SDL_bgi 3.0.2 Quick Reference

Guido Gonzato, PhD

April 17, 2024

Contents

3   Non-Graphics   Functions   and Macros   28

4   SDL_bgi Additions   29

1 Introduction to SDL_bgi

SDL_bgi is a graphics library (GRAPHICS.H) for C, C++, WebAssembly, and Python.

In more detail: SDL_bgi is a multiplatform, fast, SDL2-based implementation of BGI written in C. BGI, the Borland Graphics Interface also known as GRAPHICS.H, was a graphics library provided by Borland Turbo C / Borland C++ compilers for MS-DOS (pre-Windows era!). BGI was very popular in the late eighties–early nineties of the last century, and it became the *de facto* standard for computer graphics. It is still widely used in colleges and universities in India.

SDL_bgi aims to provide a modern implementation of BGI and replace the old DOS-based compilers, that are tricky to run on modern operating systems. However, SDL_bgi is not just a BGI clone; it is a larger and more powerful superset. It provides many extensions for ARGB colours, multiple windows, bitmap and vector fonts, and mouse support. SDL_bgi implements nearly all extensions provided by another popular BGI implementation, WinBGIm, and adds several more.

SDL_bgi is one of the easiest libraries to do graphics programming. It is much simpler to use than plain SDL2, OpenGL and the like; obviously, it's also less complete. Teachers may find SDL_bgi a useful tool for introductory computer graphics courses.

This manual provides a quick reference to SDL_bgi functions. For more complete explanations and usage examples, please see the demo and test files in the source archive.

1.1 Sample Programs

This is a minimal program written in C that creates a window and draws 1000 random lines:

```c
#include <graphics.h>

int main (int argc, char *argv[])
{
  int i, gd = DETECT, gm;
  initgraph (&gd, &gm, ""); // default: 800 x 600
  setbkcolor (BLACK);
  cleardevice ();
  outtextxy (0, 0, "Drawing 1000 lines...");
  for (i = 0; i < 1000; i++) {
    setcolor (1 + random (MAXCOLORS));
    line (random(getmaxx()), random(getmaxy()),
    random (getmaxx()), random(getmaxy()) );
  }
  getch ();
  closegraph ();
  return 0;
}
```

The program includes the header file `graphics.h`, which in turn includes `SDL_bgi.h` that contains all necessary definitions. `initgraph()` creates a window; from this point on, graphics functions can be called. `closegraph()` closes the window. The program can also be turned to WebAssembly and run in a web browser; please find compilation details in file `using.md`.

The above program can be easily rewritten in Python:

```python
from sdl_bgi import *

gd, gm = DETECT, 0
initgraph (gd, gm, "")   # default: 800 x 600
setbkcolor (BLACK)
cleardevice ()
outtextxy (0, 0, "Drawing 1000 lines...")

for i in range (1000):
    setcolor (1 + random (MAXCOLORS))
    line (random (getmaxx ()), random (getmaxy ()),
          random (getmaxx ()), random (getmaxy ()))

getch ()
closegraph ()
```

The program imports definitions from the `sdl_bgi.py` module. As you can see, the syntax of Python functions closely resembles that of corresponding C functions. Some syntax differences exist, though; they are explained in Section 1.9.

1.2 The Graphics Environment

- graphics must be initialised using `initgraph()` or `initwindow()`; one or more windows can be created. Only one of them is active, i.e. being drawn on, at any given moment.

- within a window, pixel coordinates range from the upper left corner at (0, 0) to the lower right corner at (`getmaxx()`, `getmaxy()`).

- some graphic functions set the coordinates of the last drawing position, defined as CP (Current Pointer): `cleardevice()`, `clearviewport()`, `graphdefaults()`, `linerel()`, `lineto()`, `moverel()`, `moveto()`.

- at any given moment, a foreground, background and fill colour, line style, line thickness, and fill pattern are defined. These parameters are used by functions like `line()` or `floodfill()`.

- line drawing functions can perform binary operations between plotted pixels and the background (XOR, OR, AND, NOT); please see `setwritemode()`.

- a viewport (subwindow) may also be defined, with or without clipping.

- in addition to the visualised window, hidden ones ("pages") can be defined and drawn on; please see `setactivepage()` and `setvisualpage()`.

- BGI uses its own fonts; system fonts are not available.

1.3 Getting Help

The `graphics` man page is available for GNU/Linux, macOS, and MSYS2 platforms.

In Python, you can get help on any function using one of the following commands:

```
>>> from sdl_bgi import *
>>> help (initgraph)
Help on function initgraph in module sdl_bgi:

initgraph(graphdriver, graphmode, pathtodriver)
    Initializes the graphics system. Use initgraph (0, 0, "")
    for default settings.
>>> # or:
>>> print (initgraph.__doc__)

    Initializes the graphics system. Use initgraph (0, 0, "")
    for default settings.
>>>
```

1.4 Definitions

C programs must include definitions from header file SDL_bgi.h:

```c
#ifndef _SDL_BGI_H
#define _SDL_BGI_H

#ifndef __GRAPHICS_H
#define __GRAPHICS_H

// SDL2 stuff
#include <SDL2/SDL.h>
#include <SDL2/SDL_keycode.h>
#include <SDL2/SDL_mouse.h>
#include <stdio.h>    // for fprintf()
#include <stdlib.h>   // for exit(), calloc()
#include <math.h>     // for sin(), cos()
#include <string.h>   // for strlen(), memcpy()

#ifdef __EMSCRIPTEN__
#include <emscripten.h>
#endif

#define SDL_BGI_VERSION 3.0.2

enum { NOPE, YEAH } ;
#define BGI_WINTITLE_LEN 512 // more than enough

// number of concurrent windows that can be created

#define NUM_BGI_WIN 16

// everything gets drawn here

extern SDL_Window   *bgi_window;
extern SDL_Renderer *bgi_renderer;
extern SDL_Texture  *bgi_texture;
extern Uint32        PALETTE_SIZE;

// available visual pages

#define VPAGES 4

// BGI fonts

enum {
  DEFAULT_FONT,       // 8x8 bitmap
  TRIPLEX_FONT,       // trip.h
  SMALL_FONT,         // litt.h
  SANS_SERIF_FONT,    // sans.h
  GOTHIC_FONT,        // goth.h
  SCRIPT_FONT,        // scri.h
  SIMPLEX_FONT,       // simp.h
  TRIPLEX_SCR_FONT,   // tscr.h
  COMPLEX_FONT,       // lcom.h
  EUROPEAN_FONT,      // euro.h
  BOLD_FONT,          // bold.h
  LAST_SPEC_FONT
};

enum { HORIZ_DIR, VERT_DIR };

#define USER_CHAR_SIZE 0

enum {
  LEFT_TEXT, CENTER_TEXT, RIGHT_TEXT,
  BOTTOM_TEXT = 0, TOP_TEXT = 2
};
```

