

From Webots 5 to Webots 6

What changed in Webots API ?

Yvan Bourquin

Cyberbotics Ltd.

Summary

In the transition from Webots 5 to Webots 6 an important cleanup and redesign of Webots APIs (Application Programming Interfaces) was undertaken. All C functions were renamed and several were deprecated. In addition, an entirely new JAVA API was added and replaces the JAVA API of Webots 5. The C++ and Python APIs were also largely modified. The deprecated functions and methods of Webots 5 are no longer documented in Webots 6. This document explains how to port existing Webots 5 code to Webots 6.

C controller API

Controllers using the C API and compiled with Webots 5 will usually work right away with Webots 6 thanks to a binary compatibility layer. Recompiling these controllers with Webots 6 will print a bunch of deprecation warnings. Despite the warnings the recompilation will usually not give any error and will run unmodified. For a transitional period the deprecated C API of Webots 5 will remain functional in Webots 6. However, on the long run, it is recommended to adapt your code to Webots 6 API.

In the case you are porting a C++ controller using the C API you will have to add this flag to your project's Makefile for Webots 6:

Webots 6
<code>CFLAGS=-I"\${WEBOTS_HOME_PATH}/include/controller/c"</code>

The reason is that in Webots 6, unlike Webots 5, the C API is not automatically enabled for C++ controller code. Instead, the default API for C++ controller is now the C++ API.

JAVA controller API

- **Webots 6 JAVA API was completely redesigned**

Webots 5 JAVA API is entirely deprecated. Recompiling the Webots 5 JAVA controllers will give deprecation warnings when compiled with Webots 6. However the controllers should compile and run unmodified. For a transitional period the deprecated JAVA API will remain functional.

Python and C++ controller APIs

- **Webots 6 C++ and Python APIs were completely redesigned**

You will need to adapt your C++ and Python controllers when upgrading to Webots 6. Unlike the C and JAVA API, there are no transitional compatibility layer.

Physics plugin API

- **Physics plugins will need to be recompiled for Webots 6 !**

Executing in Webots 6 a plugin that was compiled with Webots 5 will usually give this error message:

```
[WARNING] Error while loading required plugin: "my_physics"
```

```
[WARNING] physics/my_physics/my_physics.so: undefined symbol: dWebotsConsolePrint
```

Just recompiling the plugin will fix the problem. The recompilation should normally not produce any error because the physics plugin API remained unchanged from Webots 5 to Webots 6.

World files

- **Webots 5 .wbt files can be opened with Webots 6**
- **Webots 6 .wbt files cannot be opened with Webots 5**

The CustomRobot node was renamed Robot. When a Webots 5 .wbt file is opened and saved with Webots 6, all CustomRobot nodes are automatically renamed Robot for compatibility.

Several other things have changed in the .wbt file format, these are the most important changes:

The version number in the file header:

Webots 5	Webots 6
#VRML_SIM V5.0 utf8	#VRML_SIM V6.0 utf8

The “follow object” state is now saved in the Viewpoint node instead of in a comment:

Webots 5	Webots 6
#!robotView: "ghostdog"	Viewpoint { ... follow "ghostdog" }

The GUI layout and list of open files is no longer saved in the .wbt file. This information is now saved in separate “project” files (hidden):

Webots 5	Webots 6
<pre>#!mainWindow: 0 0 0.5 0.7 #!sceneTreeWindow: 0 0.7 0.5 0.3 0.25 #!textEditorWindow: 0.5 0 0.5 0.7 0.37 0 "controllers/ghostdog/ghostdog.c" #!logWindow: 0.5 0.7 0.5 0.3</pre>	Now saved separately in "project" files

C controller API

Function, type and constant names

To avoid name space clashes, in Webots 6, a prefix was added to the names of all functions, types and constants. Examples:

Webots 5 (deprecated)	Webots 6
robot_get_device() DeviceTag NodeRef SERVO_INFINITY CHANNEL_BROADCAST ...	wb_robot_get_device() WbDeviceTag WbNodeRef WB_SERVO_INFINITY WB_CHANNEL_BROADCAST ...

Includes

In Webots 6, the C header files must now be included from the “webots” folder instead of the “device” folder. Examples:

Webots 5 (deprecated)	Webots 6
#include <device/light_sensor.h>	#include <webots/light_sensor.h> #include <webots/utils/motion.h>

These two different folders contain distinct sets of header files. The “device” folder contains Webots 5 deprecated header files that will issue warnings, while the “webots” folder contains Webots 6 new header files.

