

## Fundamentals

# Programming in ROBOTC ROBOTC Rules

In this lesson, you will learn the basic rules for writing ROBOTC programs.

### ROBOTC is a text-based programming language

Commands to the robot are first written as text on the screen. They are then processed by the ROBOTC compiler into a machine language file that the robot can understand. Finally, they are loaded onto the robot, where they can be run.

```

1 task main()
2 {
3
4     motor[port3] = 127;
5     wait1Msec(3000);
6
7 }
```

**Program Code**  
Commands to the robot written as text.

Text written as part of a program is called **code**. You type code just like you type normal text. Keep in mind that capitalization is important to the computer. Replacing a lowercase letter with a capital letter (or a capital letter with a lowercase letter) will cause the robot to become confused.

```

1 Task main()
2 {
3
4     motor[port3] = 127;
5     wait1Msec(3000);
6
7 }
```

**Capitalization**  
Capitalization (paying attention to UPPERCASE vs. lowercase) is important in ROBOTC.  
  
If you capitalize the 'T' in task, ROBOTC no longer recognizes this command.

As you type, ROBOTC will try to help you out by coloring the words it recognizes. If a word appears in a different color, it means ROBOTC recognizes it as an important word in the programming language.

```

1 task main()
2 {
3
4     motor[port3] = 127;
5     wait1Msec(3000);
6
7 }
```

**Code coloring**  
ROBOTC automatically colors key words that it recognizes.  
  
Compare this correctly-capitalized "task" command with the incorrectly-capitalized version in the previous example. The correct one is recognized as a command and turns blue.

## Fundamentals

### Programming in ROBOTC ROBOTC Rules (cont.)

Now, we will look at some of the important parts of the program code itself.

**Statements** are instructions for the robot. The most basic kind of statement in ROBOTC simply gives a command to the robot. The `motor[port3] = 127;` statement in the sample program you downloaded is a simple statement that gives a command. It instructs the motor plugged into Motor Port 3 to turn on at full power.

<pre> 1  task main() 2  { 3 4      motor[port3] = 127; 5      wait1Msec(3000); 6 7  }</pre>	<p><b>Simple statement</b> A straightforward command to the robot.</p> <p>This statement tells the robot to turn on the motor attached to motor port 3 at full power.</p>
<pre> 6 7  }</pre>	<p><b>Simple statement (2)</b> This is also a simple statement. It tells the robot to wait for 3000 milliseconds (3 seconds).</p>

Statements are run in order as quickly as the robot is able to reach them. Running this program on the robot turns the motor on, then waits for 3000 milliseconds (3 seconds) with the motor still running, and then ends.

<pre> 1  task main() 2  { 3 4      1st motor[port3] = 127; 5      2nd wait1Msec(3000); 6 7  } End</pre>	<p><b>Sequence</b> Statements run in English reading order (left-to-right, top-to-bottom). As soon as a command is complete, the next one runs. These two statements cause the motors to turn on (<i>1st command</i>). The robot then immediately begins a three second wait (<i>2nd command</i>) while the motors remain on.</p>
<pre> 7  } End</pre>	<p><b>End</b> When the program runs out of statements and reaches the } symbol in <code>task main</code>, all motors stop and the program ends.</p>

## Fundamentals

### Programming in ROBOTC ROBOTC Rules (cont.)

How did ROBOTC know that `motor[port3]= 127` and `wait1Msec[3000]` were two separate commands. Was it because they appeared on two different lines?

No. Spaces and line breaks in ROBOTC are only used to separate words from each other in multi-word commands. Spaces, tabs, and lines don't affect the way a program is interpreted by the machine.

```

1 task main()
2 {
3
4     motor[port3] = 127;
5     wait1Msec(3000);
6
7 }
```

#### Whitespace

Spaces, tabs, and line breaks are generally unimportant to ROBOTC and the robot.

They are sometimes needed to separate words in multi-word commands, but are otherwise ignored by the machine.

So why ARE they on separate lines? For the programmer. Programming languages are designed for humans and machines to communicate. Using spaces, tabs, and lines helps human programmers read the code more easily. Making good use of spacing in your program is a very good habit for your own sake.

```

1 task main() {motor[port3]
2 =127;wait1Msec(3000);}
```

#### No Whitespace

To ROBOTC, this program is the same as the last one. To the human programmer, however, this is close to gibberish.

Whitespace is used to make programs readable to humans.

But what about ROBOTC? How DID it know where one statement ended and the other began? It knew because of the semicolon (;) at the end of each line. Every statement ends with a semicolon. It's like the period at the end of a sentence.

```

1 task main()
2 {
3
4     motor[port3] = 127;
5     wait1Msec(3000);
6
7 }
```

#### Semicolons

Like periods in an English sentence, semicolons mark the end of every ROBOTC statement.

