

Quickstart - Fortran Modernization with Himeno and compile_commands.json

1. This is a continuation of the Quickstart Guide to Codee, which helps to start using Codee integration with the build system (i.e. cmake, make, ninja, etc.) of your code. It works with the [performance-demos-fortran](#) github repository, more specifically with [Himeno/serial](#).

2. In order to generate the compile_commands.json file required by Codee, we need to first have a Makefile available. Himeno already ships with a Makefile, but we could also leverage the CMake configuration to generate it if needed:

```
$ cmake . -DCMAKE_Fortran_COMPILER=gfortran -DCMAKE_BUILD_TYPE=Release -G "Unix Makefiles" -B build
```

3. Use *make* with the *bear* tool (version 3 or later) for command invocation interception:

```
$ bear --output build/compile_commands.json -- make -C build
```

This command will produce the compile_commands.json file with all the compiler invocations needed to build the source files.

Note: although CMake can actually generate the compile_commands.json file by itself, for Fortran we strongly recommend using Bear (version 3.x or later) to properly manage file dependencies due to Fortran modules.

Example of a compile_commands.json generated with Bear:

```
[
{
  "arguments": [
    "/usr/bin/gfortran",
    "-O3",
    "-DNDEBUG",
    "-O3",
    "-c",
    "-o",
    "CMakeFiles/himeno.dir/himeno.f90.o",
    "/home/user/performance-demos-fortran/Himeno/serial/himeno.f90"
  ],
  "directory": "/home/user/performance-demos-fortran/Himeno/serial/build",
  "file": "/home/user/performance-demos-fortran/Himeno/serial/himeno.f90",
  "output":
"/home/user/performance-demos-fortran/Himeno/serial/build/CMakeFiles/himeno.dir/himeno.f90.o"
}
]
```

4. Produce the Screening Report of Codee using the compilation database (`--config`), noting that the compiler flags are obtained from the JSON file:

```
codee screening --check-id PWR001,PWR002,PWR003,PWR007,PWR008,PWR012,PWR063 --config
build/compile_commands.json
```

```
1 total entry detected
```

```
Configuration file 'build/compile_commands.json' successfully parsed.
Date: 2024-04-22 Codee version: 2024.2.2
```

```

[Fortran] target compiler: <none> (Compiler Agnostic Mode)

[1/1] /home/ulises/Appentra/repos/performance-demos-fortran/Himeno/serial/himeno.f90 ... Done

SCREENING REPORT

---Number of files---
Total | C C++ Fortran
-----|-----
1      | 0 0 1

Lines of code Analysis time # checks Profiling
-----
214      165 ms      6      n/a

CHECKS PER CATEGORY AND PRIORITY LEVELS

| -----Checks per category----- | Priority |
| Scalar Control Memory Vector Multi Offload Quality | L1 L2 L3 |
| -----|-----|-----|-----|
| n/a n/a n/a n/a n/a n/a 6 | 1 0 5 |

Lines of code : total lines of code found in the target (computed the same way as the sloccount tool)
Analysis time : time required to analyze the target
# checks : total actionable items (opportunities, recommendations, defects and remarks) detected
Profiling : estimation of overall execution time required by this target

RANKING OF CHECKERS

Checker Level Priority # Title
-----
PWR063 L1 P12 1 Avoid using legacy Fortran constructs
PWR001 L3 P3 5 Declare global variables as function parameters

SUGGESTIONS

Use 'checks' to find out details about the detected checks:
codee checks --check-id PWR001,PWR002,PWR003,PWR007,PWR008,PWR012,PWR063 --config
build/compile_commands.json

Use --target-arch to focus on the checks most relevant to your hardware type [cpu | gpu | mcu],
e.g.:
codee screening --target-arch cpu --check-id PWR001,PWR002,PWR003,PWR007,PWR008,PWR012,PWR063
--config build/compile_commands.json

1 file, 7 functions, 5 loops successfully analyzed and 0 non-analyzed files in 165 ms

```

- Similarly, produce the detailed Checkers Report of Codee using the compilation database:

```
codee checks --check-id PWR001,PWR002,PWR003,PWR007,PWR008,PWR012,PWR063 --verbose --config
build/compile_commands.json
```

```

1 total entry detected

Configuration file 'build/compile_commands.json' successfully parsed.
Date: 2024-04-22 Codee version: 2024.2.2
[Fortran] target compiler: <none> (Compiler Agnostic Mode)

[1/1] himeno.f90 ... Done

CHECKS REPORT

himeno.f90 [PWR063] (level: L1): Avoid using legacy Fortran constructs
PAUSE:
131: pause
Suggestion: Remove the legacy fortran constructs and refactor the code to comply with modern Fortran
standards.
Documentation: https://github.com/codee-com/open-catalog/tree/main/Checks/PWR063
. . .

1 file, 7 functions, 5 loops successfully analyzed and 0 non-analyzed files in 132 ms

```

From this point, the steps are the same as in the previous quickstart, basically, rewriting the code applying the suggestion made by the PWR063, and then running the checks report again to verify that the checker no longer appears.

Appendix

Example of a compile_commands.json generated with CMake:

```
[
{
  "directory": "/home/user/performance-demos-fortran/Himeno/serial/build",
  "command": "/usr/bin/gfortran -O3 -DNDEBUG -O3 -c
/home/user/performance-demos-fortran/Himeno/serial/himeno.f90 -o CMakeFiles/himeno.dir/himeno.f90.o",
  "file": "/home/user/performance-demos-fortran/Himeno/serial/himeno.f90"
}
]
```

Example of a compile_commands.json generated with Bear:

```
[
{
  "arguments": [
    "/usr/bin/gfortran",
    "-O3",
    "-DNDEBUG",
    "-O3",
    "-c",
    "-o",
    "CMakeFiles/himeno.dir/himeno.f90.o",
    "/home/user/performance-demos-fortran/Himeno/serial/himeno.f90"
  ],
  "directory": "/home/user/performance-demos-fortran/Himeno/serial/build",
  "file": "/home/user/performance-demos-fortran/Himeno/serial/himeno.f90",
  "output":
"/home/user/performance-demos-fortran/Himeno/serial/build/CMakeFiles/himeno.dir/himeno.f90.o"
}
]
```