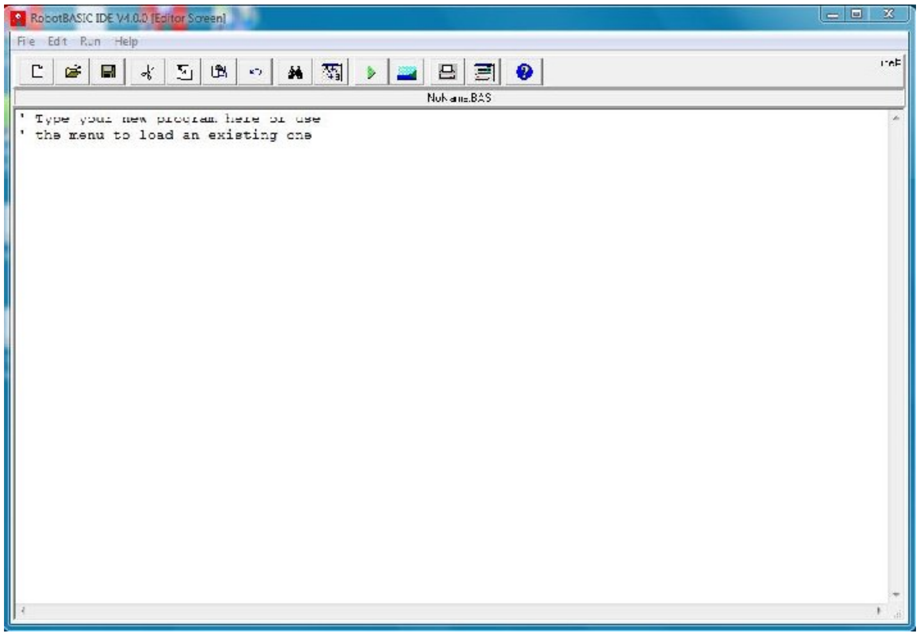
RobotBASIC Tutorial

This tutorial is designed to give a quick introduction to RobotBASIC in general, and the integrated simulator in particular. For more detailed information explore the HELP file (available from within RobotBASIC), or view our YouTube videos, or consider one of our books (full information and TOC are available on www.RobotBASIC.com).

When you first start RobotBASIC, read and accept the license, and you will see the following startup screen.



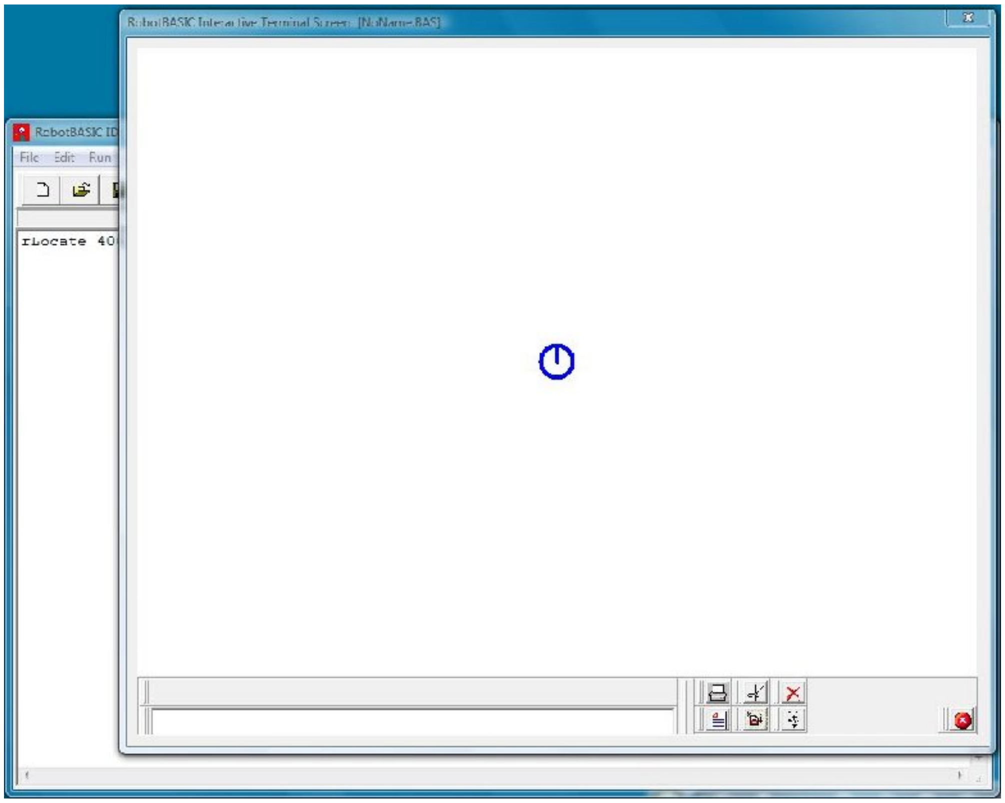
```
' Type your new program here or use
' the menu to load an existing one
```