C Program Structure

In Webots 6, the initialization and cleanup code does no longer need to be placed in special functions. The robot_live(), robot_run() and robot_die() functions are deprecated. Instead it is now required to call wb_robot_init() before, and wb_robot_cleanup() after, any other call to a Webots function. Here is an example:

Webots 5 (deprecated)	Webots 6
#include <device/robot.h> #include <device/distance_sensor.h> #define STEP 16 static DeviceTag ds; static void reset() { ds=robot_get_device(“ds”); }	#include <webots/robot.h> #include <webots/distance_sensor.h> #define STEP 16 int main() { wb_robot_init(); WbDeviceTag ds=wb_robot_get_device(“ds”); }

<pre> distance_sensor_enable(ds, STEP); } static void cleanup() { /* your cleanup code here */ } static int run(int ms) { /* sense & actuate */ unsigned short v; v=distance_sensor_get_value(ds); return STEP; } int main() { robot_live(reset); robot_die(cleanup); robot_run(run); return 0; } </pre>	<pre> wb_distance_sensor_enable(ds, STEP); do { /* sense & actuate */ double v; v=wb_distance_sensor_get_value(ds); } while (wb_robot_step(STEP) != -1); /* your cleanup code here */ wb_robot_cleanup(); return 0; } </pre>
---	---

The `wb_robot_step()` function returns -1 to indicate that Webots 6 will terminate the controller (this happens when the user opens a world file, reverts the simulation or quits Webots). Checking for this condition is the only way to be informed of the upcoming controller termination and to have a chance to cleanup: close files and free data, etc. This method now replaces the previous mechanism based on `robot_die()`.

In case a cleanup is not necessary, the simpler alternative below can also be used. Note that in this case it is not necessary to call `wb_robot_cleanup()`.

Webots 5 (deprecated)	Webots 6
<pre> #include <device/robot.h> #include <device/light_sensor.h> #define STEP 16 static DeviceTag ls; static void reset() { ls = robot_get_device("ls"); light_sensor_enable(ls, STEP); } int main() { robot_live(reset); for (;;) { /* sense & actuate */ unsigned short v; v = light_sensor_get_value(ls); robot_step(STEP); } return 0; } </pre>	<pre> #include <webots/robot.h> #include <webots/light_sensor.h> #define STEP 16 int main() { wb_robot_init(); WbDeviceTag ls=wb_robot_get_device("ls"); wb_light_sensor_enable(ls, STEP); for (;;) { /* sense & actuate */ double v; v = wb_light_sensor_get_value(ls); wb_robot_step(STEP); } return 0; } </pre>

Webots 5 (deprecated)	Webots 6
}	

Floating point values

Webots 5 C, C++ and JAVA API used have the float type (32 bit) to represent floating point values. In Webots 6 this was systematically changed to double (64 bit). Most functions like the examples below will recompile without any problem with Webots 6.

Webots 5 (deprecated)
void servo_set_position(DeviceTag tag, float pos) float servo_get_position(DeviceTag tag) ...
Webots 6
void wb_servo_set_position(WbDeviceTag tag, double pos) double wb_servo_get_position(WbDeviceTag tag) ...

However functions returning float* will cause some trouble because Webots 6 equivalents now return double* which is not compatible. Fortunately there were only two functions of this type in Webots 5 (see below). You will have to update your code if you used one of them:

Webots 5 (deprecated)
const float *accelerometer_get_values(DeviceTag tag) const float *receiver_get_emitter_direction(DeviceTag tag)
Webots 6
const double *wb_accelerometer_get_values(WbDeviceTag tag) const double *wb_receiver_get_emitter_direction(WbDeviceTag tag)

Printing to Webots console

Webots 6 automatically redirects the standard output and error streams of controllers to the built-in console. So any method for printing to the standard output and error streams can now be used in controller code. This replaces the robot_console_printf() function of Webots 5.

	Webots 5 (deprecated)	Webots 6
C	robot_console_printf("Hello!");	printf("Hello!"); ...
C++	robot_console_printf("Hello!");	printf("Hello!"); cout << "Hello!"; ...

LookupTables and related functions

In Webots 5 all the function based on lookup tables used to return unsigned short values. In Webots 6 the equivalent functions now return double values. This change is backwards compatible; it may produce compilation warnings but it will normally not affect the behavior of your controllers.

