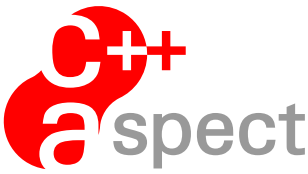

Documentation:
AspectC++ First Steps

The AspectC++ Developers

Version 2.4

February 4, 2025

Contents

1 Getting Started

Let's get in touch with AspectC++. This document provides a step-by-step introduction to:

- Installing the AspectC++ compiler on Linux, macOS, and Windows (Section 2)
- Writing a program that traces function execution (Section 3)
- Using the CMake build system to manage AspectC++ projects (Section 4)
- Generating binary code for different processor architectures with a cross compiler (Section 5)

2 Installation

The first step is to install AspectC++. We recommend downloading the latest release from https://www.aspectc.org in the Download section.

2.1 Installing AspectC++ on Linux

Note: If you are using Debian, Ubuntu, or derivatives, you can install AspectC++ readily from the distribution's repository by typing the following command in a terminal:

```
sudo apt install aspectc++
```

That's all!

If you use a non-Debian/Ubuntu Linux distribution or want to install the most recent version of AspectC++, you can choose between a 32-bit (Linux/x86) and 64-bit (Linux/x86_64) version. Open a terminal and enter the following command:

```
uname -m
```

If the output is `x86_64`, download the 64-bit version (Linux/x86_64) from https://www.aspectc.org. Otherwise, download the 32-bit version (Linux/x86). Open a terminal, change to the download directory and unpack the downloaded file:

```
tar xzvf ac-bin-linux-*.tar.gz
```

The unpacked contents are stored in the folder `aspect++/`. The next step is to add that folder to your `PATH` system variable. Assuming you have downloaded AspectC++ to a folder named `Downloads/` in your home directory (`$HOME`), just type the following command in the terminal:

```
export PATH=$HOME/Downloads/aspectc++/:$PATH
```

Add the line from above to `$HOME/.bashrc` or the like to set this path permanently.

Linux users should also install the GNU C++ compiler (g++). If you are using Debian, Ubuntu, or derivatives, you can install the `build-essential` package from the distribution's repository.

2.2 Installing AspectC++ on macOS

On macOS, go to https://www.aspectc.org and download the respective release file. Open a terminal, change to the download directory and unpack the downloaded file:

```
tar xzvf ac-bin-macosx-*.tar.gz
```

The unpacked contents are stored in the folder `aspect++/`. The next step is to add that folder to your `PATH` system variable. Assuming you have downloaded AspectC++ to a folder named `Downloads/` in your home directory (`$HOME`), just type the following command in the terminal:

```
export PATH=$HOME/Downloads/aspectc++/:$PATH
```

Add the line from above to `$HOME/.zshrc` or the like to set this path permanently.

On macOS, you should also install the Clang C++ (`clang++`) compiler by using the following command:

```
xcode-select --install
```

2.3 Installing AspectC++ on Windows

On Windows, go to https://www.aspectc.org and download the respective release file. Open the file manager, change to the download directory and unpack the downloaded zip file. The unpacked contents are stored in the folder `aspect++\`. The next step is to add that folder to your `PATH` system variable:

1. Open the Windows start menu (Start) and search for **`environment variables`** and open it.
2. Click on the button “Environment Variables”.
3. In the section “User variables” select the item **`Path`** and click on the “Edit” button.
4. Click on the “New” button. Assuming you have downloaded AspectC++ to a folder named `Downloads\` in your home directory on drive `C:\`, enter the following path: `C:\Users\<Your Name>\Downloads\aspectc++`

5. Click on the “OK” button of every opened window.

Windows users should also install MSYS2 and the GNU C/C++ compiler (GCC). Go to https://www.msys2.org/ and follow the instructions for installing MSYS2. In particular, take care of installing GCC using *pacman* as described there. You should add the installation path of that GCC to your `PATH` system variable:

1. Open the Windows start menu (Start) and search for **`environment variables`** and open it.
2. Click on the button “Environment Variables”.
3. In the section “System variables” select the item **`Path`** and click on the “Edit” button.
4. Click on the “New” button. Assuming you have installed MSYS2 to `C:\msys64\`, enter the following path: `C:\msys64\ucrt64\bin`
5. Click on the “OK” button of every opened window.

