

WEBOTS 2.0

Reference Manual

© 1998,1999 Cyberbotics
www.cyberbotics.com

September 13, 1999

© 1998,1999 Cyberbotics S.á r.l.
All Rights Reserved

CYBERBOTICS S.A R.L. ("CYBERBOTICS") MAKES NO WARRANTY OR CONDITION, EITHER EXPRESSED OR IMPLIED, INCLUDING BUT NOT LIMITED TO ANY IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE, REGARDING THIS MANUAL AND THE ASSOCIATED SOFTWARE. THIS MANUAL IS PROVIDED ON AN "AS-IS" BASIS. NEITHER CYBERBOTICS NOR ANY APPLICABLE LICENSOR WILL BE LIABLE FOR ANY INCIDENTAL OR CONSEQUENTIAL DAMAGES.

This software was initially developed at the Laboratoire de Micro-Informatique (LAMI) of the Swiss Federal Institute of Technology, Lausanne, Switzerland (EPFL). The EPFL makes no warranties of any kind on this software. In no event shall the EPFL be liable for incidental or consequential damages of any kind in connection with use and exploitation of this software.

Trademark information

CodeWarrior is a registered trademark of Metrowerks, Inc.

IRIX, O2 and OpenGL are registered trademark of Silicon Graphics Inc.

Khepera is a registered trademark of K-Team S.A.

Linux is a registered trademark of Linus Torwalds.

Macintosh is a registered trademark of Apple Computer, Inc.

Pentium is a registered trademark of Intel Corp.

PowerPC is a registered trademark of IBM Corp.

Red Hat is a registered trademark of Red Hat Software, Inc.

Solaris and Solaris OpenGL are registered trademarks of Sun Microsystems, Inc.

All SPARC trademarks are used under the license and are trademarks or registered trademarks of SPARC International, Inc. Products bearing SPARC trademarks are based upon an architecture developed by Sun Microsystems, Inc.

SuSE is a registered trademark of SuSE GmbH.

UNIX is a registered trademark licensed exclusively by X/Open Company, Ltd.

VisualC++, Windows, Windows 95, Windows 98 and Windows NT are registered trademarks of Microsoft, Corp.

Contents

1 Controller API Reference	5
1.1 The GUI: Graphical User Interface	5
1.1.1 Arc, Circle, Ellipse	5
1.1.2 Buttons	13
1.1.3 Check boxes	19
1.1.4 Colors	24
1.1.5 Events	26
1.1.6 Images	37
1.1.7 Labels	43
1.1.8 Lines	48
1.1.9 Miscellaneous	54
1.1.10 Pixmaps	60
1.1.11 Popup menus	63
1.1.12 Rectangles	72
1.1.13 Textfields	77
1.1.14 Windows	83
2 Khepera API Reference	93
2.1 Khepera	93
2.2 Khepera Gripper	107
2.3 Khepera K213	115
2.4 Khepera K6300	118
2.5 Khepera Panoramic	122

3	Supervisor API Reference	129
3.1	Supervisor	129
3.2	EAI: External Authoring Interface	133
4	Webots File Format Version 2.0	141
4.1	File Structure	141
4.2	VRML97 nodes partially supported in Webots	141
4.3	Webots nodes	142
4.3.1	Ball	142
4.3.2	Can	143
4.3.3	KheperaFeeder	143
4.3.4	Lamp	143
4.3.5	Wall	143
4.3.6	Supervisor	144
4.3.7	Alice	144
4.3.8	AliceIRCom	144
4.3.9	Khepera	144
4.3.10	KheperaK213	145
4.3.11	KheperaK6300	145
4.3.12	KheperaGripper	145
4.3.13	KheperaPanoramic	145

Chapter 1

Controller API Reference

1.1 The GUI: Graphical User Interface

1.1.1 Arc, Circle, Ellipse

```
gui_arc  
gui_arc_change_color  
gui_arc_hide  
gui_arc_show  
gui_arc_is_visible  
gui_arc_delete  
gui_arc_new  
gui_arc_new_circle  
gui_arc_new_ellipse
```

TYPE

gui_arc – C type representing an arc

SYNOPSIS

```
#include <gui.h>  
gui_arc arc;
```

DESCRIPTION

This type is used to represent circular or elliptical arcs, circles and ellipses.

NAME

gui_arc_change_color - change the color of an arc, ellipse or circle

SYNOPSIS

```
#include <gui.h>
void gui_arc_change_color(gui_arc arc,uint32 color);
```

DESCRIPTION

This function changes the color of an arc. It causes the arc, ellipse or circle to be redrawn.

PARAMETERS

- **arc**: Specifies an arc, circle or ellipse.
- **color**: Specifies the new color of the arc, circle or ellipse.

SEE ALSO

[gui_get_color](#)

NAME

`gui_arc_hide`, `gui_arc_show`, `gui_arc_is_visible` - set and get the visibility of an arc, circle or ellipse

SYNOPSIS

```
#include <gui.h>
void gui_arc_show(gui_arc arc);
void gui_arc_hide(gui_arc arc);
bool gui_arc_is_visible(gui_arc arc);
```

DESCRIPTION

When an arc, circle or ellipse is created, it is visible.

`gui_arc_hide` makes an arc, circle or ellipse disappear.

`gui_arc_show` makes an arc, circle or ellipse appear.

`gui_arc_is_visible` gets the visibility of an arc, circle or ellipse.

PARAMETERS

- `arc`: Specifies an arc, circle or ellipse.

RETURN VALUE

`gui_arc_is_visible` returns `true` if the arc, circle or ellipse is visible and `false` otherwise.

SEE ALSO

`gui_arc_new`, `gui_arc_delete`

NAME

`gui_arc_delete` - delete an existing arc, circle or ellipse

SYNOPSIS

```
#include <gui.h>
void gui_arc_delete(gui_arc arc);
```

DESCRIPTION

`gui_arc_delete` deletes an arc, circle or ellipse. Note that an arc is automatically deleted when its window is deleted.

PARAMETERS

- `arc`: Specifies an arc, circle or ellipse.

SEE ALSO

`gui_arc_new`, `gui_arc_new_ellipse`, `gui_arc_new_circle`,
`gui_window_delete`

NAME

gui_arc_new - create an arc

SYNOPSIS

```
#include <gui.h>
gui_arc gui_arc_new(gui_window window,int16 x,int16 y,
                     uint16 width,uint16 height,
                     int16 start,int16 length,
                     uint32 color,bool filled);
```

DESCRIPTION

gui_arc_new creates an arc in a window. The arc is built as a portion of an ellipse. The ellipse is defined by the upper-left corner of its bounding box, its width, its height, its color and a boolean value defining if it is empty or filled. The portion of the ellipse is defined by its starting angle, and its length. These angles are calculated in 64ths of degrees in the trigonometric circle.

PARAMETERS

- **window:** Specifies the window, in which the arc is created.
- **x, y:** Specify the x and y coordinates of the upper-left corner of the bounding box containing the arc, relative to the origin of the window.
- **height, width:** Specify the width and height in pixels of the bounding box. These are the major and the minor axes of the arc.
- **start:** Specifies the start of the arc relative to the three-o'clock position from the center. Angles are specified in 64ths of degrees.
- **length:** Specifies the path and extent of the arc relative to the start of the arc. Angles are specified in 64ths of degrees.
- **color:** Specifies the color of the arc.
- **filled:** Specifies whether the arc must be filled (`true`) or not (`false`).

RETURN VALUE

gui_arc_new returns the gui_arc just created.

SEE ALSO

`gui_arc_new_circle`, `gui_arc_new_ellipse`, `gui_arc_delete`,
`gui_color_get`

NAME

`gui_arc_new_circle` - create an circle

SYNOPSIS

```
#include <gui.h>
gui_arc gui_arc_new_circle(gui_window window,
                           int16 x,int16 y,
                           uint16 diameter,uint32
                           color,
                           bool filled);
```

DESCRIPTION

`gui_arc_new_circle` creates a circle in a window. It is defined by the upper-left corner of its bounding box, its diameter, its color, and a boolean value defining if it is empty or filled.

PARAMETERS

- `window`: Specifies the window, in which the circle is created.
- `x`, `y`: Specify the `x` and `y` coordinates of the upper-left corner of the bounding box containing the circle, relative to the origin of the window.
- `diameter`: Specifies the diameter of the circle.
- `color`: Specifies the color of the circle.
- `filled`: Specifies whether the circle must be filled (`true`) or not (`false`)

RETURN VALUE

`gui_arc_new_circle` returns the `gui_arc` just created.

SEE ALSO

`gui_arc_new`, `gui_arc_new_ellipse`, `gui_arc_delete`, `gui_color_get`

NAME

gui_arc_new_ellipse - create an ellipse

SYNOPSIS

```
#include <gui.h>
gui_arc gui_arc_new_ellipse(gui_window window,
                            int16 x,int16 y,
                            uint16 width,uint16 height,
                            uint32 color,bool filled);
```

DESCRIPTION

gui_arc_new_ellipse creates an ellipse in a window. It is defined by the upper-left corner of its bounding box, its width, its height, its color, and a boolean value defining if it is empty or filled.

PARAMETERS

- **window**: Specifies the window, in which the ellipse is created.
- **x**, **y**: Specify the x and y coordinates of the upper-left corner of the bounding box containing the ellipse, relative to the origin of the window.
- **height**, **width**: Specify the width and height in pixels of the bounding box. These are the major and the minor axes of the ellipse.
- **color**: Specifies the color of the ellipse.
- **filled**: Specifies whether the ellipse must be filled (`true`) or not (`false`)

RETURN VALUE

gui_arc_new_ellipse returns the `gui_arc` just created.

SEE ALSO

`gui_arc_new`, `gui_arc_new_circle`, `gui_arc_delete`,
`gui_color_get`

1.1.2 Buttons

```
gui_button  
gui_button_change_text  
gui_button_activate  
gui_button_desactivate  
gui_button_is_active  
gui_button_enable  
gui_button_disable  
gui_button_is_enabled  
gui_button_show  
gui_button_hide  
gui_button_is_visible  
gui_button_set_width  
gui_button_delete  
gui_button_new
```

TYPE

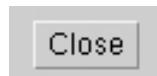
gui_button – C type representing a button

SYNOPSIS

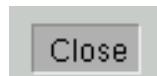
```
#include <gui.h>  
gui_button button;
```

SCREENSHOTS

a button:



an active button:



NAME

`gui_button_activate`, `gui_button_desactivate`,
`gui_button_is_active` - get or set the activation state of a button

SYNOPSIS

```
#include <gui.h>

void gui_button_activate(gui_button button);
void gui_button_desactivate(gui_button button);
bool gui_button_is_active(gui_button button);
```

DESCRIPTION

`gui_button_activate` activates a button, i.e., visually pushes it.

`gui_button_desactivate` desactivates a button, i.e., visually releases it. When a button has been pushed by user, this function must be used to release it after the completion of the action associated to the button.

`gui_button_is_active` returns the activation state of a button.

PARAMETERS

- `button`: Specifies a button

RETURN VALUE

`gui_button_is_active` returns `true` if the button is down, otherwise it returns `false`.

NAME

gui_button_enable, gui_button_disable, gui_button_is_enabled
- set and get the enable state of a button

SYNOPSIS

```
#include <gui.h>
void gui_button_enable(gui_button button);
void gui_button_disable(gui_button button);
bool gui_button_is_enabled(gui_button button);
```

DESCRIPTION

gui_button_disable makes a button disabled.

gui_button_enable makes a button enabled.

gui_button_is_enabled checks whether a button is enabled.

When a button is disabled, the user cannot push it, hence, it cannot produce events.

PARAMETERS

- o button: Specifies a button

RETURN VALUE

gui_button_is_enabled returns `true` if the button is enabled and `false` otherwise.

NAME

`gui_button_hide`, `gui_button_show`, `gui_button_is_visible`
- set and get the visibility of a button

SYNOPSIS

```
#include <gui.h>
void gui_button_show(gui_button button);
void gui_button_hide(gui_button button);
bool gui_button_is_visible(gui_button button);
```

DESCRIPTION

When a button is created, it is visible.
`gui_button_hide` makes a button disappear.
`gui_button_show` makes a button appear.
`gui_button_is_visible` gets the visibility of a button.
When a button is hidden, it cannot produce events.

PARAMETERS

- o `button`: Specifies a button.

RETURN VALUE

`gui_button_is_visible` returns `true` if the button is visible and `false` otherwise.

