VSCode Configuration and Tips for Flang Development

Anthony Cabrera
Research Scientist
Oak Ridge National Laboratory
January 4, 2023

ORNL is managed by UT-Battelle, LLC for the US Department of Energy

OAK RIDGE National Laboratory

Preliminaries

- You can find my slides at https://cabreraam.github.io/files/presentations/2023_01_05-hcbb_vscode_mlir_presentation.pdf
- You can find my configuration files at https://github.com/cabreraam/flang_vscode_pres_supplement
- If you have any suggestions, feedback, or errata, please email me at `cabreraam AT ornl DOT gov`

Overview

VSCode Configuration

Stepping Through Code

Case Study

Current Topic

VSCode Configuration
- VSCode Extensions
- Configuration Files

Stepping Through Code

Case Study

Current Topic

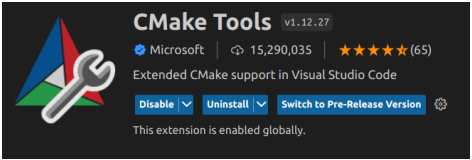
VSCode Configuration
- VSCode Extensions
- Configuration Files

Stepping Through Code

Case Study

Extensions to Download

- C/C++
- CMake Tools



Current Topic

VSCode Configuration

- VSCode Extensions
- Configuration Files
  - `CMakePresets.json` and `CMakeUserPresets.json`
  - `settings.json`
  - `launch.json`
  - `tasks.json`

Stepping Through Code

Case Study

`CMakePresets.json` and `CMakeUserPresets.json`

These files allow you to create multiple configurations for the configure, generate, and build steps of any CMake project.

`CMakePresets.json` and `CMakeUserPresets.json`

These files allow you to create multiple configurations for the configure, generate, and build steps of any CMake project.

- Place these files in the `llvm` directory

`CMakePresets.json` and `CMakeUserPresets.json`

These files allow you to create multiple configurations for the configure, generate, and build steps of any CMake project.

- Place these files in the `llvm` directory
- `CMakePresets.json` are for common configurations, and `CMakeUserPresets.json` are on a per-user basis

`CMakePresets.json` and `CMakeUserPresets.json`

These files allow you to create multiple configurations for the configure, generate, and build steps of any CMake project.

- Place these files in the `llvm` directory
- `CMakePresets.json` are for common configurations, and `CMakeUserPresets.json` are on a per-user basis
  - Specifically, in the Flang project, the `CMakePresets.json` config can contain the default build configuration from the Flang `README.md`, and the `CMakeUserPresets.json` can contain your personal system-specific details for building on your machine

`CMakePresets.json` and `CMakeUserPresets.json`

These files allow you to create multiple configurations for the configure, generate, and build steps of any CMake project.

- Place these files in the `llvm` directory
- `CMakePresets.json` are for common configurations, and `CMakeUserPresets.json` are on a per-user basis
  - Specifically, in the Flang project, the `CMakePresets.json` config can contain the default build configuration from the Flang `README.md`, and the `CMakeUserPresets.json` can contain your personal system-specific details for building on your machine

Use these config files over `cmake-kits.json` and `cmake-variants.json`

The presets files are cross-plaform; you can share your `CMakePresets.json` and `CMakeUserPresets.json` files with someone who does not use VSCode as their IDE, since the presets files are a ‘CMake‘ feature and not a VSCode feature. Also, this is now the recommended method from Microsoft developers.