2.4 Troubleshooting

To check your AspectC++ installation, open a new terminal (Linux and macOS) or PowerShell (Windows) and enter the following command:

```
ac++ --version && ag++ --version
```

The output should show the version numbers and build dates for your installed AspectC++ release, in the following format:

```
ac++ x.y (<date>, clang a.b.c)
ag++ u.w  built: <date>
```

If you see these numbers and dates, you have installed AspectC++ successfully! Otherwise, check whether AspectC++ is in your `PATH` system variable.
On Linux and macOS, examine the output of the following command:

```
echo $PATH
```

On Windows using PowerShell, examine the output of:

```
echo $env:Path
```

3 Tracing Aspect

After installing AspectC++, it is time to write your first AspectC++ program. We will start by making a directory to store AspectC++ code. We propose to create a new directory called `tracing_aspectc++` in your home directory. Open a new terminal on Linux, macOS, or Windows (using PowerShell) and enter the following commands:

```
mkdir $HOME/tracing_aspectc++
cd $HOME/tracing_aspectc++
```

Inside this directory, create a new source file and call it `factorial.cpp`. Open that file and enter the following C++ code:

```
#include <iostream>

unsigned int factorial(unsigned int n) {
   return (n > 1) ?  n * factorial(n - 1) : 1;
}

int main() {
    std::cout << factorial(3) << " == 3!" << std::endl;
    return 0;
}
```

Next, make a new aspect header file and call it `tracing.ah`. By convention, AspectC++ header files always end with the `.ah` extension. Open the `tracing.ah` file with your text editor and enter the following AspectC++ code:

```
#ifndef __TRACING_AH__
#define __TRACING_AH__

#include <iostream>
#include <string>

aspect Tracing {
    unsigned int level = 0;

    advice execution("% ...::%(...)") : before() {
        std::cout << std::string(level++, ' ') << "> "
            << JoinPoint::signature() << std::endl;
    }

    advice execution("% ...::%(...)") : after() {
        level--;
    }
};

#endif
```

Save both files and go back to your terminal in the `$HOME/tracing_aspectc++` directory. Enter the following command to compile both files to an executable program:

```
ag++ factorial.cpp
```

To run the program on Linux or macOS, type the following command in the terminal:

```
./a.out
```

On Windows, enter:

```
.\a.exe
```

Now, you should see the output of the program:

```
> int main()
 > unsigned int factorial(unsigned int)
  > unsigned int factorial(unsigned int)
   > unsigned int factorial(unsigned int)
6 == 3!
```

The file `factorial.cpp` just computes the factorial of three and prints it on the final line (6 == 3!). The aspect header file `tracing.ah` makes the program to trace the C++ function execution and, therefore, prints the first four lines (beginning with `>`). That is, the function `main()` runs first, and then the function `factorial(unsigned int)` gets called three times recursively. In short, the AspectC++ expression `advice execution("% ...::%(...)")` captures the *execution* of any function by the wildcard expression `"%"` for return type and function name, and the wildcard expression `"..."` for any sequence of scopes and function arguments. The aspect file generates the tracing output eventually by AspectC++'s built-in function `JoinPoint::signature()`, which returns a formatted string per captured function that describes the function's return type, its name, and argument types.

3.1 Project Directory Tree

By default, the AspectC++ compiler uses the current working directory `./` as *project directory tree*. That is, the AspectC++ compiler searches recursively there for aspect header files and applies the advice to all C++ source and header files in that directory. The *project directory tree* can also be specified on the command line explicitly. In our example, the following two commands are equivalent:

```
ag++ factorial.cpp
ag++ -p $HOME/tracing_aspectc++ factorial.cpp
```

This means, that the compiler applies all pieces of advice to source and header files in the `$HOME/tracing_aspectc++` directory only. In particular, the compiler applies no advice to functions of the C/C++ standard library, because the standard library files are typically stored elsewhere. Therefore, the tracing aspect in our example does not print the `operator <<` function on the `std::cout` object of the standard library.