NAME

gui_button_set_width - set the button width

SYNOPSIS

```
#include <gui.h>
void gui_button_set_width(gui_button button,
                          uint16 width);
```

DESCRIPTION

The default button width is computed from the width of the text or the image contained in the button. `gui_button_set_width` allows you to change of this default width.

PARAMETERS

- `button`: Specifies a button.
- `width`: Specifies the width in pixels.

SEE ALSO

`gui_button_new`

NAME

`gui_button_new`, `gui_button_delete` - create or delete a button

SYNOPSIS

```
#include <gui.h>
gui_button gui_button_new(gui_window window,
int16 x,int16 y,char *text);
void         gui_button_delete(gui_button button);
```

DESCRIPTION

The `gui_button_new` function creates a button in a window. The size of the button is computed from the size of the text or the image it contains.

The `gui_button_delete` deletes a button. A button is automatically deleted when its window is deleted.

PARAMETERS

`gui_button_new`

- `window`: Specifies the window, in which the button is created.
- `x`, `y`: Specify the `x` and `y` coordinates of the upper-left corner of the button, relative to the origin of the window.
- `text`: if `text` ends up with ``.png'', it specify a PNG image which is loaded inside the button. Otherwise, `text` specify the text label of the button.

`gui_button_delete`

- `button`: Specifiesthe button to be deleted.

RETURN VALUE

`gui_button_new` returns the `gui_button` just created.

1.1.3 Check boxes

```
gui_checkbox  
gui_checkbox_activate  
gui_checkbox_desactivate  
gui_checkbox_is_activated  
gui_checkbox_enable  
gui_checkbox_disable  
gui_checkbox_is_enabled  
gui_checkbox_hide  
gui_checkbox_show  
gui_checkbox_is_visible  
gui_checkbox_new  
gui_checkbox_delete
```

TYPE

gui_checkbox - C type representing a checkbox

SYNOPSIS

```
#include <gui.h>  
gui_checkbox checkbox;
```

SCREENSHOT

a check box:



an active check box:



NAME

gui_checkbox_activate, gui_checkbox_desactivate,
gui_checkbox_is_activated - get or set the check box activation state

SYNOPSIS

```
#include <gui.h>
void gui_checkbox_activate(gui_checkbox checkbox);
void gui_checkbox_desactivate(gui_checkbox checkbox);
bool gui_checkbox_is_active(gui_checkbox checkbox);
```

DESCRIPTION

gui_checkbox_activate activates a checkbox, i.e., visually the box appears checked.
gui_checkbox_desactivate deactivates a checkbox, i.e., visually removes the check mark from the box.

gui_checkbox_is_active returns the checkbox activation state.

PARAMETERS

- o checkbox: Specifies a checkbox.

RETURN VALUE

If the box is checked, gui_checkbox_is_active returns true, otherwise, it returns false.

NAME

gui_checkbox_disable, gui_checkbox_enable,
gui_checkbox_is_enabled - set and get the enable state of a checkbox

SYNOPSIS

```
#include <gui.h>
void gui_checkbox_disable(gui_checkbox checkbox);
void gui_checkbox_enable(gui_checkbox checkbox);
bool gui_checkbox_is_enabled(gui_checkbox checkbox);
```

DESCRIPTION

gui_checkbox_disable makes a checkbox disabled.
gui_checkbox_enable makes a checkbox enabled.
gui_checkbox_is_enabled checks whether a checkbox is enabled or not.
When a checkbox is disabled, it cannot produce events.

PARAMETERS

- o **checkbox:** Specifies a checkbox.

RETURN VALUE

gui_checkbox_is_enabled returns **true** if the checkbox is enabled and **false** otherwise.

NAME

gui_checkbox_hide, gui_checkbox_show, gui_checkbox_is_visible
- set and get the visibility of a checkbox

SYNOPSIS

```
#include <gui.h>
void gui_checkbox_hide(gui_checkbox checkbox);
void gui_checkbox_show(gui_checkbox checkbox);
bool gui_checkbox_is_visible(gui_checkbox checkbox);
```

DESCRIPTION

When a checkbox is created, it is visible.
gui_checkbox_hide makes a checkbox disappear.
gui_checkbox_show makes a checkbox appear.
gui_checkbox_is_visible gets the visibility of a checkbox.
When a checkbox is hidden, it cannot produce events.

PARAMETERS

- o checkbox: Specifies a checkbox.

RETURN VALUE

gui_checkbox_is_visible returns true if the checkbox is visible and false otherwise.

NAME

gui_checkbox_new, gui_checkbox_delete - create or delete a checkbox

SYNOPSIS

```
#include <gui.h>
gui_checkbox gui_checkbox_new(gui_window window,
                             int16 x,int16 y,
                             char *text,bool active);
void gui_checkbox_delete(gui_checkbox checkbox);
```

DESCRIPTION

gui_checkbox_new creates a checkbox in a window. The size of the checkbox is computed from the size of the text it contains.

gui_checkbox_delete deletes a checkbox. Note that a checkbox is automatically deleted when its window is deleted.

PARAMETERS

gui_checkbox_new

- **window**: Specifies the window, in which the checkbox is created.
- **text**: Specifies the text label if the checkbox.
- **x, y**: Specify the x and y coordinates of the upper-left corner of the checkbox, relative to the origin of the window.
- **active**: Specifies if the checkbox is active (`true`), i.e., checked, or not (`false`).

gui_checkbox_delete

- **checkbox**: Specifies a checkbox.

RETURN VALUE

gui_checkbox_new returns the `gui_checkbox` just created.

SEE ALSO

gui_window_delete

1.1.4 Colors

`gui_color_get`

CONSTANTS

12 colors are pre-defined with the following global variables :

- `uint32 GUI_RED;`
- `uint32 GUI_GREEN;`
- `uint32 GUI_BLUE;`
- `uint32 GUI_WHITE;`
- `uint32 GUI_BLACK;`
- `uint32 GUI_MAGENTA;`
- `uint32 GUI_CYAN;`
- `uint32 GUI_YELLOW;`
- `uint32 GUI_WHITEGREY;`
- `uint32 GUI_LIGHTGREY;`
- `uint32 GUI_DARKGREY;`
- `uint32 GUI_BLACKGREY;`

NAME

gui_color_get - get a color corresponding to RGB values

SYNOPSIS

```
#include <gui.h>
uint32 gui_color_get(uint8 r, uint8 g, uint8 b);
```

DESCRIPTION

The `gui_color_get` function returns a new color based on RGB information. Each parameter represents one color component (red, green and blue). Color component values range from 0 to 255.

PARAMETERS

- `r`: Specifies the red component value (between 0 and 255).
- `g`: Specifies the green component value (between 0 and 255).
- `b`: Specifies the blue component value (between 0 and 255).

RETURN VALUE

The value returned by `gui_color_get` is an identifier for the closest color corresponding to the RGB components specified as parameters.

1.1.5 Events

```
gui_event_callback  
gui_event_get_key  
gui_event_get_info  
gui_event_get_modifier  
gui_event_get_mouse_x  
gui_event_get_mouse_y  
gui_event_get_type  
gui_event_get_window  
gui_event_get_widget  
gui_event_get_timer
```

NAME

gui_event_callback - define a callback function for handling events

SYNOPSIS

```
#include <gui.h>
void gui_event_callback(void (*callback)());
```

DESCRIPTION

This function define a `callback` function that will be called whenever a GUI event occurs. This callback function will have to handle the event: First, it will determine what kind of event it is, by using the `gui_event_get_type`, `gui_event_get_info`, `gui_event_get_key`, etc. functions. Second, it will execute the instructions associated to that event.

EXAMPLE

```
#include <gui.h>

gui_window window;
gui_button button_beep, button_hide, button_show;

void callback()
{
    if (window == gui_event_get_window())
    {
        if (gui_event_get_type() == GUI_WINDOW_CLOSE)
            gui_window_delete(window);
        else if (button_beep == gui_event_get_widget())
        {
            gui_beep();                                /* rings the bell      */
            gui_button_desactivate(button_beep); /* le-
lease the button */
        }
        else if (button_hide == gui_event_get_widget())
        {
            gui_button_hide(button_beep);
            gui_button_hide(button_hide);
            gui_button_show(button_show);
            gui_button_desactivate(button_hide);
            /* release the button */
        }
    }
}
```

```
    }
    else if (button_show == gui_event_get_widget( ))
    {
        gui_button_show(button_beep);
        gui_button_show(button_hide);
        gui_button_hide(button_show);
        gui_button_desactivate(button_show); /* release the button */
    }
}
}

int main()
{
    supervisor_live();
    gui_event_callback(callback);
    window = gui_window_new("Testing event callback",
                           10,10,200,50);
    button_hide = gui_button_new(window, 30,20,"hide");
    button_show = gui_button_new(window, 30,20,"show");
    button_beep = gui_button_new(window,130,20,"beep");
    gui_button_hide(button_show);
    gui_window_show(window);
    for(;;) supervisor_step(64); /* never ending */
}
```

SEE ALSO

`gui_event_get_type`, `gui_event_get_info`, `gui_event_get_key`,
`gui_event_get_modifier`, `gui_event_get_mouse_x`, `gui_event_get_mouse_y`,
`gui_event_get_window`, `gui_event_get_widget`, `gui_event_get_timer`

NAME

gui_event_get_key - get the key for a GUI_KEY_DOWN or GUI_KEY_UP event

SYNOPSIS

```
#include <gui.h>
char gui_event_get_key();
```

DESCRIPTION

The `gui_event_get_key` returns the key that caused a GUI_KEY_DOWN or GUI_KEY_UP event.

RETURN VALUE

This function return the ASCII code of the corresponding key. For special keys, a number of constants has been defined :

- GUI_RIGHT
- GUI_UP
- GUI_LEFT
- GUI_DOWN
- GUI_CUT
- GUI_COPY
- GUI_PASTE
- GUI_BACKSPACE
- GUI_TAB
- GUI_ESCAPE
- GUI_DELETE

SEE ALSO

`gui_event_callback`, `gui_event_get_type`, `gui_event_get_modifier`

NAME

`gui_event_get_info` - get more information about the current event

SYNOPSIS

```
#include <gui.h>
uint32 gui_event_get_info();
```

DESCRIPTION

The `gui_event_get_info` function is used to get more information about the current event in the callback function.

RETURN VALUE

This function returns a value in the following list of constant:

- `GUI_WIDGET`: the current event is a widget event (i.e., produced by a `gui_checkbox`, `gui_button`, `gui_popup` or `gui_textfield`).
- `GUI_DOUBLE_CLICK`: the current event is a double click.
- 0: otherwise.

SEE ALSO

`gui_event_callback`, `gui_event_get_info`, `gui_event_get_key`,
`gui_event_get_modifier`, `gui_event_get_mouse_x`, `gui_event_get_mouse_y`,
`gui_event_get_window`, `gui_event_get_widget`, `gui_event_get_timer`

NAME

`gui_event_get_modifier`

- get the modifier key for a GUI_KEY_DOWN or GUI_KEY_UP event

SYNOPSIS

```
#include <gui.h>
uint32 gui_event_get_modifier();
```

DESCRIPTION

The `gui_event_get_modifier` returns the modifier key pressed when the GUI_KEY_DOWN or GUI_KEY_UP event occurred.

RETURN VALUE

Three constants represent the three modifier keys available on common keyboards (shift, control and alt). If no modifier key is pressed, the function returns 0.

- GUI_SHIFT
- GUI_CONTROL
- GUI_ALT

SEE ALSO

`gui_event_callback`, `gui_event_get_type`, `gui_event_get_key`

NAME

`gui_event_get_mouse_x`, `gui_event_get_mouse_y`
- get the position of the mouse pointer for a mouse event.

SYNOPSIS

```
#include <gui.h>
int16 gui_event_get_mouse_x();
int16 gui_event_get_mouse_y();
```

DESCRIPTION

The `gui_event_get_mouse_x` and `gui_event_get_mouse_y` functions return the position of the mouse corresponding to an event of the following type:

- `GUI_MOUSE_DOWN`
- `GUI_MOUSE_UP`
- `GUI_MOUSE_MOVE`

SEE ALSO

`gui_event_callback`, `gui_event_get_type`

NAME

`gui_event_get_timer` - get the timer that caused a time up event

SYNOPSIS

```
#include <gui.h>
gui_timer gui_event_get_timer();
```

DESCRIPTION

If `get_event_type()` returns `GUI_TIME_UP`, `gui_event_get_timer()` will return the timer corresponding to this event.

