How to use FBdebugger

1. Create a debug build of your code

The debugger needs additional information in the executable to be able to display corresponding source code (position) to the currently executed program code (position) and for some other useful things like variable inspection.

To enable inclusion of debug information in the executable rebuild your executable with the compiler parameter “-g”.

If you are using a IDE ensure that “-g” is contained in the used compiler command line (see respective documentation for your IDE).

e.g. if you are normally building with “fbc -exx myprogram.bas” change it to:

```
fbc -g -exx myprogram.bas
```

Ensure that you (re)compile/(re)build your executable once you changed the command line in case you are using an IDE.

2. Launch your executable using FBdebugger

First make sure to choose the correct version of FBdebugger: 32-bit for 32-bit executables and 64-bit for 64-bit executables. Do not mix 32-bit and 64-bit.

To find out whether you are compiling a 32-bit or 64-bit executable there are multiple options. An easy one is to add “**-v**” to the compiler command line, which will generate a detailed compiler output similar to this (assembling/linking details stripped):

```
C:\Program Files (x86)\FreeBASIC\fbc -v "test.bas"
FreeBASIC Compiler - Version 1.02.1 (04-25-2015), built for win32 (32bit)
Copyright (C) 2004-2015 The FreeBASIC development team.
standalone
target:       win32, 486, 32bit
compiling:    test.bas -o test.asm (main module)
assembling:   […]
linking:      […]

Make done
```

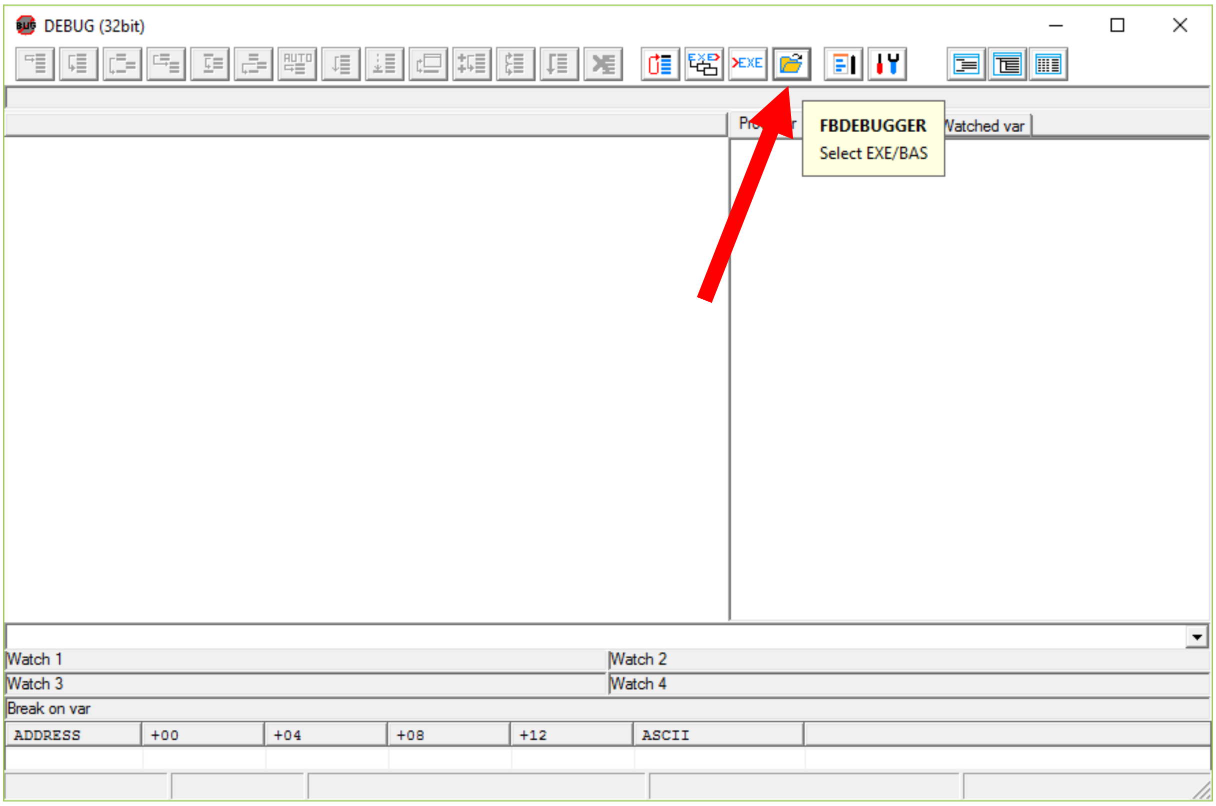
Note the information displayed as “target”, which tells you exactly what you need.