CMakePresets.json Example

```
1   {
2     "version": 5,
3     "cmakeMinimumRequired": {
4       "major": 3,
5       "minor": 23,
6       "patch": 0
7     },
8     "include": [],
9     "configurePresets": [
10      {
11        "name": "Flang Default Configure",
12        "displayName": "Flang Default Configure",
13        "description": "Flang Default configure recipe given from Flang docs",
14        "generator": "Ninja",
15        "binaryDir": "${sourceDir}/../build_flang_default",
16        "cacheVariables": {
17          "CMAKE_INSTALL_PREFIX": "${sourceDir}/../install_flang_default",
18          "CMAKE_CXX_STANDARD": "17",
19          "CMAKE_BUILD_TYPE": "Release",
20          "CMAKE_EXPORT_COMPILE_COMMANDS": "ON",
21          "CMAKE_CXX_LINK_FLAGS": "-Wl,-rpath,$LD_LIBRARY_PATH",
22          "FLANG_ENABLE_WERROR": "ON",
23          "LLVM_ENABLE_ASSERTIONS": "ON",
24          "LLVM_TARGETS_TO_BUILD": "host",
25          "LLVM_LIT_ARGS": "-v",
26          "LLVM_ENABLE_PROJECTS": "clang;mlir;flang;openmp",
27          "LLVM_ENABLE_RUNTIMES": "compiler-rt"
28        }
29      }
30    ],
31  > "buildPresets": [ …
36    ],
37  > "testPresets": [ …
49    ]
50  }
```

CMakeUserPresets.json Example

llvm > {} CMakeUserPresets.json > [ ] testPresets

```
1   {
2     "version": 5,
3     "cmakeMinimumRequired": {
4       "major": 3,
5       "minor": 23,
6       "patch": 0
7     },
8     "include": [
9     ],
10    "configurePresets": [
11 >    {…
30      },
31 >    {…
49      },
50      {
51        "name": "default_local_relwithdebuginfo_with_gcc",
52        "inherits": [
53          "Flang Default Configure"
54        ],
55        "binaryDir": "${sourceDir}/../build_relwithdebuginfo_gcc",
56        "displayName": "local Default relwithdebuginfo Config with gcc",
57        "description": "local Default relwithdebuginfo build using Ninja generator",
58        "environment": {
59        },
60        "cacheVariables": {
61          "CMAKE_C_COMPILER": "gcc",
62          "CMAKE_CXX_COMPILER": "g++",
63          "CMAKE_BUILD_TYPE": "RelWithDebInfo"
64        }
65      }
66    ],
67    "buildPresets": [
```

```
67    "buildPresets": [
68 >    {…
74      },
75 >    {…
81      },
82      {
83        "name": "build_from_user_presets_gcc_relwithdebuginfo",
84        "configurePreset": "default_local_relwithdebuginfo_with_gcc",
85        "displayName": "Build from user presets gcc relwithdebuginfo",
86        "description": "local Default build using Ninja generator",
87        "jobs": 128
88      }
89    ],
90    "testPresets": [
91    ]
92  }
```

settings.json

This config file configures your project in the current VSCode workspace.

`settings.json`

This config file configures your project in the current VSCode workspace.

- Place this file in the `.vscode` directory

`settings.json`

This config file configures your project in the current VSCode workspace.

- Place this file in the `.vscode` directory
- Specifically in LLVM and Flang, we need to associate `.inc` files `c++` files

`settings.json`

This config file configures your project in the current VSCode workspace.

- Place this file in the `.vscode` directory
- Specifically in LLVM and Flang, we need to associate `.inc` files `c++` files
- You can't step through auto-generated `.inc` files unless you make this association

settings.json

This config file configures your project in the current VSCode workspace.

- Place this file in the `.vscode` directory
- Specifically in LLVM and Flang, we need to associate `.inc` files `c++` files
- You can't step through auto-generated `.inc` files unless you make this association
- Example `settings.json` file:

.vscode > {} settings.json > ...

```
1  {
2    "files.associations": {
3      "*.inc": "cpp",
4      "*.tcc": "cpp",
5      "string": "cpp",
6      "regex": "cpp"
7    },
8    "cmake.sourceDirectory": "${workspaceFolder}/llvm",
9    "cmake.parallelJobs": 16,
10   "git.ignoreLimitWarning": true,
11 }
```

