Webots Reference Manual

release 3.2.9

www.cyberbotics.com

July 10, 2002

Trademark information

Thanks

Cyberbotics is grateful to all the people who contributed to the development of Webots, Webots sample applications, the Webots User Guide, the Webots Reference Manual, and the Webots web site, including Jordi Porta, Emanuele Ornella, Yuri Lopez de Meneses, Auke-Jan Ijspeert, Gerald Foliot, Allen Johnson, Michael Kertesz, Aude Billiard, and many others.

Moreover, many thanks are due to Prof. J.-D. Nicoud (LAMI-EPFL) and Dr. F. Mondada for their valuable support.

Finally, thanks to Skye Legon, who proof-read this manual.

Contents

Chapter 1

Introduction

This reference manual contains all the information needed to program robot controllers in Webots. Moreover, it contains reference information on the world description language used in Webots, which is an extension of a subset of VRML 2.0.

The programming of graphical user interfaces (GUI) is not covered in this manual since Webots 3 uses GTK+ for creating user interfaces for controllers. GTK+ Tutorial and Reference Manual are included on the Webots CD-ROM.

Chapter 2

Controller API

2.1 Robot

`robot_die`

NAME

`robot_die` – *declare an exit function*

SYNOPSIS

```
#include <device/robot.h>
void robot_die(void (*exit_function)(void));
```

DESCRIPTION

This function declares an exit function to be used whenever a controller quits. A controller can quit for the following reasons: the simulator quits, or the robot quits the simulator by entering an HyperGate to be transfered to another simulation server. In the latter case, it might be useful for the robot to save important data (like an acquired behaviour) before it quits, so that this data can be transfered to the target simulator corresponding to the HyperGate. Hence, when the robot restarts on the other side of the HyperGate, it can retrieve its data in its reset function before it starts running again.

SEE ALSO

`robot_live`

robot_get_device

NAME

`robot_get_device` – *get a pointer to a device*

SYNOPSIS

```
#include <device/robot.h>
DeviceTag robot_get_device(const gchar *name);
```

DESCRIPTION

This function returns a pointer to a device corresponding to a specified DEF name. For example, if the robot contains a `DistanceSensor` node which `DEF name` is ”ds1”, the function will return a pointer to that device. This `DeviceTag` pointer will be used subsequently for enabling, sending command to, or reading data from this device.

SEE ALSO

`robot_live`

robot_live

NAME

`robot_live` – *initialize a robot controller*

SYNOPSIS

```
#include <device/robot.h>
void robot_live(void (*reset_function)(void));
```

DESCRIPTION

This function must be called before any other controller API function. It is necessary to initialize the robot controller and optionally to provide a reset function to the controller. This reset function is useful to perform some initializations, so that the controller knows which sensors and actuators

are available. The reset function should be a void function without any argument. It is called once at the beginning of the simualtion and may be called again if the simulator needs to reset the robot. However, this rarely happens in practise.

EXAMPLE

```
#include <device/robot.h>

static DeviceTag my_sensor, my_actuator;

void my_reset_function() { /* called at init. */
  printf("hello!\n");
  my_sensor = robot_get_device("my_sensor");
  my_actuator = robot_get_device("my_actuator");
}

void my_exit_function() { /* called before quitting  */
  printf("bye bye!\n");
}

int main() {
 robot_live(my_reset_function); /* called when robot starts */
 robot_die(my_exit_function); /* called when robot quits */
 for(;;) { /* infinit loop */
    /* read the sensors and write to the actuators */
    ...
    robot_step(64);
  }
  return 0; /* this statement will never be reached */
}
```

SEE ALSO

`robot_get_device`

`robot_die`

`robot_step`

robot_step

NAME

`robot_step` – *execute a simulation step*

SYNOPSIS

```
#include <device/robot.h>
guint32 robot_step(guint32 ms);
```

DESCRIPTION

This function requests the simulator to perform a simulation step of `ms` milliseconds, that is to advance in the simulated time of this amount of time. In synchronous simulation mode, the request is always fulfilled and the function always return 0. In asynchronous mode, the request may not be fulfilled. In this case, the return value `dt`, representing the delay, may not be 0. Let `controller_date` be the current time of the controller, the return value be interpreted as follow:

- if `dt` = 0, then, the behavior is equivalent to the one of synchronous mode.
- if 0 ¡= `dt` ¿ `ms`, then the actuator values were set at `controller_date` + `dt` and the sensor values where measured at `controller_date` + `ms`, as requested. It means that the step actually lasted the requested number of milliseconds, but the actuators command could not be executed on time.
- if `dt` ¿ `ms`, then the actuators values were set at `controller_date` + `dt` and the sensors values where measured also at `controller_date` + `dt`. It means that the requested step duration could not be respected.

SEE ALSO

`robot_live`

2.2 CustomRobot

`custom_robot_move`

NAME

`custom_robot_move` – *control the position of the robot*

SYNOPSIS

```
#include <device/custom_robot.h>
void custom_robot_move(gfloat tx, gfloat ty, gfloat tz, gfloat rx, gfloat ry, gfloat tz, gfloat alpha);
```

DESCRIPTION

This function allows the user to modify the position and orientation of a custom robot. The move will be performed at the beginning of the next simulation step. If the collision detection system detects a collision between the `CustomRobot` node and any another `Solid` object, the move will not be performed and the custom robot position and orientation will remain unchanged. The `tx`, `ty` and `tz` values represent the requested translation relative to the current translation value of the robot. The `rx`, `ry`, `rz` and `alpha` values represent the offsets to be added to the current rotation vector and angle of the robot.