6

```
// BGI colours, including CGA and EGA palettes

enum {
  BLACK        = 0,                                  EGA_BLACK        = 0,
  BLUE         = 1,                                  EGA_BLUE         = 1,
  GREEN        = 2,   CGA_GREEN      = 2,   EGA_GREEN        = 2,
  CYAN         = 3,   CGA_CYAN       = 3,   EGA_CYAN         = 3,
  RED          = 4,   CGA_RED        = 4,   EGA_RED          = 4,
  MAGENTA      = 5,   CGA_MAGENTA    = 5,   EGA_MAGENTA      = 5,
  BROWN        = 6,   CGA_BROWN      = 6,   EGA_BROWN        = 6,
  LIGHTGRAY    = 7,   CGA_LIGHTGRAY  = 7,   EGA_LIGHTGRAY    = 7,
  DARKGRAY     = 8,                                  EGA_DARKGRAY     = 8,
  LIGHTBLUE    = 9,                                  EGA_LIGHTBLUE    = 9,
  LIGHTGREEN   = 10, CGA_LIGHTGREEN   = 10, EGA_LIGHTGREEN   = 10,
  LIGHTCYAN    = 11, CGA_LIGHTCYAN    = 11, EGA_LIGHTCYAN    = 11,
  LIGHTRED     = 12, CGA_LIGHTRED     = 12, EGA_LIGHTRED     = 12,
  LIGHTMAGENTA = 13, CGA_LIGHTMAGENTA = 13, EGA_LIGHTMAGENTA = 13,
  YELLOW       = 14, CGA_YELLOW       = 14, EGA_YELLOW       = 14,
  WHITE        = 15, CGA_WHITE        = 15, EGA_WHITE        = 15,
  MAXCOLORS    = 15
};

// ARGB colours, set by COLOR (), COLOR32 (), and RGBPALETTE ()

enum {
  ARGB_FG_COL   = 16,
  ARGB_BG_COL   = 17,
  ARGB_FILL_COL = 18,
  ARGB_TMP_COL  = 19,
  TMP_COLORS    = 4
};

// line style, thickness, and drawing mode

enum { NORM_WIDTH = 1, THICK_WIDTH = 3 };

enum { SOLID_LINE, DOTTED_LINE, CENTER_LINE, DASHED_LINE,
       USERBIT_LINE };

enum { COPY_PUT, XOR_PUT, OR_PUT, AND_PUT, NOT_PUT };

// fill styles

enum {
  EMPTY_FILL,        // fills area in background color
  SOLID_FILL,        // fills area in solid fill color
  LINE_FILL,         // --- fill
  LTSLASH_FILL,      // /// fill
  SLASH_FILL,        // /// fill with thick lines
  BKSLASH_FILL,      // \\\ fill with thick lines
  LTBKSLASH_FILL,    // \\\ fill
  HATCH_FILL,        // light hatch fill
  XHATCH_FILL,       // heavy cross hatch fill
  INTERLEAVE_FILL,   // interleaving line fill
  WIDE_DOT_FILL,     // Widely spaced dot fill
  CLOSE_DOT_FILL,    // Closely spaced dot fill
  USER_FILL          // user defined fill
};

// mouse events - compatible with WinBGIm

#define WM_MOUSEMOVE        SDL_MOUSEMOTION

#define WM_LBUTTONDOWN      SDL_BUTTON_LEFT
#define WM_LBUTTONUP        SDL_MOUSEBUTTONUP   + SDL_BUTTON_LEFT
#define WM_LBUTTONDBLCLK    SDL_MOUSEBUTTONDOWN + SDL_BUTTON_LEFT + 2

#define WM_MBUTTONDOWN      SDL_BUTTON_MIDDLE
```

```
#define WM_MBUTTONUP        SDL_MOUSEBUTTONUP   + 10*SDL_BUTTON_MIDDLE
#define WM_MBUTTONDBLCLK    SDL_MOUSEBUTTONDOWN + 10*SDL_BUTTON_MIDDLE+2

#define WM_RBUTTONDOWN      SDL_BUTTON_RIGHT
#define WM_RBUTTONUP        SDL_MOUSEBUTTONUP   + 20*SDL_BUTTON_RIGHT
#define WM_RBUTTONDBLCLK    SDL_MOUSEBUTTONDOWN + 20*SDL_BUTTON_RIGHT+2

#define WM_WHEEL            SDL_MOUSEWHEEL
#define WM_WHEELUP          SDL_BUTTON_RIGHT + 1
#define WM_WHEELDOWN        SDL_BUTTON_RIGHT + 2

// keys
#define KEY_HOME            SDLK_HOME
#define KEY_LEFT            SDLK_LEFT
#define KEY_UP              SDLK_UP
#define KEY_RIGHT           SDLK_RIGHT
#define KEY_DOWN            SDLK_DOWN
#define KEY_PGUP            SDLK_PAGEUP
#define KEY_PGDN            SDLK_PAGEDOWN
#define KEY_END             SDLK_END
#define KEY_INSERT          SDLK_INSERT
#define KEY_DELETE          SDLK_DELETE
#define KEY_F1              SDLK_F1
#define KEY_F2              SDLK_F2
#define KEY_F3              SDLK_F3
#define KEY_F4              SDLK_F4
#define KEY_F5              SDLK_F5
#define KEY_F6              SDLK_F6
#define KEY_F7              SDLK_F7
#define KEY_F8              SDLK_F8
#define KEY_F9              SDLK_F9
#define KEY_F10             SDLK_F10
#define KEY_F11             SDLK_F11
#define KEY_F12             SDLK_F12
#define KEY_CAPSLOCK        SDLK_CAPSLOCK
#define KEY_LEFT_CTRL       SDLK_LCTRL
#define KEY_RIGHT_CTRL      SDLK_RCTRL
#define KEY_LEFT_SHIFT      SDLK_LSHIFT
#define KEY_RIGHT_SHIFT     SDLK_RSHIFT
#define KEY_LEFT_ALT        SDLK_LALT
#define KEY_RIGHT_ALT       SDLK_RALT
#define KEY_ALT_GR          SDLK_MODE
#define KEY_LGUI            SDLK_LGUI
#define KEY_RGUI            SDLK_RGUI
#define KEY_MENU            SDLK_MENU
#define KEY_TAB             SDLK_TAB
#define KEY_BS              SDLK_BACKSPACE
#define KEY_RET             SDLK_RETURN
#define KEY_PAUSE           SDLK_PAUSE
#define KEY_SCR_LOCK        SDLK_SCROLLOCK
#define KEY_ESC             SDLK_ESCAPE

#define QUIT                SDL_QUIT

// graphics modes. Expanded from the original GRAPHICS.H

enum {
  DETECT = -1,
  SDL = 0,
  // all modes @ 320x200
  SDL_320x200 = 1, SDL_CGALO = 1, CGA = 1, CGAC0 = 1, CGAC1 = 1,
  CGAC2 = 1, CGAC3 = 1, MCGAC0 = 1, MCGAC1 = 1, MCGAC2 = 1,
  MCGAC3 = 1, ATT400C0 = 1, ATT400C1 = 1, ATT400C2 = 1, ATT400C3 = 1,
  // all modes @ 640x200
  SDL_640x200 = 2, SDL_CGAHI = 2, CGAHI = 2, MCGAMED = 2,
  EGALO = 2, EGA64LO = 2,
  // all modes @ 640x350
  SDL_640x350 = 3, SDL_EGA = 3, EGA = 3, EGAHI = 3,
  EGA64HI = 3, EGAMONOHI = 3,
```

```
  // all modes @ 640x480
  SDL_640x480 = 4, SDL_VGA = 4, VGA = 4, MCGAHI = 4, VGAHI = 4,
  IBM8514LO = 4,
  // all modes @ 720x348
  SDL_720x348 = 5, SDL_HERC = 5,
  // all modes @ 720x350
  SDL_720x350 = 6, SDL_PC3270 = 6, HERCMONOHI = 6,
  // all modes @ 800x600
  SDL_800x600 = 7, SDL_SVGALO = 7, SVGA = 7,
  // all modes @ 1024x768
  SDL_1024x768 = 8, SDL_SVGAMED1 = 8,
  // all modes @ 1152x900
  SDL_1152x900 = 9, SDL_SVGAMED2 = 9,
  // all modes @ 1280x1024
  SDL_1280x1024 = 10, SDL_SVGAHI = 10,
  // all modes @ 1366x768
  SDL_1366x768 = 11, SDL_WXGA = 11,
  // other
  SDL_USER = 12, SDL_FULLSCREEN = 13
};

// error messages
enum graphics_errors {
  grOk              =    0,
  grNoInitGraph     =   -1,
  grNotDetected     =   -2,
  grFileNotFound    =   -3,
  grInvalidDriver   =   -4,
  grNoLoadMem       =   -5,
  grNoScanMem       =   -6,
  grNoFloodMem      =   -7,
  grFontNotFound    =   -8,
  grNoFontMem       =   -9,
  grInvalidMode     =  -10,
  grError           =  -11,
  grIOerror         =  -12,
  grInvalidFont     =  -13,
  grInvalidFontNum  =  -14,
  grInvalidVersion  =  -18
};

// libXbgi compatibility

#define X11_CGALO        SDL_CGALO
#define X11_CGAHI        SDL_CGAHI
#define X11_EGA          SDL_EGA
#define X11             SDL
#define X11_VGA          SDL_VGA
#define X11_640x480      SDL_640x480
#define X11_HERC         SDL_HERC
#define X11_PC3270       SDL_PC3270
#define X11_SVGALO       SDL_SVGALO
#define X11_800x600      SDL_800x600
#define X11_SVGAMED1     SDL_SVGAMED1
#define X11_1024x768     SDL_1024x768
#define X11_SVGAMED2     SDL_SVGAMED2
#define X11_1152x900     SDL_1152x900
#define X11_SVGAHI       SDL_SVGAHI
#define X11_1280x1024    SDL_1280x1024
#define X11_WXGA         SDL_WXGA
#define X11_1366x768     SDL_1366x768
#define X11_USER         SDL_USER
#define X11_FULLSCREEN   SDL_FULLSCREEN

// structs

struct arccoordstype {
  int x;
  int y;
```

```c
  int xstart;
  int ystart;
  int xend;
  int yend;
};

struct date {
  int da_year;
  int da_day;
  int da_mon;
};

struct fillsettingstype {
  int pattern;
  int color;
};

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

struct palettetype {
  unsigned char size;
  Uint32 colors[MAXCOLORS + 1];
};
// SDL_bgi extension
struct rgbpalettetype {
  Uint32 size;
  Uint32 *colors;
};

struct textsettingstype {
  int font;
  int direction;
  int charsize;
  int horiz;
  int vert;
};

struct viewporttype {
  int left;
  int top;
  int right;
  int bottom;
  int clip;
};
```

