arc

Syntax

```
#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

ellipse

fillellipse

getarcoords

getaspectratio

pieslice

sector

Example

```
/* 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 */

      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 arc */
   arc(midx, midy, stangle, endangle, radius);

   /* clean up */
   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

Syntax

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

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

fillellipse

getaspectratio

sector

setaspectratio

Example

```
/* 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

```
#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

```
/* cleardevice 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;
   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_LBUTTONDBLCLK`

...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_MBUTTONDBLCLK`

...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_RBUTTONDBLCLK`

...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)
{
    const int LIMIT = 10; // Number of clicks to stop program.
    int maxx, maxy;  // Maximum x and y pixel coordinates
    int count = 0;   // Number of mouse clicks
    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 " << 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

cleardevice

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("W");

   /* 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 the specified number of milliseconds.

Return Value

None.

See also None.

Example

```
/* 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

```
#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:

graphics_drivers constant	Numeric value

DETECT	0 (requests autodetect)
CGA	1
MCGA	2
EGA	3
EGA64	4
EGAMONO	5
IBM8514	6
HERCMONO	7
ATT400	8
VGA	9
PC3270	10

*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.

Graphics Driver	**graphics_mode**	**Value**	**Columns x Rows**	**Palette**	**Pages**
CGA	CGAC0	0	320 x 200	C0	1
	CGAC1	1	320 x 200	C1	1
	CGAC2	2	320 x 200	C2	1
	CGAC3	3	320 x 200	C3	1
	CGAHI	4	640 x 200	2 color	1
MCGA	MCGAC0	0	320 x 200	C0	1
	MCGAC1	1	320 x 200	C1	1
	MCGAC2	2	320 x 200	C2	1
	MCGAC3	3	320 x 200	C3	1
	MCGAMED	4	640 x 200	2 color	1
	MCGAHI	5	640 x 480	2 color	1
EGA	EGALO	0	640 x 200	16 color	4
	EGAHI	1	640 x 350	16 color	2
EGA64	EGA64LO	0	640 x 200	16 color	1
	EGA64HI	1	640 x 350	4 color	1
EGA-MONO	EGAMONOHI	3	640 x 350	2 color	1 w/64K
	EGAMONOHI	3	640 x 350	2 color	2 w/256K
HERC	HERCMONOHI	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	?

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

```
#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

```
/* 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;
```

```
   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;
}
```

---

ellipse

Syntax

```
#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

```
/* 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

```
#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

```
#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

```
/* fillpoly example */

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

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

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

   /* 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, 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

```
#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

```
/* 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

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 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 active page.

See also

getvisualpage

setactivepage

---

getarccoords

Syntax

```
#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:

```
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

```
/* 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 sstr[80], estr[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)", arcinfo.xstart, arcinfo.ystart);

   sprintf(estr, "*- (%d, %d)", 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;
}
```

getaspectration

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

Syntax

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

Description

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

Return Value

getbkcolor returns the current background color.

Windows Notes

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 */
   itoa(bkcolor, bkname, 10);
   strcat(bkname, " is the current background color.");



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

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

---

getch

Syntax

```
#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:

```
#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

```
#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; break;
                case KEY_UP:      cout << "Up key."     << endl; break;
                case KEY_PGUP:    cout << "PgUp key."   << endl; break;
                case KEY_LEFT:    cout << "Left key."   << endl; break;
                case KEY_CENTER:  cout << "Center key." << endl; break;
                case KEY_RIGHT:   cout << "Right key."  << endl; break;
                case KEY_END:     cout << "End key."    << endl; break;
                case KEY_DOWN:    cout << "Down key."   << endl; break;
                case KEY_PGDN:    cout << "PgDn key."   << endl; break;
                case KEY_INSERT:  cout << "Insert key." << endl; break;
                case KEY_DELETE:  cout << "Delete key." << endl; break;
                case KEY_F1:      cout << "F1 key."     << endl; break;
                case KEY_F2:      cout << "F2 key."     << endl; break;
                case KEY_F3:      cout << "F3 key."     << endl; break;
                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

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

getbkcolor

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

```
/* 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

```
#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:

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

Return Value

None.

See also

getfillsettings

setfillpattern

Example

/* 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); getch(); /* get the current user-defined fill pattern */ getfillpattern(pattern); /* alter the pattern we grabbed */ pattern[4] -= 1; pattern[5] -= 3; pattern[6] += 3; pattern[7] -= 4; /* select our new pattern */ setfillpattern(pattern, getmaxcolor()); /* fill the screen with the new pattern */ bar(0, 0, maxx, maxy); /* clean up */ getch(); closegraph(); return 0; }

getfillsettings

Syntax

```
#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:

```
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:

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

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

```
/* 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;
}
```

getgraphmode

Syntax

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

Description

Your program must make a successful call to initgraph before calling getgraphmode.

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

```
/* 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 = 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;
}
```

---

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);
   }

   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

getlinesettings 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:

```
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:

Name	Value	Description
SOLID_LINE	0	Solid line
DOTTED_LINE	1	Dotted line
CENTER_LINE	2	Centered line
DASHED_LINE	3	Dashed line
USERBIT_LINE	4	User-defined line style

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

Name	Value	Description
NORM_WIDTH	1	1 pixel wide
THICK_WIDTH	3	3 pixels wide

Return Value

None.

See also

setlinestyle

Example

```
/* 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

The winbgim 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\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 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

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

```
#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

getmaxy

getx

Example

```
/* 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);


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