



Speare Code Editor Debugger Quick Reference


Copyright (C) 2020 Sevenuc Consulting
Version 1.0
update: Jan 18 2020

Speare (<http://sevenuc.com/en/Speare.html>) is an ultra lightweight code editor and small IDE that has an efficient code navigation and call routines tracing ability, and has integrated debugging environment for programming languages including C, C++, Ruby, mruby, Lua, Python, PHP, Perl and Tcl. It was originally developed to providing a native scripting language debugging environment that seamlessly integrated with C and C++. 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.llvm.org/) (<http://lldb.llvm.org/>), and supports extend it by yourself. 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](http://lldb.llvm.org/) 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.

mruby Debugger

The mruby debugger of Speare code editor is a patched version of [mruby](http://mruby.org) (<http://mruby.org>) that support remote debugging mruby project, currently support mruby version 2.0.1 and 2.1.0.

1. Install mruby debugging server

Download mruby remote debugger:

→ http://sevenuc.com/download/mruby_debugger.tar.gz (733KB)

Download mruby-2.0.1.tar.gz (518KB) or mruby-2.1.0.tar.gz (585KB) from <https://github.com/mruby/mruby>

```
$ cd mruby-2.0.1 or mruby-2.1.0
```

```
$ make
```

compile mruby and replace mrdb under bin directory with the corresponding version.

```
$ cd bin
```

```
$ ./mrdb : start the mruby remote debugger.
```

2. Configuring Speare code editor

Launch Speare and open the Preferences of Speare and select the tab of "**Debug Settings**" then check on "**Enable mruby debugging**".

Please remember to turn the option off when you switched to debug common Ruby applications.

3. 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 ...

Tips: Separate modules of your app with mruby gems instead of using require.

Ruby Debugger

The Ruby debugger of Speare code editor implemented as a client of rdebug-ide, and Ruby interpreter that has a rdebug-ide installed will be running as the debug server.

Ruby debugging environment support all kinds of Ruby interpreters, the version includes: 1.8.x, 1.9.x, 2.x, and JRuby.

Steps of start debugging session:

1. Download and install debug gems

For Ruby 1.8.x: download: ruby-debug-base (0.10.4)

```
$ gem install --force --local ruby-debug-base-0.10.4.gem
```

```
$ gem install ruby-debug-ide
```

For Ruby 1.9.x: download: ruby-debug-base19 (0.11.25)

```
$ gem install --force --local ruby-debug-base19x-0.11.32.gem
```

```
$ gem install ruby-debug-ide
```

For Ruby 2.x:

```
$ gem install debug
```

```
$ gem install ruby-debug-ide
```

2. Start the debug server

```
$ rdebug-ide --host 127.0.0.1 --port 1234 --dispatcher-port 1234 --main.rb
```

(**Note:** Please replace the main.rb file with your script file.)

For Ruby on Rails:

```
$ rdebug-ide --host 0.0.0.0 --port 1234 --dispatcher-port 1234 --bin/rails s
```

3. 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 ...

4. Add condition breakpoint

Right click on the breakpoint, on the prompt menu, → select "Condition" and then input expression or use empty string to remove the condition, left click outside of the input box to close it and execute the command. e.g. $x > 5$ means: Pause on the breakpoint only if $x > 5$ is true.

5. Run Extra Commands

Right click in the stackview (bottom left side) and then input extra 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. Variables

- . **var const object**: show constants of object.
- . **var instance object**: show instance variables of object, object can be given by its id or an expression.
- . **var inspect**: reference inspection results in order to save them from the GC.

b. Expression

- . **p expression**: evaluate expression and print its value.
- . **pp expression**: evaluate expression and print its value.
- . **eval expression**: evaluate expression and print its value, alias for p.
- . **expression_info expression**: returns parser-related information for the expression given 'incomplete'=true | false indicates whether expression is a complete ruby expression and can be evaluated without getting syntax errors.

c. Backtrace

- . **where**: display frames.
- . **bt | backtrace**: alias for where.
- . **up | down [count]**: move to higher or lower frame.
- . **frame [frame-number]**: Move the current frame to the specified frame number. (A negative number indicates position from the other end. So 'frame -1' moves to the oldest frame, and 'frame 0' moves to the newest frame.)

d. Jump

Change the next line of code to be executed.

