LEGO NXT Robots using NXC – Part 1

Writing your first program

We are going begin by programming a robot to move forwards for 4 seconds, then backwards for another 4 seconds, and then stop.

Now type in the following program:

```
task main()
{
        OnFwd(OUT_A, 75);
        OnFwd(OUT_C, 75);
        Wait(4000);
        OnRev(OUT_AC, 75);
        Wait(4000);
        Off(OUT_AC);
}
```

Let us analyze the code:

Each program needs to have a task called `main` which is the one that will be executed by the robot. There are brackets around the statements such that it is clear that they all belong to this task. Each statement ends with a semicolon. In this way it is clear where a statement ends and where the next statement begins.

For example:

```
task main()
{
        statement1;
        statement2;
        …
}
```

Our program has six statements. Let us look at them one at the time:

`OnFwd(OUT_A, 75);` Tells the robot to start the motor connected to the output labeled A to move forwards. The speed of the motor to 75% of maximum speed.

`OnFwd(OUT_C, 75);` Tells the robot to start the motor connected to the output labeled C to move forwards. After these two statements, both motors are running, and the robot moves forwards.

`Wait(4000);` This statement tells us to wait for 4 seconds.

`OnRev(OUT_AC, 75);` We tell it to move in reverse direction backwards. Note that we can set both motors at once using `OUT_AC` as argument. We could also have combined the first two statements this way.

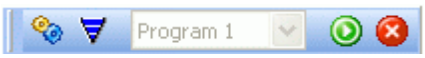
`Wait(4000);` Again we wait for 4 seconds.

`Off(OUT_AC);` Finally we switch both motors off.

That is the whole program. It moves both motors forwards for 4 seconds, then backwards for 4 seconds, and finally switches them off.

Running the program

Once you have written a program, it needs to be compiled and sent to the robot using USB cable.



Here you can see the button that allows you to (from left to right) compile, download, run and stop the program.

Changing the speed

To change the speed you just change the second parameter inside parentheses. The power is a number between 0 and 100. 100 is the fastest, 0 means stop. Here is a new version of our program in which the robot moves slowly:

```
task main()
{
	OnFwd(OUT_AC, 30);
	Wait(4000);
	OnRev(OUT_AC, 30);
	Wait(4000);
	Off(OUT_AC);
}
```

Summary

Each program has one task named `main` that is always executed by the robot. Also the four basic motor commands: `OnFwd()`, `OnRev()`, `Off()` and `Wait()`.

Making turns

You can make your robot turn by stopping or reversing the direction of one of the two motors. Here is an example. Type it in, download it to your robot and let it run. It should drive a bit and then make a 90-degree right turn.

```
task main()
{
	OnFwd(OUT_AC, 75);
	Wait(800);
	OnRev(OUT_C, 75);
	Wait(360);
	Off(OUT_AC);
}
```

You might have to try some slightly different numbers than 500 in the second `Wait()` command to make a 90 degree turn. This depends on the type of surface on which the robot runs. Rather than changing this in the program it is easier to use a name for this number. In NXC you can define constant values as shown in the following program.

```
#define MOVE_TIME 1000
#define TURN_TIME 360
task main()
{
	OnFwd(OUT_AC, 75);
	Wait(MOVE_TIME);
	OnRev(OUT_C, 75);
	Wait(TURN_TIME);
	Off(OUT_AC);
}
```

The first two lines define two constants. These can now be used throughout the program. Defining constants is good for two reasons: it makes the program more readable, and it is easier to change the values. Note that BricxCC gives the define statements its own color. As we will see in Chapter VI, you can also define things other than constants.

Repeating commands

Let us now try to write a program that makes the robot drive in a square. Going in a square means: driving forwards, turning 90 degrees, driving forwards again, turning 90 degrees, etc. We could repeat the above piece of code four times but this can be done a lot easier with the **`repeat`** statement.

```
#define MOVE_TIME 500
#define TURN_TIME 500
task main()
{
	repeat(4)
	{
		OnFwd(OUT_AC, 75);
		Wait(MOVE_TIME);
		OnRev(OUT_C, 75);
		Wait(TURN_TIME);
	}
	Off(OUT_AC);
}
```

The number inside the **repeat** statement's parentheses indicates how many times the code inside its brackets must be repeated. Note that, in the above program, we also indent the statements. This is not necessary, but it makes the program more readable.