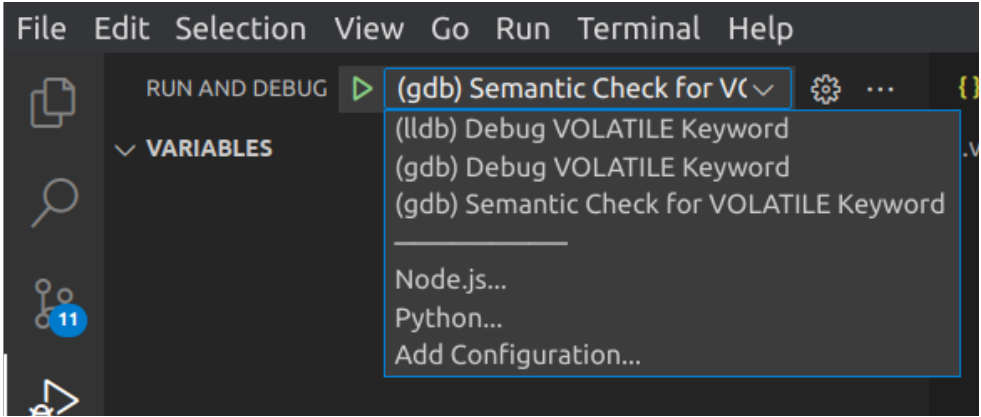
launch.json

A `launch.json` file is used to configure the debugger in VSCode

launch.json

A `launch.json` file is used to configure the debugger in VSCode

- This is useful for quickly selecting between debugging tasks



launch.json example

.vscode > {} launch.json > Launch Targets > {} LD_LIBRARY_PATH

```
1   {
2     "version": "0.2.0",
3     "configurations": [
4 >     { …
16      },
17 >    { …
57      },
58      {
59        "name": "(gdb) Semantic Check for VOLATILE Keyword",
60        "type": "cppdbg",
61        "request": "launch",
62        "program": "${workspaceFolder}/build_debug_gcc/bin/flang-new",
63        "args": [
64          "-fc1",
65          "/noback/93u/Sandbox/issue_58973_volatile_dummy_arg/snem0601_012_.f90"
66        ],
67        "stopAtEntry": false,
68        "cwd": "${workspaceFolder}",
69        "environment": [
70          {
71            "name": "LD_LIBRARY_PATH",
72            "value": "${env:LD_LIBRARY_PATH}:/auto/software/swtree/ubuntu20.04/x86_64/gcc/12.1.0/lib64"
73          }
74        ],
75        "externalConsole": false,
76        "MIMode": "gdb",
77        "miDebuggerPath": "/noback/93u/spack/opt/spack/linux-ubuntu20.04-zen2/gcc-12.1.0/gdb-12.1-6ajvwcxw666sgncms2zimnnomn7cwmsw/bin/gdb",
78 >      "setupCommands": [ …
95        ],
96        "logging": {"engineLogging": false}
97      }
98    ]
99  }
```

tasks.json

This config file allows you to create tasks with external tools, e.g., through shell commands, that are easily navigatable and can be executed within the VSCode environment

tasks.json

This config file allows you to create tasks with external tools, e.g., through shell commands, that are easily navigatable and can be executed within the VSCode environment

- For example, this is helpful for *quickly* compiling a given test to check if LLVMFlang behaved as expected or not

tasks.json - llvm-project [SSH: secretariat] - Visual Studio Code

File Edit Selection View Go Run Terminal Help

EXPLORER

OPEN EDITORS
- tasks.json .vscode

LLVM-PROJECT [SSH: SECRETARIAT]
- .github
- .vscode
  - launch.json
  - settings.json
  - tasks.json
- bolt
- build
- build_debug
- build_debug_gcc
- build_fujitsu
- build_relwithdebuginfo_gcc
- clang
- clang-tools-extra
- cmake

OUTLINE
- version 2.0.0
- [ ] tasks
  - { } 0
    - type shell
    - label build_and_run
    - command #bash build_and_run.sh
  - { } 1
    - type shell
    - label Anthony's Semantic Check with test_errors.py
    - command module load gcc/12.1.0 && python /noback/93u...
    - [ ] dependsOn []
    - dependsOrder sequence
  - { } 2
    - type shell
    - label load LLVM 14.0.0
    - command module load llvm/14.0.0
  - { } 3
    - type shell
    - label Add llvm build_debug/bin binary dir to path
    - command export PATH=/noback/93u/Research/secretaria...
  - { } 4
    - label Run Before debugging build_debug
    - [ ] dependsOn
      - 0
      - 1
  - { } 5
    - type shell