Python bindings to SDL_bgi are implemented via ctypes. Programs must import definitions from the sdl_bgi.py module:

```python
from ctypes        import *
from sysconfig     import get_platform
from random        import randint

# try to load the SDL_bgi library

if get_platform () == 'win-amd64':        # Windows, IDLE
    sdlbgilib = ".\SDL_bgi.dll"
elif get_platform () == 'mingw_x86_64':   # Windows, MSYS2/ucrt64
    sdlbgilib = '/ucrt64/bin/SDL_bgi.dll'
else: # GNU/Linux, macOS
    sdlbgilib = 'libSDL_bgi.so'
```

```
try:
    sb = CDLL (sdlbgilib)
except:
    print ("The sdl_bgi.py module needs the SDL_bgi library binary;")
    print ("please get it from https://sdl-bgi.sourceforge.io")
    print ("Exiting.")
    quit ()

SDL_BGI_VERSION = "3.0.2"
NOPE = False
YEAH = True
BGI_WINTITLE_LEN = 512

# number of concurrent windows that can be created
NUM_BGI_WIN = 16

VPAGES = 4

# BGI fonts

DEFAULT_FONT      = 0    # 8x8 bitmap
TRIPLEX_FONT      = 1    # trip.h
SMALL_FONT        = 2    # litt.h
SANS_SERIF_FONT   = 3    # sans.h
GOTHIC_FONT       = 4    # goth.h
SCRIPT_FONT       = 5    # scri.h
SIMPLEX_FONT      = 6    # simp.h
TRIPLEX_SCR_FONT  = 7    # tscr.h
COMPLEX_FONT      = 8    # lcom.h
EUROPEAN_FONT     = 9    # euro.h
BOLD_FONT         = 10   # bold.h
LAST_SPEC_FONT    = 11

HORIZ_DIR = 0
VERT_DIR  = 1

USER_CHAR_SIZE = 0

LEFT_TEXT   = 0
CENTER_TEXT = 1
RIGHT_TEXT  = 2
BOTTOM_TEXT = 0
TOP_TEXT    = 2

# BGI colours

BLACK         = 0
BLUE          = 1
GREEN         = 2
CYAN          = 3
RED           = 4
MAGENTA       = 5
BROWN         = 6
LIGHTGRAY     = 7
DARKGRAY      = 8
LIGHTBLUE     = 9
LIGHTGREEN    = 10
LIGHTCYAN     = 11
LIGHTRED      = 12
LIGHTMAGENTA  = 13
YELLOW        = 14
WHITE         = 15
MAXCOLORS     = 15

# ARGB colours set by COLOR () COLOR32 () and RGBPALETTE ()

ARGB_FG_COL   = 16
ARGB_BG_COL   = 17
```

```
ARGB_FILL_COL  = 18
ARGB_TMP_COL   = 19
TMP_COLORS     = 4

# line style thickness and drawing mode

NORM_WIDTH  = 1
THICK_WIDTH = 3

SOLID_LINE    = 0
DOTTED_LINE   = 1
CENTER_LINE   = 2
DASHED_LINE   = 3
USERBIT_LINE  = 4

COPY_PUT = 0
XOR_PUT  = 1
OR_PUT   = 2
AND_PUT  = 3
NOT_PUT  = 4

# fill styles

EMPTY_FILL      = 0    # fills area in background color
SOLID_FILL      = 1    # fills area in solid fill color
LINE_FILL       = 2    # --- fill
LTSLASH_FILL    = 3    # /// fill
SLASH_FILL      = 4    # /// fill with thick lines
BKSLASH_FILL    = 5    # \\\ fill with thick lines
LTBKSLASH_FILL  = 6    # \\\ fill
HATCH_FILL      = 7    # light hatch fill
XHATCH_FILL     = 8    # heavy cross hatch 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

# window properties
SDL_WINDOW_FULLSCREEN = 0x00000001
SDL_BLENDMODE_NONE    = 0x00000000
SDL_BLENDMODE_BLEND   = 0x00000001

# mouse events - compatible with WinBGIm

SDL_BUTTON_LEFT        = 1
SDL_BUTTON_MIDDLE      = 2
SDL_BUTTON_RIGHT       = 3
SDL_MOUSEMOTION        = 0x400   # from SDL_events.h
SDL_MOUSEBUTTONDOWN    = 0x401
SDL_MOUSEBUTTONUP      = 0x402
SDL_MOUSEWHEEL         = 0x403

WM_MOUSEMOVE        = SDL_MOUSEMOTION
WM_LBUTTONDOWN      = SDL_BUTTON_LEFT # from SDL_mouse.h
WM_LBUTTONUP        = SDL_MOUSEBUTTONUP   + SDL_BUTTON_LEFT
WM_LBUTTONDBLCLK    = SDL_MOUSEBUTTONDOWN + SDL_BUTTON_LEFT + 2

WM_MBUTTONDOWN      = SDL_BUTTON_MIDDLE
WM_MBUTTONUP        = SDL_MOUSEBUTTONUP   + 10*SDL_BUTTON_MIDDLE
WM_MBUTTONDBLCLK    = SDL_MOUSEBUTTONDOWN + 10*SDL_BUTTON_MIDDLE + 2

WM_RBUTTONDOWN      = SDL_BUTTON_RIGHT
WM_RBUTTONUP        = SDL_MOUSEBUTTONUP   + 20*SDL_BUTTON_RIGHT
WM_RBUTTONDBLCLK    = SDL_MOUSEBUTTONDOWN + 20*SDL_BUTTON_RIGHT + 2

WM_WHEEL            = SDL_MOUSEWHEEL
WM_WHEELUP          = SDL_BUTTON_RIGHT + 1
WM_WHEELDOWN        = SDL_BUTTON_RIGHT + 2
```

```
# keys

SDL_SCANCODE_TO_KEYCODE = lambda key: key | (1 << 30)
# 30 = SDLK_SCANCODE_MASK

KEY_HOME          = SDL_SCANCODE_TO_KEYCODE (74) # from SDL_scancode.h
KEY_LEFT          = SDL_SCANCODE_TO_KEYCODE (80)
KEY_UP            = SDL_SCANCODE_TO_KEYCODE (82)
KEY_RIGHT         = SDL_SCANCODE_TO_KEYCODE (79)
KEY_DOWN          = SDL_SCANCODE_TO_KEYCODE (81)
KEY_PGUP          = SDL_SCANCODE_TO_KEYCODE (75)
KEY_PGDN          = SDL_SCANCODE_TO_KEYCODE (78)
KEY_END           = SDL_SCANCODE_TO_KEYCODE (77)
KEY_INSERT        = SDL_SCANCODE_TO_KEYCODE (73)
KEY_DELETE        = SDL_SCANCODE_TO_KEYCODE (76)
KEY_F1            = SDL_SCANCODE_TO_KEYCODE (58)
KEY_F2            = SDL_SCANCODE_TO_KEYCODE (59)
KEY_F3            = SDL_SCANCODE_TO_KEYCODE (60)
KEY_F4            = SDL_SCANCODE_TO_KEYCODE (61)
KEY_F5            = SDL_SCANCODE_TO_KEYCODE (62)
KEY_F6            = SDL_SCANCODE_TO_KEYCODE (63)
KEY_F7            = SDL_SCANCODE_TO_KEYCODE (64)
KEY_F8            = SDL_SCANCODE_TO_KEYCODE (65)
KEY_F9            = SDL_SCANCODE_TO_KEYCODE (66)
KEY_F10           = SDL_SCANCODE_TO_KEYCODE (67)
KEY_F11           = SDL_SCANCODE_TO_KEYCODE (68)
KEY_F12           = SDL_SCANCODE_TO_KEYCODE (69)
KEY_CAPSLOCK      = SDL_SCANCODE_TO_KEYCODE (57)
KEY_LEFT_CTRL     = SDL_SCANCODE_TO_KEYCODE (224)
KEY_RIGHT_CTRL    = SDL_SCANCODE_TO_KEYCODE (228)
KEY_LEFT_SHIFT    = SDL_SCANCODE_TO_KEYCODE (225)
KEY_RIGHT_SHIFT   = SDL_SCANCODE_TO_KEYCODE (229)
KEY_LEFT_ALT      = SDL_SCANCODE_TO_KEYCODE (226)
KEY_RIGHT_ALT     = SDL_SCANCODE_TO_KEYCODE (230)
KEY_ALT_GR        = SDL_SCANCODE_TO_KEYCODE (230)
KEY_LGUI          = SDL_SCANCODE_TO_KEYCODE (227)
KEY_RGUI          = SDL_SCANCODE_TO_KEYCODE (231)
KEY_MENU          = SDL_SCANCODE_TO_KEYCODE (118)
KEY_TAB           = SDL_SCANCODE_TO_KEYCODE (43)
KEY_BS            = SDL_SCANCODE_TO_KEYCODE (42)
KEY_RET           = SDL_SCANCODE_TO_KEYCODE (40)
KEY_PAUSE         = SDL_SCANCODE_TO_KEYCODE (72)
KEY_SCR_LOCK      = SDL_SCANCODE_TO_KEYCODE (71)
KEY_ESC           = SDL_SCANCODE_TO_KEYCODE (41)

SDL_KEYDOWN       = 0x300
SDL_QUIT          = 0x100
QUIT              = SDL_QUIT

# graphics modes. Expanded from the original GRAPHICS.H

DETECT            = -1
SDL               = 0
SDL_320x200       = 1
CGA               = 1
SDL_640x200       = 2
SDL_640x350       = 3
SDL_EGA           = 3
EGA               = 3
SDL_640x480       = 4
SDL_VGA           = 4
VGA               = 4
SDL_720x348       = 5
SDL_720x350       = 6
SDL_800x600       = 7
SVGA              = 7
SDL_1024x768      = 8
SDL_1152x900      = 9
SDL_1280x1024     = 10
```

```
SDL_1366x768  = 11
SDL_USER      = 12
SDL_FULLSCREEN = 13

# error messages

grOk                  =    0
grNoInitGraph         =   -1
grNotDetected         =   -2
grFileNotFound        =   -3
grInvalidDriver       =   -4
grNoLoadMem           =   -5
grNoScanMem           =   -6
grNoFloodMem          =   -7
grFontNotFound        =   -8
grNoFontMem           =   -9
grInvalidMode         =  -10
grError               =  -11
grIOerror             =  -12
grInvalidFont         =  -13
grInvalidFontNum      =  -14
grInvalidVersion      =  -18

# C structs, implemented as classes

class arccoordstype (Structure):
    _fields_ = [ ("x", c_int),
                 ("y", c_int),
                 ("xstart", c_int),
                 ("ystart", c_int),
                 ("xend", c_int),
                 ("yend", c_int) ]

class date (Structure):
    _fields_ = [ ("da_year", c_int),
                 ("da_day", c_int),
                 ("da_mon", c_int) ]

class fillsettingstype (Structure):
    _fields_ = [ ("pattern", c_int),
                 ("color", c_int) ]

class linesettingstype (Structure):
    _fields_ = [ ("linestyle", c_int),
                 ("upattern", c_uint),
                 ("thickness", c_uint) ]

class palettetype (Structure):
    _fields_ = [ ("size", c_ubyte),
                 ("colors", c_char * (MAXCOLORS + 1)) ]

# SDL_bgi extensions

class rgbpalettetype (Structure):
    _fields_ = [ ("size", c_uint),
                 ("colors", c_char_p) ]

class textsettingstype (Structure):
    _fields_ = [ ("font", c_int),
                 ("direction", c_int),
                 ("charsize", c_int),
                 ("horiz", c_int),
                 ("vert", c_int) ]

class viewporttype (Structure):
    _fields_ = [ ("left", c_int),
                 ("top", c_int),
                 ("right", c_int),
                 ("bottom", c_int),
```

```
            ("clip", c_int) ]
```