The menu and tool bar across the top allows you to do many things, including SAVE and OPEN programs, get HELP, and RUN programs. The open space below the tool bar is where you write your program. Entering a program is not all that different from using a word processor.

We can create a simulated robot on the screen by "locating" it at some starting point. Since the output screen is 800 pixels wide by 600 pixels tall (slightly smaller on many NetBook computers) we can create the robot near the middle of the screen with the following command.

```
rLocate 400,300
```

All robot related commands in RobotBASIC start with an 'r'. Commands are NOT case-sensitive, so you do not need a capital 'L', but using it makes the command easier to read. Type the above line in the program area discussed above, and run the program using the menus, or by clicking on the green triangle in the tool bar. When you do, the OUTPUT or TERMINAL window will pop up over the programming screen as shown below.



```
rLocate 40
```

As you can see, a tiny robot has appeared in the center of the output screen. It looks simple, but this robot has MANY sensors (such as infrared perimeter detectors, an electronic compass, laser distance measurement, etc) that can be used just like the sensors on a REAL robot. Let's see how we can move this robot.

Type in the following program.

```
rLocate 400,300
rSpeed 10
rForward 100
rTurn 90
```

The `rSpeed` command slows the robot down a little so that it is easier for you to watch (in case you have a very fast computer). The larger the number given in `rSpeed`, the slower the robot will move.

When you run the program, the robot will move forward (in this case upward) 100 pixels, and then turn 90 degrees to the right. Negative numbers will cause the robot to move backwards or left.

If you add three more sets of `rForward` and `rTurn`, the robot will move in a complete square. Add the lines (as shown below) and then run the program.

```
rLocate 400,300
rSpeed 10
rForward 100
rTurn 90
rForward 100
rTurn 90
rForward 100
rTurn 90
rForward 100
rTurn 90
```

Notice how many lines it took to move the robot. One of the powerful things about programming, is that there are special statements called Flow-Control-Structures, that allow you to tell the computer to do the same things over and over. Let's see how this works.

One control structure is a counting-loop. It is implemented in RobotBASIC as a `FOR-NEXT` loop as shown below.

```
For a= 1 to 10
     // everything typed in here
     // will be repeated
Next
```

This structure starts with a `For` and ends with a `Next`. Everything inside the loop will be repeated a specified number of times. In this case, we will use a variable called **a** to keep track of the counting. The example above starts a at the number 1 and increments it each time through the loop, until it gets to 10. We could use any variable (**a, b, cat, Hello**, etc). Unlike commands, variable names ARE case sensitive, so the variable **Hello** is not the same as **hello**.

Notice that the lines inside the loop above start with two backslashes, which tells RobotBASIC to ignore everything past this point on a line. This is an easy way to place comments in a program to remind you of what this portion of the program is doing. Also notice that the items inside the loop are indented. This is not required, but it helps someone reading your program see the *block* of code that will be repeating.