### Checkpoint

**Statements** are commands for the robot. Each statement ends in a **semicolon** so that ROBOTC can identify it. Each statement is also usually written on its own line to make it easier for humans to read. Statements are run in reading order, from left to right and top to bottom. Each statement is run as soon as the previous one is complete. When there are no more statements, the program ends.

## Fundamentals

### Programming in ROBOTC ROBOTC Rules (cont.)

ROBOTC uses far more punctuation than English. Punctuation in programming languages is generally used to separate important areas of code from each other. Most ROBOTC punctuation comes in pairs.

Punctuation pairs, like the parentheses and square brackets in these two statements, are used to mark off special areas of code. Every punctuation pair consists of an **opening** punctuation mark and a **closing** punctuation mark. The punctuation pair designates the area *between them* as having special meaning to the command that they are part of.

```

1 task main()
2 {
3
4     motor[port3] = 127;
5     wait1Msec(3000);
6
7 }
```

#### Punctuation pair: Square brackets [ ]

The code written between the square brackets of the `motor` command indicates which motor the command should use. In this case, it is the motor on port 3.

```

1 task main()
2 {
3
4     motor[port3] = 127;
5     wait1Msec(3000);
6
7 }
```

#### Punctuation pair: Parentheses ( )

The code written between the parentheses of the `wait1Msec` command tell it how many milliseconds to wait before starting a new command. In this case, it waits 3000 milliseconds, or three seconds.

### Checkpoint

**Paired punctuation marks** (such as square brackets and parentheses) are always used together. They surround specific important parts of a statement to set them apart.

Different commands make use of different kinds of paired punctuation. The `motor` command uses square brackets and the `wait1Msec` command uses parentheses. This is just the way the commands are set up. You will have to remember to use the right punctuation with the right commands or plan.

## Fundamentals

### Programming in ROBOTC ROBOTC Rules (cont.)

Simple statements do the work in ROBOTC, but control structures do the thinking. **Control structures** (or control statements) are pieces of code that control the flow of the program's commands, rather than issue direct orders to the robot.

Simple statements can only run one after another in order. However, control structures allow the program to choose the order in which statements are run. For instance, a control structure may tell the program to choose between two different groups of statements and only run one of them. Sometimes, control structures repeat a group of statements over and over again.

One important control structure is **task main**. Every ROBOTC program includes a special section called **task main**. This control structure determines which code the robot will run as part of the main program.

```

1 task main()
2 {
3
4     motor[port3] = 127;
5     wait1Msec(3000);
6
7 }
```

#### Control structure: task main

The control structure **task main** directs the program to the main body of the code. When you click the **Start** button in ROBOTC or turn on the robot, the program immediately goes to **task main** and runs the code it finds there.

The left and right curly braces { } belong to the **task main** structure. They surround the commands which will be run in the program.

```

while (SensorValue(touchSensor) == 0)
{
    motor[port3] = 127;
    motor[port2] = 127;
}
```

#### Control structure: while loop

The while loop repeats the code between its curly braces { } as long as certain conditions are met.

Normally, statements run only once. But with a **while** loop, they can be told to repeat over and over for as long as you want!

### Checkpoint

**Control structures** like **task main** determine which lines of code are run and specify when they are run. They control the order in which commands are executed in your program. Control structures enable your robot to make decisions and respond intelligently to its environment.

## Fundamentals

### Programming in ROBOTC ROBOTC Rules (cont.)

Programming languages are meant to be readable by both humans and machines. Sometimes, a programmer needs to leave a note for other human readers to help them understand what the code is doing. ROBOTC allows comments to be made for this purpose.

**Comments** are text that the program ignores. A comment can contain notes, messages, and symbols that may help a human, but would be meaningless to the robot. ROBOTC simply skips over them. Comments appear in green in ROBOTC.

```

1 // Motor port 3 forward with 100% power
2
3 task main()
4 {
5
6     /*
7      Port 3 forward with 100% power
8      Do this for 3 seconds
9     */
10
11     motor[port3] = 127;
12     wait1Msec(3000);
13
14 }
```

**Comments: // Single line**

Any section of text that follows a // (two forward slash characters) on a line is considered to be a comment. Any text to the left of the // is treated as normal code.

**Comments: /\* Any length \*/**

A comment can be created in ROBOTC using another type of paired punctuation, which starts with /\* and ends with \*/. This type of comment can span multiple lines, so be sure to include both the opening and closing marks!

#### End of Section

What you have just seen are some of the primary features of the ROBOTC language. **Code** is entered as text, which builds **statements**. Statements are used to issue commands to the robots. **Control structures** decide which statements to run at what times. **Punctuation**, both single like semicolons and paired like parentheses, is used to set apart important parts of commands.

A number of features in ROBOTC code are designed to help the human, rather than the robot. **Comments** let programmers leave notes for themselves and others. **Whitespace** like tabs and spaces helps to keep your code organized and readable.