1.5 Environment Variables

These variables only apply to C and Python programs.

SDL_BGI_RES: when set to VGA, default resolution will be 640 × 480 instead of default 800 × 600. Please see initgraph() (page 22) for details.

SDL_BGI_RATE: when set to auto, automatic screen refresh will be performed. Please see initgraph() (page 22) for details.

SDL_BGI_PALETTE: when set to BGI, the first 16 colours will use the same RGB values as Turbo C 2.01. Please see initpalette() (page 33) for details.

WebAssembly programs can use "environment files", which are text files with the same name as the corresponding environment variable. These files contain a string containing the desired value. Please see examples in test/assets/.

1.6 Colours

SDL_bgi provides two palettes that can be used at the same time.

The default BGI palette includes 16 named colours (BLACK... WHITE); functions getbkcolor(), getcolor(), putpixel(), setbkcolor() setbkcolor(), setcolor(), setfillpattern(), setfillstyle() and setpalette() use this palette by default.

An extended ARGB palette of PALETTE_SIZE additional colours can be created and accessed using functions getrgbpalette(), setallpalette(), setbkrgbcolor(), setrbgcolor() and setrbgpalette(). These functions are functionally equivalent to their standard BGI counterparts. PALETTE_SIZE is 4096 by default; the palette can be resized using resizepalette().

Standard BGI functions can also use ARGB colours using COLOR(), COLOR32(), and RGBPALETTE() as colour parameter; see sample programs in directory test/.

At any given moment, either the BGI or the ARGB palette is being used. Functions IS_BGI_COLOR() and IS_RGB_COLOR() return 1 if the corresponding palette is being used. Functions with *rgb* in their name trigger ARGB palette mode.

Constants ARGB_FG_COL, ARGB_BG_COL, ARGB_FILL_COL, and ARGB_TMP_COL denote the foreground, background, fill, and temporary ARGB colours that can be set with functions COLOR(), COLOR32(), and RGBPALETTE().

1.7 Fonts

SDL_bgi provides an 8×8 bitmap font and vector fonts decoded from original CHR files; loading CHR fonts from disk is also possible. Please see settextstyle() for details.

CHR font support was added by Marco Diego Aurélio Mesquita.

1.8   Notes for SDL2 Programmers

The following variables are declared in SDL_bgi.h, and are accessible to the programmer (C/C++ only):

```
SDL_Window   *bgi_window;
SDL_Renderer *bgi_renderer;
SDL_Texture  *bgi_texture;
Uint32        PALETTE_SIZE;
```

so they can be used by native SDL2 functions. In fact, you can use BGI and native SDL2 functions together, as in the following code snippet:

```
SDL_Surface *bitmap;
SDL_Texture *texture;
...
bitmap = SDL_LoadBMP ("picture.bmp");
texture = SDL_CreateTextureFromSurface (bgi_renderer, bitmap);
SDL_RenderCopy (bgi_renderer, texture, NULL, NULL);
SDL_RenderPresent (bgi_renderer);
...
```

Please see test/loadimage.c for a complete example.

1.9   Notes for Python Programmers

Even though the syntax of C and Python versions of SDL_bgi is quite similar, ctypes introduces a few minor differences in some cases. Please consult howto_Python.md for further explanations.

2   Standard BGI Graphics Functions

The following are standard BGI functions, as implemented for example in Turbo C. They are all prototyped in SDL_bgi.h, and implemented in sdl_bgi.py.

Unless otherwise specified, graphics routines draw shapes using the current drawing colour, i.e. as specified by setcolor().

In most cases, Python functions use the same syntax and data types as their C counterparts. When syntax differs, a Python template is provided. Variables passed by reference are implemented using the byref() function, provided by ctypes.

> **C** void arc (int *x*, int *y*, int *stangle*, int *endangle*, int *radius*);
> **🐍** arc (*x*, *y*, *stangle*, *endangle*, *radius*)

Draws a circular arc centered at (*x*, *y*), with a radius given by *radius*, traveling from *stangle* to *endangle*. The angle for arc() is measured counterclockwise, with 0 degrees at 3 o' clock, 90 degrees at 12 o' clock, etc.

Note: The *linestyle* parameter does not affect arcs, circles, ellipses, or pieslices. Only the *thickness* parameter is used.

```
C void bar (int left, int top, int right, int bottom);
  bar (left, top, right, bottom)
```

Draws a filled-in rectangle (bar), using the current fill colour and fill pattern. The bar is not outlined; to draw an outlined two-dimensional bar, use `bar3d()` with *depth* equal to 0.

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

Draws a three-dimensional, filled-in rectangle (bar), using the current fill colour and fill pattern. The three-dimensional outline of the bar is drawn in the current line style and colour. The bar's depth, in pixels, is given by *depth*. If *topflag* is nonzero, a top is put on.

```
C void circle (int x, int y, int radius);
  circle (x, y, radius)
```

Draws a circle of the given *radius* at (*x*, *y*).