- . **jump line**: jump to line number (absolute).
- . **jump -line**: jump back to line (relative).
- . **jump +line**: jump ahead to line (relative).

e. Thread

- . **thread list**: list all threads.
- . **thread current**: show current thread.
- . **thread switch <nnn>**: **switch** thread context to nnn.
- . **thread inspect <nnn>**: switch thread context to nnn but don't resume any threads.
- . **thread resume <nnn>**: resume thread nnn.
- . **thread stop <nnn>**: stop thread nnn.

f. Type Set

- . **set_type <var> <type>**: Change the type of <var> to <type>.

g. File Operation

- . **load file**: read and parse file every time instead of require.
- . **file-filter on | off**: enable or disable file filtering.
- . **include file | dir**: adds file or dir to file filter (either remove already excluded or add as included).
- . **exclude file | dir**: exclude file or dir from file filter (either remove already included or add as exclude).

Switch Ruby Interpreter

You can directly switch between any Ruby interpreter or your own version of Ruby and then config it to support rdebug-ide.

Appendix:

Make a fresh Ruby debugging environment.

Step 1. build an openssl library

```
$ download https://www.openssl.org/source/openssl-1.0.2t.tar.gz
$ tar -zxvf openssl-1.0.2t.tar.gz
$ cd openssl-1.0.2t
$ export KERNEL_BITS=64
$ ./config no-ssl2 no-ssl3 no-shared enable-ec_nistp_64_gcc_128 \
  --prefix=/Users/yeung/Public/Rdebug/openssl \
  --openssldir=/Users/yeung/Public/Rdebug/openssl
$ make && make install
```

Step 2. build a ruby interpreter

```
$ download https://cache.ruby-lang.org/pub/ruby/2.1/ruby-
2.1.2.tar.bz2
$ tar -jxvf ruby-2.1.2.tar.bz2
$ export LDFLAGS=-L/Users/yeung/Public/Rdebug/openssl/lib -
lcrypto -lssl
$ export CFLAGS=-I/Users/yeung/Public/Rdebug/openssl/include
$ export
PKG_CONFIG_PATH=/Users/yeung/Public/Rdebug/openssl/pkgconf
ig
$ cd ruby-2.1.2
$ ./configure --prefix=/Users/yeung/Public/Rdebug/2.x/ruby2 \
  --with-openssl-dir=/Users/yeung/Public/Rdebug/openssl
```

Alternative: directly modify Makefile to add openssl library link options

```
LDFLAGS = $(CFLAGS) -L. -fstack-protector -L/usr/local/lib -
L/Users/yeung/Public/Rdebug/openssl -lcrypto -lssl
```

```
$ make && make install
```

Step 3. install debug gems

```
$ download https://rubygems.org/downloads/debase-  
ruby_core_source-0.10.5.gem  
$ download https://rubygems.org/downloads/debase-  
0.2.5.beta1.gem  
$ download https://rubygems.org/downloads/ruby-debug-ide-  
0.7.0.gem  
$ export PATH=/Users/yeung/Public/Rdebug/2.x/ruby2/bin:$PATH  
$ gem install --force --local debase-ruby_core_source-0.10.5.gem  
$ gem install --force --local debase-0.2.5.beta1.gem  
$ gem install --force --local ruby-debug-ide-0.7.0.gem
```

Step 4. start debugging session

```
$ rdebug-ide --host 127.0.0.1 --port 1234 --dispatcher-port 1234 --  
xxx/xxx/xxx/main.rb
```

Add breakpoints in Speare code editor.

