simple_graphical_module.h

```
//
// simple_graphical_module.h
//
// Version 1.0, September-October 2003
//
// Miguel Oliveira e Silva (mos@det.ua.pt)
//
// Implemented with Qt (http://www.trolltech.com) version 2.3.x
//


#ifndef SIMPLE_GRAPHICAL_MODULE_H
#define SIMPLE_GRAPHICAL_MODULE_H


// Colors
#define min_color      0
#define aquamarine     0
#define black          1
#define blue           2
#define navy_blue      3
#define coral          4
#define cyan           5
#define brown          6
#define gold           7
#define green          8
#define dark_green     9
#define forest_green  10
#define gray          11
#define magenta       12
#define orange        13
#define orchid        14
#define pink          15
#define red           16
#define salmon        17
#define violet        18
#define wheat         19
#define white         20
#define yellow        21
#define max_color     21


// Font weight
#define min_font_weight 0
#define fw_light        1
#define fw_normal       2
#define fw_demi_bold    3
#define fw_bold         4
#define fw_black        5
#define max_font_weight 5


// Mouse buttons
#define mouse_left_button       0x1000
#define mouse_right_button      0x1001


// Special keys
#define key_escape      0x2000
#define key_backspace   0x2001
#define key_insert      0x2002
#define key_delete      0x2003
#define key_pause       0x2004
#define key_home        0x2005
#define key_end         0x2006
#define key_left        0x2007
#define key_up          0x2008
```

```
#define key_right       0x2009
#define key_down        0x200A
#define key_f1          0x200B
#define key_f2          0x200C
#define key_f3          0x200D
#define key_f4          0x200E
#define key_f5          0x200F
#define key_f6          0x2010
#define key_f7          0x2011
#define key_f8          0x2012
#define key_f9          0x2013
#define key_f10         0x2014
#define key_f11         0x2015
#define key_f12         0x2016
#define key_page_up     0x2017
#define key_page_down   0x2018


// Window closed event
#define exit_event      0x3000


typedef struct
{
  int x;
  int y;
} POINT_TYPE; // used for polygon drawing


typedef void *DRAWABLE_ID;


class SIMPLE_GRAPHICAL_OUTPUT
  // comment: singleton
{
public: // window construction and initialization


  SIMPLE_GRAPHICAL_OUTPUT(void);
    // comment: Default (sgout) constructor


  SIMPLE_GRAPHICAL_OUTPUT(int argc,char **argv);
    // comment: If is desired to pass system arguments (-geometry, etc.)


  void initialize(void);
    // comment: Initializes a new (clean) drawing window
    //          Default geometry is (5,22,790,350)


  void initialize(int x,int y,int width,int height);
    // require: width > 0 && height > 0


public: // window attributes


  int window_x(void);
    // comment: Current window's x position


  int window_y(void);
    // comment: Current window's y position
```

```
  int window_width(void);
    // comment: Current window's width


  int window_height(void);
    // comment: Current window's height


public: // colors


  int valid_color(char *color_name);
    // comment: Is "color_name" a valid color name?
    //          (Only the names of the color codes defined in the top
    //            of this file are valid)


  int translate_color(char *color_name);
    // require: valid_color(color_name)
    //
    // comment: Returns the SGM code for "color_name"


public: // defining text fonts


  void set_font(char *family,int size,int weight,int is_italic,
                int is_underline);
    // require: family != "" &&
    //          size > 0 &&
    //          weight >= min_font_weight && weight <= max_font_weight
    //
    // comment: font to use in future 'add_text' or 'change_text' calls
    //          family example strings are:
    //            Helvetica, Times, Courier and OldEnglish


  void reset_font(void);
    // comment: sets default font settings


  int text_width(char *text);
    // require: text != ""
    //
    // comment: the text width with current font.


  int text_height(char *text);
    // require: text != ""
    //
    // comment: the text height with current font.