2.3 DifferentialWheels

`differential_wheels_set_speed`

NAME

`differential_wheels_set_speed` – *control the speed of the robot*

SYNOPSIS

```
#include <device/differential_wheels.h>
void differential_wheels_set_speed(gint16 left, gint16 right);
```

DESCRIPTION

This function allows the user to specify a speed for the differentially wheeled robot. This speed will be send to the motors of the robot at the beginning of the next simulation step. The speed unit is defined by the `speedUnit` field of the `DifferentialWheels` node. The default value is 0.1 radian per seconds. Hence a speed value of 20 will make the wheel rotate at a speed of 2 radian per seconds. The linear speed of the robot can then be computed from the angular speed of each wheel, the wheel radius and the noise on the command. Both the wheel radius and the noise on the command are documented in the `DifferentialWheels` node.

differential_wheels_enable_encoders

NAME

differential_wheels_enable_encoders, differential_wheels_disable_encoders – *enable or disable the incremental encoders of the robot wheels*

SYNOPSIS

```
#include <device/differential_wheels.h>
void differential_wheels_enable_encoders(guint16 ms);
void differential_wheels_disable_encoders (void);
```

DESCRIPTION

These functions allow the user to enable or disable the incremental wheel encoders for both wheels of the `DifferentialWheels` robot. Incremental encoder are counters that incremented each time a wheel turns with the value of the speed for the corresponding wheel. For example, if you set a speed of 400 for the left wheel and perform a robot step of 64 ms, the left encoder will increase its value of 400 * 0.064 = 25.6 after this step. However, if the initial value of the encoder was 0, the value you will read will be 25 and not 25.6. But if you repeat the same step, the computed value will be 51.2 and you will be able to read 51. Please note that when the `DifferentialWheels` robot faces an obstacle while trying to move forward, the wheels of the robot do not slip, hence the encoder values are not increased. This is very useful to detect that the robot has hit an obstacle.

differential_wheels_get_left_encoder

NAME

differential_wheels_get_left_encoder, differential_wheels_get_right_encoder, differential_wheels_set_encoders – *read or set the encoders of the robot wheels*

SYNOPSIS

```
#include <device/differential_wheels.h>
 gint32 differential_wheels_get_left_encoder (void);
 gint32 differential_wheels_get_right_encoder (void);
```

```
void differential_wheels_set_encoders (gint32 left, gint32 right);
```

DESCRIPTION

These functions are used to read or set the values of the left and right encoders. The encoders have to be enabled with `differential_wheels_enable_encoders`, so that the functions can read correct values. Moreover, the `encoderNoise` of the corresponding `DifferentialWheels` node should be positive. Setting encoders value will not make the wheels rotate to reach the specified value, instead, it will simply reset the encoders with the specified value.

2.4 DistanceSensor

`distance_sensor_enable`

NAME

`distance_sensor_enable`, `distance_sensor_disable` – *enable and disable the distance sensor measurements*

SYNOPSIS

```
#include <device/distance_sensor.h>
 void distance_sensor_enable (DeviceTag sensor, guint16 ms);
 void distance_sensor_disable (DeviceTag sensor);
```

DESCRIPTION

`distance_sensor_enable` allows the user to enable a distance sensor measurement each `ms` milliseconds.

`distance_sensor_disable` turns the distance sensor off, saving computation time.

`distance_sensor_get_value`

NAME

`distance_sensor_get_value` – *get the distance sensor measure*

SYNOPSIS

```
#include <device/distance_sensor.h>
 guint16 distance_sensor_get_value (DeviceTag sensor);
```

DESCRIPTION

`distance_sensor_get_value` returns the last value measured by the specified distance sensor. This value is computed by the simulator according to the lookup table of the `DistanceSensor` node. Hence, the value range for the return value is defined by this lookup table.

2.5 Camera

`camera_enable`

NAME

`camera_enable`, `camera_disable` – *enable and disable the camera measurements*

SYNOPSIS

```
#include <device/camera.h>
 void camera_enable (DeviceTag camera, guint16 ms);
 void camera_disable (DeviceTag camera);
```

DESCRIPTION

`camera_enable` allows the user to enable a camera measurement each `ms` milliseconds.

`camera_disable` turns the camera off, saving computation time.