TIMELINE

tasks.json

.vscode > tasks.json > [ ] tasks > { } 1 > [

```
1  {
2    "version": "2.0.0",
3    "tasks": [
4      {
5        "type": "shell",
6        "label": "build_and
7        "command": "#bash b
8      },
9      {
10       "type": "shell",
11       "label": "Anthony's
12       "command": "module                                  /flang/test/Semantics/test_errors.py /noback/93u/Research/secretariat/
         llvm-project/flang/                                 /build_debug_gcc/bin/flang-new -fc1 -Werror",
13       "dependsOn": [
14       ],
15       "dependsOrder": "se
16     },
17 >   {…
21     },
22 >   {…
26     },
```

Select the task to run

- Anthony's Semantic Check with test_errors.py — recently used
- Anthony's Semantic Check
- Compile volatile example with -fc1, build_debug_gcc
- echo User
- load GCC 12.1.0
- Run Before debugging build_debug
- Add llvm build_debug/bin binary dir to path — configured
- build_and_run
- Compile volatile example with -fc1, build_fujitsu
- Compile volatile example with -flang-experiental-exec, build_debug_gcc
- load LLVM 14.0.0
- grunt — contributed
- gulp
- jake
- npm
- typescript
- docker-build
- docker-run

PROBLEMS OUTPUT TERMINAL PORTS DEBUG CONSOLE

```
Executing task: module load gcc/12.1.0 && python /noback/93u/Research/secretariat/llvm-project/flang/test/Semantics/test_errors.py /noback/93u/Research/secretariat/llvm-projec
t/flang/test/Semantics/call30.f90 /noback/93u/Research/secretariat/llvm-project/build_debug_gcc/bin/flang-new -fc1 -Werror

PASS
Terminal will be reused by tasks, press any key to close it.
```

bash
cppdbg: flan...
Anthony's...

SSH: secretariat | main* | 0 0 | 0 | (gdb) Semantic Check for VOLATILE Keyword (llvm-project) | Live Share | [local Default relwithdebuginfo Config with gcc] | Build | [Build from user presets gcc relwithdebuginfo] | [check-flang] | No Test Preset Selected | -- NORMAL -- | JSON with Comments | Linu

tasks.json - llvm-project [SSH: secretariat] - Visual Studio Code

File Edit Selection View Go Run Terminal Help

EXPLORER

OPEN EDITORS
- tasks.json .vscode

LLVM-PROJECT [SSH: SECRETARIAT]
- .github
- .vscode
  - launch.json
  - settings.json
  - tasks.json
- bolt
- build
- build_debug
- build_debug_gcc
- build_fujitsu
- build_relwithdebuginfo_gcc
- clang
- clang-tools-extra
- cmake

OUTLINE
- version 2.0.0
- [ ] tasks
  - { } 0
    - type shell
    - label build_and_run
    - command #bash build_and_run.sh
  - { } 1
    - type shell
    - label Anthony's Semantic Check with test_errors.py
    - command module load gcc/12.1.0 && python /noback/93u...
    - [ ] dependsOn []
    - dependsOrder sequence
  - { } 2
    - type shell
    - label load LLVM 14.0.0
    - command module load llvm/14.0.0
  - { } 3
    - type shell
    - label Add llvm build_debug/bin binary dir to path
    - command export PATH=/noback/93u/Research/secretaria...
  - { } 4
    - label Run Before debugging build_debug
    - [ ] dependsOn
      - 0
      - 1
  - { } 5
    - type shell

TIMELINE

tasks.json

.vscode > tasks.json > [ ] tasks > { } 1 > label

```
1  {
2    "version": "2.0.0",
3    "tasks": [
4      {
5        "type": "shell",
6        "label": "build_and_run",
7        "command": "#bash build_and_run.sh"
8      },
9      {
10       "type": "shell",
11       "label": "Anthony's Semantic Check with test_errors.py",
12       "command": "module load gcc/12.1.0 && python /noback/93u/Research/secretariat/llvm-project/flang/test/Semantics/test_errors.py /noback/93u/Research/secretariat/
         llvm-project/flang/test/Semantics/call30.f90 /noback/93u/Research/secretariat/llvm-project/build_debug_gcc/bin/flang-new -fc1 -Werror",
13       "dependsOn": [
14       ],
15       "dependsOrder": "sequence"
16     },
17 >   {...
21     },
22 >   {...
26     },
```