public: // adding drawable figures


  void add_text(int x,int y,int color,char *text);
    // require: color >= min_color && color <= max_color &&
    //          text != ""


  void add_point(int x,int y,int color);
    // require: color >= min_color && color <= max_color


  void add_line(int x1,int y1,int x2,int y2,int color);
    // require: color >= min_color && color <= max_color
```

```
void add_rectangle(int x,int y,int width,int height,int color);
  // require: color >= min_color && color <= max_color &&
  //          width > 0 && height > 0


void add_fill_rectangle(int x,int y,int width,int height,int color);
  // require: color >= min_color && color <= max_color &&
  //          width > 0 && height > 0


void add_round_rectangle(int x,int y,int width,int height,int x_rnd,
                         int y_rnd,int color);
  // require: color >= min_color && color <= max_color &&
  //          width > 0 && height > 0 &&
  //          x_rnd >= 0 && x_rnd <= 99 &&
  //          y_rnd >= 0 && y_rnd <= 99
  //
  // comment: ?_rnd = 0 -> angled corners
  //          ?_rnd = 99 -> maximum roundness


void add_round_fill_rectangle(int x,int y,int width,int height,int x_rnd,
                              int y_rnd,int color);
  // require: color >= min_color && color <= max_color &&
  //          width > 0 && height > 0 &&
  //          x_rnd >= 0 && x_rnd <= 99 &&
  //          y_rnd >= 0 && y_rnd <= 99


void add_ellipse(int x,int y,int width,int height,int color);
  // require: color >= min_color && color <= max_color &&
  //          width > 0 && height > 0


void add_fill_ellipse(int x,int y,int width,int height,int color);
  // require: color >= min_color && color <= max_color &&
  //          width > 0 && height > 0


void add_circle(int x,int y,int radius,int color);
  // require: color >= min_color && color <= max_color &&
  //          radius > 0


void add_fill_circle(int x,int y,int radius,int color);
  // require: color >= min_color && color <= max_color &&
  //          radius > 0


void add_arc(int x,int y,int width,int height,int angle,int arc_len,
             int color);
  // require: color >= min_color && color <= max_color &&
  //          width > 0 && height > 0
  //
  // comment: Draws an arc defined by the rectangle (x,y,width,height),
  //          the start angle (angle) and the arc length (arc_len).
  //          The angles (angle) and (arc_len) are 1/16th of a degree,
  //          i.e. a full circle equals 5760 (16*360). Positive values
  //          of (angle) and (arc_len) mean counter-clockwise while
  //          negative values mean clockwise direction. Zero degrees
  //          is at the 3'o clock position.
  //


void add_pie(int x,int y,int width,int height,int angle,int arc_len,
```

```
                int color);
    // require: color >= min_color && color <= max_color &&
    //          width > 0 && height > 0
    //
    // comment: Look at add_arc


  void add_fill_pie(int x,int y,int width,int height,int angle,int arc_len,
                    int color);
    // require: color >= min_color && color <= max_color &&
    //          width > 0 && height > 0
    //
    // comment: Look at add_arc


  void add_chord(int x,int y,int width,int height,int angle,int arc_len,
                 int color);
    // require: color >= min_color && color <= max_color &&
    //          width > 0 && height > 0
    //
    // comment: Look at add_arc


  void add_fill_chord(int x,int y,int width,int height,int angle,int arc_len,
                      int color);
    // require: color >= min_color && color <= max_color &&
    //          width > 0 && height > 0
    //
    // comment: Look at add_arc


  void add_polygon(POINT_TYPE *points,int num_points,int color);
    // require: color >= min_color && color <= max_color &&
    //          points != NULL && num_points > 2 && !(collinear)


  void add_fill_polygon(POINT_TYPE *points,int num_points,int color);
    // require: color >= min_color && color <= max_color &&
    //          points != NULL && num_points > 2 && !(collinear)


public: // console output emulation