`camera_new_widget`

NAME

`camera_new_widget` – *create a GTK+ widget for displaying the camera view*

SYNOPSIS

```
#include <device/camera.h>

GtkWidget *camera_new_widget (DeviceTag camera);
```

DESCRIPTION

`camera_new_widget` creates a new GTK+ widget for a camera. Creating such a widget is useful only if you want to embed it into your own graphical user interface, otherwise, if you don't create any camera widget, the controller library will do it automatically in a separate window. It is mandatory to have such a widget displayed to allow OpenGL rendering. This constraint is mainly due to 3D hardware which often cannot render an off screen image.

camera_get_fov

NAME

`camera_get_fov`, `camera_set_fov` – *get and set field of view for a camera*

SYNOPSIS

```
#include <device/camera.h>

gfloat camera_get_fov (DeviceTag camera);

void camera_set_fov (DeviceTag camera, gfloat fov);
```

DESCRIPTION

These functions allow the controller to get and set the value for the field of view (fov) of a camera. The original value for this field of view is defined in the `Camera` node, as `fieldOfView`. Note however, that changing the field of view using `camera_set_fov` will not change the value of the `fieldOfView` field on the simulator side. It will only affect the controller side, making new rendered images use the specified field of view for the specified camera.

camera_get_width

NAME

`camera_get_width`, `camera_get_height` – *get the size of the camera image*

SYNOPSIS

```
#include <device/camera.h>
 guint16 camera_get_width (DeviceTag camera);
 guint16 camera_get_height (DeviceTag camera);
```

DESCRIPTION

These functions return the width and height of a camera image as defined in the corresponding `Camera` node.

camera_get_type

NAME

`camera_get_type` – *get the type of the camera*

SYNOPSIS

```
#include <device/camera.h>
 gchar camera_get_type (DeviceTag camera);
```

DESCRIPTION

This function returns the type of a camera as defined in the corresponding `Camera` node. If the type is ”black and white” or ”grey”, then the return value is ’g’, otherwise, the return value is ’c’, standing for color.

camera_get_image

NAME

`camera_get_image`, `camera_image_get_red`, `camera_image_get_green`, `camera_image_get_blue` – *get the image data from a camera*

SYNOPSIS

```
#include <device/camera.h>
 guint8 *camera_get_image (DeviceTag camera);
```

```
guint8 camera_image_get_red (image, width, x, y);
guint8 camera_image_get_green (image, width, x, y);
guint8 camera_image_get_blue (image, width, x, y);
```

DESCRIPTION

The `camera_get_image` function allows you to read the contents of the last image grabbed by the camera. The image is coded as a series of three bytes coding for the red, green and blue levels of a pixel. Pixels are stored in lines ranging from the bottom left hand side of the image up to top right hand side. The memory chunck returned by this function doesn't need to be released, as it is handled by the camera itself. The size in bytes of this memory chunk can be computed as follow:

`size` = `camera_width` * `camera_height` * 3

Attempting to read outside the bounds of this chunk will cause an error.

The `camera_image_get_` macros are useful helpers for accessing directly the pixel colors from the pixel coordinates. They are defined as follow:

```
#define camera_image_get_red(image,width,x,y)   (image[3*((y)*(width)+(x))])
#define camera_image_get_green(image,width,x,y) (image[3*((y)*(width)+(x))+1])
#define camera_image_get_blue(image,width,x,y)  (image[3*((y)*(width)+(x))+2])
```

2.6 Emitter

`emitter_get_buffer`

NAME

`emitter_get_buffer`, `emitter_get_buffer_size` – *get information on the emitter buffer*

SYNOPSIS

```
#include <device/emitter.h>
gpointer emitter_get_buffer (DeviceTag emitter);
gint32 emitter_get_buffer_size (DeviceTag emitter);
```

DESCRIPTION

The `emitter_get_buffer` function returns a pointer to the buffer used by the emitter to send data. The `emitter_get_buffer_size` function returns the size of this buffer, expressed in bytes.

emitter_send

NAME

`emitter_send` – *send a message through the emitter*

SYNOPSIS

```
#include <device/emitter.h>
 void emitter_send (DeviceTag emitter, guint32 size);
```

DESCRIPTION

The `emitter_send` function sends `size` bytes of data contained in the beginning of the emitter buffer.

emitter_get_channel

NAME

`emitter_get_channel`, `emitter_set_channel` – *get or set channel information for an emitter.*

SYNOPSIS

```
#include <device/emitter.h>
 gint32 emitter_get_channel (DeviceTag emitter);
 void emitter_set_channel (DeviceTag emitter, gint32 channel);
```

DESCRIPTION

The `emitter_get_channel` function returns the channel value of the `Emitter` node. Only receivers set to the same channel of the emitter can receive message from this emitter.

The `emitter_set_channel` function allows the controller to change the emission channel, so that different receivers may receive the messages of the emitter. Calling this function will change the channel field of the `Emitter` node.

2.7 LightSensor

light_sensor_enable

NAME

light_sensor_enable, light_sensor_disable – *enable and disable the light sensor measurements*

SYNOPSIS

```
#include <device/light_sensor.h>
void light_sensor_enable (DeviceTag sensor, guint16 ms);
void light_sensor_disable (DeviceTag sensor);
```

DESCRIPTION

light_sensor_enable allows the user to enable a light sensor measurement each ms milliseconds.

light_sensor_disable turns the light sensor off, saving computation time.

light_sensor_get_value

NAME

light_sensor_get_value – *get the light sensor measure*

SYNOPSIS

```
#include <device/light_sensor.h>
guint16 light_sensor_get_value (DeviceTag sensor);
```

DESCRIPTION

light_sensor_get_value returns the last value measured by the specified light sensor. This value is computed by the simulator according to the lookup table of the LightSensor node. Hence, the value range for the return value is defined by this lookup table.