Note: The *linestyle* parameter does not affect arcs, circles, ellipses, or pieslices. Only the *thickness* parameter is used.

```
C void cleardevice (void);
  cleardevice ()
```

Clears the graphics screen, filling it with the current background colour. The CP is moved to (0, 0).

```
C void clearviewport (void);
  clearviewport ()
```

Clears the viewport, filling it with the current background colour. The CP is moved to (0, 0), relative to the viewport.

```
C void closegraph (void);
  closegraph ()
```

Closes the graphics system. In Emscripten, it closes the browser tab or window.

```
C void detectgraph (int *graphdriver, int *graphmode);
  graphdriver, graphmode = c_int (), c_int ()
  detectgraph (byref (graphdriver), byref (graphmode))
```

Detects the graphics driver and default graphics mode to use; SDL and SDL_FULLSCREEN, respectively.

C `void drawpoly (int numpoints, int *polypoints);`
🐍 `drawpoly (numpoints, polypoints);`

Draws a polygon of `numpoints` vertices. In C, `polypoints` is a pointer to a sequence of (2 * `numpoints`) integers; each pair gives the x and y coordinate of each vertex. The polygon is not closed automatically.

In Python, `polypoints` is a list.

C `void ellipse (int x, int y, int stangle, int endangle, int xradius,`
`int yradius);`
🐍 `ellipse (x, y, stangle, endangle, xradius, yradius)`

Draws an elliptical arc centered at (x, y), with axes given by `xradius` and `yradius`, traveling from `stangle` to `endangle`.

C `void fillellipse (int x, int y, int xradius, int yradius);`
🐍 `fillellipse (x, y, xradius, yradius)`

Draws an ellipse centered at (x, y), with axes given by `xradius` and `yradius`, and fills it using the current fill colour and fill pattern.

C `void fillpoly (int numpoints, int *polypoints);`
🐍 `fillpoly (numpoints, polypoints)`

Draws a polygon of `numpoints` vertices and fills it using the current fill colour.

C `void floodfill (int x, int y, int border);`
🐍 `floodfill (x, y, border)`

Fills an enclosed area, starting from point (x, y) bounded by the `border` colour. The area is filled using the current fill colour.

C `int getactivepage (void);`
🐍 `getactivepage ()`

Returns the active page number.

C `void getarccoords (struct arccoordstype *arccoords);`
🐍 `arccoords = arccoordstype ()`
🐍 `getarccoords (byref (arccoords))`

Gets the coordinates of the last call to `arc()`, filling the *arccoords* struct/class.

> **C**   void getaspectratio (int **xasp*, int **yasp*);
> **🐍**   *xasp*, *yasp* = c_int (), c_int ()
> **🐍**   getaspectratio (byref (*xasp*), byref (*yasp*))

Retrieves the current graphics mode's aspect ratio. *xasp* and *yasp* are always 10000 (i.e. pixels are square).

> **C**   int getbkcolor (void);
> **🐍**   getbkcolor ()

Returns the current background colour.

> **C**   int getcolor (void);
> **🐍**   getcolor ()

Returns the current drawing (foreground) colour in the default palette. If the foregroud colour was set by `COLOR()`, `getcolor()` returns -1.

> **C**   struct palettetype *getdefaultpalette (void);
> **🐍**   *palette* = palettetype ()
> **🐍**   getdefaultpalette (byref (*palette*))

Returns the palette definition struct/class.

> **C**   char *getdrivername (void);
> **🐍**   getdrivername ()

Returns a (pointer to a) string containing the name of the current graphics driver.

> **C**   void getfillpattern (char **pattern*);
> **🐍**   getfillpattern (*pattern*)

Copies the user-defined fill pattern, as set by `setfillpattern()`, into the 8-byte area pointed to by *pattern*. In Python, *pattern* is an 8-byte string.

> **C**   void getfillsettings (struct fillsettingstype **fillinfo*);
> **🐍**   *fillinfo* = fillsettingstype ()
> **🐍**   getfillsettings (byref (*fillinfo*))

Fills the `fillsettingstype` struct/class pointed to by *fillinfo* with information about the current fill pattern and fill colour.

C `int getgraphmode (void);`
🐍 `getgraphmode ()`

Returns the current graphics mode.

C `void getimage (int left, int top, int right, int bottom, void *bitmap);`
🐍 `getimage (left, top, right, bottom, *bitmap)`

Copies a bit image of the specified region into the memory pointed by *bitmap*, previously allocated with `malloc (imagesize ())` (C) or `create_string_buffer (imagesize ())` (Python).

C `void getlinesettings (struct linesettingstype *lineinfo);`
🐍 `lineinfo = linesettingstype ()`
🐍 `getlinesettings (byref (lineinfo))`

Fills the `linesettingstype` struct/class pointed by *lineinfo* with information about the current line style, pattern, and thickness.

C `int getmaxcolor (void);`
🐍 `getmaxcolor ()`

Returns the maximum colour value available. If the default BGI palette is being used, it returns `MAXCOLORS`; otherwise (ARGB palette), it returns `PALETTE_SIZE`.

C `int getmaxmode (void);`
🐍 `getmaxmode ()`

Returns the maximum mode number for the current driver. In `SDL_bgi`, the default is `SDL_FULLSCREEN`.

C `int getmaxx (void);`
🐍 `getmaxx ()`

Returns the maximum x screen coordinate.

C `int getmaxy (void);`
🐍 `getmaxy ()`

Returns the maximum y screen coordinate.

C `char* getmodename (int mode_number);`
🐍 `getmodename (mode_number)`

Returns a (pointer to a) string containing the name of the specified graphics mode.

C `void getmoderange (int graphdriver, int *lomode, int *himode);`
🐍 `lomode, himode = c_int (), c_int ()`
🐍 `getmoderange (0, byref (lomode), byref (himode))`

Returns the range of valid graphics modes. The *graphdriver* parameter is ignored.

C `void getpalette (struct palettetype *palette);`
🐍 `palette = palettetype ()`
🐍 `getpalette (palette)`

Fills the `palettetype` struct/class pointed by *palette* with information about the current palette's size and colours.

C `int getpalettesize (void);`
🐍 `getpalettesize ()`

Returns the size of the palette (`MAXCOLORS` + 1 in BGI mode, or `MAXRGBCOLORS` + 1 in ARGB mode).

C `int getpixel (int x, int y);`
🐍 `getpixel (x, y)`

Returns the colour of the pixel located at (*x*, *y*).

C `void gettextsettings (struct textsettingstype *texttypeinfo);`
🐍 `texttypeinfo = textsettingstype ()`
🐍 `gettextsettings (texttypeinfo)`

Fills the `textsettingstype` struct/class pointed to by *texttypeinfo* with information about the current text font, direction, size, and justification.

C `void getviewsettings (struct viewporttype *viewport);`
🐍 `viewport = viewporttype ()`
🐍 `getviewsettings (viewport)`

Fills the `viewporttype` struct/class pointed to by *viewport* with information about the current viewport.

C `int getvisualpage (void);`
P `getvisualpage ()`

Returns the visual page number.

C `int getx (void);`
P `getx ()`

Returns the current CP x coordinate, relative to the viewport.

C `int gety (void);`
P `gety ()`

Returns the current CP y coordinate, relative to the viewport.

C `void graphdefaults (void);`
P `graphdefaults ()`

Resets all graphics settings to their default values: sets the viewport to the entire screen, moves the CP to (0, 0), sets the default palette colours, the default drawing and background colour, the default fill style and pattern, the default text font and justification.

C `char* grapherrormsg (int errorcode);`
P `grapherrormsg (errorcode)`

Returns the (pointer to the) error message string associated with *errorcode*, returned by `graphresult()`.

C `int graphresult (void);`
P `graphresult ()`

Returns the error code for the last unsuccessful graphics operation and resets the error level to `grOk`.

C `unsigned imagesize (int left, int top, int right, int bottom);`
P `imagesize (left, top, right, bottom)`

Returns the size in bytes of the memory area required to store a bit image. This value must be allocated in a buffer before copying the image with `getimage()`.

C `void initgraph (int *graphdriver, int *graphmode, char *pathtodriver);`

🐍 `initgraph (graphdriver, graphmode, pathtodriver);`

Initializes the graphics system. In SDL‗bgi, you can use SDL as *graphdriver*, then choose a suitable graphics mode (listed in `graphics.h`) as *graphmode*. The *pathtodriver* argument is ignored. Typically, *graphdriver* is set to DETECT, and *graphmode* is not set; these values will set the default resolution (800 × 600) as SVGA. If the environment variable SDL‗BGI‗RES equals VGA or vga, then VGA resolution (640 × 480) will be forced.