  void set_console_window(int x,int y,int width,int height,
                          int color = black,int background = white);
    // require: color >= min_color && color <= max_color &&
    //          background >= min_color && background <= max_color &&
    //          width > 0 && height > 0
    //
    // comment: defines cursor window (default is the entire window)
    //          (if necessary the width and height values are changed
    //           to allow the visibility of at least one character)


  void set_console_font(char *family,int size,int weight,int is_italic,
                        int is_underline);
    // require: family != "" &&
    //          size > 0 &&
    //          weight >= min_font_weight && weight <= max_font_weight
    //
    // comment: Look at set_font


  void set_console_color(int color);
    // require: color >= min_color && color <= max_color
```

```
  void set_console_background_color(int color);
    // require: color >= min_color && color <= max_color


  int console_line_height(void);
    // comment: returns the actual console font line height

  int console_enabled(void);


  void enable_console(void);
    // require: !console_enabled()


  void disable_console(void);
    // require: console_enabled()


  void clear_console(void);
    // require: console_enabled()


  SIMPLE_GRAPHICAL_OUTPUT &operator <<(char c);
    // require: console_enabled()


  SIMPLE_GRAPHICAL_OUTPUT &operator <<(char *str);
    // require: console_enabled() &&
    //          str != ""


  SIMPLE_GRAPHICAL_OUTPUT &operator <<(int i);
    // require: console_enabled()


  SIMPLE_GRAPHICAL_OUTPUT &operator <<(double d);
    // require: console_enabled()


public:


  void update_window(void);
    // comment: updates graphical window and returns to the caller


public: // DRAWABLE_ID managing


  DRAWABLE_ID last_added_drawable_id(void);
    // comment: returns an opaque internal drawable identifier, or NULL
    //            if none exists.
    //          this value can be used to delete individual drawables
    //            with 'remove_drawable' method


  int drawable_exists(DRAWABLE_ID id);
    // comment: true if (id) exists in current output window


  int drawable_visible(DRAWABLE_ID id);
    // comment: true if drawable (id) is visible in the output window


  void show_drawable(DRAWABLE_ID id);
    // require: drawable_exists(id) &&
```

```
  //          !drawable_visible(id)
  //
  // comment: makes drawable visible (the default when add_*
  //            methods are called)


void hide_drawable(DRAWABLE_ID id);
  // require: drawable_exists(id) &&
  //          drawable_visible(id)
  //
  // comment: makes drawable invisible


void raise_drawable(DRAWABLE_ID id);
  // require: drawable_exists(id)
  //
  // comment: prints drawable on top of any other


void remove_drawable(DRAWABLE_ID id);
  // require: drawable_exists(id)
  //
  // comment: removes a drawable from the output window


void translate_drawable(DRAWABLE_ID id,int delta_x,int delta_y);
  // require: drawable_exists(id)
  //
  // comment: translates drawable by (delta_x,delta_y)


int drawable_color(DRAWABLE_ID id);
  // require: drawable_exists(id)


void change_drawable_color(DRAWABLE_ID id,int color);
  // require: drawable_exists(id) &&
  //          color >= min_color && color <= max_color
  //
  // comment: sets 'color' as drawable color


int drawable_x(DRAWABLE_ID id);
  // require: drawable_exists(id)
  //
  // comment: drawable bounding box x position


int drawable_y(DRAWABLE_ID id);
  // require: drawable_exists(id)
  //
  // comment: drawable bounding box y position


int drawable_width(DRAWABLE_ID id);
  // require: drawable_exists(id)
  //
  // comment: drawable bounding box width


int drawable_height(DRAWABLE_ID id);
  // require: drawable_exists(id)
  //
  // comment: drawable bounding box height


int draw_bounding_box_enable(DRAWABLE_ID id);
```

```
    // require: drawable_exists(id)
    //
    // comment: true if the drawing of a enclosing rectangle is enabled


  void set_draw_bounding_box(DRAWABLE_ID id,int distance,int color);
