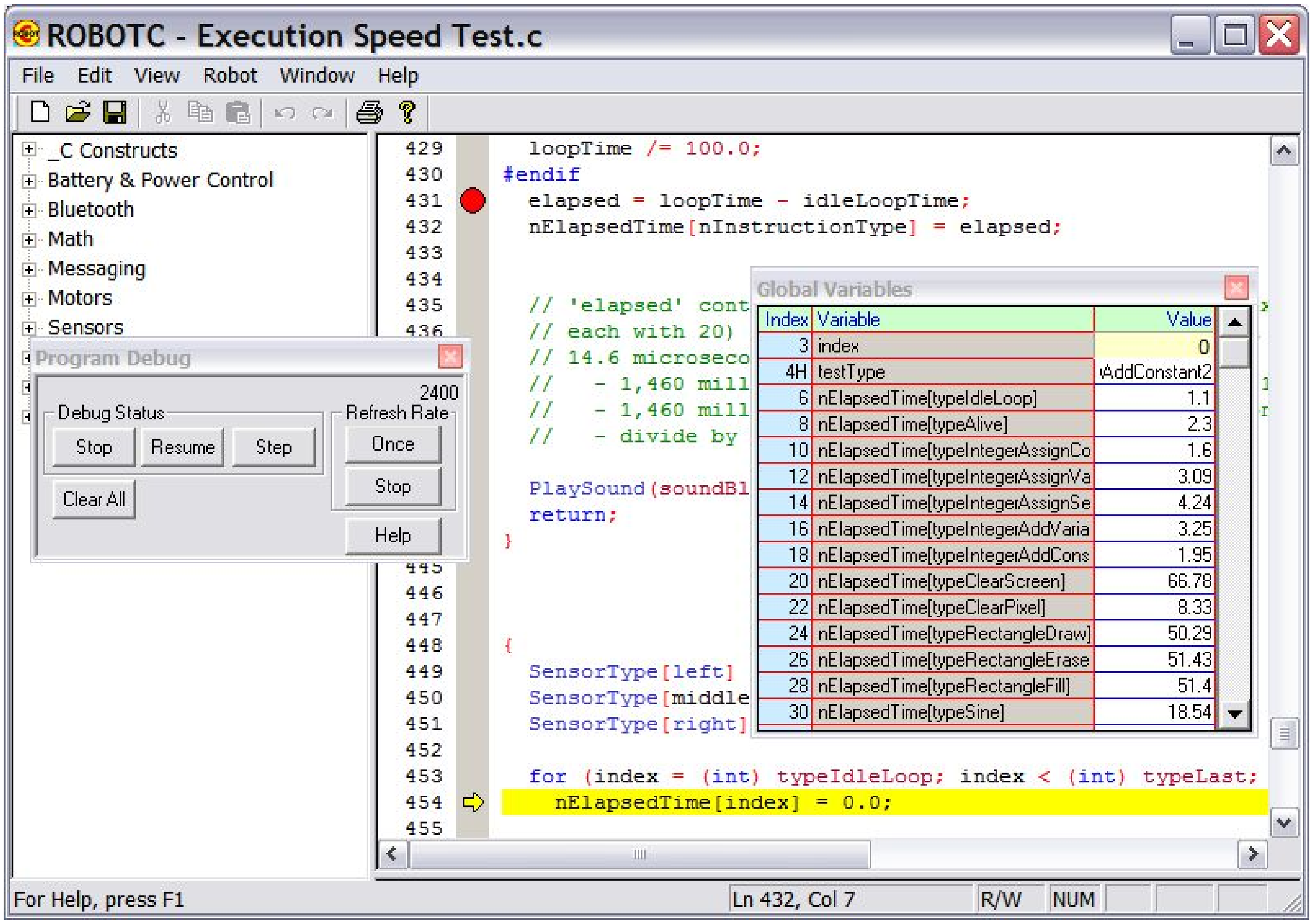
Program Debugging with ROBOTC

ROBOTC has a debugging capability that enables unparalleled interactive real-time access to the robot as your program is running. This has the potential to significantly reduce the time it takes to find and correct errors in your programs. With the debugger you can:

- Start and stop your program execution from the PC
- "Single step" through your program executing one line of code at a time and examining the results (i.e. the values of variables) the flow of execution.
- Define "breakpoints in your code that will cause program execution to be suspended when they are reached during program execution
- Read and write the values of all the variables defined in your program
- Read the write the "values" of motors and sensors.