Click "Start" button on the debug toolbar of Speare code editor.
step in, step out, step next ...

Lua Debugger

The Lua debugger of Speare code editor implemented as a module of Lua (<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.

Python Debugger

The Python debugger of Speare code editor supports Python version 2.5, 2.6, 2.7 and 3.x, and MicroPython. You can enjoy debugging Python scripts as same as debugging web applications that based on web frameworks such as Flask and Django under the lightweight environment of Speare code editor.

Steps of Start Debugging Session:

1. Download Speare Debug Server:

→ http://sevenuc.com/download/python_debugger.tar.gz (30KB)

The source code of the Python 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

```
$ cd ~/Desktop/debugger/2.x  
$ python server.py
```

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. Add condition breakpoint

Described in the Ruby debug section of this page.

6. Run Extra Commands

Right click in the stackview (bottom left side) and then input extra 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. Remove all breakpoints

. **clear**: clear all breakpoints of current file.

b. Stack trace and frame operation

- . where:** Print stack trace, an arrow indicates the "current frame".
- . up [count]:** Move stack trace to older frame.
- . down [count]:** Move stack trace to newer frame.

c. Display argument list

- . args:** Print the argument list of the current function.

d. Display return value

- . retval:** Print the return value for the last return of a function.

e. Display value of expression

- . p expression:** Print the value of the expression.
- . pp expression:** Pretty-print the value of the expression.
- . display expression:** Display the value of the expression, Python 3.x only.
- . undisplay:** Clear all display expressions for the current frame, Python 3.x only.
- . undisplay expression:** Do not display the expression any more in the current frame, Python 3.x only.

f. Display argument type

- . whatis argument:** Print the type of the argument.

g. Display source code of object

- . source expression:** Python 3.x only.

h. Continue execution

- . until:** continue execution until the line number greater than the current is reached.
- . until [lineno]:** continue execution until line number greater or equal to the lineno is reached.

i. Add module search path

- . basedir directory:** insert a directory in sys.path.

Switch Python interpreter

You can directly switch CPython interpreter to MicroPython or your own self-compiled version of Python, or others such as PyPy, Jython and IronPython.

PHP Debugger

The PHP debugger of Speare code editor supports all kinds debugging of PHP applications and any version of PHP interpreter that has Xdebug support from PHP 5.x to PHP 7.x. Different with Lua, Ruby and Python debugging, this time Speare acts as debug server and Xdebug as the client.

Setting up Xdebug for PHP Debugging:

1. Download Xdebug:

<https://xdebug.org/files/xdebug-2.6.0.tgz>

2. Compile and install Xdebug:

```
$ rm configure.in (optional)
$ rm configure.ac (optional)
$ phpize && ./configure --enable-xdebug && make clean && make all
$ sudo make install
```

At this step Terminal will report:

/usr/lib/php/extensions Operation not permitted.

This is because SIP default set to be enabled by macOS, even you execute sudo operation, the system protected directories still can't be writable.

WARNING: THE FOLLOWING OPERATION IS VERY DANGERS.

Assuming that you know what you're doing, here is how to change SIP (System Integrity Protection) settings on your Mac. Turn off your Mac (Apple → Shut Down...), hold down Command-R and press the Power button. Keep holding Command-R until the Apple logo appears and wait for OS X to boot into the "OS X Utility" window and then choose Utilities → Terminal.