Webots 5 (deprecated)
unsigned short distance_sensor_get_value(DeviceTag tag) unsigned short light_sensor_get_value(DeviceTag tag) unsigned short touch_sensor_get_value(DeviceTag tag)
Webots 6
double wb_distance_sensor_get_value(WbDeviceTag tag) double wb_light_sensor_get_value(WbDeviceTag tag) double wb_touch_sensor_get_value(WbDeviceTag tag)

Note that, in order to be consistent with the function return values, the output values (second column) of all lookup tables in Webots 6 are now interpreted as double values (see below). As a consequence lookup tables can now also return negative values.

```
lookupTable [  
  -19.62 -19.62 0.01  
  19.62  19.62 0.01  
]
```

Supervisor functions

Webots 5 supervisor functions for accessing nodes in the scene tree are now deprecated. There are two reasons for this deprecation: first, users often misunderstood how to setup the parameters and when to call these function. Second, these functions worked only with a small subset of all nodes and fields types.

In Webots 6 a new set of functions is available and offers more flexibility. In particular it is now possible to read every field and every node of the scene tree.

Webots 5 (deprecated)
NodeRef supervisor_node_get_from_def(const char *DEF) bool supervisor_node_was_found(NodeRef)
Webots 6
WbNodeRef wb_supervisor_node_get_root() WbNodeRef wb_supervisor_node_get_from_def(const char *def) WbNodeType wb_supervisor_node_get_type(WbNodeRef node) const char *wb_supervisor_node_get_name(WbNodeRef) WbFieldRef wb_supervisor_node_get_field(WbNodeRef node, const char *field_name) WbFieldType wb_supervisor_field_get_type(WbFieldRef field) const char *wb_supervisor_field_get_type_name(WbFieldRef field) int wb_supervisor_field_get_count(WbFieldRef field)

There is a new set of field “accessor” functions: four functions for each type of field: one “getter”, one “setter”, one “single” and one “multiple” field variation.

Webots 5 (deprecated)	
<code>supervisor_field_set(NodeRef, FieldType, const void *data)</code> <code>supervisor_field_get(NodeRef, FieldType, void *data, unsigned short ms)</code>	
Webots 6	
<code>wb_supervisor_field_get_sf_bool()</code> <code>wb_supervisor_field_get_sf_int32()</code> <code>wb_supervisor_field_get_sf_float()</code> <code>wb_supervisor_field_get_sf_vec2f()</code> <code>wb_supervisor_field_get_sf_vec3f()</code> <code>wb_supervisor_field_get_sf_rotation()</code> <code>wb_supervisor_field_get_sf_color()</code> <code>wb_supervisor_field_get_sf_string()</code> <code>wb_supervisor_field_get_sf_node()</code> <code>wb_supervisor_field_set_sf_bool()</code> <code>wb_supervisor_field_set_sf_int32()</code> <code>wb_supervisor_field_set_sf_float()</code> <code>wb_supervisor_field_set_sf_vec2f()</code> <code>wb_supervisor_field_set_sf_vec3f()</code> <code>wb_supervisor_field_set_sf_rotation()</code> <code>wb_supervisor_field_set_sf_color()</code> <code>wb_supervisor_field_set_sf_string()</code>	<code>wb_supervisor_field_get_mf_bool()</code> <code>wb_supervisor_field_get_mf_int32()</code> <code>wb_supervisor_field_get_mf_float()</code> <code>wb_supervisor_field_get_mf_vec2f()</code> <code>wb_supervisor_field_get_mf_vec3f()</code> <code>wb_supervisor_field_get_mf_rotation()</code> <code>wb_supervisor_field_get_mf_color()</code> <code>wb_supervisor_field_get_mf_string()</code> <code>wb_supervisor_field_get_mf_node()</code> <code>wb_supervisor_field_set_mf_bool()</code> <code>wb_supervisor_field_set_mf_int32()</code> <code>wb_supervisor_field_set_mf_float()</code> <code>wb_supervisor_field_set_mf_vec2f()</code> <code>wb_supervisor_field_set_mf_vec3f()</code> <code>wb_supervisor_field_set_mf_rotation()</code> <code>wb_supervisor_field_set_mf_color()</code> <code>wb_supervisor_field_set_mf_string()</code>

Here are some examples showing how to port existing Supervisor code to the new API. The first example shows how to read the position of a robot:

Webots 5 (deprecated)
<pre>static float translation[3]; static void reset() { ... // this must be done once only NodeRef robot = supervisor_node_get_from_def("MY_ROBOT"); supervisor_field_get(robot, SUPERVISOR_FIELD_TRANSLATION, translation, STEP); ... } static int run(int ms) { ... // this is done repeatedly robot_console_printf("MY_ROBOT is at position: %g %g %g\n", translation[0], translation[1], translation[2]); ... }</pre>
Webots 6
...


```

// do this once only
WbNodeRef robot = wb_supervisor_node_get_from_def("MY_ROBOT");
WbFieldRef trans_field = wb_supervisor_node_get_field(robot, "translation");
...
for (;;) {
    ...
    // this is done repeatedly
    const double *translation = wb_supervisor_field_get_sf_vec3f(trans_field);
    printf("MY_ROBOT is at position: %g %g %g\n",
        translation[0], translation[1], translation[2]);
    ...
}

```

Note that the `ms` (STEP) parameter has disappeared in Webots 6, because the new functions return the result synchronously. This second example shows how to change the position of a robot:

Webots 5 (deprecated)
<pre> static NodeRef robot = NULL; static void reset() { ... // do this once only robot = supervisor_node_get_from_def("MY_ROBOT"); ... } static int run(int ms) { ... // do this repeatedly float pos[3] = { 0.0, 1.0, 4.0 }; supervisor_field_set(robot, SUPERVISOR_FIELD_TRANSLATION, pos); ... } </pre>
Webots 6
<pre> ... // need to do this once only WbNodeRef robot = wb_supervisor_node_get_from_def("MY_ROBOT"); WbFieldRef trans_field = wb_supervisor_node_get_field(robot, "translation"); ... for (;;) { ... // do this repeatedly double pos[3] = { 0.0, 1.0, 4.0 }; wb_supervisor_field_set_sf_vec3f(trans_field, pos); ... } </pre>

This third example show how to change the controller of a robot:

Webots 5 (deprecated)
<pre>... NodeRef robot = supervisor_node_get_from_def("MY_ROBOT"); ... supervisor_robot_set_controller(robot, "braitenberg"); ...</pre>
Webots 6
<pre>... WbNodeRef robot = wb_supervisor_node_get_from_def("MY_ROBOT"); WbFieldRef controller_field = wb_supervisor_node_get_field(robot, "controller"); ... wb_supervisor_field_set_sf_string(controller_field, "braitenberg"); ...</pre>

GPS functions

In Webots 6, the concept of “matrix” was removed from the GPS device. So the `gps_get_matrix()` and `gps_euler()` functions are deprecated. As a consequence in Webots 6, the GPS device can no longer give rotational information; it only gives positional information (like a real world GPS). The rotation of an object must now be estimated either by using the new Compass device, or with the new `wb_supervisor_field_get_sf_rotation()` function.

Webots 5 (deprecated)	Webots 6
<pre>gps_get_matrix() gps_euler()</pre>	<pre>wb_compass_get_values() wb_supervisor_field_get_sf_rotation()</pre>

The three access macros (below) were removed because they became obsolete as a consequence of the removal of `gps_get_matrix()`. They are now replaced by the new `wb_gps_get_values()` function.

Webots 5 (deprecated)	Webots 6
<pre>#define gps_position_x() #define gps_position_y() #define gps_position_z()</pre>	<pre>wb_gps_get_values()</pre>

This example shows the difference between Webots 5 and Webots 6:

Webots 5 (deprecated)	Webots 6
<pre>const float *matrix; matrix = gps_get_matrix(gps); float x = gps_position_x(matrix); float y = gps_position_y(matrix); float z = gps_position_z(matrix);</pre>	<pre>const double *pos; pos = wb_gps_get_values(gps); double x = pos[0]; double y = pos[1]; double z = pos[2];</pre>

Note that the returned vector (`pos`) is a pointer to internal data managed by the GPS node. It is illegal to free this pointer. Furthermore the returned vector (`pos`) has exactly three components, therefore only

the indexes 0, 1 and 2 are valid to access it:

```
double wrong = pos[3]; // WRONG WRONG WRONG !
```

Accelerometer functions

In Webots 6, the following macros were removed:

Webots 5 (deprecated)
<pre>#define accelerometer_value_x() #define accelerometer_value_y() #define accelerometer_value_z()</pre>

It is now recommended to access the returned values directly as shown here:

Webots 5 (deprecated)
<pre>const float *acc = accelerometer_get_values(tag); float xacc = accelerometer_value_x(acc); float yacc = accelerometer_value_y(acc); float zacc = accelerometer_value_z(acc);</pre>
Webots 6
<pre>const double *acc = wb_accelerometer_get_values(tag); double xacc = acc[0]; double yacc = acc[1]; double zacc = acc[2];</pre>

Note that the returned vector (acc) is a pointer to internal data managed by the Accelerometer node. It is illegal to free this pointer. Furthermore the returned vector (acc) has exactly three components, therefore only the indexes 0, 1 and 2 are valid to access it:

```
double wrong = acc[3]; // WRONG WRONG WRONG !
```

Emitter/Receiver functions

The `emitter_send_packet()` function was renamed `wb_emitter_send()`.

