Visual Studio Code Verbose Feedback Extension Developer Manual

Dirk Vet
dirk.vet@student.uva.nl

June 10, 2022

Contents

1 Functionality

The Verbose Feedback extension is made for novice programmers to teach them C more quickly and effectively. The extension is able to return errors for the Clang and GCC compilers through various methods. To teach the learners in a efficient manner, the errors from the Clang compiler are delivered together with a verbose description and possibly with a hint to solve the issue. The extension implements a Clang language server and client to deliver the feedback[5][4]. The extension is based on the "C/C++ Advanced Lint for VS Code" extension from Joseph Benden and makes also use of the Microsoft C/C++ extension. [1][3].

2 Systems used for development

Operating System:
Description: Ubuntu 20.04.4 LTS
Release: 20.04
Codename: focal

Project dependencies:
VScode: 1.67.2
Node.js: 16.2.0
npm: 8.7.0
GCC: 9.4.0
Clang: 10.0.0-4ubuntu1
bash timeout command

3 Build the project

To build the project use `npm install` inside the project root Next use `npm run compile` or `npm run watch` to build the server. This will create the `client/out` and `server/out` directory with the compiled server. You will also need to install the Microsoft C/C++ extension, but that can also be installed during debugging.

In case of problems look at the "Development Setup" in the README.md file from Joseph Benden [1].

4 Debugging the extension

When changing the code of the extension, you should recompile with `npm run compile` in the root folder of the project. Then `F5` will start the Extension Development Host, which runs a VScode instance with your extension. It is important to have a ./.vscode/tasks.json file in your development host in case you run `Verbose Feedback: Compile and run the program`. Parameters for the compile command for tasks.json that includes all .c files in the current directory could be for example:

```
"args": [
"-fdiagnostics-color=always",
"-g",
"${fileDirname}/*.c",
"-o",
"${fileDirname}/${fileBasenameNoExtension}"
]
```

For information on the usage of the extension, please refer to the student manual.

5 File structure

5.1 client/src/extension.ts

This is the client code, which contains all the static error handling and the definitions of the commands.

5.2 server/src/server.ts

This implements the language server, which will do all the dynamic error handling.

5.3 server/src/error-regex.json

The language server matches the errors from the Clang compiler with regular expressions to determine what feedback should be returned. The `error-regex.json` file consists of 3 elements that are required for the error handling of dynamic errors. The 3 elements for each error are:

- **Regex** - A regular expression to capture the original error message. If the original error matches with the regular expression, the corresponding Translation and Phase will be used for error handling.

- **Translation** - The simplified error message.
- **Phase** - A label for the compiler phase at which the error takes place.

The errors that have been covered are based on the course content of the course Inleiding Programmeren from the Bachelor program Computing science of the University of Amsterdam and common mistakes made by novice C programmers [2].

6 Error types

The extension produces four types of error. These differ in the way that they appear in the Problems window.

- Static diagnostics: diagnostics appear and disappear by executing a command through ctrl+P. These are prefixed with *[S]*.
- Dynamic diagnostics: diagnostics appear and disappear while typing. These are prefixed with *[D]*.

7 Commands

By using ctrl+P and typing "Verbose Feedback" you will see a list of available commands as shown below:

- Verbose Feedback: Switch on/off Clang compiler with feedback.
  - Switch between dynamic 'C/C++' and '[Clang] Verbose Feedback' diagnostics to turn on/off feedback support.
- Verbose Feedback: Check for GCC runtime errors/warnings.
  - Check for runtime errors by compiling all .c files in the current directory using the GCC compiler and by running the executable. Returns static '[GCC] Verbose Feedback' diagnostics.
- Verbose Feedback: Check for Clang runtime errors/warnings.
  - Check for runtime errors by compiling all .c files in the current directory using the Clang compiler and by running the executable. Returns static '[Clang] Verbose Feedback' diagnostics.
- Verbose Feedback: Compile/link the active document with GCC.