RETURN VALUE

`gui_event_get_timer` returns the `gui_timer` that caused a `GUI_TIME_UP` event.

SEE ALSO

`gui_event_callback`, `gui_event_get_type`

NAME

`gui_event_get_type` - get the type of the current event

SYNOPSIS

```
#include <gui.h>
uint32 gui_event_get_type();
```

DESCRIPTION

The `gui_event_get_type` function is used to get the type of the current event in the call-back function.

RETURN VALUE

This function returns a `uint32` value in the following list of constants:

- `GUI_MOUSE_DOWN`: the user pressed one of the buttons of the mouse.
- `GUI_MOUSE_UP`: the user released one the button of the mouse.
- `GUI_MOUSE_MOVE`: the user moved the mouse pointer.
- `GUI_KEY_DOWN`: the user pressed a key.
- `GUI_KEY_UP`: the user released a key.
- `GUI_WINDOW_CLOSE`: the user requested to close the window.
- `GUI_WINDOW_RESIZE`: a window was resized by the user.
- `GUI_TIME_UP`: a time up event occurred.
- `GUI_WINDOW_ENTER`: the mouse pointer entered a window.
- `GUI_WINDOW_LEAVE`: the mouse pointer left a window.

SEE ALSO

`gui_event_callback`, `gui_event_get_info`, `gui_event_get_key`,
`gui_event_get_modifier`, `gui_event_get_mouse_x`, `gui_event_get_mouse_y`,
`gui_event_get_window`, `gui_event_get_widget`, `gui_event_get_timer`

NAME

gui_event_get_window - get the window related to the current event

SYNOPSIS

```
#include <gui.h>
gui_window gui_event_get_window();
```

DESCRIPTION

The `gui_event_get_window` function returns the window in which the current event occurred. It might be `NULL` if the event is not associated to a window (like `GUI_TIME_UP`).

SEE ALSO

`gui_event_callback`

NAME

gui_event_get_widget - get the widget involved in the current event

SYNOPSIS

```
#include <gui.h>
gui_widget gui_event_get_widget();
```

DESCRIPTION

The `gui_event_get_widget` function returns the widget corresponding to the current event. The `gui_event_get_info` function should be used before to check if the event was produced by a widget.

SEE ALSO

`gui_event_callback`, `gui_event_get_type`, `gui_event_get_info`

1.1.6 Images

```
gui_image  
gui_image_delete  
gui_image_hide  
gui_image_show  
gui_image_is_visible  
gui_image_new  
gui_image_new_from_pixmap  
gui_image_get_width  
gui_image_get_height
```

TYPE

gui_image - C type representing an image

SYNOPSIS

```
#include <gui.h>  
gui_image image;
```

NAME

`gui_image_delete` - delete an image

SYNOPSIS

```
#include <gui.h>
void gui_image_delete(gui_image image);
```

DESCRIPTION

`gui_image_delete` is used to delete an image, and to release its memory. Note that an image is automatically deleted when its window is deleted. If this image was created from a pixmap, `gui_image_delete` must be called before the corresponding `gui_pixmap_delete`.

PARAMETERS

- o `image`: Specifies an image

SEE ALSO

`gui_image_new`, `gui_image_new_from_pixmap`, `gui_pixmap_delete`,
`gui_window_delete`

NAME

`gui_image_hide`, `gui_image_show`, `gui_image_is_visible` - get or set the visibility of an image

SYNOPSIS

```
#include <gui.h>
void gui_image_hide(gui_image image);
void gui_image_show(gui_image image);
bool gui_image_is_visible(gui_image image);
```

DESCRIPTION

When an image is created, it is visible.

`gui_image_hide` makes an image disappear.

`gui_image_show` makes an image appear.

`gui_image_is_visible` gets the visibility of the image.

PARAMETERS

- o `image`: Specifies an image

RETURN VALUE

`gui_image_is_visible` returns `true` if the image is visible and `false` otherwise.

NAME

`gui_image_new`, `gui_image_new_from_pixmap` - create an image

SYNOPSIS

```
#include <gui.h>
gui_image gui_image_new(gui_window window,
                        int16 x,int16 y,
                        char *png_file);
gui_image gui_image_new_from_pixmap(gui_window window,
                                      int16 x,int16 y,
                                      gui_pixmap pixmap);
```

DESCRIPTION

`gui_image_new` creates an image in a window from a file.

`gui_image_new_from_pixmap` creates an image in a window from a pixmap. Several images can share the same pixmap.

PARAMETERS

`gui_image_new`

- `window`: Specifies the window, in which the image is created.
- `x`, `y`: Specify the x and y coordinates of the upper-left corner of the image, relative to the origin of window.
- `png_file`: Specify a PNG file name, the path to find the images must be set before by using the `gui_image_set_path` function.

`gui_image_new_from_pixmap`

- `window`: Specifies the window, in which the image is created.
- `x`, `y`: Specify the x and y coordinates of the upper-left corner of the image, relative to the origin of window.
- `pixmap`: Specifies a pixmap.

RETURN VALUE

`gui_image_new` returns the `gui_image` just created.

`gui_image_new_from_pixmap` returns the `gui_image` just created.

SEE ALSO

`guiPixmap`, `gui_setImagePath`

NAME

gui_image_get_width, gui_image_get_height - get the image dimensions

SYNOPSIS

```
#include <gui.h>
uint16 gui_image_get_width(gui_image image);
uint16 gui_image_get_height(gui_image image);
```

DESCRIPTION

These functions get the dimensions of an image.

PARAMETERS

- o **image**: Specifies an image

RETURN VALUE

gui_image_get_width returns the width of the image .
gui_image_get_height return the height of the image.

1.1.7 Labels

```
gui_label  
gui_label_delete  
gui_label_new  
gui_label_change_text  
gui_label_hide  
gui_label_show  
gui_label_is_visible  
gui_label_align_left  
gui_label_align_center  
gui_label_align_right
```

TYPE

`gui_label` - C type representing a label

SYNOPSIS

```
#include <gui.h>  
gui_label label;
```

SCREENSHOT



NAME

`gui_label_new`, `gui_label_delete` - create or delete a label

SYNOPSIS

```
#include <gui.h>
gui_label gui_label_new(gui_window window,
                        int16 x,int16 y,char *text);
void gui_label_delete(gui_label label);
```

DESCRIPTIONS

`gui_label_new` creates a label in a window. The width of the label is computed according to the width of the text.

`gui_label_delete` delete a label. Note that a label is automatically deleted when its window is deleted.

PARAMETERS

`gui_label_new`

- `window`: Specifies the window, in which the label is created.
- `x`, `y`: Specify the `x` and `y` coordinates of the label reference position, relative to the origin of `window`; by default this position is the left extremity of the text line.
- `text`: Specifies text of the label.

`gui_label_delete`

- `label`: Specifies a label.

RETURN VALUE

`gui_label_new` returns the `gui_label` just created.

SEE ALSO

`gui_window_delete`

NAME

gui_label_change_text – change the text of a label

SYNOPSIS

```
#include <gui.h>
void gui_label_change_text(gui_label la-
bel,char *new_text);
```

DESCRIPTIONS

The `gui_label_change_text` changes the text of a label.

PARAMETERS

- `label`: Specifies a label
- `new_text`: Specifies the new text for the label

NAME

`gui_label_hide`, `gui_label_show`, `gui_label_is_visible` - get or set the visibility of a label

SYNOPSIS

```
#include <gui.h>
void gui_label_hide(gui_label label);
void gui_label_show(gui_label label);
bool gui_label_is_visible(gui_label label);
```

DESCRIPTION

When a label is created, it is visible.
`gui_label_hide` makes a label disappear.
`gui_label_show` makes a label appear.
`gui_label_is_visible` gets the visibility of a label.
When a label is hidden, it cannot produce events.

PARAMETERS

- o `label`: Specifies a label.

RETURN VALUE

`gui_label_is_visible` returns `true` if the label is visible, and `false` otherwise.

NAME

gui_label_align_left, gui_label_align_center,
gui_label_align_right - change the alignment of a label

SYNOPSIS

```
#include <gui.h>
void gui_label_align_left(gui_label label);
void gui_label_align_center(gui_label label);
void gui_label_align_right(gui_label label);
```

DESCRIPTIONS

The position of the label position is determined by a point, that has been defined at the creation of the label. By default, this point is the left hand extremity of the text line. It can be changed, and become the center point, or the right hand extremity. This transformation doesn't move the point position, but the label position referring to the point.

PARAMETERS

- o label: Specifies a label.

SEE ALSO

gui_label_new

1.1.8 Lines

```
gui_line  
gui_line_new  
gui_line_delete  
gui_line_change_x2y2  
gui_line_get_x1  
gui_line_get_y1  
gui_line_get_x2  
gui_line_get_y2  
gui_line_show  
gui_line_hide  
gui_line_is_visible
```

TYPE

gui_line - C type representing a line

SYNOPSIS

```
#include <gui.h>  
gui_line line;
```

NAME

`gui_line_new`, `gui_line_delete` - create or delete a line

SYNOPSIS

```
#include <gui.h>
gui_line gui_line_new(gui_window window,
                      int16 x1,int16 y1,
                      int16 x2,int16 y2,uint32 color);
void      gui_line_delete(gui_line line);
```

DESCRIPTION

`gui_line_new` creates a new line in a window. This line is created visible. The line is defined by 2 points : P1(x1, y1), P2(x2, y2) and a color.

`gui_line_delete` deletes a line. Note that a line is automatically deleted when its window is deleted.

PARAMETERS

`gui_line_new`

- `window`: Specifies the window, in which the line is created.
- `x1`, `y1`: Specify the x and y coordinates of P1, the first extremity of the line, relative to the origin of window.
- `x2`, `y2`: Specify the x and y coordinates of P2, the second extremity of the line, relative to the origin of window.
- `color`: Specifies the color of the line.

`gui_line_delete`

- `line`: Specifies a line

RETURN VALUE

`gui_line_new` returns the `gui_line` just created.

SEE ALSO

`gui_window_delete`, `gui_color_get`

NAME

gui_line_change_color - change the color of a line

SYNOPSIS

```
#include <gui.h>
void gui_line_change_color(gui_line line,uint32 color);
```

DESCRIPTION

gui_line_change_color changes the color of a line. It causes the line to be redrawn

PARAMETERS

- o line: Specifies a line.
- o color: Specifies the new color of the line.

SEE ALSO

gui_color_get

NAME

gui_line_hide, gui_line_show, gui_line_is_visible
- get or set the visibility of a line

SYNOPSIS

```
#include <gui.h>
void gui_line_hide(gui_line line);
void gui_line_show(gui_line line);
bool gui_line_is_visible(gui_line line);
```

DESCRIPTION

When a line is created, it is visible.
gui_line_hide makes a line disappear.
gui_line_show makes a line appear.
gui_line_is_visible gets the visibility of a line.

PARAMETERS

- o line: Specifies a line.

RETURN VALUE

gui_line_is_visible returns true if the line is visible and false otherwise.

NAME

gui_line_get_x1, gui_line_get_x2,
gui_line_get_y1, gui_line_get_y2
- get the coordinates of a line

SYNOPSIS

```
#include <gui.h>
int16 gui_line_get_x1(gui_line line);
int16 gui_line_get_x2(gui_line line);
int16 gui_line_get_y1(gui_line line);
int16 gui_line_get_y2(gui_line line);
```

DESCRIPTION

These functions get the position of the two points P1(x1,y1) and P2(x2,y2) that define a line.

PARAMETERS

- o line: Specifies a line.

RETURN VALUE

gui_line_get_x1 returns the x coordinate of the first point.
gui_line_get_y1 returns the y coordinate of the first point.
gui_line_get_x2 returns the x coordinate of the second point.
gui_line_get_y2 returns the y coordinate of the second point.

NAME

gui_line_change_x2y2 - change one extremity of a line

SYNOPSIS

```
#include <gui.h>
void gui_line_change_x2y2(gui_line line,int16 x2,int16 y2);
```

DESCRIPTION

gui_line_change_x2y2 changes one extremity of a line, and redraws it.

PARAMETERS

- line: Specifies a line.
- x2, y2: Specify the new x and y coordinates of P2, the second extremity of the line, relative to the origin of window.

SEE ALSO

gui_line_new

1.1.9 Miscellaneous

```
gui_beep  
gui_message  
gui_warning  
gui_screen_get_width  
gui_screen_get_height  
gui_set_image_path  
gui_exit_function
```

NAME

gui_beep() - ring the bell

SYNOPSIS

```
#include <gui.h>
void gui_beep();
```

DESCRIPTION

This function rings the system bell.