You can also use NULL for **graphdriver* and **graphmode* to get the default resolution (800 × 600), or use `detectgraph()` (see above) to get fullscreen.

Unless a fullscreen window is already present, multiple windows can be created.

Using `initgraph()`, the default 16-colour palette uses the same ARGB values as the original palette in Turbo C. Using `initwindow()`, the default 16-colour palette uses different (possibly, better-looking) ARGB values.

After `initgraph()`, all graphics commands are immediately displayed, as in the original BGI. This could make drawing very slow; you may want to use `initwindow()` instead.

Alternatively, automatic screen refresh can be performed according to the value of the SDL‗BGI‗RATE environment variable. If the variable is set to `auto`, screen refresh is automatically performed every *msec* milliseconds; this value is the current screen refresh rate, as given by `SDL‗GetDisplayMode()`. If the variable is set to an integer value *msec*, automatic screen refresh will be performed every *msec* milliseconds.

Automatic screen refresh is much faster than the default behaviour; however, this feature may not work on some graphic cards.

C `int installuserdriver (char *name, int (*detect)(void));`

🐍 `installuserdriver ()`

Unimplemented; not used by SDL‗bgi.

C `int installuserfont (char *name);`

🐍 `installuserfont (name)`

Loads and installs a CHR font from disk. The function returns an integer to be used as first argument in `settextstyle()`.

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

🐍 `line (x1, y1, x2, y2)`

Draws a line between two specified points; the CP is not updated.

C `void linerel (int `*`dx`*`, int `*`dy`*`);`
P `linerel (`*`dx`*`, `*`dy`*`)`

Draws a line from the CP to a point that is (*dx*, *dy*) pixels from the CP. The CP is then advanced by (*dx*, *dy*).

C `void lineto (int `*`x`*`, int `*`y`*`);`
P `lineto (`*`x`*`, `*`y`*`)`

Draws a line from the CP to (*x*, *y*), then moves the CP to (*dx*, *dy*).

C `void moverel (int `*`dx`*`, int `*`dy`*`);`
P `moverel (`*`dx`*`, `*`dy`*`)`

Moves the CP by (*dx*, *dy*) pixels.

C `void moveto (int `*`x`*`, int `*`y`*`);`
P `moveto (`*`x`*`, `*`x`*`)`

Moves the CP to the position (*x*, *y*), relative to the viewport.

C `void outtext (char *`*`textstring`*`);`
P `outtext (`*`textstr`*`)`

Outputs *textstring* at the CP.

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

Outputs *textstring* at (*x*, *y*).

C `void pieslice (int `*`x`*`, int `*`y`*`, int `*`stangle`*`, int `*`endangle`*`, int `*`radius`*`);`
P `pieslice (`*`x`*`, `*`y`*`, `*`stangle`*`, `*`endangle`*`, `*`radius`*`)`

Draws and fills a pie slice centered at (*x*, *y*), with a radius given by *radius*, traveling from *stangle* to *endangle*. The pie slice is filled using the current fill colour.

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

Puts the bit image pointed to by *bitmap* onto the screen, with the upper left corner of the image placed at (*left*, *top*). *op* specifies the drawing mode (COPY_PUT, etc).

C `void putpixel (int x, int y, int color);`
P `putpixel (x, y, color)`

Plots a pixel at (x, y) in the colour defined by `color`.

C `void rectangle (int left, int top, int right, int bottom);`
P `rectangle (left, top, right, bottom)`

Draws a rectangle delimited by (`left`, `top`) and (`right`, `bottom`).

C `int registerbgidriver (void (*driver)(void));`
P `registerbgidriver ()`

Unimplemented; not used by SDL_bgi.

C `int registerbgifont (void (*font)(void));`
P `registerbgifont ()`

Unimplemented; not used by SDL_bgi.

C `void restorecrtmode (void);`
P `restorecrtmode ()`

Hides the graphics window and turns on text mode.

C `void sector (int x, int y, int stangle, int endangle, int xradius,`
`int yradius);`
P `sector (x, y, stangle, endangle, xradius, yradius)`

Draws and fills an elliptical pie slice centered at (x, y), horizontal and vertical radii given by `xradius` and `yradius`, traveling from `stangle` to `endangle`.

C `void setactivepage (int page);`
P `setactivepage (page)`

Makes `page` the active page for all subsequent graphics output. In multi-window mode, `setactivepage()` only works for the first window.

C `void setallpalette (struct palettetype *palette);`
P `palette = palettetype ()`
P `setallpalette (palette)`

Sets the current palette to the values stored in `palette`.

```
C  void setaspectratio (int xasp, int yasp);
   xasp, yasp = c_int (), c_int ()
   setaspectratio (xasp, yasp)
```

Changes the default aspect ratio of the graphics. In `SDL_bgi`, this function is not necessary since the pixels are square on modern hardware.

```
void setbkcolor (int color);
```

Sets the current background colour in the default BGI palette. If ARGB colours are not being used and `color > MAXCOLORS`, then sets `color % MAXCOLORS`.

```
C  void setcolor (int color);
   setcolor (color)
```

Sets the current drawing colour in the default BGI palette. If ARGB colours are not being used and `color > MAXCOLORS`, then sets `color % MAXCOLORS`.

```
C  void setfillpattern (char *upattern, int color);
   setfillpattern (upattern, color)
```

Sets a user-defined fill pattern. `upattern` is a pointer to an 8-byte area; in Python, `upattern` is an 8-byte string. Each bit set to 1 in each byte is plotted as a pixel.

```
C  void setfillstyle (int upattern, int color);
   setfillstyle (upattern, color)
```

Sets the fill pattern and fill colour. `upattern` is a pointer to an 8-byte area; in Python, `upattern` is an 8-byte string. Each bit set to 1 in each byte is plotted as a pixel.

```
C  unsigned setgraphbufsize (unsigned bufsize);
   setgraphbufsize (bufsize)
```

Unimplemented; not used by `SDL_bgi`.

```
C  void setgraphmode (int mode);
   setgraphmode (mode)
```

Shows the window that was hidden by `restorecrtmode()`. The `mode` parameter is ignored,

C `void setlinestyle (int linestyle, unsigned upattern,`
`int thickness);`

🐍 `setlinestyle (linestyle, upattern, thickness)`

Sets the line width and style for all lines drawn by `line()`, `lineto()`, `rectangle()`, `drawpoly()`, etc. The line style can be `SOLID_LINE`, `DOTTED_LINE`, `CENTER_LINE`, `DASHED_LINE`, or `USERBIT_LINE`; in the latter case, the user provides a 16-bit number (*upattern*) whose bits set to 1 will be plotted as pixels.

The line thickness can be set with `NORM_WIDTH` or `THICK_WIDTH`.

Arcs, circles, ellipses, and pieslices are not affected by *linestyle*, but are affected by *thickness*.

C `void setpalette (int colornum, int color);`

🐍 `setpalette (colornum, color)`

Changes the standard palette *colornum* to *color*, which can also be specified using the `COLOR()` function; it also changes the colour of currently drawn pixels.

C `void settextjustify (int horiz, int vert);`

🐍 `settextjustify (horiz, vert)`

Sets text justification. Text output will be justified around the CP horizontally and vertically; settings are `LEFT_TEXT`, `CENTER_TEXT`, `RIGHT_TEXT`, `BOTTOM_TEXT`, and `TOP_TEXT`.

C `void settextstyle (int font, int direction, int charsize);`

🐍 `settextstyle (font, direction, charsize)`

Sets the text font (8×8 bitmap font `DEFAULT_FONT` and vector fonts `TRIPLEX_FONT`, `SMALL_FONT`, `SANS_SERIF_FONT`, `GOTHIC_FONT`, `SCRIPT_FONT`, `SIMPLEX_FONT`, `TRIPLEX_SCR_FONT`), the text direction (`HORIZ_DIR`, `VERT_DIR`), and the size of the characters.

charsize is a scaling factor for the text (max. 10). If *charsize* is 0, the text will either use the default size, or it will be scaled by the values set with `setusercharsize()`.

Experimental feature: if a `CHR` font is available in the same directory as the running program, it will be loaded and used instead of its internal equivalent.

C `void setusercharsize (int multx, int divx, int multy, int divy);`

🐍 `setusercharsize (multx, divx, multy, divy)`

Lets the user change the character width and height. If a previous call to `settextstyle()` set *charsize* to 0, the default width is scaled by *multx/divx*, and the default height is scaled by *multy/divy*.