The following picture shows the ROBOTC windows during an interactive debugging session.

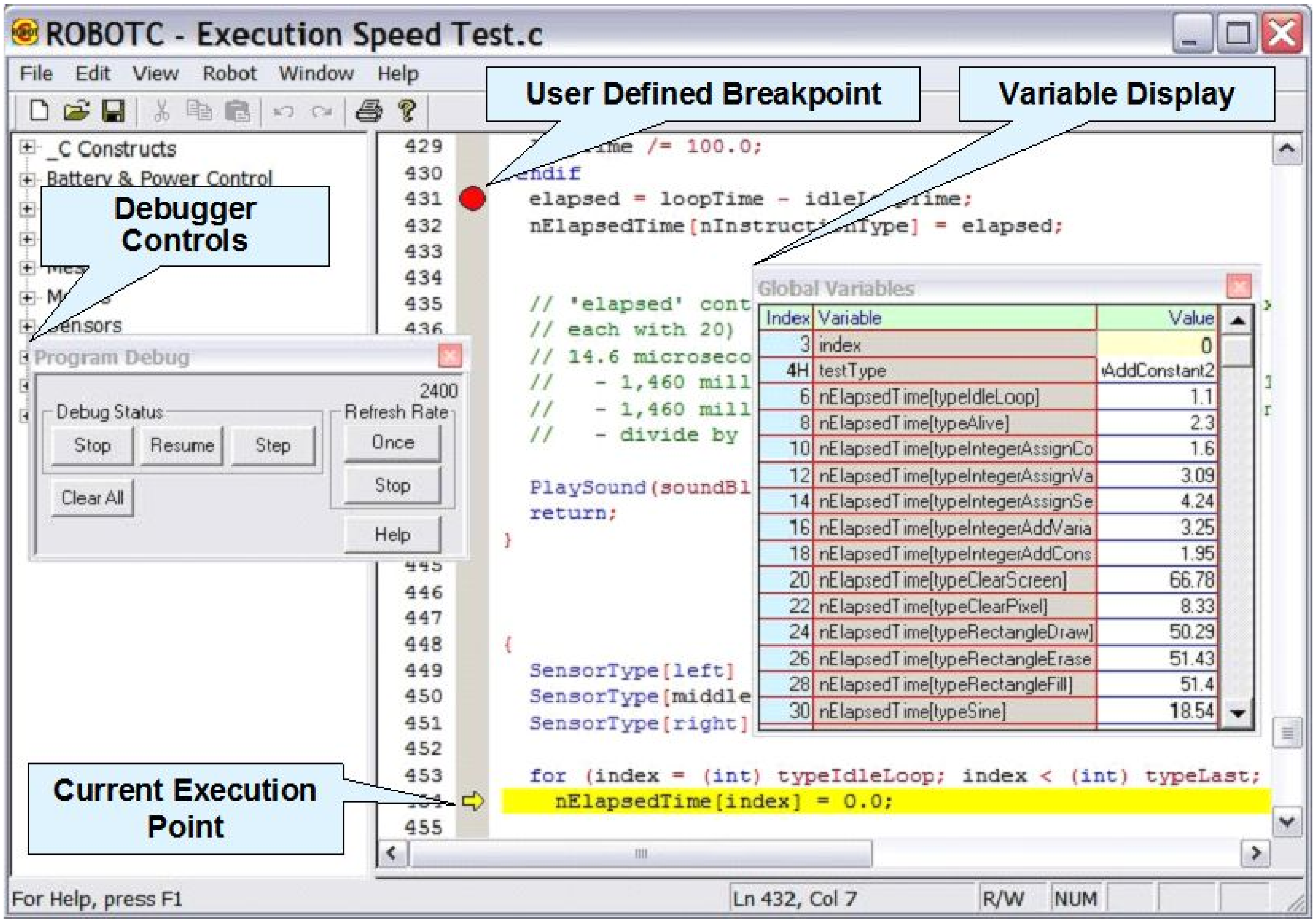


```
   loopTime /= 100.0;
#endif
   elapsed = loopTime - idleLoopTime;
   nElapsedTime[nInstructionType] = elapsed;


   // 'elapsed' cont
   // each with 20)
   // 14.6 microseco
   //    - 1,460 mill
   //    - 1,460 mill
   //    - divide by

   PlaySound(soundBl
   return;
}



{
   SensorType[left]
   SensorType[middle
   SensorType[right]

   for (index = (int) typeIdleLoop; index < (int) typeLast;
      nElapsedTime[index] = 0.0;
```

