Programming Language Extension Protocol for Speare Code Editor

Version 1.2
Update: Jan 30 2020

Speare (http://www.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 integrated debugging environment for C, C++, Ruby, mruby, Lua, Python, PHP, Perl and Tcl. It not only has very concise user interface but also has a very flexible architecture to easily extend it using the definitions specified in this document. Speare code editor adopts the definitions of this document to describe how to add a new programming language code runner, parser, syntax highlighting, code formatter and debugger in it.

This document specifies the general interface, data format and communication protocol of an IDE (Integrated Development Environment) for a generic programming language code runner, parser, syntax highlighting, code formatter and debugger originally (can be found at http://www.sevenuc.com/download/language_extension_protocol.pdf). The definitions are very simple but very efficient and have broad applicability and Sevenuc Consulting reserves all rights of the definitions.

Language Parser

```
#!/bin/bash
# full path of the language parser
path = /usr/local/xxx/bin/xxparser
$path $1
```

The above is a sample shell script that accepts a file path as input parameter and output parsed symbols in stdout.

1. The output format:

Speare code editor parse the output line by line, each line contains four data fields, each data field separated by a character '\t', the first field is the symbol definition, and then symbol class (e.g. header, module, package, library, declare etc), and then the full path of the file (optional) and end with line number.

e.g. myfunction\tfunction\t/Users/henry/Desktop/uclib/list.c\t210

2. The location of the language parsers:

~/Library/Application Scripts/com.sevenuc.SpeareHelper/parsers
Speare code editor call the parsers in the above directory and parsing anything printed in stdout to fetch symbol definitions in source code files.

Syntax Highlighting

Speare code editor use the extension name of the source code files to determine which language syntax highlighting definition should be called.

1. Configure file extension name

Open the "Extra" panel of Preferences, input the language name (or document type) and it's file extension names.

2. Create syntax highlighting definitions

Assume the new programming language named MyLang and it source code file has .my and .myd extension file names, then it should has a correspond syntax definition file that be named as MyLang.plist and be correctly put in:
"~/Library/Application Scripts/com.sevenuc.SpeareHelper/languages",
details of such type of file please refer the exists files under "/Applications/Speare.app/Contents/Resources/Languages". Of course, you can copy a similar language definition file as a start.

Code Formatter

```
#!/bin/bash
# full path of the code formatter
path = /usr/local/xxx/bin/xxfmt
$path $1 $2
```

The above is a sample shell script that accepts two input parameters (a file path and a “number of space per tab”) and print formatted code block in stdout.

1. The rule of code formatter:

Speare code editor will give up formatted code block if there's something printed in stderr or nothing printed in stdout otherwise anything printed in stdout will be used to replace the selected code block.

2. The location of the language formatters:

```
~/Library/Application Scripts/com.sevenuc.SpeareHelper/formatters
```

Speare code editor call the formatters in the above directory, follow the above rule and replace the current selected text with the output of a formatter.

Speare Debugging Protocol

This section describes how to implement a generic debugger to support a new programming language debugging in Speare code editor. Generally, a language debugger acts as the debug server and Speare code editor as the client, and they communicates with TCP socket on a special port defined in the Preferences of Speare code editor. Speare code editor send debugging command line by line and then the debugger parse the command line by line. Speare code editor parse the debugger’s response data with JSON data format, if the data is not formatted JSON data, it will be directly printed in the debug output. JSON data block allows containing multiple data lines.

To prevent common data output mixed with JSON data block, the JSON data block should be quoted with “\r\n”, i.e. the JSON data block must starts with “\r\n” and ends with “\r\n”.

1. Initialize debugging session

When Speare code editor successfully connected with the debugger, it will send a file path (full path of the current selected file) as the start parameter of the debugging session, after 5 seconds, if the connection failed, it will give up the connection.

The command is the path of the init file.