Webots 5 (deprecated)	Webots 6
<code>emitter_send_packet()</code>	<code>wb_emitter_send()</code>

Pen functions

In Webots 6, the color passed to the `wb_pen_set_ink_color()` function must now be specified as a single `int` argument instead of three floats (RGB) as it was in Webots 5. The `int` contains the bit values for the red green and blue components each specified with 8 bits. For example: 0x000000 is black, 0xffffffff is white, 0xff0000 is red, 0x00ff00 is green, 0x0000ff blue, etc.

Webots 5 (deprecated)
<pre>void pen_set_ink_color(DeviceTag,float red,float green,float blue,float density);</pre>

Webots 6
<code>void wb_pen_set_ink_color(WbDeviceTag, int color, double density);</code>

Here is an example:

Webots 5 (deprecated)
<code>pen_set_ink_color(tag, 1.0, 0.0, 0.0, 1.0); // red and opaque</code>
Webots 6
<code>wb_pen_set_ink_color(tag, 0xff0000, 1.0); // red and opaque</code>

CustomRobot functions

All the `custom_robot_*`() functions are deprecated. In particular, the `custom_robot_move()` function was deprecated because it was inconsistent with the fact that a real robot can not “magically” move itself to a random location.

Randomly moving a robot is now reserved to Supervisor controllers and must be achieved through the `wb_supervisor_field_set_sf_vec3f()` and / or `wb_supervisor_field_set_sf_rotation()` functions:

Webots 5 (deprecated)
<code>void custom_robot_move(float tx, float ty, float tz, float rx, float ry, float rz, float alpha)</code>
Webots 6
<code>void wb_supervisor_field_set_sf_vec3f(WbFieldRef field, const double *values)</code> <code>void wb_supervisor_field_set_sf_rotation(WbFieldRef field, const double *values)</code>

Similarly the two functions below were removed because a real robot controller cannot apply a random force to its own body. There is no controller API replacement for these functions. If you want to add random forces or torques to your simulation your are now advised to do this with a physics plugin:

Webots 5 (deprecated)
<code>custom_robot_set_rel_force_and_torque()</code> <code>custom_robot_set_abs_force_and_torque()</code>
Webots 6
<code>// There is no replacement in the controller API</code> <code>// You need to use these ODE functions in the physics plugin</code> <code>dBodyAddForce()</code> <code>dBodyAddTorque()</code> <code>dBodyAddRelForce()</code> <code>dBodyAddRelTorque()</code>

JAVA Language API

JAVA Program Structure

In the Webots 5 JAVA API, all the methods (~130) were contained in a single class named Controller. In Webots 6, this was completely redesigned in order to offer a more object-oriented structure. The Java API is now automatically generated from the C++ API using SWIG. Note that the C++, Python and JAVA APIs now offer exactly the same structure and functionality.

The Controller class is now replaced by the new Robot class and each device is now defined in its own class. For example there is now a class for DistanceSensor, Servo, etc... These new device classes provide operations such as enable(), disable(), getValue() etc.

The new Robot class has now dedicated methods to get the various device instances. This replaces the previous Controller.robot_get_device() method and also the “device tags” that are now completely encapsulated in the device classes. Please have a look at Illustration 1.

Webots 5 Controller.reset() and Controller.die() methods are deprecated. The program structure of Webots 6 JAVA controllers looks now different:

Webots 5 (deprecated)
<pre>import com.cyberbotics.webots.Controller; public class MyController extends Controller { private int tag; private final int STEP = 64; public static void reset() { tag = get_device("ls"); } public static void die() { // your cleanup code here } public static void main(String[] args) { for (;;) { /* sense & actuate */ int v = light_sensor_get_value(tag); robot_step(STEP); } } }</pre>
Webots 6
<pre>import com.cyberbotics.webots.controller.Robot; import com.cyberbotics.webots.controller.LightSensor; public class MyRobot extends Robot { private LightSensor ls;</pre>

```

private final int STEP = 64;

public MyRobot() {
    super();
    ls = getLightSensor("ls");
}

public void run() {
    do {
        /* sense & actuate */
        double v = ls.getValue();
    } while (step(STEP) != -1);

    // your cleanup code here
}

public static void main(String[] args) {
    MyRobot robot = new MyRobot();
    robot.run();
}
}