Index	Variable	Value
3	index	0
4H	testType	AddConstant2
6	nElapsedTime[typeIdleLoop]	1.1
8	nElapsedTime[typeAlive]	2.3
10	nElapsedTime[typeIntegerAssignCo	1.6
12	nElapsedTime[typeIntegerAssignVa	3.09
14	nElapsedTime[typeIntegerAssignSe	4.24
16	nElapsedTime[typeIntegerAddVaria	3.25
18	nElapsedTime[typeIntegerAddCons	1.95
20	nElapsedTime[typeClearScreen]	66.78
22	nElapsedTime[typeClearPixel]	8.33
24	nElapsedTime[typeRectangleDraw]	50.29
26	nElapsedTime[typeRectangleErase	51.43
28	nElapsedTime[typeRectangleFill]	51.4
30	nElapsedTime[typeSine]	18.54

Note: To minimize the size of the picture, the two debugger windows in the above picture were positioned on top of the main ROBOTC window. This is not a requirement. They can be positioned anywhere on the PC desktop.

Four debugger capabilities are shown in the above picture. Each of these is highlighted in the following picture.

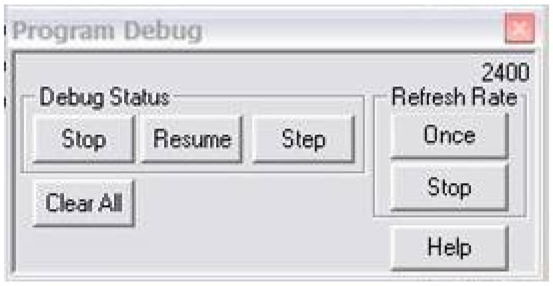


```
        me /= 100.0;
#endif
    elapsed = loopTime - idleLoopTime;
    nElapsedTime[nInstructionType] = elapsed;


    // 'elapsed' cont
    // each with 20)
    // 14.6 microseco
    //    - 1,460 mill
    //    - 1,460 mill
    //    - divide by

    PlaySound(soundBl
    return;
}



{
    SensorType[left]
    SensorType[middle
    SensorType[right]

    for (index = (int) typeIdleLoop; index < (int) typeLast;
      nElapsedTime[index] = 0.0;
```

Index	Variable	Value
3	index	0
4H	testType	AddConstant2
6	nElapsedTime[typeIdleLoop]	1.1
8	nElapsedTime[typeAlive]	2.3
10	nElapsedTime[typeIntegerAssignCo	1.6
12	nElapsedTime[typeIntegerAssignVa	3.09
14	nElapsedTime[typeIntegerAssignSe	4.24
16	nElapsedTime[typeIntegerAddVaria	3.25
18	nElapsedTime[typeIntegerAddCons	1.95
20	nElapsedTime[typeClearScreen]	66.78
22	nElapsedTime[typeClearPixel]	8.33
24	nElapsedTime[typeRectangleDraw]	50.29
26	nElapsedTime[typeRectangleErase	51.43
28	nElapsedTime[typeRectangleFill]	51.4
30	nElapsedTime[typeSine]	18.54