NAME

`gui_message`, `gui_warning` - display a modal dialog box

SYNOPSIS

```
#include <gui.h>
void gui_message(char *string);
bool gui_warning(char *string);
```

DESCRIPTION

These functions can be used to display two kinds of dialog box:

The `gui_message` function displays a dialog box with only an ``Ok'' button. It returns when the button is pressed or the window is closed:



The `gui_warning` function displays a dialog box with 2 buttons : ``Ok'' and ``Cancel'':



RETURN VALUE

`gui_warning` returns `true` if the ‘‘Ok’’ button is pressed and `false` if the ‘‘Cancel’’ button is pressed or the window is closed.

NAME

`gui_screen_get_width`, `gui_screen_get_height`

- return the width and the height of the screen.

SYNOPSIS

```
#include <gui.h>
uint16 gui_screen_get_width();
uint16 gui_screen_get_height();
```

DESCRIPTION

These functions return the width and the height of the screen in pixels.

RETURN VALUES

`gui_screen_get_width` returns the width of the screen in pixels .

`gui_screen_get_height` returns the height of the screen in pixels.

NAME

gui_set_image_path - set the image path

SYNOPSIS

```
#include <gui.h>
void gui_set_image_path();
```

DESCRIPTION

This function sets the path where image files can be found. This path must be set appropriately before using `gui_image_new` or `gui_pixmap_new`, so that PNG images are searched in that directory.

SEE ALSO

`gui_image_new`, `gui_pixmap_new`

EXAMPLE

```
set_image_path(``/home/john/images/png``); /* all my PNG images are in this directory */
```

1.1.10 Pixmaps

```
gui_pixmap  
gui_pixmap_delete  
gui_pixmap_new  
gui_pixmap_get_width  
gui_pixmap_get_height
```

TYPE

`gui_pixmap` - C type representing a pixmap

SYNOPSIS

```
#include <gui.h>  
gui_pixmap pixmap;
```

DESCRIPTION

A pixmap is an image stored in memory. Using a pixmap may be a convenient way to create several images using the same image file. This file can be loaded in a pixmap, then images can be created from with this pixmap using `gui_image_new_from_pixmap` without additional memory allocation. Note: a pixmap must not be deleted before deleting the image(s) using this pixmap.

SEE ALSO

`gui_image_new_from_pixmap`

NAME

`gui_pixmap_new`, `gui_pixmap_delete`
- create or delete a pixmap

SYNOPSIS

```
#include <gui.h>
gui_pixmap gui_pixmap_new(char *png_file);
void gui_pixmap_delete(gui_pixmap pixmap);
```

DESCRIPTION

`gui_pixmap_new` creates a pixmap from a file.
`gui_pixmap_delete` deletes a pixmap. This function must not be called before deleting the images created from this pixmap.

PARAMETERS

`gui_pixmap_new`

- `png_file`: Specify a PNG file name, the path to find the images must be set before by using the `gui_set_image_path` function.

`gui_pixmap_delete`

- `pixmap`: Specifies a pixmap.

RETURN VALUE

`gui_pixmap_new` returns the `gui_pixmap` just created.

SEE ALSO

`gui_image_new_from_pixmap`, `gui_set_image_path`

NAME

`gui_pixmap_get_width`, `gui_pixmap_get_height`
- get the dimensions of a pixmap

SYNOPSIS

```
#include <gui.h>
uint16 gui_pixmap_get_width(gui_pixmap pixmap);
uint16 gui_pixmap_get_height(gui_pixmap pixmap);
```

DESCRIPTION

These functions get the dimensions of a pixmap.

PARAMETERS

- o `pixmap`: Specifies a pixmap.

RETURN VALUES

`gui_pixmap_get_width` returns the width of the pixmap in pixels.
`gui_pixmap_get_height` returns the height of the pixmap in pixels.

1.1.11 Popup menus

```
gui_popup  
gui_popup_new  
gui_popup_delete  
gui_popup_change_text  
gui_popup_enable  
gui_popup_disable  
gui_popup_is_enabled  
gui_popup_show  
gui_popup_hide  
gui_popup_is_visible  
gui_popup_get_selection  
gui_popup_get_selected_text  
gui_popup_set_selection  
gui_popup_set_width
```

TYPE

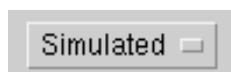
gui_popup - C type representing a popup menu

SYNOPSIS

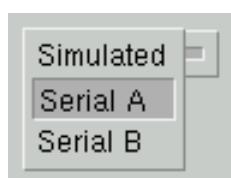
```
#include <gui.h>  
gui_popup menu;
```

SCREENSHOTS

a popup menu:



an open popup menu:



NAME

`gui_popup_new`, `gui_popup_delete`

- create or delete a popup menu

SYNOPSIS

```
#include <gui.h>
gui_popup gui_popup_new(gui_window window,
                        int16 x,int16 y,uint16 nb_item,
                        char *item[],uint16 selection);
void gui_popup_delete(gui_popup popup);
```

DESCRIPTION

`gui_popup_new` creates a popup menu in a window. It is defined by its the upper-left corner, a list of items and the default selected item.

`gui_popup_delete` deletes a popup menu. Note that a popup menu is automatically deleted when its window is deleted.

PARAMETERS

`gui_popup_new`

- `window`: Specifies the window, in which the popup menu is created.
- `x`, `y`: Specify the `x` and `y` coordinates of the upper-left corner to the popup menu, relative to the origin of window.
- `nb_item`: Specifies the number of items in the popup menu.
- `item`: Specify the items of the popup menu.
- `selection`: Specify the default selected item. This value ranges from 0 to `nb_item - 1`.

`gui_popup_delete`

- `popup`: Specifies a popup menu.

RETURN VALUE

`gui_popup_new` returns the `gui_popup` just created.

SEE ALSO

`gui_window_delete`

NAME

gui_popup_disable, gui_popup_enable, gui_popup_is_enabled - set and get the enable state of a popup menu

SYNOPSIS

```
#include <gui.h>
void gui_popup_disable(gui_popup popup);
void gui_popup_enable(gui_popup popup);
bool gui_popup_is_enabled(gui_popup popup);
```

DESCRIPTION

gui_popup_disable makes the popup disabled.

gui_popup_enable makes the popup enabled.

gui_popup_is_enabled checks whether a popup is enabled or disabled.

When a popup is disabled, it cannot produce events.

PARAMETERS

- o **popup:** Specifies a popup menu.

RETURN VALUE

gui_popup_is_enabled returns `true` if the popup menu is enabled and `false` otherwise.

NAME

`gui_popup_hide`, `gui_popup_show`, `gui_popup_is_visible`
- set and get the visibility of a popup menu

SYNOPSIS

```
#include <gui.h>
void gui_popup_hide(gui_popup popup);
void gui_popup_show(gui_popup popup);
bool gui_popup_is_visible(gui_popup popup);
```

DESCRIPTION

When a popup menu is created, it is visible.
`gui_popup_hide` makes a popup menu disappear.
`gui_popup_show` makes a popup to appear.
`gui_popup_is_visible` gets the visibility of a popup menu.
When a popup is hidden, it cannot produce events.

PARAMETERS

- o `popup`: Specifies a popup menu.

RETURN VALUE

`gui_popup_is_visible` returns `true` if the poupu menu is visible and `false` otherwise.

NAME

gui_popup_change_text - change the text of an item in a popup menu

SYNOPSIS

```
#include <gui.h>
void gui_popup_change_text(gui_popup popup,uint16 item,
char *text);
```

DESCRIPTION

gui_popup_change_text replaces the text of an item in a popup menu.

PARAMETERS

- **popup:** Specifies a popup menu.
- **item:** Specifies the item position in the menu. This position ranges between 0 and nb_item - 1.
- **text:** Specifies the new text.

NAME

gui_popup_get_selection, gui_popup_get_selected_text
- get the selected item

SYNOPSIS

```
#include <gui.h>
uint16 gui_popup_get_selection(gui_popup popup);
char*  gui_popup_get_selected_text(gui_popup popup);
```

DESCRIPTION

These functions get the current selection of the popup menu.

RETURN VALUES `gui_popup_get_selection` returns the position of the item currently selected. It ranges between 0 and `nb_item - 1`.

`gui_popup_get_selected_text` returns the text of the item currently selected.

PARAMETERS

- `popup`: Specifies a popup menu.

NAME

gui_popup_set_selection - set the selected item in a popup menu.

SYNOPSIS

```
#include <gui.h>
void gui_popup_set_selection(gui_popup popup,uint16 item);
```

DESCRIPTION

gui_popup_set_selection changes the selected item in a popup menu.

PARAMETERS

- **popup:** Specifies a popup menu
- **item:** Specifies the position of the item in the popup menu. This position ranges between 0 and nb_item - 1.

NAME

gui_popup_set_width - set the width of a popup menu

SYNOPSIS

```
#include <gui.h>
void gui_popup_set_width(gui_popup popup, uint16 width);
```

DESCRIPTION

The default popup menu width is computed from the width of the longest item text. `gui_popup_set_width` allows you to change of this default width.

PARAMETERS

- `popup`: Specifies a popup menu.
- `width`: Specifies the width in pixels.

SEE ALSO

`gui_popup_new`

1.1.12 Rectangles

```
gui_rectangle  
gui_rectangle_new  
gui_rectangle_delete  
gui_rectangle_show  
gui_rectangle_hide  
gui_rectangle_is_visible  
gui_rectangle_change_color
```

TYPE

gui_rectangle - C type representing a rectangle

SYNOPSIS

```
#include <gui.h>  
gui_rectangle rectangle;
```

NAME

`gui_rectangle_new`, `gui_rectangle_delete`
- create or delete a rectangle

SYNOPSIS

```
#include <gui.h>
gui_rectangle gui_rectangle_new(gui_window window,
                               int16 x,int16 y,
                               uint16 width,uint16 height,
                               uint32 color,bool filled);
void gui_rectangle_delete(gui_rectangle rectangle);
```

DESCRIPTION

`gui_rectangle_new` creates a rectangle in a window. The rectangle is defined by its upper-left corner, its width, its height, the color, and a boolean value defining if it is empty or filled. `gui_delete_rectangle` deletes a rectangle. A rectangle is automatically deleted when its window is deleted.

PARAMETERS

`gui_rectangle_new`

- `window`: Specifies the window, in which the rectangle is created.
- `x`, `y`: Specify the `x` and `y` coordinates of the upper-left corner of the rectangle, relative to the origin of window.
- `height`, `width`: Specify the width and height of the rectangle in pixels.
- `color`: Specifies the color of the rectangle.
- `filled`: Specifies whether the rectangle is filled (`true`) or not (`false`).

`gui_popup_delete`

- `rectangle`: Specifies a rectangle.

RETURN VALUE

`gui_rectangle_new` returns the `gui_rectangle` just created.

SEE ALSO

`gui_window_delete`, `gui_color_get`

NAME

gui_rectangle_hide, gui_rectangle_show,
gui_rectangle_is_visible - get or set the visibility of a rectangle

SYNOPSIS

```
#include <gui.h>
void gui_rectangle_hide(gui_rectangle rectangle);
void gui_rectangle_show(gui_rectangle rectangle);
bool gui_rectangle_is_visible(gui_rectangle rectangle);
```

DESCRIPTION

When a rectangle is created, it is visible.
gui_rectangle_hide makes a rectangle disappear.
gui_rectangle_show makes a rectangle appear.
gui_rectangle_is_visible gets the visibility of a rectangle.

PARAMETERS

- o rectangle: Specifies a rectangle.

RETURN VALUE

gui_rectangle_is_visible returns `true` if the rectangle is visible and `false` otherwise.

NAME

gui_rectangle_change_color - change the color of a rectangle

SYNOPSIS

```
#include <gui.h>
void gui_rectangle_change_color(gui_rectangle rectangle,
                               uint32 color);
```

DESCRIPTION

gui_rectangle_change_color changes the color of a rectangle. The rectangle is redrawn.

PARAMETERS

- rectangle: Specifies a rectangle.
- color: Specifies the new color for the rectangle.

SEE ALSO

gui_color_get

1.1.13 Textfields

```
gui_textfield  
gui_textfield_new  
gui_textfield_delete  
gui_textfield_activate  
gui_textfield_desactivate  
gui_textfield_is_active  
gui_textfield_get_text  
gui_textfield_set_text  
gui_textfield_enable  
gui_textfield_disable  
gui_textfield_is_enabled  
gui_textfield_show  
gui_textfield_hide  
gui_textfield_is_visible
```

TYPE

gui_textfield – C type representing a textfield

SYNOPSIS

```
#include <gui.h>  
gui_textfield textfield;
```

DESCRIPTIONS

A gui_textfield is a text area, in which the user can edit a line of text.