```

Printing to Webots console

Webots 6 automatically redirects the standard output and error streams of the controllers to the console. So any method for printing to the standard output and error streams can now be used in controller code. This replaces the `robot_console_printf()` and `robot_console_println()` methods (Controller class) of Webots 5:

Webots 5 (deprecated)	Webots 6
<code>robot_console_print("Hello!");</code> <code>robot_console_println("Hello!");</code>	<code>System.out.print("Hello!");</code> <code>System.out.println("Hello!");</code> <code>System.out.printf("Hello!");</code> ...

C++ and Python APIs

Webots 6 Python and C++ APIs were completely redesigned. You will need to adapt your controller code when upgrading to Webots 6. Unlike the C and JAVA API, the backwards compatibility is not ensured. This is the simplified class diagram of Webots 6:

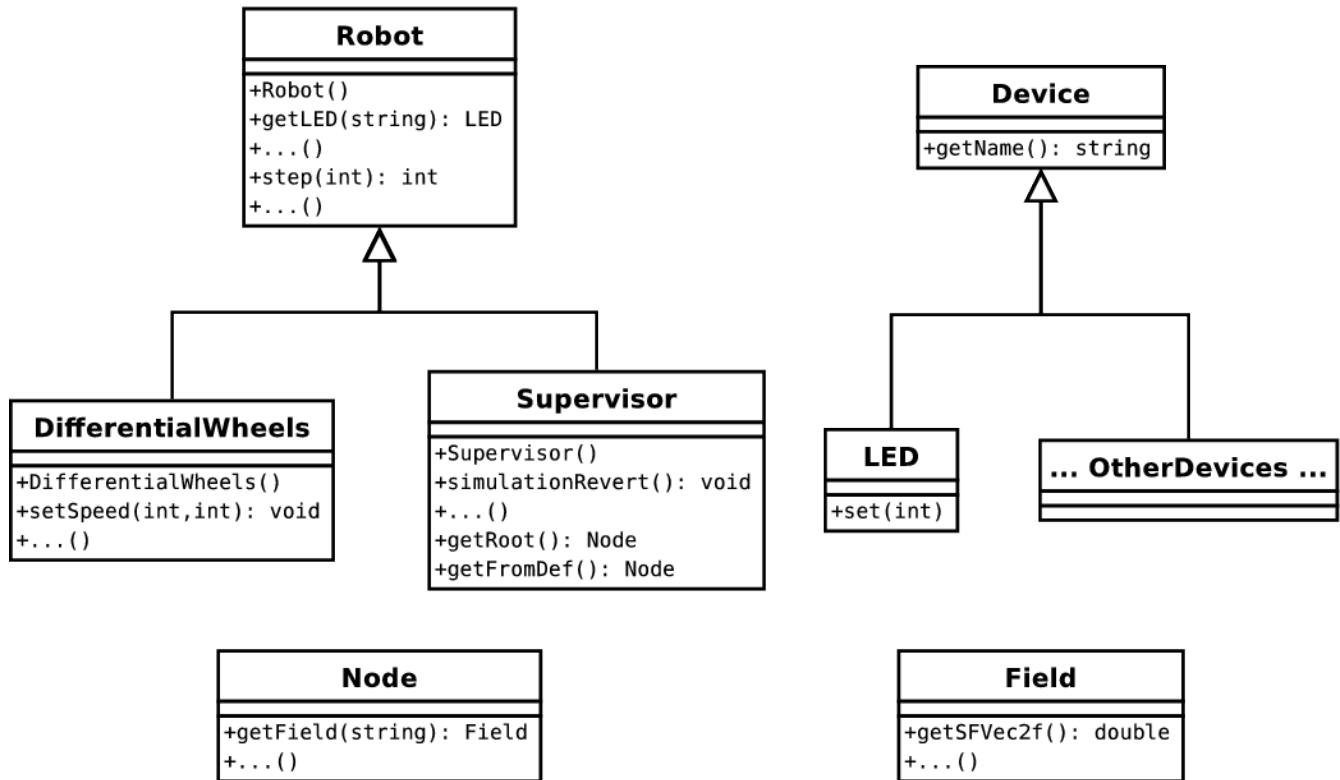


Illustration 1: Simplified class diagram for the C++, JAVA and Python APIs of Webots 6