Item	Description
Debugger Controls	This is the main debugger window for controlling user program execution
Current Execution Point	The yellow left arrow indicates the next line of code to be executed in the program. Yellow is used to indicate that program execution is currently suspended. Green is used when the program is "free running".
User Defined Breakpoint	The red "stop sign" indicates a user defined break point. Program execution will be suspended if this line of code is reached during program execution. Breakpoints are easily defined by left-clicking in the "gray" column corresponding to the desired breakpoint.
Variable Display	This is a display of all the variables that are defined in your program. The values are periodically retrieved from your robot and displayed here. If desired you can select a variable and rewrite its value.

Debugger Controls



This is the main debugger window for controlling user program execution. The buttons on this window allow you to start/stopm suspend/resume and program execution. Other buttons allow you to control the rate at which information from your robot is refreshed.

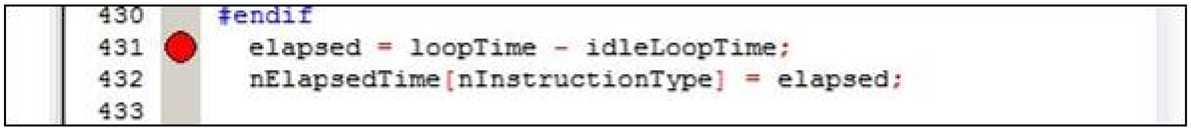
The names of the buttons change based on the state of your program execution. For example "Start" is shown if no program is running and "Stop" is displayed when a program is executing.

Each of the buttons is described in the following table.

Button	Action Performed
Start / Stop	Starts or stops the execution of a program
Suspend / Resume	Temporarily suspends (or resumes) the execution of your program.
Step	Causes a suspended program to execute the next line of code in your program. **Note:** Some "complicated" lines of code, especially code lines that cause a break in sequential program execution like a "for" or "while" clause may only be partially executed by a single "Step" command.
Once	Will trigger a single refresh of the debugger windows. It also stops continuous refresh of the debugger windows.
Continuous / Stop	Starts or stops continuous refresh of the debugger windows.

User Defined Breakpoints

"Break points" are locations in your source code where program execution can be interrupted for manual intervention.



```
#endif
  elapsed = loopTime - idleLoopTime;
  nElapsedTime[nInstructionType] = elapsed;
```

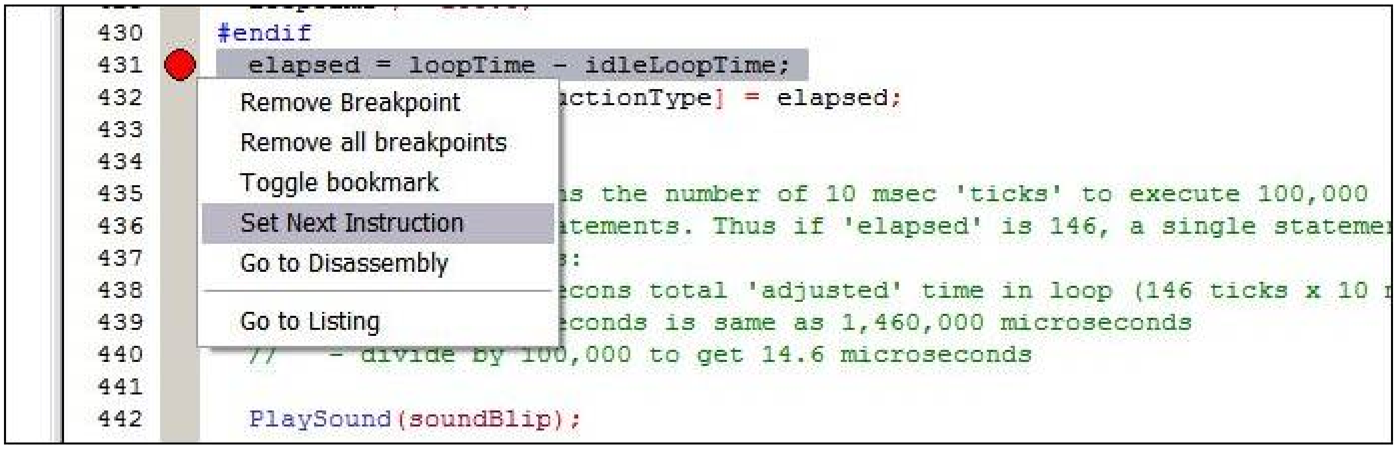
The red "stop sign" indicates a user defined break point. Program execution will be suspended if this line of code is reached during program execution. Breakpoints are easily defined by left-clicking in the "gray" column on the source code line corresponding to the desired breakpoint.