SEE ALSO

gui_label

SCREENSHOT



NAME

`gui_textfield_new`, `gui_textfield_delete` – create or delete a textfield

SYNOPSIS

```
#include <gui.h>
gui_textfield gui_textfield_new(gui_window window,
                                int16 x,int16 y,
                                char *text,uint16 size);
void gui_textfield_delete(gui_textfield textfield);
```

DESCRIPTIONS

`gui_textfield_new` creates a textfield.

`gui_textfield_delete` deletes a textfield. Note that a textfield is automatically deleted when its window is deleted.

PARAMETERS

`gui_textfield_new`

- `window`: Specifies the window, in which the textfield is created.
- `x`, `y`: Specify the x and y coordinates of the upper-left corner of the textfield, relative to the origin of window.
- `text`: Specifies the default text in the textfield.
- `size`: Specifies the maximum number of characters that can be written and displayed in the textfield.

`gui_textfield_delete`

- `textfield`: Specifies a textfield.

RETURN VALUE

`gui_textfield_new` returns the `gui_textfield` just created.

SEE ALSO

`gui_window_delete`

NAME

gui_textfield_activate, gui_textfield_desactivate,
gui_textfield_is_active - activate or deactivate a textfield

SYNOPSIS

```
#include <gui.h>
void gui_textfield_activate(gui_textfield textfield);
void gui_textfield_desactivate(gui_textfield textfield);
bool gui_textfield_is_active(gui_textfield textfield);
```

DESCRIPTION

gui_textfield_activate activates the cursor in the textfield.
gui_textfield_desactivate removes the cursor from the textfield.
gui_textfield_is_active checks the activation of a textfield.

PARAMETERS

- o textfield: Specifies a textfield.

RETURN VALUE

gui_textfield_is_active returns true if the cursor is active in the textfield and false otherwise.

NAME

gui_textfield_get_text, gui_textfield_set_text
- get or set the text in a textfield

SYNOPSIS

```
#include <gui.h>
char* gui_textfield_get_text(gui_textfield textfield);
void gui_textfield_set_text(gui_textfield textfield, char *text);
```

DESCRIPTION

gui_textfield_get_text returns the text of the textfield.
gui_textfield_set_text sets the text of the textfield. If the text is longer than the textfield size, it is truncated.

PARAMETERS

gui_textfield_get_text

- o textfield: Specifies a textfield.

gui_textfield_set_text

- o textfield: Specifies a textfield.
- o text: Specifies the new text .

SEE ALSO

gui_textfield_new

NAME

gui_textfield_enable, gui_textfield_disable,
gui_textfield_is_enabled - set and get the enable state of a textfield

SYNOPSIS

```
#include <gui.h>
void gui_textfield_enable(gui_textfield textfield);
void gui_textfield_disable(gui_textfield textfield);
bool gui_textfield_is_enabled(gui_textfield textfield);
```

DESCRIPTION

gui_textfield_enable makes a textfield enabled.
gui_textfield_disable makes a textfield disabled.
gui_textfield_is_enabled checks whether a textfield is enabled.
When a textfield is disabled, it cannot produce events.

PARAMETERS

- o textfield: Specifies a textfield.

RETURN VALUE

gui_textfield_is_enabled returns `true` if the textfield is enabled and `false` otherwise.

NAME

gui_textfield_hide, gui_textfield_show,
gui_textfield_is_visible - set or get the visibility of a textfield

SYNOPSIS

```
#include <gui.h>
void gui_textfield_hide(gui_textfield textfield);
void gui_textfield_show(gui_textfield textfield);
bool gui_textfield_is_visible(gui_textfield textfield);
```

DESCRIPTION

When a textfield is created, it is visible.
gui_textfield_hide makes a textfield disappear.
gui_textfield_show makes a textfield to appear.
gui_textfield_is_visible gets the visibility of a textfield.
When a textfield is hidden, it cannot produce events.

PARAMETERS

- o textfield: Specifies a textfield.

RETURN VALUE

gui_textfield_is_visible returns true if the textfield is visible and false otherwise.

1.1.14 Windows

```
gui_window  
gui_window_new  
gui_window_delete  
gui_window_allow_resize  
gui_window_allow_all_resize  
gui_window_forbid_resize  
gui_window_forbid_all_resize  
gui_window_set_min_max_size  
gui_window_set_max_size  
gui_window_show  
gui_window_hide  
gui_window_is_visible  
gui_window_set_title  
gui_window_set_icon_title  
gui_window_set_size  
gui_window_set_position  
gui_window_get_size  
gui_window_get_position
```

TYPE

gui_window - C type representing a window

SYNOPSIS

```
#include <gui.h>  
gui_window window;
```

NAME

gui_window_new, gui_window_delete

- create or delete a window

SYNOPSIS

```
#include <gui.h>
gui_window gui_window_new(char *title,int16 x,int16 y,
                           uint16 width,uint16 height);
void gui_window_delete(gui_window window);
```

DESCRIPTION

gui_window_new creates a window. However, the window is not immediately displayed. gui_window_show should be called to make the window appear. This way, the window can be prepared by adding graphical objects and widgets before its is shown. The window is defined by its upper-left corner, its width, and its height. The x and y coordinates of the upper-left corner are only a request to the window manager, that can set the new window anywhere on the screen. gui_window_delete deletes the window and releases the associated memory.

PARAMETERS

gui_window_new

- title: Specifies window title, that will appear in the banner and below the icon.
- x, y: Specify the requested x and y coordinates of the upper-left corner of the window relative to the origin of the screen.
- width, height: Specify width and height of the window (in pixels).

gui_window_delete

- window: Specifies a window.

RETURN VALUE

gui_window_new returns the gui_window just created.

SEE ALSO

gui_window_show, gui_window_set_title, gui_window_set_icon_title

NAME

gui_window_forbid_resize, gui_window_forbid_all_resize
- forbid resizing of windows

SYNOPSIS

```
#include <gui.h>
gui_window_forbid_resize(gui_window window);
gui_window_forbid_all_resize();
```

DESCRIPTION

gui_window_forbid_resize forbids the resizing of a window.
gui_window_forbid_all_resize forbids the resizing of any windows in the current application.

PARAMETERS

gui_window_forbid_resize

- o window: Specifies a window.

SEE ALSO

gui_window_allow_resize, gui_window_allow_all_resize,
gui_window_set_min_max_size, gui_window_set_max_size

NAME

gui_window_allow_resize, gui_window_allow_all_resize
- allow users to resize windows

SYNOPSIS

```
#include <gui.h>
gui_window_allow_resize(gui_window window);
gui_window_allow_all_resize();
```

DESCRIPTION

gui_window_allow_resize allows users to resize the specified window.
gui_window_allow_all_resize allows users to resize of any windows in the current application.

PARAMETERS

gui_window_allow_resize

- o window: Specifies a window .

SEE ALSO

gui_window_forbid_resize, gui_window_forbid_all_resize,
gui_window_set_min_max_size, gui_window_set_max_size

NAME

gui_window_set_max_size, gui_window_set_min_max_size
- set range for resizing a window

SYNOPSIS

```
#include <gui.h>
void gui_window_set_max_size(gui_window window,
                             uint16 width,uint16
                             height);
void gui_window_set_min_max_size(gui_window window,
                                 uint16 min_w,uint16
                                 min_h,
                                 uint16 max_w,uint16
                                 max_h);
```

DESCRIPTION

gui_window_set_max_size sets the maximum value for the window width and height.
gui_window_set_min_max_size sets the maximum and the minimum values for the window width and height.

PARAMETERS

gui_window_set_max_size

- **window:** Specifies a window.
- **width, height:** Specify the width and the height of the window (in pixels).

gui_window_set_max_size

- **window:** Specifies a window.
- **min_w, min_h:** Specify the minimal width and height for the window in pixels.
- **max_w, max_h:** Specify the maximal width and height for the window in pixels.

SEE ALSO

gui_window_allow_resize, gui_window_allow_all_resize,
gui_window_forbid_resize, gui_window_forbid_all_resize

NAME

`gui_window_show`, `gui_window_hide`,
`gui_window_is_visible` - set and get the visibility of a window

SYNOPSIS

```
#include <gui.h>
void gui_window_show(gui_window window);
void gui_window_hide(gui_window window);
bool gui_window_is_visible(gui_window window);
```

DESCRIPTION

When a window is created, it is not visible.

`gui_window_show` makes a window appear.

`gui_window_hide` makes a window disappear.

`gui_window_is_visible` returns the visibility of a window.

When a window is hidden, it cannot produce events.

PARAMETERS

- `window`: Specifies a window.

RETURN VALUE

`gui_window_is_visible` returns `true` if the window is visible and `false` otherwise.

NAME

`gui_window_set_title`, `gui_window_set_icon_title`
- set the titles for a window and its icon

SYNOPSIS

```
#include <gui.h>
void gui_window_set_title(gui_window window,
                           char *title);
void gui_window_set_icon_title(gui_window window,
                               char *title);
```

DESCRIPTION

`gui_window_set_title` changes the title of a window.

`gui_window_set_icon_title` changes the title of the icon of a window.

These titles are initialized with the same value when a window is created by `gui_window_new`.

PARAMETERS

`gui_window_set_title`

- `window`: Specifies a window.
- `title`: Specify the new title of the window.

`gui_window_set_icon_title`

- `window`: Specifies a window .
- `title`: Specify the new title of the window icon.

SEE ALSO

`gui_window_new`

NAME

`gui_window_get_size`, `gui_window_set_size`
- get and set the dimensions of a window

SYNOPSIS

```
#include <gui.h>
void gui_window_get_size(gui_window window,
                         uint16 *width,uint16 *height);
void gui_window_set_size(gui_window window,
                         uint16 width,uint16 height);
```

DESCRIPTION

`gui_window_get_size` gets the window dimensions.
`gui_window_set_size` changes the window dimensions. The upper-left corner position remains unchanged.

PARAMETERS

`gui_window_get_size`

- `window`: Specifies a window
- `width`, `height`: Receive the dimensions of the window in pixels

`gui_window_set_position`

- `window`: Specifies a window
- `width`, `height`: Specify the new dimensions of the window in pixels

SEE ALSO

`gui_window_get_position`, `gui_window_set_position`

NAME

`gui_window_get_position`, `gui_window_set_position`
- get and set the position of a window

SYNOPSIS

```
#include <gui.h>

void gui_window_get_position(gui_window window,
    int16 *x,int16 *y);
void gui_window_set_position(gui_window window,
    int16 x,int16 y);
```

DESCRIPTION

`gui_window_get_position` gets the position of a window.
`gui_window_set_position` changes the position of a window.

PARAMETERS

`gui_window_get_position`

- `window`: Specifies a window.
- `x`, `y`: Receive the `x` and `y` coordinates of the upper-left corner of the window relative to the origin of the screen.

`gui_window_set_position`

- `window`: Specifies a window
- `x`, `y`: Specify the `x` and `y` coordinates of the upper-left corner of the window relative to the origin of the screen.

SEE ALSO

`gui_window_get_size`, `gui_window_set_size`

Chapter 2

Khepera API Reference

2.1 Khepera

```
khepera_die  
khepera_disable_jumper  
khepera_disable_light,  
khepera_disable_position  
khepera_disable_proximity  
khepera_disable_speed  
khepera_enable_jumper  
khepera_enable_light  
khepera_enable_position  
khepera_enable_proximity  
khepera_enable_speed  
khepera_get_jumper  
khepera_get_light  
khepera_get_proximity  
khepera_get_position  
khepera_get_speed  
khepera_live  
khepera_receive_serial  
khepera_send_serial  
khepera_set_led  
khepera_set_position  
khepera_set_speed  
khepera_step
```

NAME

khepera_die - set the exit function for a Khepera controller

SYNOPSIS

```
#include <Khepera.h>
void khepera_die(void (*)());
```

DESCRIPTION

This function declares an exit function to be called automatically when the controller exits. This might be useful to perform some clean-up before quitting, i.e., save data to a file, print out some statistics, etc. The exit function should be a void function without any argument.