In order to see that this example loop does in fact repeat its contents 10 times, and that the variable **a**, will have a value of 1 the first time through the loop, a value of 2 the second time through the loop, etc, lets place a `Print` command inside the loop as shown below.

```
For a= 1 to 10
     Print a
Next
```

If you run this program, you will see it print the numbers from 1 to 10, starting at the top of the screen. Change the program so it will only print from 1 to 5, or from 1 to 20, or perhaps from 10 to 15.

We can use the For loop to make our robot move around in a square more efficiently by using the program below.

```
rLocate 400,300
rSpeed 10
For a= 1 to 4
   rForward 100
   rTurn 90
Next
```

Can you change the program so it makes a bigger square? How do you think the robot's behavior would change if you changed the **90** to **-90** in the `rTurn` command? You must be careful when you move the robot. If it runs into a wall or any object that is drawn on the screen, then the program will cause an error and stop (the error will be reported in a pop-up window, and the line that caused the error will be highlighted when you close the pop-up).

The robot has a PEN that can be raised and lowered so that it can draw a line as it moves. As mentioned above, if something is drawn on the screen, the robot will normally think it is an object in the room, and if it bumps into the object, an error will occur. We do not want the robot to see the line it draws as an object, so we can tell the robot that the line is invisible as shown in the program below. By default, the robot will use the first invisible color to draw its lines.

```
rLocate 400,300
rInvisible RED
rSpeed 10
rPen DOWN
For a= 1 to 4
   rForward 100
   rTurn 90
Next
rPen UP
rForward 150
```

Run the program and watch the robot draw a square. Notice how the robot raises its PEN at the end and moves away from the drawing. Can you draw a triangle? You can use a loop if you wish, but depending on the type of triangle you wish to draw, a loop might not be the proper choice. (A loop would be useful only for drawing an equilateral triangle.)

Now that you have a little experience with RobotBASIC let's write a program that allows you to control the robot using the mouse. RobotBASIC has a `ReadMouse` command that can tell you where the mouse is on the screen, and what buttons are or are not pressed. The short program below shows and example of how to use the `ReadMouse` command.

```
while TRUE
  ReadMouse x,y,b
  if b<>0
     print x;y;b
     delay 500
  endif
wend
```

The above program introduces a new type of loop that begins with `WHILE` and ends with `WEND` (for while-end). A `while` loop repeats its contents WHILE something is true. In this case, we want it to repeat forever (until we close the output window) so we just use the word TRUE to indicate that it should loop continuously.

The `ReadMouse` command places the **x,y** coordinates of the mouse's current position into the first two variables (here called, **x** and **y**). The third variable in the command will be set to 0 if no mouse buttons are pressed, 1 if the left button is pressed, and 2 if the right button is pressed. Notice that the loop causes the mouse to be read over and over.

A new statement allows the program to decide if some statements are to be executed or not. The IF-decision structure starts with `IF` and ends with `ENDIF`. If the statement following the `IF` is TRUE, then all the lines inside the IF-structure will be executed. If the statement is FALSE, then all the statements inside the IF-structure will be skipped. This means that, when you run this program, the mouse data will be read into the three variables specified, but the information will ONLY be printed IF the variable **b** is NOT zero, meaning that the information will only be printed IF one of the mouse buttons has been pressed.

If you run this program and click one of mouse buttons, you will see the program print the location of the mouse (its **x,y** coordinates) and either a 1 or 2 depending on which mouse button you pressed (either LEFT or RIGHT). Try placing the mouse in different positions on the screen and verify that the numbers printed equate to the mouse's actual position. The upper left corner of the screen is 0,0 and the bottom right is 799,599. The `delay` command in the program pauses for 500 ms or ½ second. If you remove this delay, the program will print many times because it prints as long as you have the mouse button pressed. Remove the delay or comment it out with // and see the results.

Now that we have a good start, let's use the mouse to control our robot. The program below demonstrates many new principles. First, it draws six rectangles of various colors. They will have black borders (of width 3). The `rectangle` command requires that you give it the coordinates of the upper-left corner and the lower-right corner of the rectangle you wish to draw. These rectangle will serve as buttons that, when pressed, will control the movements of our robot.