C `void setviewport (int` *left*`, int` *top*`, int` *right*`, int` *bottom*`,`
`int` *clip*`);`
🐍 `setviewport (`*left*`,` *top*`,` *right*`,` *bottom*`,` *clip*`)`

Sets the current viewport for graphics output. If `clip` is nonzero, all drawings will be clipped (truncated) to the current viewport.

C `void setvisualpage (int` *page*`);`
🐍 `setvisualpage (`*page*`)`

Sets the visual graphics page number to *page*. In "fast mode", the screen is not cleared.

C `void setwritemode (int` *mode*`);`
🐍 `setwritemode (`*mode*`)`

Sets the writing mode for line drawing. *mode* can be COPY_PUT, XOR_PUT, OR_PUT, AND_-PUT, and NOT_PUT.

C `int textheight (char *`*textstring*`);`
🐍 `textheight (`*textstring*`)`

Returns the height in pixels of *textstring*.

C `int textwidth (char *`*textstring*`);`
🐍 `textwidth (`*textstring*`)`

Returns the width in pixels of *textstring*.

3   Non-Graphics Functions and Macros

C `void delay (int` *millisec*`);`
🐍 `delay (`*millisec*`)`

Waits for *millisec* milliseconds. In "slow mode", a screen refresh is performed.
Note: in Turbo C, this function was provided by `DOS.H`.

C `int getch (void);`
🐍 `getch ()`

Waits for a key and returns its ASCII or key code (i.e. KEY_*). In "slow mode", a screen refresh is performed. If an SDL_QUIT event occurs, QUIT is returned.
Note: in Turbo C, this function was provided by `CONIO.H`.

```
C  int kbhit (void);
   kbhit ()
```

Returns 1 when a key is pressed, excluding special keys (Ctrl, Shift, etc.); in "slow mode", a screen refresh is performed. If an SDL_QUIT event occurs, QUIT is returned.

Note: in Turbo C, this function was provided by CONIO.H.

```
C  int lastkey (void);
   lastkey ()
```

Returns the last key that was detected by kbhit().

```
C  int random (int range) (macro)
   random (range)
```

Returns an integer random number between 0 and range - 1.

Note: in Turbo C, this function was provided by STDLIB.H.

4   SDL_bgi **Additions**

The following SDL_bgi extensions are mostly compatible with those made available by WinBGIm.

```
C  int ALPHA_VALUE (int color);
   ALPHA_VALUE (color)
```

Returns the alpha (transparency) component of an ARGB colour in the ARGB palette.

```
C  int BLUE_VALUE (int color);
   BLUE_VALUE (color)
```

Returns the blue component of an ARGB colour in the ARGB palette.

```
C  void closewindow (int id);
   closewindow (id)
```

Closes the window identified by id. If id is ALL_WINDOWS, closegraph() is called.

```
C  int COLOR (int r, int g, int b);
   COLOR (r, g, b)
```

Can be used as colour argument for getbkcolor(), getcolor(), putpixel(), set-bkcolor() setbkcolor(), setcolor(), setfillpattern(), setfillstyle(), and setpalette() to set a colour specifying its ARGB components. The colour index is ARGB_TMP_COL.

Functions ALPHA_VALUE(), BLUE_VALUE(), GREEN_VALUE(), and RED_VALUE() do not work on temporary colours.

C int COLOR32 (Uint32 *color*);
🐍 COLOR32 (*color*)

Can be used as colour argument for getbkcolor(), getcolor(), putpixel(), set-bkcolor(), setcolor(), setfillpattern(), setfillstyle(), and setpalette() to set a colour as ARGB integer. The colour index is ARGB_TMP_COL.

Functions ALPHA_VALUE(), BLUE_VALUE(), GREEN_VALUE(), and RED_VALUE() do not work on temporary colours.

C void clearmousebutton (int *kind*);
🐍 clearmousebutton (*kind*)

Clears mouse events; subsequent calls to ismouseclick() will return 0.

C char* colorname (int *color*);
🐍 colorname (*color*)

Returns a string containing the colour name ("BLACK", "BLUE", etc.)

C int colorRGB (int *r*, int *g*, int *b*) (macro)
🐍 colorRGB (*r*, *g*, *b*)

Can be used to compose a 32 bit colour with *r* *g* *b* components; the alpha value is set to 0xff. This macro/function is typically used to set values in memory buffers.

C void copysurface (SDL_Surface *surface*, int *x1*, int *y1*, int *x2*,
int *y2*);
🐍 copysurface (*surface*, *x1*, *y1*, *x2*, *y2*)

Copies *surface* to the rectangle defined by *x1*, *y1*, *x2*, *y2* and displays it immediately. If *x2* or *y2* equals 0, then the original surface size will be used.

C int doubleclick (void);
🐍 doubleclick ()

Returns 1 if the last mouse click was a double click.

C `int edelay (int `*`msec`*`);`

P `edelay (`*`msec`*`)`

Waits for `msec` milliseconds. In "slow mode", a screen refresh is performed. If an event occurs during the delay, this function returns 1, otherwise 0. Use `eventtype()` to get the last event.

C `int event (void);`

P `event ()`

Returns 1 if one of the following events has occurred: SDL_KEYDOWN, SDL_MOUSEBUT-TONDOWN, SDL_MOUSEWHEEL, or SDL_QUIT; 0 otherwise.

C `int eventtype (void);`

P `eventtype ()`

Returns the type of the last event. Reported events are SDL_KEYDOWN, SDL_MOUSEMOTION, SDL_MOUSEBUTTONDOWN, SDL_MOUSEBUTTONUP, SDL_MOUSEWHEEL, and SDL_QUIT.

C `void fputpixel (int `*`x`*`, int `*`y`*`);`

P `fputpixel (`*`x`*`, int `*`y`*`)`

Plots a point at (*x*, *y*) using the current drawing colour. This function is usually faster than `putpixel()`.

C `void getbuffer (Uint32 `*`*buffer`*`);`

P `getbuffer (buffer)`

Copies the contents of the active window to *`buffer`*, which must be a (`getmaxy()` + 1) × (`getmaxx()` + 1) array of *`Uint32`* (C) or a string buffer (Python). Copied elements are in ARGB format.

C `int getclick (void);`

P `getclick ()`

Waits for a mouse click and returns the button that was clicked.

C `int getcurrentwindow (void);`

P `getcurrentwindow ()`

Returns the *`id`* of the current window.

```
C  int getevent (void);
   getevent ()
```

Waits for one of the following events: SDL_KEYDOWN, SDL_MOUSEBUTTONDOWN, SDL_MOU-SEWHEEL, and SDL_QUIT. It returns the code of the key, or the mouse button, either WM_WHEELUP or WM_WHEELDOWN, or SDL_QUIT.

x

```
C  void getleftclick (void);
   getleftclick ()
```

Waits for the left mouse button to be clicked and released.

```
C  void getlinebuffer (int y, Uint32 *linebuffer);
   getlinebuffer (y, linebuffer)
```

Copies the *y*-th screen line to *linebuffer*, which must be a getmaxx()+1 array of *Uint32* in ARGB format (C), or a string buffer allocated with create_string_buffer (imagesize ()) (Python).

```
C  int getmaxheight (void);
   getmaxheight ()
```

Returns the maximum possible height for a new window (actual screen height in pixels). This function may be called before graphics initialisation.

```
C  int getmaxwidth (void);
   getmaxwidth ()
```

Returns the maximum possible width for a new window (actual screen width in pixels). This function may be called before graphics initialisation.

```
C  void getmiddleclick (void);
   getmiddleclick ()
```

Waits for the middle mouse button to be clicked and released.

```
C  void getmouseclick (int kind, int *x, int *y);
   x, y = c_int (), c_int ()
   getmouseclick (kind, byref (x), byref (y))
```

Sets the *x*, *y* coordinates of the last *kind* button click expected by ismouseclick().

```
C  void getrgbpalette (struct rgbpalettetype *palette int size);
🐍 palette = rgbpalettetype ()
🐍 getrgbpalette (palette, size)
```

Fills the rgbpalettetype struct/class pointed by *palette* with information about the current ARGB palette's size and colours.

```
C  void getrightclick (void);
🐍 getrightclick ()
```

Waits for the right mouse button to be clicked and released.

```
C  void getscreensize (int *width, int *height);
🐍 x, y = c_int (), c_int ()
🐍 getscreensize (byref (x), byref (y))
```

Reports the screen width and height in *width* and *height*, regardless of current window dimensions. This function may be called before graphics initialisation.
(macro)

```
C  void getwindowheight (void);
🐍 getwindowheight ()
```

Equivalent to getmaxy() (WinBGIm compatibility).
(macro)

```
C  void getwindowwidth (void);
🐍 getwindowwidth ()
```

Equivalent to getmaxx() (WinBGIm compatibility).

```
C  int GREEN_VALUE (int color);
🐍 GREEN_VALUE (color)
```

Returns the green component of an ARGB colour in the ARGB palette.

```
C  void initpalette (void);
🐍 initpalette ()
```

Initialises the BGI palette to the standard 16 colours. If the environment variable SDL_-BGI_PALETTE equals BGI, the first 16 colours will use the same RGB values as Turbo C 2.01; otherwise, a brighter palette will be used.