2. Disconnect debugging session

exit|quit

3. Breakpoint operations

breakpoint [add|remove|enable|disable] file line
breakpoint set file line “condition”
breakpoint setf “function name”
breakpoint rmf “function name”

Speare code editor has four breakpoint operation commands, each command contains different data fields, each data field separated by a character '\t', the fist command has four data fields, and the second has five data fields, the last two commands add or remove function breakpoint and send manually by user via extra command.

4. Debugging step commands

step [into|out|over|to]

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. A character '\t' used to separate the “step” and the second word instead of a space.

5. Evaluate expression

eval “expression”

A character '\t' used to separate the “eval” and the expression, and the expression is a quoted string.

6. Extra debug command
Right click in the stackview (bottom left side) of Speare code editor and then input any 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 debugger.

Note: In order to make the debugger happy and can easily parse extra commands, the command format have the same format as the mentioned previous commands will more better, and the data fields can also be separated by a character ‘\t’.

JSON Data Block

Debugger send response to Speare code editor with JSON data format, they are:

a. Debugging session exit
'{"command": "exit"}'

b. Debugging session paused
'{"command": "paused", "file": "%s", "line": %d}'

c. Stack dump
'{"command": "stack", “data”: ... }'

Debugger sends “stack” command to tell Speare code editor displaying values of global and local variable, the “data” can be any structure that has a correct JSON format, usually it is a structured tree grouped with variable name and their values.

d. Expression evaluate
'{"command": "expression", "string": “%s”, "value": “%s” }'

Watches: “printed value of variables” or “expression evaluated values”, all of them send by one command on IDE side:

eval “variable name or expression”

In many debugger, it just equals to command, “**print variable**”, in fact, print variable is not necessary in most situation. Speare code editor will merge the response data into the stackview (bottom left side) and always placed on the top of the stackview if it has a correct JSON format.

Note: In the respond JSON data block, “string” is the raw expression sent from IDE side and the “value” is the evaluated result.

Syntax Checking and Run Editing Code Instantly

Speare code editor provides a common interface that enables end user to run code for various tasks immediately, usually includes syntax checking, unit test, and run the editing code instantly.

The location of the code runners:

~/Library/Application Scripts/com.sevenuc.SpeareHelper/commands

End user can put different of shell scripts in this directory to instantly run other tasks, such as automatic compile, building and packing, or run static code analysis tool (source code analyzer, such as PC-Lint, clang-analyzer, PHPLint, OCLint, SwiftLint, and PMD etc), and any other tasks. Speare code editor determine which shell script should be called by the extension name of the source code files (e.g. c.sh: *.c, cc.sh: *.cc, *.cxx, *.cpp, *.c++, ruby.sh: *.rb, lua.sh: *.lua, python.sh: *.py etc). One shell script service for one programming language. Every script can accept two input parameters, the full path of the current selected file and the selected code block, and the last parameter is optional and can be empty string.

```
#!/bin/bash
# $1 is the path of the current selected file
# $2 is the selected code block (optional)
ruby $1
```

The above is a sample shell script that accepts a file as input parameter and run it, if there's some syntax error, the error message will be printed in the debug output. Speare code editor will merge stderr and stdout together and print them in the debug output.

File Extension Name Association

Speare code editor will try to connect debugger and determine which language parser and code formatter should be called by the extension name of the source code files. Open the “Preferences” of Speare code editor and select the "Extra" panel and then configure the association between a programming language and its source code file extension names. Each extension name should be separated by a space.

e.g: a sample programming language named Mylang, and file extension name of its source code file are my and myd (without dot). Speare code editor make a lower case string of the programming language name to find the shell scripts. It will find a shell script named mylang.sh and call it to parse source code files under:

~/Library/Application Scripts/com.sevenuc.SpeareHelper/parsers

and find a shell script named mylang.sh and call it to format code block under:

~/Library/Application Scripts/com.sevenuc.SpeareHelper/formatters