ROBOTC breakpoints do not impact the speed with which your program executes. The breakpoint capability is built-into the ROBOTC firmware and does not require insertion of extra instructions in your compiled code.

You can define (virtually) unlimited breakpoints within your program. There are no restrictions on the number of breakpoints within a single function or task.

Changing Flow of Program Execution

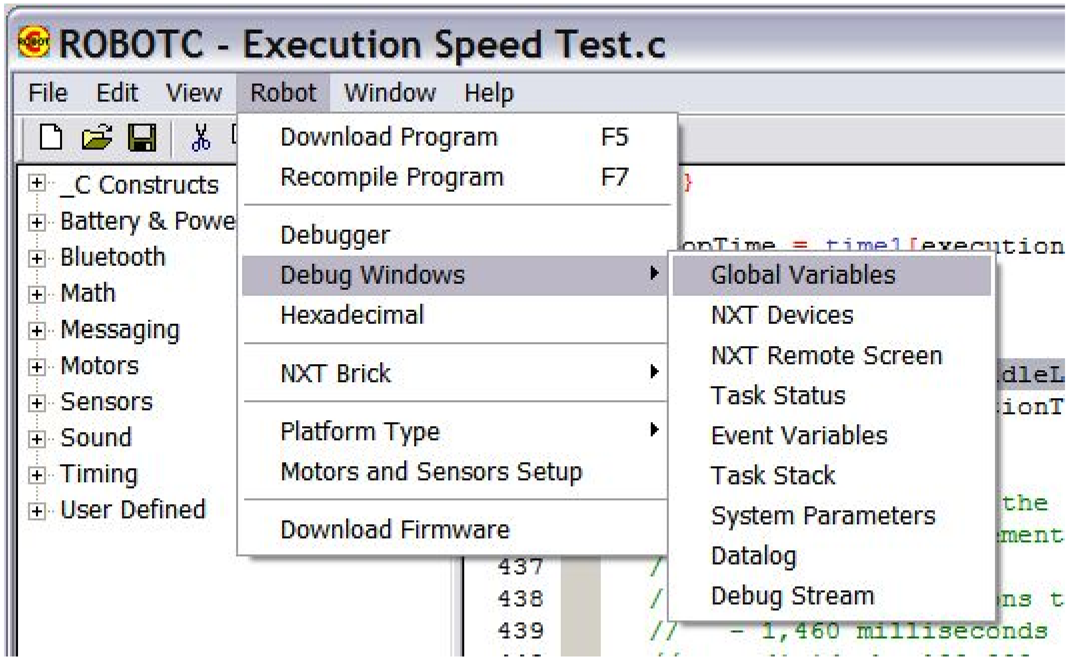
Sometime during debugging you may want to alter the normal flow of program execution. If you right-click instead of left-clicking with the mouse a pop-up menu appears. One of the commands is "Set Next Instruction" which allows you to change the currently executing point of your program to this selected line! The pop-up menu is shown in the picture below.



```
#endif
 elapsed = loopTime - idleLoopTime;
                                 ctionType] = elapsed;

                                 s the number of 10 msec 'ticks' to execute 100,000
                                 tements. Thus if 'elapsed' is 146, a single stateme
                                 :
                                 econs total 'adjusted' time in loop (146 ticks x 10
                                 conds is same as 1,460,000 microseconds
 //    - divide by 100,000 to get 14.6 microseconds

 PlaySound(soundBlip);
```