    // require: drawable_exists(id) &&
    //          !draw_bounding_box_enable(id) &&
    //          distance >= 0 &&
    //          color >= min_color && color <= max_color
    //
    // comment: draws a rectangle enclosing the drawable with 'distance'
    //          pixels of interval from the drawable's frontier


  void reset_draw_bounding_box(DRAWABLE_ID id);
    // require: drawable_exists(id) &&
    //          draw_bounding_box_enable(id)
    //
    // comment: removes the drawing of an enclosing rectangle


public: // changing existent (added) drawable figures


  int is_text(DRAWABLE_ID id);
    // require: drawable_exists(id)
    //
    // comment: is (id) a text drawable


  void change_text(DRAWABLE_ID id,int x,int y,int color,char *text);
    // require: is_text(id) &&
    //          color >= min_color && color <= max_color &&
    //          text != ""
    //
    // comment: font can be redefined with a previous call to 'set_font'


  int is_point(DRAWABLE_ID id);
    // require: drawable_exists(id)
    //
    // comment: is (id) a point drawable


  void change_point(DRAWABLE_ID id,int x,int y,int color);
    // require: is_point(id) &&
    //          color >= min_color && color <= max_color


  int is_line(DRAWABLE_ID id);
    // require: drawable_exists(id)
    //
    // comment: is (id) a line drawable


  void change_line(DRAWABLE_ID id,int x1,int y1,int x2,int y2,int color);
    // require: is_line(id) &&
    //          color >= min_color && color <= max_color


  int is_rectangle(DRAWABLE_ID id);
    // require: drawable_exists(id)
    //
    // comment: is (id) a rectangle drawable
```

```
void change_rectangle(DRAWABLE_ID id,int x,int y,int width,int height,
                      int color,int filled);
  // require: is_rectangle(id) &&
  //          color >= min_color && color <= max_color &&
  //          width > 0 && height > 0
  //
  // comment: filled is a boolean (0,!=0) argument


int is_round_rectangle(DRAWABLE_ID id);
  // require: drawable_exists(id)
  //
  // comment: is (id) a round_rectangle drawable


void change_round_rectangle(DRAWABLE_ID id,int x,int y,int width,int height,
                            int x_rnd,int y_rnd,int color,int filled);
  // require: is_round_rectangle(id) &&
  //          color >= min_color && color <= max_color &&
  //          width > 0 && height > 0 &&
  //          x_rnd >= 0 && x_rnd <= 99 &&
  //          y_rnd >= 0 && y_rnd <= 99
  //
  // comment: filled is a boolean (0,!=0) argument


int is_ellipse(DRAWABLE_ID id);
  // require: drawable_exists(id)
  //
  // comment: is (id) a ellipse drawable


void change_ellipse(DRAWABLE_ID id,int x,int y,int width,int height,
                    int color,int filled);
  // require: is_ellipse(id) &&
  //          color >= min_color && color <= max_color &&
  //          width > 0 && height > 0
  //
  // comment: filled is a boolean (0,!=0) argument


int is_circle(DRAWABLE_ID id);
  // require: drawable_exists(id)
  //
  // comment: is (id) a circle drawable


void change_circle(DRAWABLE_ID id,int x,int y,int radius,
                   int color,int filled);
  // require: is_circle(id) &&
  //          color >= min_color && color <= max_color &&
  //          radius > 0
  //
  // comment: filled is a boolean (0,!=0) argument


int is_arc(DRAWABLE_ID id);
  // require: drawable_exists(id)
  //
  // comment: is (id) a arc drawable


void change_arc(DRAWABLE_ID id,int x,int y,int width,int height,
                int angle,int arc_len,int color);
  // require: is_arc(id) &&
  //          color >= min_color && color <= max_color &&
  //          width > 0 && height > 0
```

```
  int is_pie(DRAWABLE_ID id);
    // require: drawable_exists(id)
    //
    // comment: is (id) a pie drawable


  void change_pie(DRAWABLE_ID id,int x,int y,int width,int height,
                  int angle,int arc_len,int color,int filled);
    // require: is_pie(id) &&
    //          color >= min_color && color <= max_color &&
    //          width > 0 && height > 0
    //
    // comment: filled is a boolean (0,!=0) argument