33

C `void initwindow (int width, int height);`

🐍 `initwindow (width, height)`

Initializes the graphics system, opening a $width \times height$ window. If either `width` or `height` is 0, then SDL_FULLSCREEN will be used. Unless a fullscreen window is already present, multiple windows can be created.

The user must update the screen as needed using `refresh()`, or use `sdlbgiauto()`.

This function is also overloaded as `initwindow (int width, int height, char *title)` to specify the window title without using `setwinoptions()`.

C `int IS_BGI_COLOR (int color);`

🐍 `IS_BGI_COLOR (color)`

Returns 1 if the *current* drawing colour is a standard BGI colour (that is, not ARGB). The `color` argument is actually redundant (WinBGIm compatibility).

C `int ismouseclick (int kind);`

🐍 `ismouseclick (kind)`

Returns 1 if the `kind` mouse button was clicked.

C `int IS_RGB_COLOR (int color);`

🐍 `IS_RGB_COLOR (color)`

Returns 1 if the *current* drawing colour is ARGB. The `color` argument is actually redundant (WinBGIm compatibility).

C `int kdelay (int msec);`

🐍 `edelay (msec)`

Waits for `msec` milliseconds. In "slow mode", a screen refresh is performed. If a key is pressed during the delay, this function returns 1, otherwise 0.

C `int mouseclick (void);`

🐍 `mouseclick ()`

Returns the code of the mouse button that is being clicked, or SDL_MOUSEMOTION if the mouse is being moved, or 0 if no mouse event is occurring.

C `int mousex (void);`

🐍 `mousex ()`

Returns the X coordinate of the last mouse click.

C `int mousey (void);`

P `mousey ()`

Returns the Y coordinate of the last mouse click.

C `void putbuffer (Uint32 *buffer);`

P `putbuffer (buffer)`

Copies `buffer` to the current window. `buffer` must be a (getmaxy() + 1) × (get-maxx() + 1) array of `Uint32` in ARGB format. This function is faster than direct pixel manipulation.

C `void putlinebuffer (int y, Uint32 *linebuffer);`

P `putlinebuffer (y, linebuffer)`

Copies `linebuffer` to the `y` coordinate in the current window. `linebuffer` must be a getmaxx()+1 array of `Uint32` in ARGB format. This function is faster than direct pixel manipulation.

C `void readimagefile (char *filename, int x1, int y1, int x2, int y2);`

P `readimagefile (filename, x1, y1, x2, y2)`

Reads a .bmp file and displays it immediately at $(x1, y1)$. If $(x2, y2)$ are not 0, the bitmap is stretched to fit the rectangle $x1, y1 - x2, y2$; otherwise, the bitmap is clipped as necessary.

C `int RED_VALUE (int color);`

P `RED_VALUE (color)`

Returns the red component of an ARGB colour in the ARGB palette.

C `void refresh (void);`

P `refresh ()`

Updates the screen contents, i.e. displays all graphics.

C `void resetwinoptions (int id, char * title, int x, int y);`

P `resetwinoptions (id, title, x, y)`

Resets the window title `title` and position to (x, y) of an existing window identified by `id`. `x` and `y` can be set to `SDL_WINDOWPOS_CENTERED` or `SDL_WINDOWPOS_UNDEFINED`. If either `x` or `y` is -1, position parameters are ignored.

```C
int resizepalette (Uint32 newsize);
```
```python
resizepalette (newsize)
```

Resizes the ARGB palette to *newsize*; returns 0 if successful, 1 otherwise. The initial size of the ARGB palette is 4096.

```C
int RGBPALETTE (int color);
```
```python
RGBPALETTE (color)
```

Can be used as a colour argument for getbkcolor(), getcolor(), putpixel(), set-bkcolor() setbkcolor(), setcolor(), setfillpattern(), setfillstyle(), and setpalette() to set the colour from the ARGB palette *color* entry. The colour index is ARGB_TMP_COL.

Functions ALPHA_VALUE(), BLUE_VALUE(), GREEN_VALUE(), and RED_VALUE() do not work on temporary colours.

```C
void sdlbgiauto (void);
```
```python
sdlbgiauto ()
```

Triggers "auto mode", i.e. refresh() is performed automatically. Caveat: it may not work on some graphics cards.

```C
void sdlbgifast (void);
```
```python
sdlbgifast ()
```

Triggers "fast mode", i.e. refresh() is needed to display graphics.

```C
void sdlbgislow (void);
```
```python
sdlbgislow ()
```

Triggers "slow mode", i.e. refresh() is not needed to display graphics.

```C
void setallrgbpalette (struct rgbpalettetype *palette);
```
```python
palette = rgbpalettetype ()
setallrgbpalette (byref (palette))
```

Sets the current ARGB palette to the values stored in *palette*.

```C
void setalpha (int col, Uint8 alpha);
```
```python
setalpha (col, alpha)
```

Sets alpha transparency for colour *col* to *alpha* (0–255); 0 means full transparecy, 255 full opacity. setalpha() works with colours in both palettes.

C void setbkrgbcolor (int *n*);
P setbkrgbcolor (*n*)

Sets the current background colour using the *n*-th colour entry in the ARGB palette.

C void setblendmode (int *blendmode*);
P setblendmode (*blendmode*)

Sets the blend mode to be used with screen refresh. *blendmode* can be SDL_BLENDMO-DE_NONE (default in "slow mode") or SDL_BLENDMODE_BLEND. The latter enables alpha blending.

C void setcurrentwindow (int *id*);
P setcurrentwindow (*id*)

Sets the current active window to *id*.

C void setrgbcolor (int *n*);
P setrgbcolor (*n*)

Sets the current drawing colour using the *n*-th colour entry in the ARGB palette.

C void setrgbpalette (int *n*, int *r*, int *g*, int *b*);
P setrgbpalette (*n*, *r*, *g*, *b*)

Sets the *n*-th entry in the ARGB palette specifying the *r*, *g*, and *b* components.

Using setrgbpalette() and setrgbcolor() is faster than setting colours with setco-lor() with a COLOR() argument. It does not change the colour of currently drawn pixels.

C void setwinoptions (char **title*, int *x*, int *y*, Uint32 *flags*);
P setwinoptions (*title*, *x*, *y*, *flags*)

Sets the window title *title*, the initial position to (*x*, *y*), and SDL2 flags OR'ed together. *x* and *y* can be set to SDL_WINDOWPOS_CENTERED or SDL_WINDOWPOS_UNDEFINED.

If *title* is an empty string, the window title is set to the default value SDL_bgi.

If either *x* or *y* is -1, the position parameters are ignored.

If *flags* is -1, the parameter is ignored; otherwise, only the values SDL_WINDOW_FULL-SCREEN, SDL_WINDOW_FULLSCREEN_DESKTOP, SDL_WINDOW_SHOWN, SDL_WINDOW_HIDDEN, SDL_WINDOW_BORDERLESS, and SDL_WINDOW_MINIMIZED can be applied.

```
C  void setwintitle (int id, char *title);
```
```
setwintitle (id, title)
```

Sets the title of the window identified by *id*.

```
C  void showerrorbox (const char *message);
```
```
showerrorbox (message)
```

Opens an error message box with the specified message. The message box waits for the user to click on the OK button.

```
C  void showinfobox (const char *message);
```
```
showinfobox (message)
```

Opens an information message box with the specified message. The message box waits for the user to click on the OK button.

```
C  int swapbuffers (void);
```
```
swapbuffers ()
```

Swaps the current active and the current visual graphics pages.

```
C  void writeimagefile (char *filename, int left, int top, int right,
int bottom);
```
```
writeimagefile (filename, left, top, right, bottom)
```

Writes a .bmp file from the screen rectangle defined by *left, top − right, bottom*.

```
C  int xkbhit (void);
```
```
xkbhit ()
```

Returns 1 when any key is pressed, including special keys (Ctrl, Shift, etc.); in "slow mode", a screen refresh is performed. If an SDL_QUIT event occurs, QUIT is returned.

This document is a free manual, released under the GNU Free Documentation License (FDL) v. 1.3 or later.

Brought to you by Guido Gonzato, PhD =8-)