EXAMPLE

```
void my_exit_function()
{
    printf(``bye bye!\n'');
}

int main()
{
    khepera_live(); /* now ! */
    khepera_die(my_exit_function); /* when it quits */
    for(;;) { khepera_step(64); }
    return 0; /* this statement will never be reached */
}
```

SEE ALSO

[khepera_die](#)

[khepera_live](#)
[khepera_step](#)

NAME

khepera_disable_jumper, khepera_disable_light,
khepera_disable_position, khepera_disable_proximity,
khepera_disable_speed, khepera_enable_jumper,
khepera_enable_light, khepera_enable_position,
khepera_enable_proximity, khepera_enable_speed

- control the sensors of Khepera

SYNOPSIS

```
#include <Khepera.h>

void khepera_disable_jumper();
void khepera_disable_light(uint8 n);
void khepera_disable_position(uint8 m);
void khepera_disable_proximity(uint8 n);
void khepera_disable_speed(uint8 m);
void khepera_enable_jumper();
void khepera_enable_light(uint8 n);
void khepera_enable_position(uint8 m);
void khepera_enable_proximity(uint8 n);
void khepera_enable_speed(uint8 m);
```

DESCRIPTION

These functions allow one to disable or enable the reading of various sensor values on the Khepera robot. These sensor values include speed and position for each motor, light and proximity values for each of the 8 SFH900 sensors, and jumper configuration. Disabling sensor values results in a performance increase, but the corresponding sensor values become irrelevant. Enabling and disabling of sensors should occur before calling the `khepera_step` function.

n specifies the infra-red sensor. It can take the value `IR_SENSOR0`, `IR_SENSOR1`, `IR_SENSOR2`, `IR_SENSOR3`, `IR_SENSOR4`, `IR_SENSOR5`, `IR_SENSOR6`, `IR_SENSOR7` or `IR_ALL_SENSORS`. These values may be ORed to specify combinations of sensors: `IR_ALL_SENSORS` is equivalent to: `IR_SENSOR0 | IR_SENSOR1 | IR_SENSOR2 | IR_SENSOR3 | IR_SENSOR4 | IR_SENSOR5 | IR_SENSOR6 | IR_SENSOR7`.

m specifies the motor associated with the speed or position measurement sensor. It can be `LEFT`, `RIGHT`, or `LEFT_AND_RIGHT`.

Successive enablings (and disabling) are possible:

```
khepera_enable_proximity( IR_SENSOR0 ) ;  
khepera_enable_proximity( IR_SENSOR2 ) ;
```

is equivalent to

```
khepera_enable_proximity( IR_SENSOR0 | IR_SENSOR2 ) ;
```

By default, all sensor values are disabled, which is equivalent to the following calls:

```
khepera_disable_jumper( ) ;  
khepera_disable_light( IR_ALL_SENSORS ) ;  
khepera_disable_position( LEFT_AND_RIGHT ) ;  
khepera_disable_proximity( IR_ALL_SENSORS ) ;  
khepera_disable_speed( LEFT_AND_RIGHT ) ;
```

NAME

`khepera_get_jumper` - return the state of the jumper switches.

SYNOPSIS

```
#include <Khepera.h>
uint8 khepera_get_jumper();
```

DESCRIPTION

This function returns the state of the jumper switches situated on the top left of the Khepera robot.

RETURN VALUES

The `khepera_get_jumper` function returns a value ranging from 0 to 7 corresponding to the mode in which the jumpers are set. This value may be interpreted using combinations of the following constants: `OUTER_JUMPER`, `MIDDLE_JUMPER`, and `INNER_JUMPER`, which can be ORed to get values between 0 and 7. A return value of 0 indicates that no jumper is set, while a value equal to an OR combination of the `JUMPER` constants indicates that the corresponding jumpers are set.

EXAMPLE

The following snippet of code tests whether the inner and outer jumpers are set and the middle jumper is not set:

```
if (khepera_get_jumper() == IN-
NER_JUMPER | OUTER_JUMPER) { ... }
```

SEE ALSO

`khepera_enable_jumper`
`khepera_disable_jumper`

NAME

`khepera_get_light`, `khepera_get_proximity` - return a value measured by an infra-red sensor

SYNOPSIS

```
#include <Khepera.h>
uint16 khepera_get_light(uint8 n);
uint16 khepera_get_proximity(uint8 n);
```

DESCRIPTION

These functions return the values measured by one of the 8 SFH900 sensors available on the basic Khepera robot.

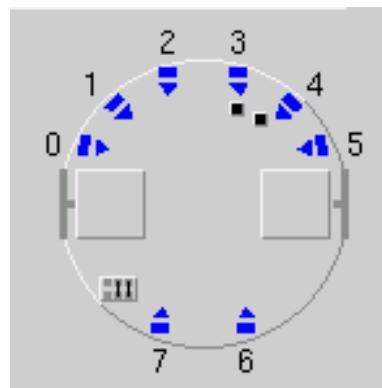
`n` specifies the number of the infra-red sensor. It must range between 0 and 7.

RETURN VALUES

The `khepera_get_light` function returns an integer value ranging from 0 to 512 corresponding to the response of the sensor in ambient light mode. A value close to 512 indicates a low infra-red luminosity while a value close to 0 indicates a high infra-red luminosity.

The `khepera_get_proximity` function returns an integer value ranging from 0 to 1023, corresponding to the response of the sensor in reflection mode. This value indicates the proximity of objects in front of the sensor. A value close to 1023 indicates that an object is close to the sensor while a value close to 0 indicates that no object is detected in front of the sensor.

The positions of the infra-red sensors on the Khepera robot are represented in the following figure:



Sensors 1 and 4 are oriented at plus or minus 45 degrees relatively to the forward direction of the robot.

SEE ALSO

`khepera_enable_light`
`khepera_enable_proximity`
`khepera_disable_light`
`khepera_disable_proximity`

NAME

`khepera_get_position`, `khepera_get_speed` - return motor wheel properties

SYNOPSIS

```
#include <Khepera.h>
int32 khepera_get_position(uint8 m);
int8 khepera_get_speed(uint8 m);
```

DESCRIPTION

These functions allow one to read the values measured by the odometry sensors of each motor on the Khepera robot. The `m` parameter specifies the motor. It must be set to `RIGHT` or `LEFT`.

RETURN VALUES

`khepera_get_position` returns the position of the specified motor wheel. The unit is the pulse, which corresponds to 0.08 mm.

`khepera_get_speed` returns the speed of the specified motor wheel. The speed unit is the pulse/10 ms that corresponds to 8 mm/s.

SEE ALSO

`khepera_enable_position`
`khepera_enable_speed`
`khepera_disable_position`
`khepera_disable_speed`

NAME

khepera_live - initialize the controller program

SYNOPSIS

```
#include <Khepera.h>
void khepera_live();
```

DESCRIPTION

This function initializes the Khepera. The call to khepera_live is mandatory before any subsequent call to khepera_step.

SEE ALSO

khepera_die
khepera_step

NAME

`khepera_receive_serial`, `khepera_send_serial` - exchange data with the supervisor

SYNOPSIS

```
#include <Khepera.h>
uint32 khepera_receive_serial(uint32 max_size,
                               char *buffer);
void khepera_send_serial(uint32 size,char *buffer);
```

DESCRIPTION

These functions are used to communicate with the supervisor through a serial character stream. Note that these functions are not blocking calls. If not data is available when calling `khepera_receive_serial`, the function returns 0. Similarly, `khepera_send_serial` doesn't wait for the data to be read by the supervisor, instead, the function returns immediatly. Similar functions (`eai_read_stream` and `eai_write_stream`) should be used on the supervisor side.

`khepera_receive_serial` reads a maximum of `max_size` bytes coming from the supervisor `eai_write_stream` function. The data is stored in the `buffer` area which should be big enough to contain at least `max_size` bytes. The function returns the number of bytes actually read from the serial stream.

`khepera_send_serial` writes `size` bytes of data to the serial stream to be read by the supervisor `eai_read_stream` function. The data is read from the `buffer` pointer.

SEE ALSO

`eai_read_stream`
`eai_write_stream`

EXAMPLE

```
char buffer[256];
int8 parameter1;
if (khepera_receive_serial(256,buffer)>0)
{ /* we received a message from the supervisor */
  if (buffer[0]=='R')
```

```
/* the supervisor asks to reset */
{
    parameter1 = buffer[1];
    /* this parameter is passed to the supervisor */
    ...
}
...
}
```

NAME

khepera_set_led - switch on or off an LED of the Khepera

SYNOPSIS

```
#include <Khepera.h>
void khepera_set_led(uint8 n,uint8 value)
```

DESCRIPTION

This function allows one to switch on or off the LEDs of the Khepera.

n specifies the LED identifier. It must be set to one of the values specified in the following table:

identifier	LED
LEFT	left front LED
RIGHT	right side LED
LEFT_AND_RIGHT	both LEDs

LED identifiers for the Khepera robot

value specifies the state of the LED and can be assigned one of the following constants: ON, OFF.

NAME

`khepera_set_position`, `khepera_set_speed`

- set the parameters for a motor wheel

SYNOPSIS

```
#include <Khepera.h>
void khepera_set_position(uint8 n,int32 position)
void khepera_set_speed(uint8 n,int8 speed)
```

DESCRIPTION

These functions allow setting of the parameters of the motor wheels of a Khepera robot.

`khepera_set_position` allows setting of the position of the odometry counter of the motor wheels.

`khepera_set_speed` allows setting of the speed of the motor wheels.

`n` specifies the motor wheel. It must be set according to the following table:

identifier	motor wheel
LEFT	left motor wheel
RIGHT	right motor wheel
LEFT_AND_RIGHT	both motor wheels

Khepera motor wheel identifiers

`speed` specifies the speed of the motor wheel. It must be between -20 and 20. The unit of speed is 8 mm/s.

`position` specifies the value to be assigned to the odometry counter. The unit of position is one pulse, corresponding to 0.08 mm.

EXAMPLE

The following snippet of code will make the Khepera robot spin round like a top:

```
khepera_set_speed(LEFT,-10);
khepera_set_speed(RIGHT,10);
```

NAME

khepera_step - proceed one simulation step

SYNOPSIS

```
#include <Khepera.h>
void khepera_step(uint32 ms);
```

DESCRIPTION

This function runs the simulator or the real robot for one time step of `ms` milliseconds. Each effector, typically the motors, of the robot will be updated according to previous `set` function calls. Hence, the robot will run for `ms` milliseconds. Then, the values of each enabled sensor will be computed and stored for subsequent call to the corresponding `get` functions. If the simulator discards the robot, this function will terminate the controller program and the exit function (if set with the `khepera_die` function) will be called. Such a situation would correspond to switching off the real robot.

EXAMPLE

The following example illustrates the use of `khepera_step`:

```
#include <Khepera.h>
int main()
{
    khepera_live();
    khepera_enable_proximity(IR_SENSOR2 | IR_SENSOR3);
    for(;;)
    {
        if ((khepera_get_proximity(2)>500)&&
            (khepera_get_proximity(3)>500))
            khepera_set_speed(LEFT,-10);
        else
            khepera_set_speed(LEFT,10);
        khepera_set_speed(RIGHT,10);
        khepera_step(64);
    }
    return 0; /* this statement is never reached */
}
```

SEE ALSO

2.2 Khepera Gripper

```
gripper_disable_arm  
gripper_disable_grip  
gripper_disable_jumper  
gripper_disable_presence,  
gripper_disable_resistivity  
gripper_enable_arm,  
gripper_enable_grip  
gripper_enable_jumper,  
gripper_enable_presence  
gripper_enable_resistivity  
gripper_get_arm  
gripper_get_grip  
gripper_get_jumper  
gripper_get_presence  
gripper_get_resistivity  
gripper_set_grip  
gripper_set_arm
```

NAME

gripper_disable_arm, gripper_disable_grip, gripper_disable_jumper,
gripper_disable_presence, gripper_disable_resistivity,
gripper_enable_arm, gripper_enable_grip, gripper_enable_jumper,
gripper_enable_presence, gripper_enable_resistivity

- control the sensors of the gripper turret

SYNOPSIS

```
#include <Khepera.h>
#include <KheperaGripper.h>
void gripper_disable_arm();
void gripper_disable_grip();
void gripper_disable_jumper();
void gripper_disable_presence();
void gripper_disable_resistivity();
void gripper_enable_arm();
void gripper_enable_grip();
void gripper_enable_jumper();
void gripper_enable_presence();
void gripper_enable_resistivity();
```

DESCRIPTION

These functions allows enabling or disabling of the corresponding sensors of the gripper turret. By default, all sensors are disabled. Enabling and disabling sensors should occur before calling of the khepera_step function.