Turn off SIP:

```
$ csrutil disable
$ csrutil status
$ reboot
```

Turn on SIP:

```
$ csrutil enable
```

```
$ csrutil status
```

```
$ reboot
```

After you turn off SIP and execute "\$ sudo make install" again, Xdebug should be successfully installed on your system, you can check it by this command:

```
$ php -v
```

It should print something like the following:

```
PHP 7.1.23 (cli) (built: Nov 27 2018 16:59:25) ( NTS )
```

```
Copyright (c) 1997-2018 The PHP Group
```

```
Zend Engine v3.1.0, Copyright (c) 1998-2018 Zend Technologies
```

```
with Xdebug v2.6.0, Copyright (c) 2002-2018, by Derick Rethans
```

3. Configuring web server:

```
$ cp /etc/apache2/httpd.conf ~/Desktop/
```

```
LoadModule userdir_module libexec/apache2/mod_userdir.so
```

```
LoadModule alias_module libexec/apache2/mod_alias.so
```

```
LoadModule rewrite_module libexec/apache2/mod_rewrite.so
```

```
LoadModule php7_module libexec/apache2/libphp7.so
```

```
Include /private/etc/apache2/other/*.conf
```

Edit file http.conf and ensure the above lines not be commented.

```
$ sudo cp ~/Desktop/httpd.conf /etc/apache2/
```

Save back the settings file.

```
$ apachectl configtest
```

Check there syntax is legal (optional).

4. Configuring PHP interpreter:

```
$ cp /etc/php.ini.default ~/Desktop/
```

```
enable_dl = On
[xdebug]
zend_extension = /usr/lib/php/extensions/no-debug-non-zts-
20160303/xdebug.so
xdebug.remote_enable = 1
xdebug.remote_host = "127.0.0.1"
xdebug.remote_port = 9000
xdebug.remote_handler = "dbgp"
xdebug.remote_mode = req
xdebug.remote_connect_back = 1
xdebug.remote_autostart=1
```

Edit file php.ini.default and carefully check the above content have been written in it.

Note: xdebug.remote_autostart=1 should be removed at product environment.

```
$ sudo cp ~/Desktop/php.ini.default /etc/
```

Save back the settings file.

```
$ sudo mv /etc/php.ini.default /etc/php.ini
```

Rename the settings file to take effect.

5. Run PHP test with Xdebug:

```
<?php
echo phpinfo();
?>
```

Save the above content in a file named test.php and put it under
/Library/WebServer/Documents/


```
$ sudo launchctl unload -w  
/System/Library/LaunchDaemons/org.apache.httpd.plist  
$ sudo launchctl load -w  
/System/Library/LaunchDaemons/org.apache.httpd.plist
```

Restart apache, launch Safari and request: <http://127.0.0.1/test.php>.
There're some text in zend engine section should be as same as
printed by execute the command "`$ php -v`" on command line.

Steps of PHP Debugging in Speare code editor:

1. Launch Speare and dragging the PHP source code folder into the "Workspace Explorer".
2. Select the startup php script and click siding bottom button to show the debug toolbar and then click "Start" button.
3. Launch Safari and request: <http://127.0.0.1/xxx/xxx/index.php>

Note: Don't use localhost but should use 127.0.0.1 instead.

4. After Speare paused and highlighting at a special line of your startup PHP script there that means the debugging session started, you can execute the common debugging command now:

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

Switch PHP interpreter

You can switch any version of PHP interpreter directly; it does not affect the server side Speare code editor. PHP is an excellent programming language to develop Web Applications based on frameworks such as Drupal, Zend Framework, CodeIgniter, Symfony and Yii framework etc, but not limited that, in fact PHP is also very suitable to develop command line applications.

Perl Debugger

The Perl debugger of Speare code editor implemented as a patched version of perl5db.pl, and support extend it by yourself. The debugger was based on the builtin debugger of Perl, so it can work with all versions of Perl interpreter that perl5db.pl supported.

Start Debugging Steps:

1. Download Speare Debug Server

→ http://sevenuc.com/download/perl_debugger.tar.gz (104KB)

The source code of the Perl 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

```
$ cd ~/Desktop/debugger
```

```
$ perl -I ~/Desktop/debugger/Speare -d:Debugger fullpath.pl
```

* **Warning:** fullpath.pl the file must input with full path.