  - Check for compiler or linking errors by compiling all .c files in the current directory using the GCC compiler. Returns static '[GCC] Verbose Feedback' diagnostics.

- Verbose Feedback: Compile/link the active document with Clang.
  - Check for compiler or linking errors by compiling all .c files in the current directory using the Clang compiler. Returns static '[Clang] Verbose Feedback' diagnostics.

- Verbose Feedback: Compile and run the program.
  - Compile and run the program.

- Verbose Feedback: Clear all errors by reloading the window.
  - Clears all errors by reloading VScode.

7.1 Verbose Feedback: Switch on/off Clang compiler with feedback

With the Verbose Feedback: Switch on/off Clang compiler with feedback. command the user can switch between diagnostics with or without feedback.

In case no feedback should be shown the command C_Cpp.EnableErrorSquiggles from the Microsoft C/C++ extension will be called to show these diagnostics. The FEEDBACK_MODE boolean will change to false and the client sends a notification to the language server that the feedback mode has changed, so the [Clang] Verbose Feedback errors will not be shown.

In case feedback should be shown the command C_Cpp.DisableErrorSquiggles from the Microsoft C/C++ extension will be called to turn off the C/C++ diagnostics. The FEEDBACK_MODE boolean will change to true and the client sends a notification to the language server that the feedback mode has changed, so the [Clang] Verbose Feedback errors will be shown.

7.2 Verbose Feedback: Compile and run the program

When running the command VScode will ask on the first call for which compiler you want to use. This results in VScode creating a `tasks.json` file in `./.vscode`. This is because in the code the C_Cpp.BuildAndRunFile command from the C/C++ extension is called. The errors that are returned are static and carry the label 'gcc' regardless of which compiler has been used. This cannot be changed in the code as this is in control of the C/C++ extension.

7.3 Verbose Feedback: Compile/link the active document with <*compiler*>

Compiling and linking are performed with the command "<compiler_cmd> -fsanitize=address -g <current_dir>/*.c -o <current_file_name>". This will compile all .c files in the current directory into one program that carries the name of the active file. Any kind of error that comes up in stderr is returned to the Problems window.

7.4 Verbose Feedback: Check for <*compiler*> runtime errors/warnings

The same command as for the compile/link is executed followed by "timeout 30s ./<current_file_name> || echo $? 1>&2". The executable is being called and is allowed to run for only 30 seconds. This is to prevent infinite while loops or infinite recursion. The exit status of the timeout command determines whether the "echo $? 1>&2" command is being executed. If the timeout did not occur the echo command is not executed and the runtime errors are being parsed. If the timeout did occur the echo command is executed and the exit status 124 from the timeout is being returned. In case of exit status 124 no runtime errors are being parsed.

Any kind of error that comes up in stderr is considered as a runtime error. The client parses the error and returns the proper feedback by matching regular expressions to stderr. If no regular expression matches stderr, then there is no support for this runtime error or the error is actually a compile or linking error. Even though there is no feedback available the error will still be shown in the Problems window.

7.5 Verbose Feedback: Clear all errors by reloading the window

All the errors in the Problems window are being cleared by clearing the diagnostic buffer. VScode will then be reloaded by executing the builtin workbench.action.reloadWindow command.

8 Notes

- Errors and command containing the label 'C/C++' are related to the Microsoft C/C++ extension. Therefore much support can be found online, which is applicable in the VerboseFeedback extension.

- 'Possible runtime errors' always contain the error location [Ln 1, Col 1]. It is not sure if the error message is a true error or a print statement to stderr from the program, so no proper error location can be given. For the true error location the user should read it directly from the error message.

9 Development remarks

- Regular expressions often make use of special characters that require an escape character (i.e. \). When adding a regular expression that requires an escape character in the JSON file, use double escape character (i.e. \\)
- The severity levels of the VScode also consist of a 'hint' and 'warning' level. Implementing the feedback through these levels has been tried, but then diagnostics are shown in order of severity. This means that corresponding errors, hints and warnings are split up and not shown together as one problem case. Therefore there has been opted for solely a additional information message for the feedback.