2.8 Gripper

gripper_set_position

NAME

gripper_set_position – *open or close the gripper*

SYNOPSIS

```
#include <device/gripper.h>
 void gripper_set_position (DeviceTag gripper, gfloat position);
```

DESCRIPTION

The gripper_set_position function allows the user to close or open the gripper depending on the specified position value which represents the aperture of the gripper device, expressed in meters. Hence a value of 0 will close the gripper and a value of 0.04 will open the gripper 4 cm wide.

gripper_enable_position

NAME

gripper_enable_position, gripper_enable_resistivity, gripper_disable_position, gripper_disable_resistivity – *enable or disable the position and resistivity sensors on a gripper*

SYNOPSIS

```
#include <device/gripper.h>
 void gripper_enable_position (DeviceTag gripper, guint16 ms);
 void gripper_enable_resistivity (DeviceTag gripper, guint16 ms);
 void gripper_disable_position (DeviceTag gripper);
 void gripper_disable_resistivity (DeviceTag gripper);
```

DESCRIPTION

These functions enable each ms milliseconds or disable the gripper position and resistivity measurement.

gripper_get_position

NAME

gripper_get_position, gripper_get_resistivity – *return the position and resistivity values measured on the gripper*

SYNOPSIS

```
#include <device/gripper.h>
gfloat gripper_get_position (DeviceTag gripper);
gfloat gripper_get_resistivity (DeviceTag gripper);
```

DESCRIPTION

The gripper_get_position function returns the position measurement performed on the specified gripper device. The position is expressed in meters and corresponds to the aperture of the gripper as with the gripper_set_position function. However, it returns the current position of the gripper and not the target position specified with gripper_set_position (which may be the same value when the target position is reached). This function may be useful to measure the size of a gripped object.

The gripper_get_resistivity function returns the resistivity measurement performed on the specified gripper device. This value is expressed in ohm. In this first version, we assume that any object has a resistivity of one ohm. It will return *Inf* when no object is gripped and 1.0 when an object is gripped.

2.9 Receiver

receiver_enable

NAME

receiver_enable, receiver_disable – *enable and disable the receiver measurements*

SYNOPSIS

```
#include <device/receiver.h>
 void receiver_enable (DeviceTag receiver, guint16 ms);
 void receiver_disable (DeviceTag receiver);
```

DESCRIPTION

`receiver_enable` allows the user to enable a receiver measurement each `ms` milliseconds.

`receiver_disable` turns the receiver off, saving computation time.

`receiver_get_buffer`

NAME

`receiver_get_buffer`, `receiver_get_buffer_size` – *get information on the receiver buffer*

SYNOPSIS

```
#include <device/receiver.h>
 gpointer receiver_get_buffer (DeviceTag receiver);
 gint32 receiver_get_buffer_size (DeviceTag receiver);
```

DESCRIPTION

The `receiver_get_buffer` function returns a pointer to the buffer used by the receiver to store received data. This function needs to be called each time new data arrives in the receiver because the address of the buffer changes when new data arrives. The returned memory chunk doesn't need to be released. Memory management is done by the receiver. Moreover calling `receiver_get_buffer` will cause the data to be flushed out of the receiver, hence calling `receiver_get_buffer_size` immediatly after will return 0;

The `receiver_get_buffer_size` function returns the size of this buffer, expressed in bytes, that is the number of bytes received and stored in the buffer. It must be called before `receiver_get_buffer`, otherwise, it returns always 0.

2.10 Servo

servo_set_position

NAME

servo_set_position, servo_set_speed, servo_set_acceleration – *set servo parameters*

SYNOPSIS

```
#include <device/servo.h>
 void servo_set_position (DeviceTag servo, gfloat position);
 void servo_set_speed (DeviceTag servo, gfloat speed);
 void servo_set_acceleration (DeviceTag servo, gfloat acc);
```

DESCRIPTION

The servo_set_position function gives a new target position the servo will try to reach. If the servo is a rotation servo, the unit of the position parameter is radian, otherwise, it is meter.

The servo_set_speed function gives the maximum speed the servo can reach in order to achieve the given position. If the servo is a rotation servo, the unit of the speed parameter is radian per second, otherwise, it is meter per second.

The servo_set_position function gives the acceleration the servo will use to reach the given position. If the servo is a rotation servo, the unit of the acc parameter is radian per second$\hat{2}$, otherwise, it is meter per second$\hat{2}$.

servo_set_position

NAME

servo_run_animation, servo_get_animation_number, servo_get_animation_range – *servo animation functions*

SYNOPSIS

```
#include <device/servo.h>
```

```
void servo_run_animation (DeviceTag servo, gint32 anim);
gint32 servo_get_animation_number (DeviceTag servo);
gfloat servo_get_animation_range (DeviceTag servo, gint32 anim);
```

DESCRIPTION

These functions are useful to perform non-robot-realistic animations. They do not refer to a real servo device, and permit to change dynamically the translation and rotation field of the Servo node. This results in more life-like animations, but should not be used in realistic simulations of real servo devices.

The `servo_run_animation` function starts the animation specified by `anim` which corresponds to the index of the `Animation` node in the `Servo` `animation` field. 0 is the first `Animation` node of the `MFNode` list. The animation is also started recursively in all the children `Servo` of the `Servo` specified by the `servo` parameter. Passing `-1` as `anim` will stop the animation in the specified `servo` and recursively in its subsequent `Servo` children.

The `servo_get_animation_number` function returns the number of `Animation` nodes present in the `animation` field of the specified `servo`.

