

# MMDebug

## 10 Minute Tutorial

This 10 minute tutorial will walk you through the usage and features of MMDebug. At the end you will have a good understanding of the debugger and how it operates and its features.

- 1) Load the appropriate PicoMite v6.01.00D firmware onto your PicoMite. The firmware in this release builds on the PicoMite v6.01.00b18 firmware. The PicoMite v6.01.00D firmware is 100% compatible with PicoMite v6.01.00 version of the firmware.
- 2) Presumably, you have already gone through the MMDebug Quick Start Guide and have a VT100 compatible terminal connected to your PicoMite.
- 3) Load the **MMDebug\_Tutorial.bas** program included in this release to your PicoMite.
- 4) At the command prompt, type: **> Run** It is probably helpful to see what the program does before watching its execution from inside the debugger. The program contains high resolution functions to calculate Pi, Sin() and Cos(). The expansion series to calculate Sin() and Cos() use a factorial function. The nesting and recursion of the functions will help demonstrate the debugger's capability. The program runs for 30 seconds and prints out the square roots of the numbers 1 to 10 (3 times).
- 5) -----
- 6) Now, at the command prompt, type: **> Run ctrl-D** This will launch the program and tell MMDebug to take control.
- 7) You will see the MMDebug window open up in the upper right corner of the screen. The debugger will skip over blank lines and comments automatically. You will see line # 6 highlighted in Yellow. The highlighted Yellow line is the next program logic that is ready to execute.
- 8) Click 'Step Over'. The array we need to demonstrate other things later will be created and the debugger will highlighted line # 9. That is the next line of logic that is ready to execute. This is a function call to generate a high precision version of Pi and assign it to a variable for use elsewhere in the program.
- 9) Click 'Step Over' in the lower left of the debug window. mypi() will be called and the return value assigned to the variable hires\_Pi. You will see the value of the just assigned hires\_Pi variable displayed in the upper right 'Variable Watch Window'. This is where all recently used or soon to be used variables are displayed. The color of the variable's value is Red because it has changed values from what it used to be.
- 10) The Yellow highlighted line has moved to line # 11, which is the main 'For' loop of the program. Click on 'Step Over' again. This will execute the entire 'For a=1 to 10' loop without stopping. The embedded logic in most loops, subroutines and functions can be executed without bogging down the debug session by using 'Step Over' on the loop, subroutine or function elements.

- 11) Notice that the Yellow highlighted line has moved to line # 21 after the 'Next a' statement. Also notice the contents of the upper right 'Variable Watch Window'. The hires\_Pi variable in is now Green. And the A! variable has been added to the list and is Red.
- 12) We skipped over all the logic inside of the 'For a=1 to 10' loop. There are a number of variables that got used and assigned inside that loop that we might want to see their values. This can be easily accomplished by clicking on the logic line with a variable of interest. For example, click on the 'For b=1 to a' statement on Line # 11. B! will be added to the top of the watch window.
- 13) Click on tmp\_value on line #14. It will be added to the watch window. You will notice the value is underscored. Underscores in the watch window mean you can't see the entire value. This applies to both the value and name of the variable. You can shift the size of the watch window with the Left and Right arrow keys. If you need to see a little more precision of the tmp\_value variable, press the Left arrow key several times. With several Left arrow key presses the underscores will disappear and you will be able to see the full precision of the tmp\_value variable.
- 14) Slowly click on the 'Step' button 5 times. You will see the Yellow highlighted line move across the logic of the If statement on line # 23. You will see each variable being added to the watch window. The watch window is limited to 20 variables. There are times where some of the variables are not important and just clutter the display.
- 15) Click on 'ROOT!' in the watch window. It will be removed. Click on 'TMP\_VALUE!' in the watch window. It will be removed.
- 16) Instead of clicking on the variable's name, this time click on the variable 'A' value of 11. Clicking on a variable's value in the watch window will allow the program to run until that variable's value changes. The program was running freely until the 'For a=1 to 10' logic was executed. 'A' changed value and the debugger took control back at Line #12 'For b=1 to a'
- 17) Click on the 'Quit' button at the bottom of the debugger window. This is to make it so you don't need to follow the tutorial perfectly. If you click in the wrong location it is easy to lose control. We will restart the program (and debugger) several times to demonstrate all the features.

**18)** -----

- 19) At the command prompt, type: **> Run ctrl-D**
- 20) You can tell the program to run and break point at a selected line of logic. Click on the White number 16 on line #16. Now click on the White number 19 on Line # 19. The Yellow highlighted line will be 'Next a'.
- 21) If you were to click on 'Step' program control will be given back to you at Line # 12 at the top of the 'For a=1 to 10' loop. Instead, click on 'Step Over'. The entire remaining loop will continue until complete.
- 22) We no longer need the current contents of the 'watch window'. Click on the line just under the last element of the watch window. The last element of the window is B!. Click on the blank line just under B!. The entire watch window will be discarded.

- 23) Let's learn about functions and subroutines. Click on the White number 14 at Line # 14 of the display. The debugger will have highlighted 'tmp\_value = mysin(a)'.
- 24) Click on 'Step' to enter the mysin() function.
- 25) Click on the White number 77 on Line #77. The program logic is going to call the recursive Factorial() function with an argument of 5. Notice in the bottom right of the debugger window a notice that the program is currently nested Level:1 deep currently in functions and subroutines.
- 26) On your keyboard type 2 9 Enter This will display the program starting at Line # 29. We want to wind up the subroutine / function call stack to demonstrate the 'Step Out' buttons use. Click on the White number 31 on Line # 31 three times. You should see the status bar display Level:4. The program is nested 4 levels deep in function calls.
- 27) Click on the 'Step Out' button four times. The program logic will return to the main program where it is calling mysin() to assign a value to tmp\_value.
- 28) Press the > key 5 times. Press the . key 4 times. This will shrink the MMDebug window to give you more screen space. Pressing < or , will increase the window size.
- 29) Press the Red 'Quit' button at the bottom of the debug window
- 30)** -----
- 31) At the command prompt, type: > **Run ctrl-D** Press the , key to expand the window enough to see Line # 28. Click on the White 23 on Line # 23.
- 32) The Variable Watch Window can not display array elements. They can be displayed by clicking the 'Edit Value' button. Do that now. From the prompt inside the Edit Value box, type ? sqr\_lookup(2)
- 33) You can also set a variable's value. CNT! Is currently displayed at the top of the Variable Watch Window. From the prompt inside the Edit Value box, type > **cnt=10** You will see the value of CNT! Change in the Variable Watch Window.
- 34) You can also call subroutines and functions from inside the Edit Value box if necessary.
- 35) Three Enter keys in a row will exit the 'Edit Value' box and return you to the debugger. Or, a single Enter key at a blank command prompt within the 'Edit Value' box will return you to the debugger.
- 36) Press the Red 'Quit' button at the bottom of the debug window
- 37) One last thing to point out. Just as there is a 'Break' key for MMBasic, there is also a 'Debug' key. It defaults to ctrl-D. But it can be changed just like the 'Break' key is changed. At the command prompt, you can type > **Option Debug 5** to switch it (in this case ctrl-E). Your program can query the current value by calling MM.Info(Option Debug). At the command prompt you can type
- > ? **MM.Info(Option Debug)** to see its current value. It can be disabled by setting it to 0.