Now you can launch the correct version of FBdebugger. There exist multiple ways of debugging your application:

- Launch your application from the debugger
- Attach the debugger to your (already running) application

We will use the first option. The other option could be used e.g. to debug a hanging application.

To launch your executable in FBdebugger open it via the “Select EXE/BAS” button in the toolbar (see picture). Alternatively, you can drag&drop your executable into the debugger window or onto the FBdebugger executable or pass it as first command line argument.

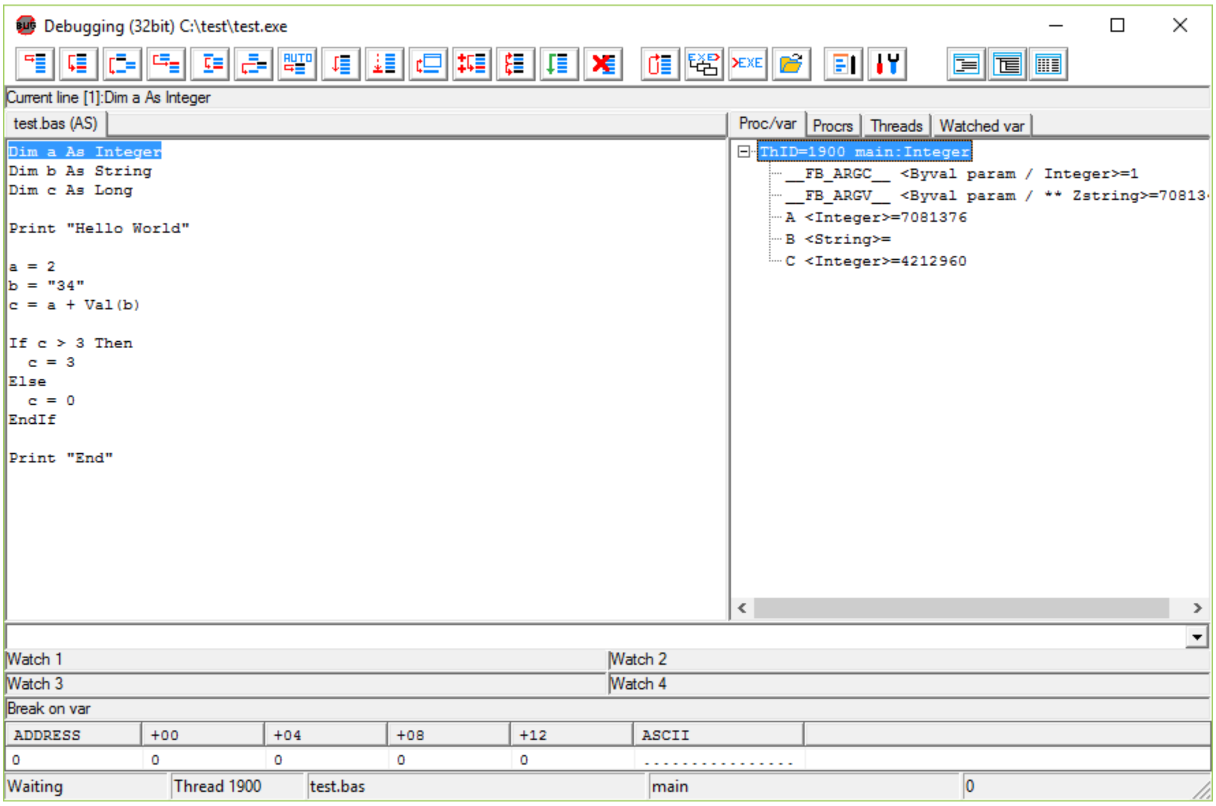


The debugger will launch your executable, but immediately halt it before the first executable line is executed.

On the left side of the window you can see your source code with the current execution position (underlined).

On the right side of the window you can get information about the running program. The following Tabs are available:

- “Proc/var” shows the variables in the current scope and its datatype and values
- “Procs” lists procedures (subs/functions) in your program
- “Threads” shows information about the different running threads and their current execution position
- “Watched Vars” allows you define special variables, whose value you want to track



```
Dim a As Integer
Dim b As String
Dim c As Long

Print "Hello World"

a = 2
b = "34"
c = a + Val(b)

If c > 3 Then
  c = 3
Else
  c = 0
EndIf

Print "End"
```

```
ThID=1900 main:Integer
   __FB_ARGC__ <Byval param / Integer>=1
   __FB_ARGV__ <Byval param / ** Zstring>=70813
   A <Integer>=7081376
   B <String>=
   C <Integer>=4212960
```

```
ADDRESS  +00  +04  +08  +12  ASCII
0        0    0    0    0    ................
```

3. Step through your program

To understand what your program does (wrong) you execute it in small steps and analyze its state (e.g. variable values) from step to step. FBdebugger provides several commands to perform different kinds of steps in your program:

Single step: execute the current line and halt at the next executable line. It steps into subs/functions.

Run to cursor: place the cursor somewhere in the source code in the debugger and press this button to run the program until the cursor position is reached. Note that you will have to place the cursor on a valid position. The program won't halt if it doesn't reach that position.