The `servo_get_animation_range` function returns the range of the animation, that is the last value of the `key` field of the `Animation` node. The `Animation` node is specified by its `anim` index like with the `servo_run_animation`. The range value corresponds to the length of the animation cycle expressed in seconds.

2.11 Supervisor

The supervisor controller is a particular case of a robot controller, hence the `robot_live`, `robot_step`, `robot_get_device`, etc. functions also apply to supervisor controllers. Moreover, as long as the supervisor contains sensors and actuators in its list of children, the corresponding sensor and actuator functions can be used (except for the `differential_wheels_*` functions that are specific to differential wheels robots).

This section covers the supervisor specific functions, allowing the supervisor controller to track the position and orientation of `Solid` nodes in the scene, to move them, to take a snapshot of the scene, etc.

`supervisor_export_image`

NAME

`supervisor_export_image` – *save the current 3D image of the simulator into a JPEG file, suitable for building a webcam system*

SYNOPSIS

```
#include <device/supervisor.h>
 void supervisor_export_image (gchar *filename, guint8 quality);
```

DESCRIPTION

The `supervisor_export_image` function saves the current 3D image of the simulator window into a `jpeg` file as specified in the `filename` parameter. The `quality` parameter defines the `jpeg` quality (in the range 0 - 100). The `filename` parameter should specify a `jpeg` file (as an absolute or relative path), i.e., `"my_image.jpeg"` or `"/var/www/html/images/shot.jpg"`. Indeed, a temporary file is first saved, and then renamed to the requested `filename`. This avoids having a temporary unfinished (and hence corrupted) file for webcam applications.

EXAMPLE

A simple example of using the `supervisor_export_image` is provided in the `photographer` directory of the `controllers` directory. An example of a webcam system using `supervisor_export_image` is provided in the `webcam` directory of the `controllers` directory.

supervisor_import_node

NAME

`supervisor_import_node` – *import a node into the scene*

SYNOPSIS

```
#include <device/supervisor.h>
 void supervisor_import_node (gchar *filename, gint position);
```

DESCRIPTION

The `supervisor_import_node` function imports a Webots node into the scene. This node should be defined in a Webots file referenced to by the `filename` parameter. Such a file can be produced easily from Webots by selecting a node in the scene tree window and using the **Export Object** button.

The `position` parameter defines the position in the scene tree where the new node is going to be inserted. It can be positive or negative. Here are a few examples for the `position` parameter:

- 0: insert at the beginning of the scene tree.
- 1: insert at the second position.
- 2: insert at the third position.
- etc.
- -1: insert at the last position.
- -2: insert at the second position from the end of the scene tree.
- -3: insert at the third position from the end.
- etc.

As in `supervisor_export_image`, the `filename` parameter can be speficied with an absolute or a relative path.

`supervisor_node_get_from_def`

NAME

`supervisor_node_get_from_def`, `supervisor_node_was_found` – *get a pointer to a node of the scene from its DEF name and check if that node exists.*

SYNOPSIS

```
#include <device/supervisor.h>
NodeRef supervisor_node_get_from_def (gchar *defname);
gboolean supervisor_node_was_found (NodeRef node);
```

DESCRIPTION

The `supervisor_node_get_from_def` function retrieves a pointer to a node of the scene from its DEF name. The return value can be used for subsequent calls to functions refering to a node of the scene. Note that this function always return a non NULL value, even if the node does not exist in the scene.

The `supervisor_node_was_found` checks whether the node refered to by `node` really exists in the scene. It returns `TRUE` if the node exists and `FALSE` otherwise. Please, note that these functions have to be called after calling the `robot_step` function, so that they are able to return correct values.

supervisor_set_label

NAME

supervisor_set_label – *display a text label over the 3D scene*

SYNOPSIS

```
#include <device/supervisor.h>

NodeRef supervisor_set_label (guint16 id, gchar *text, gfloat x, gfloat
y, gfloat size, guint32 color);
```

DESCRIPTION

The supervisor_set_label function displays a text label over the 3D scene in Webots' main window. The id parameter is a an identifier for the label, you can choose any value in the range 0-65535. It will be used later on when you want to change that label, like updating the text. The text parameter is a text string which should contain only displayable characters in the range 32-127. The x and y parameters are the coordinates of the upper left corner of the text, relative to the upper left corner of the 3D window. These floating point values are expressed in percent of the 3D window width and height, hence, they should lie in the range 0-1. The size parameter defines the size of the font to be used. It is expressed with the same unit as the y parameter. Finally, the color parameter defines the color for the label. It is expressed as 32 bits RGB integer value, when the first byte is ignored, the second byte represents the red component, the third byte represents the green component and the last byte represents the blue component.

EXAMPLE

- `supervisor_set_label(0,"hello world",0,0,0.1,0xff0000);`

  will display the label "hello world" in red at the upper left corner of the 3D window.

- `supervisor_set_label(1,"hello dad",0,0.1,0.1,0x00ff00);`

  will display the label "hello dad" in green, just below.

