**Return Value**

If invalid input is passed to `setviewport`, `graphresult` returns -11, and the current view settings remain unchanged.

**See also**

`clearviewport`
`getviewsettings`
`graphresult`

**Example**

```c
/* setviewport example */

#include <graphics.h>
```

```
#include <stdlib.h>
#include <stdio.h>
#include <conio.h>

#define CLIP_ON 1            /* activates clipping in viewport */

int main(void)
{
    /* request autodetection */
    int gdriver = DETECT, gmode, errorcode;

    /* initialize graphics and local variables */
    initgraph(&gdriver, &gmode, "");

    /* read result of initialization */
    errorcode = graphresult();
    if (errorcode != grOk)    /* an error occurred */
    {
        printf("Graphics error: %s\n", grapherrormsg(errorcode));

        printf("Press any key to halt:");
        getch();
        exit(1);               /* terminate with an error code */
    }

    setcolor(getmaxcolor());

    /* message in default full-screen viewport */
    outtextxy(0, 0, "* <-- (0, 0) in default viewport");

    /* create a smaller viewport */
    setviewport(50, 50, getmaxx()-50, getmaxy()-50, CLIP_ON);

    /* display some text */
    outtextxy(0, 0, "* <-- (0, 0) in smaller viewport");

    /* clean up */
    getch();
    closegraph();
    return 0;
}
```

**setvisualpage**

---

**Syntax**
```
#include <graphics.h>
void setvisualpage(int page);
```

**Description**
setvisualpage makes page the visual graphics page.

**Return Value**
None.

**See also**
graphresult
**setactivepage**

**Example**

```c
/* setactivepage example */

#include <graphics.h>
#include <stdlib.h>
#include <stdio.h>
#include <conio.h>

int main(void)
{
    /* select driver and mode that supports multiple pages */
    int gdriver = EGA, gmode = EGAHI, errorcode;
    int x, y, ht;

    /* initialize graphics and local variables */
    initgraph(&gdriver, &gmode, "");

    /* read result of initialization */
    errorcode = graphresult();
    if (errorcode != grOk)    /* an error occurred */
        {
            printf("Graphics error: %s\n", grapherrormsg(errorcode));
            printf("Press any key to halt:");
            getch();
            exit(1);       /* terminate with an error code */
        }

    x = getmaxx() / 2;
    y = getmaxy() / 2;
    ht = textheight("W");

    /* select the off screen page for drawing */
    setactivepage(1);

    /* draw a line on page #1 */
    line(0, 0, getmaxx(), getmaxy());

    /* output a message on page #1 */
    settextjustify(CENTER_TEXT, CENTER_TEXT);
    outtextxy(x, y, "This is page #1:");
    outtextxy(x, y+ht, "Press any key to halt:");

    /* select drawing to page #0 */
    setactivepage(0);

    /* output a message on page #0 */
    outtextxy(x, y, "This is page #0.");
    outtextxy(x, y+ht, "Press any key to view page #1:");
    getch();

    /* select page #1 as the visible page */
    setvisualpage(1);

    /* clean up */
    getch();
    closegraph();
    return 0;
}
```
setwritemodem

Syntax
#include <graphics.h>
void setwritemodem(int mode);

Description
The following constants are defined:
COPY_PUT = 0      /* MOV */
XOR_PUT = 1      /* XOR */

Each constant corresponds to a binary operation between each byte in the line and the
corresponding bytes onscreen. COPY_PUT uses the assembly language MOV
instruction, overwriting with the line whatever is on the screen. XOR_PUT uses the
XOR command to combine the line with the screen. Two successive XOR commands
will erase the line and restore the screen to its original appearance.

setwritemodem currently works only with line, linerel, lineto, rectangle, and drawpoly.

Return Value
None.

See also
lineto
putimage

Example
/* setwritemodem example */

#include <graphics.h>
#include <stdlib.h>
#include <stdio.h>
#include <conio.h>

int main()
{
    /* request autodetection */
    int gd, gm, errorcode;
    int xmax, ymax;

    /* initialize graphics and local variables */
    initgraph(&gd, &gm, "");

    /* read result of initialization */
    errorcode = graphresult();
    if (errorcode != grOk)    /* an error occurred */
    {
        printf("Graphics error: %s\n", grapherrormsg(errorcode));

        printf("Press any key to halt: ");
        getch();
        exit(1);               /* terminate with an error code */
    }

    xmax = getmaxx();
    ymax = getmaxy();