Step over: execute the current line and halt at the next executable line at the same level. Does step over subs/functions (a call is done in a single step).

Step into: execute until a sub/function is called and halt at the first executable line of that sub/function.

Step to end of sub/function: execute the current sub/function until it is left and halt at the last executable line of that sub function (which is END SUB or END FUNCTION).

Step out: execute the current sub/function until it is left and halt at the first executable line after the sub/function was already left.

Step automatically: perform single steps automatically in the debugger until the user presses the “halt” button. The steps are done at a speed that allows the eye to follow the execution (except there are jumps…)

Run/Continue: run/continue the program execution. The program can be halted again by pressing the “halt” button. Then the user can resume debugging stepwise. The execution is also halted when a (user-defined) breakpoint is hit.

Halt: halts a currently running program. Use to pause program execution if the debugger is in “step auto” or “run” mode to resume debugging afterwards. The program is not halted immediately, but as soon as the next executable line is reached.

Release: run/continue the program execution without debugging. The program can be halted again by pressing the “halt” button. Then the user can resume debugging stepwise. Breakpoints are ignored!

Restart: terminate the currently running program and restart it. The program is again halted at the first executable line.

Breakpoints are used to define certain positions in your source code that cause the debugger to halt the program, when such a position is reached. They are used to debug a specific part of a program, while the other parts can be executed normally.

Locate the cursor on an executable line in the source code and press F3 to set/unset a breakpoint on that location. You can set multiple breakpoints across your code. Finally press the “Run/Continue” button to run your program until a breakpoint is hit.

The “Proc/Var” window on the right side of FBdebugger lists all the variables known in the current program execution. You can view its current values, follow their storage location in memory (view a hexdump) or even modify them. Note that FreeBasic converts all names to uppercase during compilation.