- `supervisor_set_label(0,"hello universe",0,0,0.1,0xffff00);`

  will change the label "hello world" defined earlier into "hello universe", setting a yellow color to the new text.

supervisor_simulation_quit

NAME

supervisor_simulation_quit – *terminate the simulator and controller processes*

SYNOPSIS

```
#include <device/supervisor.h>
 void supervisor_simulation_quit ();
```

DESCRIPTION

The supervisor_simulator_quit function sends a request to the simulator process, asking to terminate and quit immediately. As a result of terminating the simulator process, all the controller processes, including the calling supervisor controller process will terminate.

supervisor_set_label

NAME

supervisor_field_get, supervisor_field_set – *get and set the contents of the field of a node in the scene*

SYNOPSIS

```
#include <device/supervisor.h>
 void supervisor_field_get (NodeRef node, field_type type, gpointer data,
guint16 ms);
 void supervisor_field_set (NodeRef node, field_type type, gpointer data);
```

DESCRIPTION

The supervisor_field_get function allows the supervisor controller to track the evolution of some fields of a node. Currently only a few fields are trackable, as described in the following list of field types. Each ms milliseconds, the new value of the field (if any) is stored at data with a specific data type (usually an array of gfloat). The type parameter should be a combination of the following primitive constants, as defined in the supervisor.h header file:

```
SUPERVISOR_FIELD_TRANSLATION_X
SUPERVISOR_FIELD_TRANSLATION_Y
SUPERVISOR_FIELD_TRANSLATION_Z
SUPERVISOR_FIELD_ROTATION_X
SUPERVISOR_FIELD_ROTATION_Y
SUPERVISOR_FIELD_ROTATION_Z
SUPERVISOR_FIELD_ROTATION_ANGLE
SUPERVISOR_FIELD_BATTERY_CURRENT
```

Some predefined combinations include:

```
SUPERVISOR_FIELD_TRANSLATION = SUPERVISOR_FIELD_TRANSLATION_X+
SUPERVISOR_FIELD_TRANSLATION_Y+SUPERVISOR_FIELD_TRANSLATION_Z

SUPERVISOR_FIELD_ROTATION = SUPERVISOR_FIELD_ROTATION_X+
SUPERVISOR_FIELD_ROTATION_Y+SUPERVISOR_FIELD_ROTATION_Z+
SUPERVISOR_FIELD_ROTATION_ANGLE

SUPERVISOR_FIELD_TRANSLATION_AND_ROTATION =
SUPERVISOR_FIELD_TRANSLATION+SUPERVISOR_FIELD_ROTATION
```

It is necessary that the `data` parameter be a pointer towards a large enough array of `gfloat`, able to contain all the requested values. One `gfloat` is necessary for each primitive value. Please note that this `data` pointer should point to a valid memory chunk at the time of the `robot_step` function. Hence, it should not be stored on the heap of a local function that won't exist any more when calling `robot_step`. Instead, it has to be dynamically allocated, or declared as a local or global static variable, or declared in the same function that (or a parent function of) the function calling `robot_step`.

In order to disable the tracking of a field, call the `supervisor_field_get` function with a `ms` parameter set to 0.

The `supervisor_field_set` function works the same way as `supervisor_field_get`, except that it changes the value of the requested `field` instead of reading it.

EXAMPLE

An simple example of using field tracking is given in the `supervisor` controller.

2.12 TouchSensor

touch_sensor_enable

NAME

touch_sensor_enable, touch_sensor_disable – *enable and disable the touch sensor measurements*

SYNOPSIS

```
#include <device/touch_sensor.h>
 void touch_sensor_enable (DeviceTag sensor, guint16 ms);
 void touch_sensor_disable (DeviceTag sensor);
```

DESCRIPTION

touch_sensor_enable allows the user to enable a touch sensor measurement each ms milliseconds.

touch_sensor_disable turns the touch sensor off, saving computation time.

touch_sensor_get_value

NAME

touch_sensor_get_value – *get the touch sensor measure*

SYNOPSIS

```
#include <device/touch_sensor.h>
 guint16 touch_sensor_get_value (DeviceTag sensor);
```

DESCRIPTION

touch_sensor_get_value returns the last value measured by the specified touch sensor. This value is computed by the simulator according to the lookup table of the TouchSensor node. Hence, the value range for the return value is defined by this lookup table.

Chapter 3

Webots File Format

3.1 File Structure

Webots files must begin with the characters:

```
#VRML_SIM V3.0 utf8
```

and the following nodes have to appear:

```
  WorldInfo
  Viewpoint
  Background
```

3.1.1 Example

```
#VRML_SIM V3.0 utf8
WorldInfo {
  info [
    "Description"
    "Author: first name last name <e-mail>"
    "Date: DD MMM YYYY"
  ]
}
Viewpoint {
  orientation 1 0 0 -0.8
  position 0.25 0.708035 0.894691
}
Background {
  skyColor [
```

```
    0.4 0.7 1
  ]
}
PointLight {
  ambientIntensity 0.54
  intensity 0.5
  location 0 1 0
}
```

The file extension is `.wbt` (standing for WeBoTs).

3.2 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 texture textureTransform }Background { skyColor }
Box { size }
Color { color }
Cone { bottomRadius height side bottom }
Coordinate { point }
Cylinder { bottom height radius side top }
DirectionalLight { ambientIntensity color direction intensity on }
ElevationGrid { color colorPerVertex height xDimension zDimension xSpacing
 zSpacing }
Fog { color fogType visibilityRange }
Group { children }
ImageTexture { url repeatS repeatT}
IndexedFaceSet { ccw coord coordIndex convex creaseAngle texCoord
 texCoordIndex}
```

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.