3.2 Aspect Header Files

As explained in the previous section, the AspectC++ compiler by default searches recursively for aspect header files in the *project directory tree*. You can alternatively specify the aspect header files explicitly on the command line with the `-a` option. In our example, the following two commands are equivalent:

```
ag++ -p $HOME/tracing_aspectc++ factorial.cpp
ag++ -p $HOME/tracing_aspectc++ -a tracing.ah factorial.cpp
```

You can use the `-a` option multiple times if there are multiple aspect files to be considered. You can use `-a 0` for not considering any aspect file at all.

4 Managing Dependencies with CMake

The CMake build system is a tool to manage growing software projects. You can use CMake to invoke the AspectC++ compiler and to manage include dependencies automatically. Install CMake as described in the download section of https://cmake.org.

Assuming you have stored the two files `factorial.cpp` and `tracing.ah` in the directory `$HOME/tracing_aspectc++` as explained in the previous section, you can manage that directory with CMake by creating a new text file there and name it `CMakeLists.txt`. Open the `CMakeLists.txt` file with your text editor and enter the following CMake commands:

```
cmake_minimum_required (VERSION 3.20)
project ("AspectC++ Tracing Example")

set(CMAKE_CXX_COMPILER "ag++")
set(CMAKE_CXX_FLAGS "--Xweaver \
-p $ENV{HOME}/tracing_aspectc++ --Xcompiler")

file(GLOB cxx_files "*.cpp")

add_executable(factorial ${cxx_files})
```

Save the file and open a terminal (Linux and macOS) or PowerShell (Windows). Create a new out-of-source build directory:

```
mkdir $HOME/tracing_aspectc++/build
cd $HOME/tracing_aspectc++/build
```

The next step is to run `cmake` in your terminal to generate the build system:

```
cmake ..
```

Then, you can use it to compile and link the executable file `factorial`.

```
cmake --build .
```

CMake takes care of managing include dependencies. That is, when a source, header, or aspect header files changes, you can use the previous command to rebuild only the affected set of files. AspectC++ currently considers every aspect header file as include dependency. That is, changes to one aspect header file cause every source file to be rebuilt. Technically, AspectC++ generates the dependency information just like `g++` with the `-MMD` and `-MD` command line options, respectively.

5 Cross Compilation

The AspectC++ compiler comes with two tools: `ag++` and `ac++`. The `ac++` tool implements a source-to-source compiler that translates AspectC++ source code to plain C++ source code. Thus, `ac++` does not directly generate the executable. You have to use a C++ compiler, such as the GNU C++ compiler (`g++`). For that reason, there is the `ag++` tool, which implements a wrapper for both `g++` and `ac++`. That is, `ag++` invokes `ac++` first to carry out the source-to-source transformation to a temporary file, and then invokes `g++` to generate the executable. Because of that, you typically should *not* invoke `ac++` manually and should use `ag++` instead.

By default, `ag++` invokes the host `g++` compiler to generate the executable. However, you can also make `ag++` use a *cross compiler*. You can compile the example program from Section 3, for instance, to ARM microcontrollers using the GNU ARM embedded toolchain with the following command:

```
ag++ --c_compiler arm-none-eabi-g++ \
--Xcompiler -specs=nosys.specs factorial.cpp
```

The `ag++` option `--c_compiler` specifies that the `arm-none-eabi-g++` compiler (if installed) shall be used to generate the executable. In addition, the above command uses the `--Xcompiler` option to tell `ag++` that every subsequent command line option shall be passed to the C++ (cross) compiler only. The other way around, any command line option after `--Xweaver` will passed to the ac++ tool only. You should pass special options to the cross compiler, such as `-specs=nosys.specs` in our example, by placing them after the `--Xcompiler` option.