NAME

gripper_set_arm - set the position of the arm of the gripper

SYNOPSIS

```
#include <Khepera.h>
#include <KheperaGripper.h>
void gripper_set_arm(uint8 position);
```

DESCRIPTION

This function allows changing of the position of the gripper arm. The position parameter ranges from 0 (bottom back) to 255 (bottom forward) and specifies the absolute position of the arm to be reached. The initial default position of the arm is 152.

NAME

`gripper_set_grip` - set the position of the gripper, open (0) or closed (1)

SYNOPSIS

```
#include <Khepera.h>
#include <KheperaGripper.h>
void gripper_set_grip(uint8 position);
```

DESCRIPTION

This function allows closing or opening of the gripper. The `position` parameter must be set to 0 or 1 and specifies whether the gripper should be open (0) or closed (1). The initial position of the gripper is open (0).

IMPORTANT WARNING: Do not execute this command continuously when you grip an object: forcing the "close" action repeatedly on a blocked gripper may cause damages with a real gripper device.

NAME

`gripper_get_resistivity` - read the resistivity value of a grasped object

SYNOPSIS

```
#include <Khepera.h>
#include <KheperaGripper.h>
uint8 gripper_get_resistivity();
```

DESCRIPTION

This function allows reading of the resistivity value of an object present in the gripper.

RETURN VALUE

The return value ranges from 0 (infinite resistivity) to 255 (zero resistivity). Conductive objects (metal, for instance) will return a value of 255 while a value close to 0 will correspond to high resistivity objects (plastic, wood, ...). A graph provided with the documentation of the gripper turret plots the return values of the sensor as a function of the resistivity of the object expressed in $k\Omega$.

SEE ALSO

`gripper_enable_resistivity`
`gripper_disable_resistivity`

NAME

`gripper_get_presence` - read the status of the optical barrier of the gripper

SYNOPSIS

```
#include <Khepera.h>
#include <KheperaGripper.h>
bool gripper_get_presence();
```

DESCRIPTION

This function allows one to detect whether an object is present in the gripper.

RETURN VALUE

A return value of `true` indicates that an object is present (i.e., the optical barrier is cut), while a return value of `false` indicates that no object is detected in the gripper.

SEE ALSO

`gripper_enable_presence`
`gripper_disable_presence`

NAME

`gripper_get_arm`, `gripper_get_grip`
- read the position of the arm of the gripper

SYNOPSIS

```
#include <Khepera.h>
#include <KheperaGripper.h>
uint8 gripper_get_arm();
uint8 gripper_get_grip();
```

DESCRIPTION

These functions allow reading of the position of the arm and of the position of the grip of the gripper turret. Reading the position of the grip is a good way to measure the size of a gripped object.

RETURN VALUE

`gripper_get_arm` returns a value ranging from 0 (bottom back) to 255 (bottom forward) which indicates the current position of the arm.

`gripper_get_grip` returns a value ranging from about 0 (closed) to about 200 (open) which indicates the current position of the grip.

SEE ALSO

`gripper_enable_arm`
`gripper_enable_grip`
`gripper_disable_arm`
`gripper_disable_grip`

NAME

`gripper_get_jumper` - read the jumper status of the gripper turret

SYNOPSIS

```
#include <Khepera.h>
#include <KheperaGripper.h>
uint8 gripper_get_jumper();
```

DESCRIPTION

This function allows one to check whether or not the two jumpers of the gripper turret are set or not.

RETURN VALUE

The return value ranges from 0 to 3 as indicated in the following table:

value	state
0	no jumper set
1	internal jumper set
2	external jumper set
3	both jumpers set

Return values for the jumpers of the gripper turret

SEE ALSO

`gripper_enable_jumper`
`gripper_disable_jumper`

2.3 Khepera K213

```
k213_disable  
k213_enable  
k213_get_image
```

CONSTANTS

K213_WIDTH contains the width in pixels of the linear image produced by the K213 turret.

NAME

k213_disable, k213_enable - control the camera of the K213 turret

SYNOPSIS

```
#include <Khepera.h>
#include <KheperaK213.h>
void k213_disable();
void k213_enable();
```

DESCRIPTION

These functions allow enabling and disabling of the camera on the K213 turret. By default, the camera is enabled. Enabling or disabling of the camera should occur before calling of the `khepera_step` function.

NAME

k213_get_image - get a pointer to the pixel array.

SYNOPSIS

```
#include <Khepera.h>
#include <KheperaK213.h>
uint8 *k213_get_image();
```

DESCRIPTION

This function returns a pointer to the image data from the camera. Warning: this pointer may change at each iteration step. Therefore, you must call k213_get_image after each khepera_step call to get a valid pointer.

RETURN VALUE

The return value (pointer) points to an array of K213_WIDTH unsigned bytes containing the grey level values of all the pixels of the image. For a pixel located at x , the grey intensity of that pixel is equal to `pointer[x]`.

SEE ALSO

`K213_enable`
`K213_disable`

2.4 Khepera K6300

```
k6300_disable, k6300_enable  
k6300_get_red, k6300_get_green, k6300_get_blue  
k6300_get_image
```

CONSTANTS

K6300_WIDTH and K6300_HEIGHT contains the width and height in pixels of the image produced by the K6300 turret.

NAME

k6300_disable, k6300_enable - control the camera of the K6300 turret

SYNOPSIS

```
#include <Khepera.h>
#include <KheperaK6300.h>
void k6300_disable();
void k6300_enable();
```

DESCRIPTION

These functions allow enabling and disabling of the camera on the K6300 turret. By default, the camera is enabled. Enabling or disabling of the camera should occur before calling of the khepera_step function.

NAME

`k6300_get_red`, `k6300_get_green`, `k6300_get_blue` - macros to read pixel colors from the camera

SYNOPSIS

```
#include <Khepera.h>
#include <KheperaK6300.h>
```

DESCRIPTION

These macros allow reading of the RGB value of the pixel located at (x,y) in the camera matrix. Since the size of the camera matrix is 80×60 , x ranges from 0 to 79 and y ranges from 0 to 59. However, use of the constants `K6300_WIDTH` and `K6300_HEIGHT` is highly recommended. Here is the definition of the macros as they can be found in the `KheperaK6300.h` header file:

```
#define k6300_get_red(image,x,y)      (im-
age[(x+y*K6300_WIDTH)*3])
#define k6300_get_green(image, x,y)   (im-
age[(x+y*K6300_WIDTH)*3+1])
#define k6300_get_blue(image,x,y)     (im-
age[(x+y*K6300_WIDTH)*3+2])
```

`image` is a pointer to the actual image which must be obtained from the `k6300_get_image` function before using these macros.

RETURN VALUE

These macros return an intensity value ranging from 0 to 255. A black pixel would return 0 for each of the three macros, whereas a white pixel would return 255 for each of the three macros.

SEE ALSO

`k6300_get_image`, `k6300_enable`, `k6300_disable`

EXAMPLE

```
uint8 *image;
...
image = k6300_get_image();
uint8 red = k6300_get_red(image,20,5); /* read red level a (20,5) */
```

NAME

k6300_get_image - get a pointer to the pixel array.

SYNOPSIS

```
#include <Khepera.h>
#include <KheperaK6300.h>
uint8 *k6300_get_image();
```

DESCRIPTION

This function returns a pointer to the image data from the camera. Warning: this pointer may change at each iteration step. Therefore, you must call k6300_get_image after each khepera_step call to get a valid pointer.

RETURN VALUE

The return value (pointer) points to an array of K6300_WIDTH×K6300_HEIGHT×3 unsigned bytes containing the red, green and blue values of all the pixels of the image. For a pixel located at (x,y) , the color intensity is equal to $\text{pointer}[3 \times (x + y \times \text{K6300_WIDTH}) + \text{color}]$ where $\text{color}=0$ for red, 1 for green, and 2 for blue.

SEE ALSO

K6300_enable
K6300_disable

2.5 Khepera Panoramic

```
panoramic_disable_highpass
panoramic_disable_lowpass1
panoramic_disable_lowpass2
panoramic_disable_odd
panoramic_disable_raw
panoramic_enable_highpass
panoramic_enable_lowpass1
panoramic_enable_lowpass2
panoramic_enable_odd
panoramic_enable_raw
panoramic_get_highpass
panoramic_get_lowpass1
panoramic_get_lowpass2
panoramic_get_odd
panoramic_get_raw
panoramic_set_parameters
panoramic_set_precision
panoramic_set_window
```

NAME

panoramic_disable_highpass, panoramic_disable_lowpass1,
panoramic_disable_lowpass2, panoramic_disable_odd,
panoramic_disable_raw, panoramic_enable_highpass,
panoramic_enable_lowpass1, panoramic_enable_lowpass2,
panoramic_enable_odd, panoramic_enable_raw
- control the artificial retina sensor

SYNOPSIS

```
#include <Khepera.h>
#include <KheperaPanoramic.h>
void panoramic_disable_highpass();
void panoramic_disable_lowpass1();
void panoramic_disable_lowpass2();
void panoramic_disable_odd();
void panoramic_disable_raw();
void panoramic_enable_highpass();
void panoramic_enable_lowpass1();
void panoramic_enable_lowpass2();
void panoramic_enable_odd();
void panoramic_enable_raw();
```

DESCRIPTION

These functions allow one to disable the filters of the artificial retina sensor of the Khepera panoramic turret when they are not needed and to enable them when needed. The aim of these functions is to optimize execution speed on both real and simulated robots. When a filter is disabled, the corresponding `panoramic_get_XXX` function will return irrelevant data.

Enabling and disabling of filters should occur before calling the `khepera_step` function.

NAME

`panoramic_get_highpass`, `panoramic_get_lowpass1`,
`panoramic_get_lowpass2`, `panoramic_get_odd`, `panoramic_get_raw`
- read pixel values

SYNOPSIS

```
#include <Khepera.h>
#include <KheperaPanoramic.h>
uint8 panoramic_get_lowpass1(uint8 position);
uint8 panoramic_get_lowpass2(uint8 position);
uint8 panoramic_get_highpass(uint8 position);
uint8 panoramic_get_odd(uint8 position);
uint8 panoramic_get_raw(uint8 position);
```

DESCRIPTION

These functions allow reading of the grayscale contents of the artificial retina sensor of the Khepera panoramic turret. `position` specifies the location of the pixel to be read. It can range from 0 to 149. It is very important to notice that only the functions for which the corresponding filter has been enabled will return correct values. Moreover, if you defined a window using `panoramic_set_window`, the values outside the window will return irrelevant values. For efficiency purposes, they are implemented as macros in the simulator and in the real robot.

RETURN VALUES

`panoramic_read_pixel` returns the value of the current pixel. This value can range from 0 to $2^p - 1$, where p is the precision specified by the function `panoramic_set_precision`. Note that the default precision is $p = 6$ bits, hence default return values range from 0 to 63.

SEE ALSO

`panoramic_enable_lowpass1`
`panoramic_enable_lowpass2`
`panoramic_enable_highpass`
`panoramic_enable_odd`
`panoramic_enable_raw`
`panoramic_disable_lowpass1`
`panoramic_disable_lowpass2`
`panoramic_disable_highpass`

```
panoramic_disable_odd  
panoramic_disable_raw  
panoramic_set_filter  
panoramic_set_precision  
panoramic_set_window
```

NAME

`panoramic_set_parameters` - set the parameters for the filters of the artificial retina

SYNOPSIS

```
#include <Khepera.h>
#include <KheperaPanoramic.h>
void panoramic_set_parameters(uint8 l1,uint8 l2,uint8 l3);
```

DESCRIPTION

This function allows setting of the parameters of the artificial retina sensor of the Khepera panoramic turret. The values l1, l2 and l3 are explained in the following tables:

filter	description
highpass	this filter corresponds to the subtraction of the image resulting from the lowpass1 and lowpass2 filters. Hence, it is defined by l1 and l2, while l3 is ignored.
lowpass1	l1 specifies the low pass filter cutoff frequency for this filter. It ranges from 0 to 7. See below for an explanation of these values. l2 and l3 are ignored.
lowpass2	l2 specifies the low pass filter cutoff frequency for this filter. It ranges from 0 to 7. See below for an explanation of these values. l1 and l3 are ignored.
odd	l3 specifies the cutoff frequency for this odd filter (derivative of low-pass). It ranges from 0 to 7. See below for an explanation of these values. l1 and l2 are ignored.
raw	no filter, provide the raw image. l1, l2, and l3 are ignored. This is the default value.