  int is_chord(DRAWABLE_ID id);
    // require: drawable_exists(id)
    //
    // comment: is (id) a chord drawable


  void change_chord(DRAWABLE_ID id,int x,int y,int width,int height,
                    int angle,int arc_len,int color,int filled);
    // require: is_chord(id) &&
    //          color >= min_color && color <= max_color &&
    //          width > 0 && height > 0
    //
    // comment: filled is a boolean (0,!=0) argument


  int is_polygon(DRAWABLE_ID id);
    // require: drawable_exists(id)
    //
    // comment: is (id) a polygon drawable


  void change_polygon(DRAWABLE_ID id,POINT_TYPE *points,int num_points,
                      int color,int filled);
    // require: is_polygon(id) &&
    //          color >= min_color && color <= max_color &&
    //          points != NULL && num_points > 2 && !(collinear)
    //
    // comment: filled is a boolean (0,!=0) argument


}; // SIMPLE_GRAPHICAL_OUTPUT


class SIMPLE_GRAPHICAL_INPUT
  // comment: singleton
{

public:


  SIMPLE_GRAPHICAL_INPUT(void);
    // comment: Default (sgin) constructor
    //          Mouse and keyboards events enabled


  int keyboard_enabled(void);
    // comment: are the keyboard events enabled?


  void enable_keyboard(void);
```

```
  void disable_keyboard(void);


  int mouse_enabled(void);
    // comment: are the mouse events enabled?


  void enable_mouse(void);


  void disable_mouse(void);


  int event_wait(void);
    // comment: passes the control to the graphical window until a key or
    //            a mouse button is pressed (if those events are enabled).
    //          It returns the key ascii code or the mouse button code.
    //          If the graphical window is closed the 'exit_event' is
    //            returned.


  void wait_until_finish(void);
    // comment: passes the control to the graphical window
    //          terminates when the window is closed


  int keyboard_event(void);
    // comment: last event triggered by the keyboard


  int mouse_event(void);
    // comment: last event triggered by the mouse


  int last_mouse_x_position(void);
    // comment: last event's mouse x position


  int last_mouse_y_position(void);
    // comment: last event's mouse y position


public: // console input emulation


  void set_string_maximum_size(int size);
    // require: size >= 0
    //
    // comment: Defines the maximum number of characters readable through
    //            the "operator >>(char *str)" operator (not counting with
    //            the string terminator character).
    //          The zero value (default) makes the input unbounded.


  void operator >>(char &c);
    // require: sgout.console_enabled()


  void operator >>(char *str);
    // require: sgout.console_enabled() &&
    //          str != ""
    //
    // comment: Reads a string until the end of line.
    //          The string maximum size can be defined with
    //            (set_string_maximum_size).
```

```
  void operator >>(int &i);
    // require: sgout.console_enabled()


  void operator >>(double &d);
    // require: sgout.console_enabled()


  void flush(void);
    // require: sgout.console_enabled()
    //
    // comment: Clears input buffer


public: // text editing


  void line_editing(int x,int y,int width,int height,char *text,int max_size,
                    int color = black,int background = white);
    // require: width > 0 && height > 0 &&
    //          text != "" && max_size >= 0 &&
    //          color >= min_color && color <= max_color &&
    //          background >= min_color && background <= max_color
    //
    // comment: line editing to the (text) string, using the box defined by
    //            (x,y,width,height).
    //          The text size won't exceed (max_size) without counting the
    //            string terminator.
    //          Uses the console default font or the one defined by (set_font)


}; // SIMPLE_GRAPHICAL_INPUT


extern SIMPLE_GRAPHICAL_OUTPUT sgout;
  // to be used in a similar way as 'cout'


extern SIMPLE_GRAPHICAL_INPUT sgin;
  // to be used in a similar way as 'cin'


#endif
```