Debugger Displays

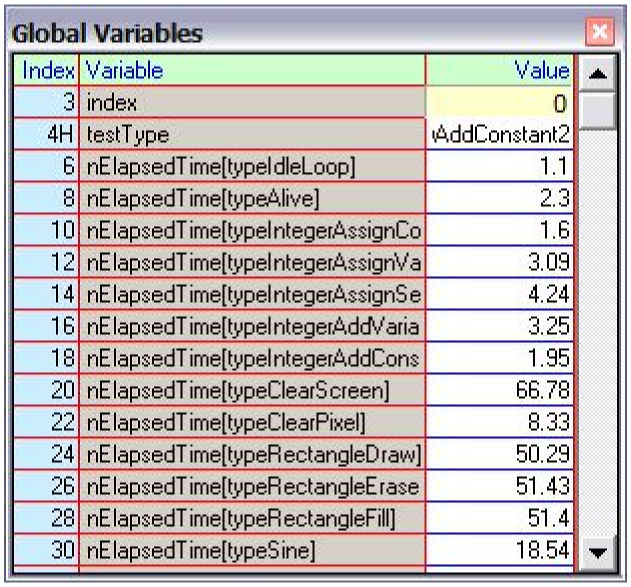
There are several different debugger windows that can be used when the ROBOTC debugger is operational. These windows give access to user program and built-in variables. Some of the windows are infrequently used but can be useful when needed. Debugging windows can be opened from the sub-menu shown in the following picture.



This picture shows the Debugger Windows available for the NXT platform in ROBOTC. Other robot platforms may have a different list of windows customized to the robot controller hardware.

The number of available windows is also impacted by the “User Level” setting found on the “Window” sub-menu. In the “basic” mode, some of the more advanced and less-frequently used windows are not available.

Several of these windows are shown below.

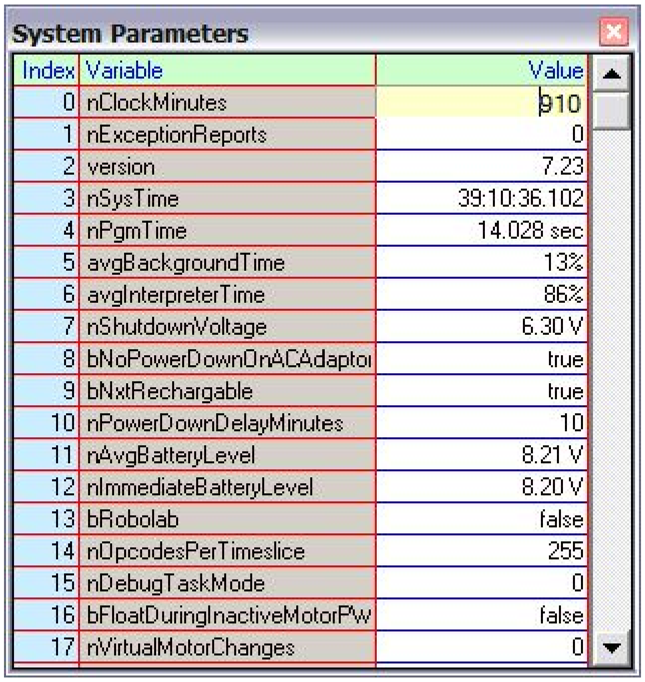


Global Variables

Index	Variable	Value
3	index	0
4H	testType	AddConstant2
6	nElapsedTime[typeIdleLoop]	1.1
8	nElapsedTime[typeAlive]	2.3
10	nElapsedTime[typeIntegerAssignCo	1.6
12	nElapsedTime[typeIntegerAssignVa	3.09
14	nElapsedTime[typeIntegerAssignSe	4.24
16	nElapsedTime[typeIntegerAddVaria	3.25
18	nElapsedTime[typeIntegerAddCons	1.95
20	nElapsedTime[typeClearScreen]	66.78
22	nElapsedTime[typeClearPixel]	8.33
24	nElapsedTime[typeRectangleDraw]	50.29
26	nElapsedTime[typeRectangleErase	51.43
28	nElapsedTime[typeRectangleFill]	51.4
30	nElapsedTime[typeSine]	18.54

This window contains a table of all the variables declared in your program along with their current values.