    /* select XOR drawing mode */
    setwritemodem(XOR_PUT);

    /* draw a line */
    line(0, 0, xmax, ymax);
getch();
/* erase the line by drawing over it */
line(0, 0, xmax, ymax);
getch();

/* select overwrite drawing mode */
setwritemode(COPY_PUT);

/* draw a line */
line(0, 0, xmax, ymax);

/* clean up */
getch();
closegraph();
return 0;
}

textheight

Syntax
#include <graphics.h>
int textheight(char *textstring);

Description
The graphics function textheight takes the current font size and multiplication factor,
and determines the height of textstring in pixels. This function is useful for adjusting
the spacing between lines, computing viewport heights, sizing a title to make it fit on a
graph or in a box, and so on.
For example, with the 8x8 bit-mapped font and a multiplication factor of 1 (set by
settextstyle), the string BorlandC++ is 8 pixels high.
Use textheight to compute the height of strings, instead of doing the computations
manually. By using this function, no source code modifications have to be made when
different fonts are selected.

Return Value
textheight returns the text height in pixels.

See also
gettextsettings
outtext
outtextxy
settextstyle
textwidth

Example
/* textheight example */
#include <graphics.h>
#include <stdlib.h>
#include <stdio.h>
#include <conio.h>
int main(void) {
    /* request autodetection */
int gdriver = DETECT, gmode, errorcode;
int y = 0; int i; char msg[80];
/* initialize graphics and local variables */
initgraph(&gdriver, &gmode, "");
/* read result of initialization */
errorcode = graphresult();
if (errorcode != grOk)
{
    /* an error occurred */
    printf("Graphics error: %s\n", grapherrormsg(errorcode));
    printf("Press any key to halt.");
    getch(); exit(1);
    /* terminate with an error code */
}
/* draw some text on the screen */
for (i=1; i<11; i++)
{
    /* select the text style, direction, and size */
    settextstyle(TRIPLEX_FONT, HORIZ_DIR, i);
    /* create a message string */
    sprintf(msg, "Size: %d", i);
    /* output the message */
    outtextxy(1, y, msg);
    /* advance to the next text line */
    y += textheight(msg); }
/* clean up */
getch();
closegraph();
return 0; }

**textwidth**

**Syntax**

```c
#include <graphics.h>
int textwidth(char *textstring);
```

**Description**

The graphics function textwidth takes the string length, current font size, and multiplication factor, and determines the width of textstring in pixels. This function is useful for computing viewport widths, sizing a title to make it fit on a graph or in a box, and so on. Use textwidth to compute the width of strings, instead of doing the computations manually. When you use this function, no source code modifications have to be made when different fonts are selected.

**Return Value**

textwidth returns the text width in pixels.

**See also**

ggettextsettings
Example

/* textwidth example */

#include <graphics.h>
#include <stdlib.h>
#include <stdio.h>
#include <conio.h>

int main(void)
{
/* request autodetection */
   int gdriver = DETECT, gmode, errorcode;
   int x = 0, y = 0;
   int i;
   char msg[80];

/* initialize graphics and local variables */
   initgraph(&gdriver, &gmode, "");

/* read result of initialization */
   errorcode = graphresult();
   if (errorcode != grOk) { /* an error occurred */
      printf("Graphics error: %s\n", grapherrormsg(errorcode));

      printf("Press any key to halt: ");
      getch();
      exit(1); /* terminate with an error code */
   }

   y = getmaxy() / 2;
   settextjustify(LEFT_TEXT, CENTER_TEXT);
   for (i = 1; i < 11; i++) {
      /* select the text style, direction, and size */
      settextstyle(TRIPLEX_FONT, HORIZ_DIR, i);

      /* create a message string */
      sprintf(msg, "Size: %d", i);

      /* output the message */
      outtextxy(x, y, msg);

      /* advance to the end of the text */
      x += textwidth(msg);
   }

   /* clean up */
   getch();
   closegraph();
   return 0;
}
Borland Graphics Interface (BGI)

The following functions compose the Borland Graphics Interface and are usually available for 16 bit DOS applications. Use them to create onscreen graphics with text. They are defined in graphics.h.

Using BGI with Windows

The BGI graphics functions may also be used with Windows programs created by the Borland 5.0 compiler or the free GNU C++ compiler. These extra functions are described in www.cs.colorado.edu/~main/bgi/doc/bgi.html. In this listing, the extra functions are indicated by WIN. Also, any of the functions that use colors can use RGB colors in addition to the 16-color BGI palette.