10 Ideas for improvement

- The runtime errors are performed with the commands "<compiler_cmd> -fsanitize=address -g <current_dir>/*.c -o <current_file_name>" and "timeout 30s ./<current_file_name> || echo $? 1>&2". The first command includes all the .c files in the current directory for compilation, so it is important that within the current directory there is only one main() function. Furthermore, a program cannot be run with arguments as the command is hardcoded. The extension can be optimized by putting these commands in a `tasks.json` file, so that the user can decide which files should be included, what parameters should be used, and after how many seconds the timeout should occur.
- The feedback errors could also contain a link to a website, database or course content, so that the student knows where to find related theory and how common errors are being solved.
- Currently there is no feedback support implemented for the linking phase. This could be improved through the same method as the runtime errors.

References

[1] Joseph Benden. *C/C++ Advanced Lint for VS Code*. Retrieved May 27 2022. URL: https://github.com/jbenden/vscode-c-cpp-flylint.

[2] Xinyu Fu et al. "Real-Time Learning Analytics for C Programming Language Courses". In: *Proceedings of the Seventh International Learning Analytics & Knowledge Conference*. LAK '17. Vancouver, British Columbia, Canada: Association for Computing Machinery, 2017, pp. 280–288. ISBN: 9781450348706. DOI: 10.1145/3027385.3027407.

[3] Microsoft. *C/C++ for Visual Studio Code*. Retrieved May 27 2022. URL: https://marketplace.visualstudio.com/items?itemName=ms-vscode.cpptools.

[4] Microsoft. *Extension API*. Retrieved 14 May 2022. URL: https://code.visualstudio.com/api.

[5] Microsoft. *Language Server Protocol*. Retrieved 03 May 2022. URL: https://microsoft.github.io/language-server-protocol/.

Appendix

Type number	Error Description	Type number	Error Description
Type 1	Unmatched data type	Type 19	Missing semicolon before “return”
Type 2	Re-declaration of variables	Type 20	Missing semicolon
Type 3	Mismatch of “{” (particularly, “{” after main)	Type 21	“,” used after variables, not “.”
Type 4	Undeclared variables (particularly, mismatched symbols and mistyping of symbols)	Type 22	Missing semicolon or comma
Type 5	Syntax errors (invalid operand or invalid suffix)	Type 23	Mistyping of standard library
Type 6	Unmatched variable type for array	Type 24	Full-width characters are used
Type 7	Mistakes on array declaration	Type 25	Misuse of switch statement (with or without use of “break”)
Type 8	Misuse of pow function	Type 26	Undeclared variables, or “;” or “}” is missing in the previous row
Type 9	Misuse of mathematical functions	Type 27	Miss input symbol such as “,”,“<=”,or the mismatch of “[]”symbol
Type 10	Missing punctuation (e.g. “=” or “,” or “;” or “asm” or “__attribute__”)—semicolons were most frequently missing	Type 28	Errors in definition of variables or the boundary of the main function is wrong
Type 11	Other error type	Type 29	Unmatched symbols such as “()”, “{}”, “[]”
Type 12	Statement is out of main class	Type 30	Variables should define before the "for" loop
Type 13	Missing “}” at the end of code	Type 31	“ ()”,“{}”symbols are not matched in if-else statement
Type 14	Mismatch of “{}”	Type 32	Parameters' type is incorrect in the function
Type 15	Mismatch of quote marks, or mismatch of “<>”	Type 33	Array size should be defined as a constant
Type 16	Mismatch of “{}”: missing semicolons or comma before “{”	Type 34	Miss input variable's type
Type 17	Re-declaration of variable type	Type 35	Misuse of do-while statement
Type 18	Two main classes in one program	Type 36	Variable's type in return statement is unmatched desired variables

Figure 1: Common mistakes made by novice C programmers [2].