If needed, you can select and change the value of any variable.



System Parameters

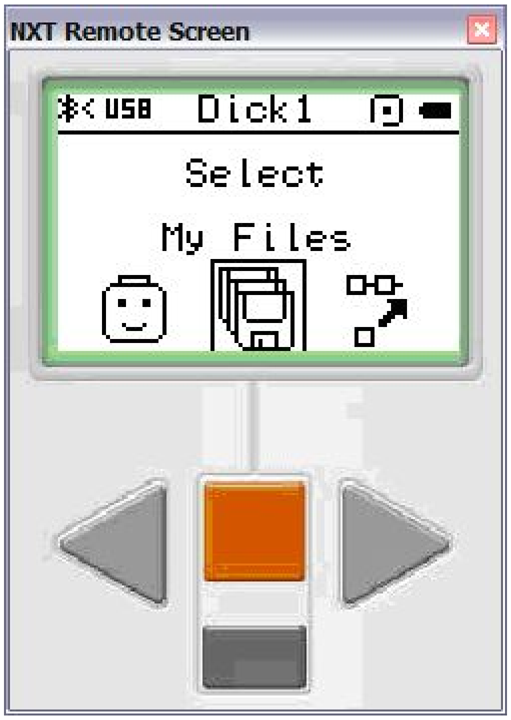
Index	Variable	Value
0	nClockMinutes	910
1	nExceptionReports	0
2	version	7.23
3	nSysTime	39:10:36.102
4	nPgmTime	14.028 sec
5	avgBackgroundTime	13%
6	avgInterpreterTime	86%
7	nShutdownVoltage	6.30 V
8	bNoPowerDownOnACAdaptor	true
9	bNxtRechargable	true
10	nPowerDownDelayMinutes	10
11	nAvgBatteryLevel	8.21 V
12	nImmediateBatteryLevel	8.20 V
13	bRobolab	false
14	nOpcodesPerTimeslice	255
15	nDebugTaskMode	0
16	bFloatDuringInactiveMotorPW	false
17	nVirtualMotorChanges	0

ROBOTC has several built-in variables that customize the performance of your robot. The “System Parameters” window provides PC access to these variables. This example is for the NXT robot controller.

Of course, all of these variables are directly accessible within your program code.

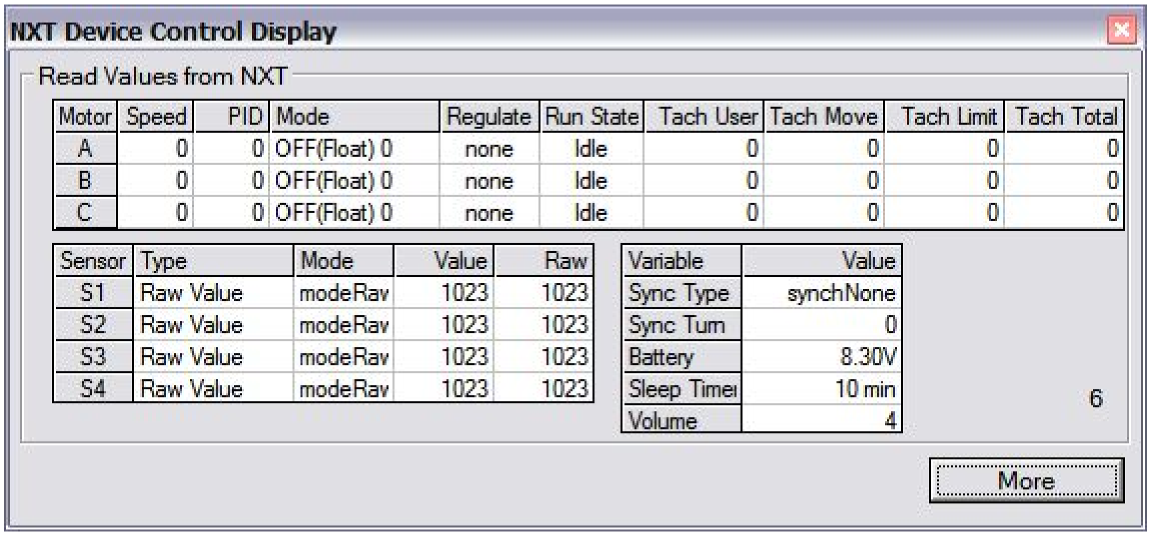
Although not usually needed, the advanced user may find a few of these fields particularly interesting. For example:

- ‘avgBackgroundTime’ is the overhead time spent in the device driver code. The remaining CPU time is available for user program execution. This example shows background overhead at 13%.
- ‘bNoPowerDownOnACAdaptor’ is a neat variable to prevent the NXT from automatically powering down if a rechargeable battery is being used and it is connected to AC power.



This debugging window is only available on the NXT robot platform.

The remote screen allows you to control your NXT from your PC. It provides an image of the NXT LCD screen. A mouse press over one of the four buttons will simulate a key press; or you can use one of the four arrow keys on your PC keyboard.



Read Values from NXT

Motor	Speed	PID	Mode	Regulate	Run State	Tach User	Tach Move	Tach Limit	Tach Total
A	0	0	OFF(Float) 0	none	Idle	0	0	0	0
B	0	0	OFF(Float) 0	none	Idle	0	0	0	0
C	0	0	OFF(Float) 0	none	Idle	0	0	0	0

Sensor	Type	Mode	Value	Raw
S1	Raw Value	modeRav	1023	1023
S2	Raw Value	modeRav	1023	1023
S3	Raw Value	modeRav	1023	1023
S4	Raw Value	modeRav	1023	1023

Variable	Value
Sync Type	synchNone
Sync Turn	0
Battery	8.30V
Sleep Timer	10 min
Volume	4

This is the "devices" debugging window for the NXT robot controller platform. Different versions of the screen are available for other platforms supported by ROBOTC.

The screen provides access to current values for the motors and sensors on the NXT.

The "More" button expands the window to provide controls that enable you to "write" to the sensors and motors to setup their initial configuration.

There are no restrictions on the number of debugger windows that can be simultaneously opened. However, each window does require the PC to "message" with the robot controller to refresh its data. The more windows that are open the more data transmission required for the refreshing activity.

ROBOTC has been optimized to minimize refresh times. Typically only one or two messages are required to refresh each window; this is valuable on the robot controllers that have a "slow" communications channel between the robot and the PC.

For example, Bluetooth messaging on the NXT platform is slow taking about 25 milliseconds per message. Out of the box, NXT Bluetooth messages are restricted to 58 bytes of data and 13 messages are required to refresh the 800 byte NXT LCD image. The ROBOTC enhanced firmware performs only requires a single message.

“Traditional” Debugging Techniques

Debugging a program – finding the errors and correcting them – can be a slow process in solutions without a run-time debugger. Without a debugger you typically resort to techniques like:

- There’s no way to determine if your program is executing the intended logic. So you add code to play different tones/sounds to your program as it executes different “blocks” of code. You determine from the sound what is being executed within your program.
- If your robot platform supports a display device (which could be a serial link to your PC) then you add “print” statements to your program code to tell you about your program execution. By examining the display, you can (hopefully) determine what’s happened without your program execution.

Both of the above techniques are available in ROBOTC. However, a real-time debugger eliminates the need to resort to them. There’s no need to add code for debugging to your program. A debugger provides better functionality without modifying your source code!