This program also uses a special print command called `xyString`, which allows you to print ANYWHERE you want, by simply specifying the coordinates of where to start printing.

Once the rectangles are drawn the robot is located at the center of the screen and speed set to a reasonable value.

```
LineWidth 3
Rectangle 300,400,500,450,BLACK,GREEN
Rectangle 300,450,500,500,BLACK,Red
Rectangle 200,425,300,475,BLACK,BLUE
Rectangle 500,425,600,475,BLACK,BLUE
Rectangle 100,425,200,475,BLACK,LIGHTBLUE
Rectangle 600,425,700,475,BLACK,LIGHTBLUE
xyString 130,480,"Rotate     Turn"
xyString 540,480,"Turn      Rotate"
xyString 180,400,"LEFT"
xyString 575,400,"RIGHT"
xyString 370,375,"FORWARD"
xyString 365,510,"BACKWARD"
rLocate 400,300
rSpeed 10
while TRUE
  ReadMouse x,y,b
  if b=1
     ReadPixel x,y,c
     if c=GREEN
       rForward 1
     elseif c=RED
       rForward -1
     elseif c=BLUE and x<400
       rForward 1
       rTurn -1
     elseif c=BLUE and x>400
       rForward 1
       rTurn 1
     elseif c=LIGHTBLUE and x<400
       rTurn -1
     elseif c=LIGHTBLUE and x>400
       rTurn 1
     endif
  endif
wend
```

An endless WHILE loop holds the instructions to control the robot. After the mouse is read, a block of statements is executed, but ONLY if the left mouse button was pressed. If it is, a new command, `ReadPixel`, reads the color of the screen at the current mouse position (**x,y**) and places that color in the variable **c**.

A special form of the IF-structure allows us to check for multiple decisions using the `ELSEIF`. If the left mouse button is pressed over GREEN, for example, the robot will move forward, but if its over RED it will move backward.

When we check for BLUE and LIGHTBLUE, we have two possibilities, because we have two blue and two light blue rectangles. Therefore, we can let the IF also check the value of the variable x, to see if the mouse is on the left or right side of the screen (allowing us to easily determine which of the rectangles was chosen). If we want to rotate the robot we simply turn it, but if we want it to move in a gradual turn, the robot moves forward as well as turning.

This is a very simple program, and it can be improved in many ways, but it gives you a great introduction to a few of RobotBASIC's capabilities. The HELP file is nearly 300 pages of information on hundreds of commands and functions. Explore it (or one of our books) to learn more about programming and RobotBASIC.

Before we close, let's examine one more idea. If you are not careful, you can easily drive the robot into a wall or even into one of the control buttons on the screen, causing an error. A robot should be smart enough to not allow you to do that. The listing below shows ANOTHER form of the IF-structure that is used to test bumper switches around the robots perimeter (see the help file or our books for more information). In this example, the robot uses the value of these bumpers to decide if it is okay to move forward or backward. The values obtained from the `rBumper()` function are binary ANDed (using the & command) to check the value of individual sensors. After making these changes try to drive the robot into objects, especially using the TURN buttons and you will see how intelligently it does what you want it to do.

NOTE: Only the while loop is shown here, as there are no changes made to the first part of the program.

```
while TRUE
  ReadMouse x,y,b
  if b=1
     ReadPixel x,y,c
     if c=GREEN
       if NOT (rBumper()&14) then rForward 1
     elseif c=RED
       if rBumper()<>1 then rForward -1
     elseif c=BLUE and x<400
       if NOT (rBumper()&12) then rForward 1
       rTurn -1
     elseif c=BLUE and x>400
       if NOT (rBumper()&6) then rForward 1
       rTurn 1
     elseif c=LIGHTBLUE and x<400
       rTurn -1
     elseif c=LIGHTBLUE and x>400
       rTurn 1
     endif
  endif
wend
```

This is only a tiny sample of what you can do with RobotBASIC. Explore our help file, YouTube videos, and books to discover the power of simplicity.