PROBLEMS OUTPUT TERMINAL PORTS DEBUG CONSOLE

```
✓ Executing task: module load gcc/12.1.0 && python /noback/93u/Research/secretariat/llvm-project/flang/test/Semantics/test_errors.py /noback/93u/Research/secretariat/llvm-projec
t/flang/test/Semantics/call30.f90 /noback/93u/Research/secretariat/llvm-project/build_debug_gcc/bin/flang-new -fc1 -Werror

PASS
✓ Terminal will be reused by tasks, press any key to close it.
```

bash
cppdbg: flan...
Anthony's... ✓

SSH: secretariat | main* | ⊗ 0 ⚠ 0 | W 0 | (gdb) Semantic Check for VOLATILE Keyword (llvm-project) | Live Share | [local Default relwithdebuginfo Config with gcc] | Build | [Build from user presets gcc relwithdebuginfo] | [check-flang] | No Test Preset Selected | -- NORMAL -- | JSON with Comments | Linux

Current Topic

VSCode Configuration

Stepping Through Code

Case Study

Using GDB to Step Through Code

Though it's somewhat slow to step through LLVM using a debugger, the tradeoff is that you can *interactively* see what the code is doing. This is particularly useful when you're still learning the codebase (like me!)

Using GDB to Step Through Code

Though it's somewhat slow to step through LLVM using a debugger, the tradeoff is that you can *interactively* see what the code is doing. This is particularly useful when you're still learning the codebase (like me!)

- You can examine the state of all variables
- You can jump through all currently active call stacks
- You can easily see where certain routines/variables are defined

RUN AND DEBUG (gdb) Semantic Check for VC

VARIABLES

```
dummyIsOptional: false
actualIsNull: false
> dummy: {...}
v dummyName: {...}
    npos: 18446744073709551615
  v _M_dataplus
    > std::allocator<char> (base): std::allocator<char>
    v _M_p: 0xf367640 "dummy argument 'n6='"
      *_M_p: 100 'd'
    _M_string_length: 20
```

WATCH

CALL STACK — Paused on breakpoint

```
Fortran::semantics::CheckExplicitDataArg(const Fortran::
operator()(const struct {...} * const __closure, const F
Fortran::common::log2visit::Log2VisitHelper<0, 0, void,
Fortran::common::log2visit::Log2VisitHelper<0, 1, void,
Fortran::common::log2visit::Log2VisitHelper<0, 2, void,
Fortran::common::log2visit::visit<Fortran::common::visit
Fortran::semantics::CheckExplicitInterfaceArg(Fortran::e
```

BREAKPOINTS

- All C++ Exceptions
- check-call.cpp flang/lib/Semantics 108
- check-call.cpp flang/lib/Semantics 117
- check-call.cpp flang/lib/Semantics 148
- check-call.cpp flang/lib/Semantics 182
- check-call.cpp flang/lib/Semantics 229
- check-call.cpp flang/lib/Semantics 239
- check-call.cpp flang/lib/Semantics 423
- CompilerInstance.cpp flang/lib/Frontend 166
- driver.cpp flang/tools/flang-driver 66
- driver.cpp flang/tools/flang-driver 109
- ExecuteCompilerInvocation.cpp flang/lib/Fronten... 103