```
IndexedLineSet { coord coordIndex }
Material { ambientIntensity diffuseColor emissiveColor shininess
 specularColor transparency}
PointLight { ambientIntensity attenuation color intensity location on radius}
```

Note:

The value of `radius` is ignored. It is considered as infinite.

```
Shape { appearance geometry }
Sphere { radius subdivision }
```

Note:

Spheres are rendered as icosaedrons with 20 faces when the subdivision field is set to 0. If the subdivision field is 1 (default value), then each face is subdivided into 4 faces, which makes 80 faces. With a subdivision parameter set to 2, 320 faces will be rendered, making the sphere very smooth.

```
TextureCoordinate { point }
TextureTransform { scale center translation rotation }
Transform { translation rotation scale children }
Viewpoint { fieldOfView orientation position }
WorldInfo { title info }
```

Note:

A texture can be mapped only on an IndexedFaceSet shape. The `texCoord` and `texCoordIndex` must be filled. The image used as a texture must be a `.png` or a `.jpg` file, its size must be 2n̂ * 2n̂ pixels (for example 8x8, 16x16, 32x32, 64x64, 128x128 pixels). Transparent PNG images are not authorized for textures in Webots.

3.3 Webots nodes

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

3.3.1 Camera

```
Camera {
  scale             1 1 1     SFVec3f
  translation       0 0 0     SFVec3f
  rotation          0 1 0 0   SFRotation
  children          []        MFNode
  name              ""        SFString
  model             ""        SFString
  author            ""        SFString
  constructor       ""        SFString
  description       ""        SFString
  boundingObject    NULL      SFNode
  physics           NULL      SFNode
  joint             NULL      SFNode
  locked            FALSE     SFBool
```

```
  fieldOfView       0.7854    SFFloat
  width             64        SFInt32
  height            64        SFInt32
  type              "color"   SFString
}
```

3.3.2 Charger

```
Charger {
  scale             1 1 1     SFVec3f
  translation       0 0 0     SFVec3f
  rotation          0 1 0 0   SFRotation
  children          []        MFNode
  name              ""        SFString
  model             ""        SFString
  author            ""        SFString
  constructor       ""        SFString
  description       ""        SFString
  boundingObject    NULL      SFNode
  physics           NULL      SFNode
  joint             NULL      SFNode
  locked            FALSE     SFBool
  battery           []        MFFloat
  radius            0.2       SFFloat
}
```

3.3.3 CustomRobot

```
CustomRobot {
  scale             1 1 1     SFVec3f
  translation       0 0 0     SFVec3f
  rotation          0 1 0 0   SFRotation
  children          []        MFNode
  name              ""        SFString
  model             ""        SFString
  author            ""        SFString
  constructor       ""        SFString
  description       ""        SFString
  boundingObject    NULL      SFNode
  physics           NULL      SFNode
  joint             NULL      SFNode
  locked            FALSE     SFBool
  controller        "void"    SFString
```

```
  synchronisation   TRUE      SFBool
  battery           []        MFFloat
  cpuConsumption    0         SFFloat
}
```

3.3.4 DifferentialWheels

```
DifferentialWheels {
  scale             1 1 1     SFVec3f
  translation       0 0 0     SFVec3f
  rotation          0 1 0 0   SFRotation
  children          []        MFNode
  name              ""        SFString
  model             ""        SFString
  author            ""        SFString
  constructor       ""        SFString
  description       ""        SFString
  boundingObject    NULL      SFNode
  physics           NULL      SFNode
  joint             NULL      SFNode
  locked            FALSE     SFBool
  controller        "void"    SFString
  synchronisation   TRUE      SFBool
  battery           []        MFFloat
  cpuConsumption    0         SFFloat
  motorConsumption  0         SFFloat
  axleLength        0.1       SFFloat
  wheelRadius       0.01      SFFloat
  maxSpeed          10        SFFloat
  maxAcceleration   10        SFFloat
  speedUnit         0.1       SFFloat
  slipNoise         0.1       SFFloat
  encoderNoise      -1        SFFloat
}
```

3.3.5 DistanceSensor

```
DistanceSensor {
  scale             1 1 1           SFVec3f
  translation       0 0 0           SFVec3f
  rotation          0 1 0 0         SFRotation
  children          []              MFNode
  name              ""              SFString
```

```
  model             ""                SFString
  author            ""                SFString
  constructor       ""                SFString
  description       ""                SFString
  boundingObject    NULL              SFNode
  physics           NULL              SFNode
  joint             NULL              SFNode
  locked            FALSE             SFBool
  lookupTable       0 0 0,0.1 1000 0  MFVec3f
  type              "infra-red"       SFString
}
```

3.3.6 Emitter

```
Emitter {
  scale             1 1 1       SFVec3f
  translation       0 0 0       SFVec3f
  rotation          0 1 0 0     SFRotation
  children          []          MFNode
  name              ""          SFString
  model             ""          SFString
  author            ""          SFString
  constructor       ""          SFString
  description       ""          SFString
  boundingObject    NULL        SFNode
  physics           NULL        SFNode
  joint             NULL        SFNode
  locked            FALSE       SFBool
  type              "infra-red" SFString
  range             0.5         SFFloat
  channel           0           SFInt32
  baudRate          9600        SFInt32
  byteSize          8           SFInt32
  bufferSize        1024        SFInt32
}
```

3.3.7 Gripper

