



Speare Code Editor The Small Lua IDE for Game and Lua Development

Copyright (C) 2020 Sevenuc Consulting
Version 1.0
Update: 4 Mar 2020


Free, Lightweight, Open Source, Extendable Flexibility

Speare (<http://sevenuc.com/en/Speare.html>) is an ultra lightweight code editor and a small IDE that has an efficient code navigation and call routines tracing ability, and has integrated debugging environment for C, C++ and Lua. It was originally developed to providing a native scripting language debugging environment that seamlessly integrated with C and C++, and well supports all kinds of game development that using Lua as its scripting language. It not only has very concise user interface but also has a very flexible architecture to easily add a new programming language code runner, parser, syntax highlighting, code formatter and debugger in it.

For general Speare code editor usage, please refer this document:
http://sevenuc.com/download/Speare_quick_reference.pdf

Debug Mode

1. Show the debug toolbar

Click  siding bottom button.

2. Debug toolbar



From left to right, Start, Stop, Step Into, Step Out, Run To, Step Over, Show Watches.

The "Step Over" is equals to "Step next", and "Step To" is equals to "Continue" in common debugging words, and the "Step To" is the command that tell the debugger run to meet a breakpoint or an exception occurred or the program meet exit.

On the rightmost there are three other function units, they are, search items in the stackview, siding stackview, and clean the debug output.

Search in the debug output

Click in the output area and use the shortcut key "Control + F" to do the searching.

3. Socket Port

You can set the socket communication port number both used by Debug Server and the Speare code editor. Open the Preferences of Speare and select the "Debug Settings" tab then input your number.

Note: Please remember to empty the port number when you switched to debugging with the default builtin programming languages with default port number.

4. Watches

Watches used to evaluate variable or expression and their values can be realtime showing in stackview when debugging session paused, the nodes normally has a green colour and always placed on the top of stackview.

Caution:

a. Please ensure all source files have been dragged in the left side Treeview (Workspace Explorer) before start a debug session, because macOS app can't be allowed to access files outside of its sandbox.

b. When your source code file moved to another folder, you must drag the source code folder in Speare again then the debugging can correctly work.

C and C++ Debugger

The C and C++ debugger of Speare code editor implemented as a script client of **LLDB** (<http://lldb.lvm.org/>), and supports extend it by yourself. The builtin module supports parsing modern C++ source code files including C++11 and C++14 syntax, e.g namespace, anonymous function, structure, union, class, enum etc and so many new features of the C++ programming language. You can enjoy debugging almost any type of C and C++ applications under the lightweight debugging environment of Speare code editor.

Start Debugging Steps:

1. Download Speare Debug Server:

→ http://sevenuc.com/download/c_debugger.tar.gz (10KB)

The source code of the C and C++ debugger can be view online here:

<https://github.com/chengdu/Speare> or here:

<https://sourceforge.net/projects/speare>

2. Uncompress the tarball:

Uncompress it to your local directory. e.g ~/Desktop and take a look at the readme.txt in it.

3. Start the debug server:

Please refer the readme.txt file.

4. Debug session start:

click "Start" button on the debug toolbar of Speare code editor.

Add breakpoint, step in, step out, step next, watch stack trace ...

5. Run extra commands:

Right click in the stackview (bottom left side) and then input any [lldb](#) command when the debugging session paused. Left click anywhere outside of the input box to close it and the command will be directly send to the debug server.

a. Add function breakpoints

- . **breakpoint set --name functionname**: add a C function breakpoint.
- . **breakpoint set --name classname::functionname**: add a C++ function breakpoint.

b. Process operation

- . **process attach --name xxx --waitfor**: attach another process by name.
- . **process attach --pid xxx**: attach another process by pid #xxx.

c. Thread operation

- . **thread list**: show all thread of current process.
- . **thread select 2**: select thread #2.
- . **thread backtrace all**: show thread info.
- . **register read**: read all CPU registers.
- . **thread step-inst**: step one machine instruction.
- . **thread step-over-inst**: step return one machine instruction.

d. Watchpoint operation

- . **watch list -v**: list watchpoints.
- . **watchpoint set variable x**: add a watchpoint x.

e. Frame operation

- . **frame list**: print all frame of the current thread.
- . **frame select 9**: select frame #9.

f. Display variable value

- . **frame variable x**: print x value.

...

Tips: Run to (run to meet a breakpoint), the source file that you want debugger stopped in it must already opened and has at least one breakpoint before run the command.

Modify the C and C++ debugger

You can directly modify the script client of lldb to satisfy your requirements.

Lua Debugger

The Lua debugger of Speare code editor implemented as a module of [Lua \(https://www.lua.org\)](https://www.lua.org), and support all common versions of Lua. You can conveniently enjoy debugging with any kinds of customised Lua interpreter and LuaJIT.

Tested Lua version includes: [5.1.4](#), [5.1.5](#), [5.2.4](#), [5.3.5](#) [5.4.0-alpha](#).

Start Debugging Steps:

1. Download Speare Debug Server:

→ http://sevenuc.com/download/lua_debugger.tar.gz (518KB)

2. Uncompress the tarball

Uncompress it to your local directory. e.g ~/Desktop and take a look at the readme.txt in it.

3. Start the debug server

```
$ cd ~/Desktop/debugger/5.1  
$ ./lua_514 server.lua
```

4. Debug session start

Click "Start" button on the debug toolbar of Speare code editor. Add breakpoint, step in, step out, step next, watch stack trace ...

Replace Lua interpreter

You can directly replace the Lua interpreter with your own customised version under the debugger directory.

Extend Speare Code Editor for Lua development and Game engines

Speare Code Editor can be easily extended to support any type of game engine or game framework that using Lua as scripting language. Lua game engine and frameworks can be easily supported, such as the LÖVE 2D open source game engine, the Moai open source game engine, the Gideros mobile game platform, the Corona 2D game engine, the Marmalade game engine, the Cocos2d and cocos2d-x game engine, the OpenResty Nginx web server, the Torch machine

learning framework, the Redis open source database, the Adobe Lightroom, the Lapis web framework, and the MoonScript dynamic scripting language, and other countless game engines.

To add scripts to support better debugging game and Lua in Speare code editor, please download the guide from here:

http://sevenuc.com/download/language_extension_protocol.pdf,

and following the description in it.