check-call.cpp 9+

flang > lib > Semantics > check-call.cpp > { } Fortran > { } semantics > CheckExplicitDataArg(const characteristics::DummyDataObject &, const std::string &, evaluate::Exp...

```
403     // Problems with polymorphism are caught in the callee's definition.
404     DefinabilityFlags flags{DefinabilityFlag::PolymorphicOkInPure};
405     if (isElemental || dummyIsValue) { // 15.5.2.4(21)
406       flags.set(DefinabilityFlag::VectorSubscriptIsOk);
407     }
408     if (actualIsPointer && dummyIsPointer) { // 19.6.8
409       flags.set(DefinabilityFlag::PointerDefinition);
410     }
411     if (auto whyNot{WhyNotDefinable(messages.at(), *scope, flags, actual)}) {
412       if (auto *msg{messages.Say(
413             "Actual argument associated with %s %s is not definable"_err_en_US,
414             reason, dummyName)}) {
415         msg->Attach(std::move(*whyNot));
416       }
417     }
418   }
419
420   // technically legal but worth emitting a warning
421   // llvm-project issue #58973: constant actual argument passed in where dummy
422   // argument is marked volatile
423   if (dummyIsVolatile && !IsVariable(actual)) {
424     messages.Say(
425         "actual argument associated with VOLATILE %s is not a variable"_warn_en_US,
426         dummyName);
427   }
428
429   // Cases when temporaries might be needed but must not be permitted.
430   bool actualIsContiguous{IsSimplyContiguous(actual, context)};
431   bool dummyIsAssumedShape{dummy.type.attrs().test(
432       characteristics::TypeAndShape::Attr::AssumedShape)};
433   bool dummyIsContiguous{
434       dummy.attrs.test(characteristics::DummyDataObject::Attr::Contiguous)};
435   if ((actualIsAsynchronous || actualIsVolatile) &&
436       (dummyIsAsynchronous || dummyIsVolatile) && !dummyIsValue) {
437     if (actualIsCoindexed) { // C1538
438       messages.Say(
439           "Coindexed ASYNCHRONOUS or VOLATILE actual argument may not be associated with %s with ASYNCHRONOUS or VOLATILE
              attributes unless VALUE"_err_en_US,
              dummyName);
440     }
441     if (actualRank > 0 && !actualIsContiguous) {
442       if (dummyIsContiguous ||
443           !(dummyIsAssumedShape || dummyIsAssumedRank ||
444               (actualIsPointer && dummyIsPointer))) { // C1539 & C1540
445         messages.Say(
446             "ASYNCHRONOUS or VOLATILE actual argument that is not simply contiguous may not be associated with a contiguous
                dummy"_err_en_US,
```

SSH: secretariat | main* | 12 ⚠ 0 | 0 | (gdb) Semantic Check for VOLATILE Keyword (llvm-project) | Live Share | [local Default relwithdebuginfo Config with gcc] | Build | [Build from user presets gcc relwithdebuginfo] | [[Targets In Preset]] | No Test Preset

Current Topic

VSCode Configuration

Stepping Through Code

Case Study

- `llvm-project` issue 58973

Current Topic

VSCode Configuration

Stepping Through Code

Case Study

- `llvm-project` issue 58973

llvm-project issue 58973

- Here is an issue that is raising an error when a warning is more appropriate
  - A little more specifically, an attribute of a dummy argument does not correspond to the actual argument being passed
- Besides knowing that I want to step through the code using `gdb`, I have no idea where to start, so I of course ask the Flang community for help
- After a tip, I find that `flang/lib/Semantics/check-call.cpp` might be where the fix should be applied
- Before going further, I start configuring VSCode + CMake
  - I set up `CMakeUserPresets.json` with the default Flang configuration, and set up `CMakeUserUserPresets.json` with details specific to my system as well as setting the `RelWithDebInfo` build option

llvm-project issue 58973 (cont.)

- I visually scan the `flang/lib/Semantics/check-call.cpp` and start setting breakpoints in places where I think might be useful
- I start implementing what I think the fix should be
- Once I have something I think works, it's time to build LLVM using the CMake extension integrated into our VSCode environment
- At this point, I can start editing the `launch.json` and `tasks.json` config files
  - The commands in `launch.json` and `tasks.json` will essentially be the same except `launch.json` contains config information for the debugger, and the `tasks.json` file will configure shell command that allows for a quick check for whether we see what we expect or not
- Once I'm happy with what I have so far, I start soliciting reviews on Phabricator
- After an iterative process, the fix was deemed acceptable but contingent on writing some tests in the `flang/test` directory

VSCode Configuration and Tips for Flang Development

Anthony Cabrera

Research Scientist
Oak Ridge National Laboratory
January 4, 2023

ORNL is managed by UT-Battelle, LLC for the US Department of Energy