```
Gripper {
  scale             1 1 1      SFVec3f
  translation       0 0 0      SFVec3f
  rotation          0 1 0 0    SFRotation
  children          []         MFNode
```

```
  name              ""          SFString
  model             ""          SFString
  author            ""          SFString
  constructor       ""          SFString
  description       ""          SFString
  boundingObject    NULL        SFNode
  physics           NULL        SFNode
  joint             NULL        SFNode
  locked            FALSE       SFBool
  position          0           SFFloat
}
```

3.3.8 HyperGate

```
HyperGate {
  scale             1 1 1     SFVec3f
  translation       0 0 0     SFVec3f
  rotation          0 1 0 0   SFRotation
  children          []        MFNode
  name              ""        SFString
  model             ""        SFString
  author            ""        SFString
  constructor       ""        SFString
  description       ""        SFString
  boundingObject    NULL      SFNode
  physics           NULL      SFNode
  joint             NULL      SFNode
  locked            FALSE     SFBool
  url               ""        SFString
  radius            0.1       SFFloat
  height            0.1       SFFloat
  maxFileSize       65536     SFInt32
}
```

3.3.9 LightSensor

```
LightSensor {
  scale             1 1 1             SFVec3f
  translation       0 0 0             SFVec3f
  rotation          0 1 0 0           SFRotation
  children          []                MFNode
  name              ""                SFString
  model             ""                SFString
```

```
  author            ""                SFString
  constructor       ""                SFString
  description       ""                SFString
  boundingObject    NULL              SFNode
  physics           NULL              SFNode
  joint             NULL              SFNode
  locked            FALSE             SFBool
  lookupTable       0 0 0,0.1 1000 0  MFVec3f
}
```

3.3.10 Physics

```
Physics {
  field SFFloat mass                   0     # expressed in kg
  field SFVec3f velocity               0 0 0 # expressed in m/s
  field SFFloat angularVelocity        0     # expressed in rad/s
  field SFFloat staticFriction         0     # coefficient
  field SFFloat kineticFriction        0     # coefficient
  field SFFloat impactEnergyAbsorbtion 0     # coefficient
  field SFFloat surfaceNoise           0     # noise due to surface defaults
}
```

3.3.11 Receiver

```
Receiver {
  scale             1 1 1       SFVec3f
  translation       0 0 0       SFVec3f
  rotation          0 1 0 0     SFRotation
  children          []          MFNode
  name              ""          SFString
  model             ""          SFString
  author            ""          SFString
  constructor       ""          SFString
  description       ""          SFString
  boundingObject    NULL        SFNode
  physics           NULL        SFNode
  joint             NULL        SFNode
  locked            FALSE       SFBool
  type              "infra-red" SFString
  channel           0           SFInt32
  baudRate          9600        SFInt32
  byteSize          8           SFInt32
  bufferSize        1024        SFInt32
}
```

3.3.12 Servo

```
Servo {
  scale             1 1 1      SFVec3f
  translation       0 0 0      SFVec3f
  rotation          0 1 0 0    SFRotation
  children          []         MFNode
  name              ""         SFString
  model             ""         SFString
  author            ""         SFString
  constructor       ""         SFString
  description       ""         SFString
  boundingObject    NULL       SFNode
  physics           NULL       SFNode
  joint             NULL       SFNode
  locked            FALSE      SFBool
  type              "rotation" SFString
  maxSpeed          10         SFFloat
  maxAcceleration   10         SFFloat
  animation         []         MFNode
}
```

3.3.13 Solid

```
Solid {
  scale             1 1 1     SFVec3f
  translation       0 0 0     SFVec3f
  rotation          0 1 0 0   SFRotation
  children          []        MFNode
  name              ""        SFString
  model             ""        SFString
  author            ""        SFString
  constructor       ""        SFString
  description       ""        SFString
  boundingObject    NULL      SFNode
  physics           NULL      SFNode
  joint             NULL      SFNode
  locked            FALSE     SFBool
}
```

3.3.14 Supervisor

```
Supervisor {
```

```
  scale             1 1 1     SFVec3f
  translation       0 0 0     SFVec3f
  rotation          0 1 0 0   SFRotation
  children          []        MFNode
  name              ""        SFString
  model             ""        SFString
  author            ""        SFString
  constructor       ""        SFString
  description       ""        SFString
  boundingObject    NULL      SFNode
  physics           NULL      SFNode
  joint             NULL      SFNode
  locked            FALSE     SFBool
  controller        "void"    SFString
  synchronisation   TRUE      SFBool
  battery           []        MFFloat
  cpuConsumption    0         SFFloat
}
```

3.3.15 TouchSensor

```
TouchSensor {
  scale             1 1 1             SFVec3f
  translation       0 0 0             SFVec3f
  rotation          0 1 0 0           SFRotation
  children          []                MFNode
  name              ""                SFString
  model             ""                SFString
  author            ""                SFString
  constructor       ""                SFString
  description       ""                SFString
  boundingObject    NULL              SFNode
  physics           NULL              SFNode
  joint             NULL              SFNode
  locked            FALSE             SFBool
  type              "bumper"          SFString
  lookupTable       0 0 0,0.1 1 0     MFVec3f
}
```