Filters parameters

Filter values	0	1	2	3	4	5	6	7
l1 and l2	0	0.926	1.414	1.897	2.449	3.162	4.243	6.481
l3	0	0.481	0.721	1.020	1.443	2.128	3.476	7.489

Parameter values expressed in KHz

NAME

panoramic_set_precision - set the precision for the artificial retina

SYNOPSIS

```
#include <Khepera.h>
#include <KheperaPanoramic.h>
void panoramic_set_precision(uint8 b);
```

DESCRIPTION

This function allows setting of the precision of the artificial retina sensor of the Khepera panoramic turret. It defines the number of grey levels for the pixels of the image. *b* defines this precision in bits. It ranges from 1 to 6, thus allowing up to 64 grey levels. The default precision is *b* = 6 bits.

NAME

`panoramic_set_window` - set a window of view for the artificial retina

SYNOPSIS

```
#include <Khepera.h>
#include <KheperaPanoramic.h>
void panoramic_set_window(uint8 start,uint8 end);
```

DESCRIPTION

This function allows setting of a window restricting the view angle of the artificial retina sensor of the Khepera panoramic turret. The parameter `start` defines where the window starts on the 150 pixel wide retina. Hence, `start` ranges from 0 to 149. `end` defines the pixel where the window ends on the retina, and also ranges from 0 to 149. The default window starts at pixel position 0 and ends at pixel position 149, corresponding to a 240 degree field of view.

Chapter 3

Supervisor API Reference

3.1 Supervisor

```
supervisor_die  
supervisor_live  
supervisor_step
```

NAME

`supervisor_die` - set the exit function for a supervisor controller

SYNOPSIS

```
#include <Supervisor.h>
void supervisor_die(void (*)());
```

DESCRIPTION

This function declares an exit function to be called automatically when the supervisor controller exits. This might be useful to perform some clean-up before quitting, i.e., save data to a file, print out some statistics, etc. The exit function should be a void function without any argument.

EXAMPLE

```
void my_exit_function()
{
    printf(``bye bye!\n'');
}

int main()
{
    supervisor_live(); /* now ! */
    supervisor_die(my_exit_function); /* when it quits */
    for(;;) { supervisor_step(64); }
    return 0; /* this statement will never be reached */
}
```

SEE ALSO

`supervisor_live`
`supervisor_step`

NAME

`supervisor_live` - initialize the supervisor program

SYNOPSIS

```
#include <Supervisor.h>
void supervisor_live();
```

DESCRIPTION

This function initializes a supervisor program. It must be called before calling any other Supervisor API function.

SEE ALSO

`supervisor_die`
`supervisor_step`

NAME

`supervisor_step` - proceed one simulation step

SYNOPSIS

```
#include <Supervisor.h>
void supervisor_step(uint32 ms);
```

DESCRIPTION

This function runs the simulator for one time step of `ms` milliseconds. Information requested by the supervisor will be gathered and the function will return to the supervisor program. The simulator may stop and destroy the supervisor during this function call if the user quits or explicitly asks to stop the supervisor. If this occurs and an exit function was set using the `supervisor_die` function, then this exit function is called before the controller exits.

SEE ALSO

`supervisor_die`
`supervisor_live`

3.2 EAI: External Authoring Interface

```
eai_delete_node  
eai_get_node  
eai_get_orientation  
eai_get_position  
eai_set_orientation  
eai_set_position  
eai_read_stream  
eai_write_stream  
eai_set_color  
eai_set_color_in_grid  
eai_get_color_in_grid  
eai_set_controller  
eai_set_height
```

NAME

eai_delete_node - delete a node from the world

SYNOPSIS

```
#include <eai.h>
void eai_delete_node(eai_node n);
```

DESCRIPTION

This function deletes the node n from the world.

SEE ALSO

eai_get_node

NAME

eai_get_node - get a pointer to a node identified by its DEF name

SYNOPSIS

```
#include <eai.h>
eai_node eai_get_node(char *DEFName);
```

DESCRIPTION

This function is used to obtain a pointer to a node in the world. The node is identified with its DEF name which is a character string. A pointer to a node may become obsolete after calling the supervisor_step function because this node can be deleted (for example by the user) in the simulator during the supervisor_step function.

RETURN VALUE

eai_get_node returns a pointer to the requested node. A return value of NULL indicates that the world contains no node with this DEF name.

SEE ALSO

[eai_delete_node](#)

NAME

eai_get_orientation, eai_get_position,
eai_set_orientation, eai_set_position
- get or set the orientation or position of a node

SYNOPSIS

```
#include <eai.h>
float32 eai_get_orientation(eai_node n);
void    eai_get_position(eai_node n,float32 *x,float32 *z);
void    eai_set_orientation(eai_node n,float32 alpha);
void    eai_set_position(eai_node n,float32 x,float32 z);
```

DESCRIPTION

These functions allow the supervisor controller to read and change the position and orientation of a node *n*. The position of a node usually corresponds to the geometrical center of this node. The *X* and *Z* parameters are used to set or retrieve the (*x,z*) position of the node *n* while the *alpha* parameter is used to set its orientation. The *x* and *z* values indicate the absolute position of the node *n* in the coordinate system of the world. They are expressed in meters (m). The *alpha* value indicates the absolute orientation of the node *n*, in the coordinate system of the world. It is expressed in radians (rad).

RETURN VALUE

eai_get_orientation returns a floating point value indicating the orientation of the node *n* in radians (rad).

NAME

eai_read_stream, eai_write_stream - exchange data with the robot controllers

SYNOPSIS

```
#include <eai.h>
int32 eai_read_stream(eai_node node, char *buffer, int32
size);
void eai_write_stream(eai_node node, char *buffer, int32
size);
```

DESCRIPTION

These functions are used to communicate with the robot controllers through a data stream. They are not blocking calls. If no data is available when `eai_read_stream` is called, then the function returns 0 immediately. Similarly, the `eai_write_stream` function doesn't wait for the data to be read by a robot controller, instead, it returns immediately. Similar functions should be used on the controller side. For example, a Khepera controller should use the `khepera_send_serial` and `khepera_receive_serial` functions to exchange data with the supervisor. The `node` parameter is a pointer to the robot with which data exchange is performed. `buffer` should point to an allocated memory address that can contain at least `size` bytes.

`eai_read_stream` reads a maximum of `size` bytes of data into `buffer`. It returns the number of bytes actually read from the stream.

`eai_write_stream` writes `size` bytes of data from `buffer`.

EXAMPLE

```
char buffer[256];
...
    buffer[0]='R';
    buffer[1]=57;
    d2r2 = eai_get_node(``D2R2``);
    eai_write_stream(d2r2,buffer,2);
```

SEE ALSO

`khepera_send_serial`
`khepera_receive_serial`

NAME

eai_set_color, eai_set_color_in_grid, eai_get_color_in_grid
- set or get the color of a node

SYNOPSIS

```
#include <eai.h>
void eai_set_color(eai_node node,
                   uint8 red,uint8 green,uint8 blue);
void eai_set_color_in_grid(eai_node node,uint32 x,uint32
z,
                           uint8 red,
                           uint8 green,
                           uint8 blue);
void eai_get_color_in_grid(eai_node node,uint32 x,uint32
z,
                           uint8 *red,
                           uint8 *green,
                           uint8 *blue);
```

DESCRIPTION

These functions are used to read or change the color of the object node in the world. red, green and blue represent the intensity of each color component.

eai_set_color_in_grid and eai_get_color_in_grid are used only with Elevation-Grid node. The x and z parameters indicate the coordinate of the plate in the Elevation-Grid for which the color will be read or changed.

NAME

eai_set_height - set the height of a node

SYNOPSIS

```
#include <eai.h>
void eai_set_height(eai_node node, float32 height);
```

DESCRIPTION

This function allows the supervisor to change the height of an object node in the world. The height parameter represents the new height to be set. Note that not all nodes have a relevant height attribute. `eai_set_height` is particularly useful for Can nodes.

NAME

`eai_set_controller` - set the controller for a robot node

SYNOPSIS

```
#include <eai.h>
void eai_set_controller(eai_node robot,
                        char *controller);
```

DESCRIPTION

This function allows the supervisor to change the controller of a robot. The robot is specified by its `eai_node` pointer `robot`. The new controller is specified its name.

EXAMPLE

```
eai_node robot;
...
robot = eai_get_node("Kheperal");
eai_set_controller(robot, "void.khepera");
```

Chapter 4

Webots File Format Version 2.0

4.1 File Structure

Webots files must begin with the characters:

```
#WEBOTS V2.0 utf8
```

The file extension is .wbt (standing for WeBoTs)

4.2 VRML97 nodes partially supported in Webots

The name of the node appears first and the names of the supported fields appear within the braces.

- Appearance { material }
- Background { skyColor }
- Box { size }
- Color { color }
- Coordinate { point }
- Cylinder { bottom height radius side top }
- DirectionalLight { ambientIntensity color direction intensity on }
- ElevationGrid { color colorPerVertex height xDimension zDimension xSpacing zSpacing }

- Group { children }
- IndexedFaceSet { coord coordIndex solid }

Note: face rendering is performed on a single side. The order of the coordinates indicate the orientation of the visible face. If you need double-sided faces, you will have to create two faces with the same coordinate lists but in reverse order.

- ImageTexture { url repeats repeatT }
- IndexedLineSet { coord coordIndex }
- Material { diffuseColor specularColor shininess transparency }
- PointLight { ambientIntensity location }
- Shape { appearance geometry }
- Sphere { radius }
- TextureTransform { scale center translation rotation }
- Transform { translation rotation children }
- Viewpoint { fieldOfView orientation position }

Note: Material transperancy field is taken as a boolean and only considered in the case that a texture is applied to the shape the material is associated to.

Note: In case of having a texture applied to a Wall object, this will be wrapped around it. In case of a box, the texture is repeated in each of its 6 faces.

4.3 Webots nodes

The nodes listed here are described using the standard VRML description syntax.

4.3.1 Ball

```
Ball
{
    SFColor color      1 1 1
    SFFloat elasticity 0.5
    SFFloat friction   1.0
```

```

SFFloat mass          0.03
SFVec3f position    0 0 0
SFFloat radius        0.02
SFVec3f speed         0 0 0
}

```

4.3.2 Can

```

Can
{
    SFColor color      1 1 1
    SFNode   imageTexture      NULL
    SFNode   textureTransform NULL
    SFFloat height       0.04
    SFVec3f position     0 0 0
    SFFloat radius        0.02
    SFFloat resistivity  0
}

```

4.3.3 KheperaFeeder

```

KheperaFeeder
{
    SFFloat orientation 0
    SFVec2f position    0 0
}

```

4.3.4 Lamp

```

Lamp
{
    SFFloat    intensity      1
    SFVec3f   position       0 0.05 0
}

```

4.3.5 Wall

```

Wall
{

```

```

SFColor    color          0.9 0.2 0.3
SFFloat   height         0.05
SFFloat   orientation    0
SFVec2f   position       0 0
MFVec2f   points         [ 0.15  0.015 ,
                           -0.15  0.015 ,
                           -0.15 -0.015 ,
                           0.15 -0.015 ]
SFNode    imageTexture    NULL
SFNode    textureTransform NULL
}

```

4.3.6 Supervisor

```

Supervisor
{
  SFString controller ""
}

```

4.3.7 Alice

```

Alice
{
  SFFloat orientation 0
  SFVec2f position    0 0
}

```

4.3.8 AliceIRCom

```

AliceIRCom
{
}

```

4.3.9 Khepera

```

Khepera
{
  SFString controller    ""
  SFBool   innerJumper   TRUE
}

```

```

SFNode      KBus           NULL
SFBool      middleJumper  TRUE
SFFloat     orientation   0
SFBool      outerJumper  FALSE
SFVec2f    position      0 0
}

```

4.3.10 KheperaK213

```

KheperaK213
{
}

```

4.3.11 KheperaK6300

```

KheperaK6300
{
  SFColor    color        0.3 0.3 0.3
}

```

4.3.12 KheperaGripper

```

KheperaGripper
{
  SFInt32    armPosition  152
  SFFloat    canZ         0
  SFFloat    canX         0
  SFNode     can          NULL
  SFInt32    gripPosition 200
  SFBool     innerJumper  FALSE
  SFBool     outerJumper  FALSE
  SFNode     KBus         NULL
}

```

4.3.13 KheperaPanoramic

```

KheperaPanoramic
{
}

```