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 Perl debug 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

. **b functionname:** add a function breakpoint.

b. Add condition breakpoints

. via breakpoint marker: Right click on the breakpoint, on the prompt menu, → select "Condition" and then input expression or use empty string to remove the condition, left click outside of the input box to close it and execute the command. e.g. `x > 5` means: Pause on the breakpoint only if `x > 5` is true.

. b fullpath.pl condition: e.g. `b /xxx/xxx/code.pl 6 $x > 5`.
`/xxx/xxx/code.pl`: fullpath of the script file. (do the same thing, optional)

c. Watchpoint operation

. w expr: add a watchpoint expr.
. W expr: delete watchpoint expr.
. W *: delete all watchpoints.

d. Evaluate express

. e expr: e.g. `"e $x+$y"`.

e. Display variable value

. p \$x: print value of variable x.
. p expr: print value of expression expr.
...

Switch Perl Interpreter

You can directly switch the Perl interpreter on the command line or debugging with your own self-compiled version of Perl.

Tcl Debugger

The Tcl debugger of Speare code editor implemented with Tcl scripts and an extension written with C to parse Tcl source code, and support extend it by yourself. You can enjoy debugging almost all kinds of Tcl applications under the lightweight debugging environment of Speare code editor.

Start Debugging Steps:

1. Download Speare Debug Server

→ http://sevenuc.com/download/tcl_debugger.tar.gz (223KB)

The source code of the Tcl 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 ...

Modify the Tcl debugger

You can directly modify the source code of the Tcl Debugger to satisfy your requirements.

Switch Tcl interpreter

You can directly switch the Tcl interpreter on the command line or debugging with your own self-compiled version of Tcl.

Programming Languages and Document Types

AMPL ASM ASP **AWK** Ada **ActionScript** Active4D AnsiblePlaybook Ant
Apache **Applescript** Asciidoc AutoIt Autoconf Automake **Basic** Batch
Beta Bibtex **C** **C#** **C++** CMake CPrePro **CSS** Scss CUDA Clojure **Cobol**
CoffeeScript ColdFusion Csound Ctags D DTS DbusIntrospect **Diff**
DosBatch PowerShell Dtd Dylan **Eiffel** **Elixir** **Elm** **Erlang** eZ Publish F-
Script Falcon Flex **Forth** **Fortran** FreeFem++ Fypp GEDCOM Gdbinit
Glade **Go** Gradle GraphViz Groovy **Haskell** **HTML** **Haxe** Header IDL
ITcl Iniconf Inko JSP Java JavaFX JavaProperties **JavaScript** **JSON** Julia
Kotlin Kuin LSL **LaTeX** LdScript **Lilypond** **Lisp** Logtalk **Lua** M4 MEL
Makefile Man **Markdown** **MATLAB** Maven2 MetaPost Metaslang
Moose **MySQL** Myrddin NASL NEURON Nemerle **Objective-C** **Octave**
Ocaml Ox **PDF** **PHP** **Parrot** **Pascal** Passwd **Perl** Perl6 Pig Plist Pod
PostScript **Prolog** Processing Protobuf Puppet PuppetManifest
Python PythonLoggingConfig QemuHX QtMoc **R** R/S-PLUS RHTML
RSpec RelaxNG **Rexx** Robot Racket RpmSpec Rst **Ruby** **mruby** **Rust**
SGML SML **SQL** **Scala** **Scheme** Sedona **Shell** Slang **Smalltalk** Snippets
Stata SuperCollider **SVG** **Swift** SystemTap SystemdUnit TTCN **Tcl**
Tcl/Tk **Tex** TorqueScript **TypeScript** Udo VB VB.NET VHDL Vera
Verilog **SystemVerilog** **Vim** WSDL WindRes **XML** XSD Xquery Xslt
Yacc **YAML** YumRepo Zephir

To add a new programming language code runner, parser, syntax highlighting, code formatter and debugger in Speare code editor, please download the guide from here:

http://sevenuc.com/download/language_extension_protocol.pdf,
and following the description in it.