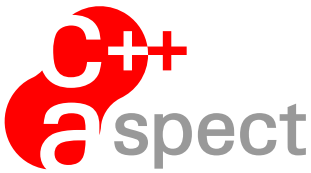

Documentation:
Ag++ Manual

The AspectC++ Developers

Version 2.5

December 22, 2025

Contents

1 Introduction

The `ag++` program provides a more intuitive frontend to the AspectC++ weaver (`ac++`) in a GNU environment. The only prerequisites are a working installation of GNU C++ compiler, which also can run within a cygwin environment. It basically wraps the functionality of the aspect weaver and the c++ compiler into one single program.

1.1 Weaving and Compiling in ag++

`ag++` calls `ac++` for each file in the parameter list. `ac++` then weaves the .cpp, .h and .ah files into a temporary .acc file. The .acc file contains all aspect headers and source code combined in regular C++ code. `g++` (or when specified, another back-end compiler) then compiles the .acc file into .o object files or an .elf executable. The generated .acc file can be saved and used for debugging using `--keep_woven` or `--weave_only` parameters (see Section 3.1). The whole translation process is illustrated in Figure 1.

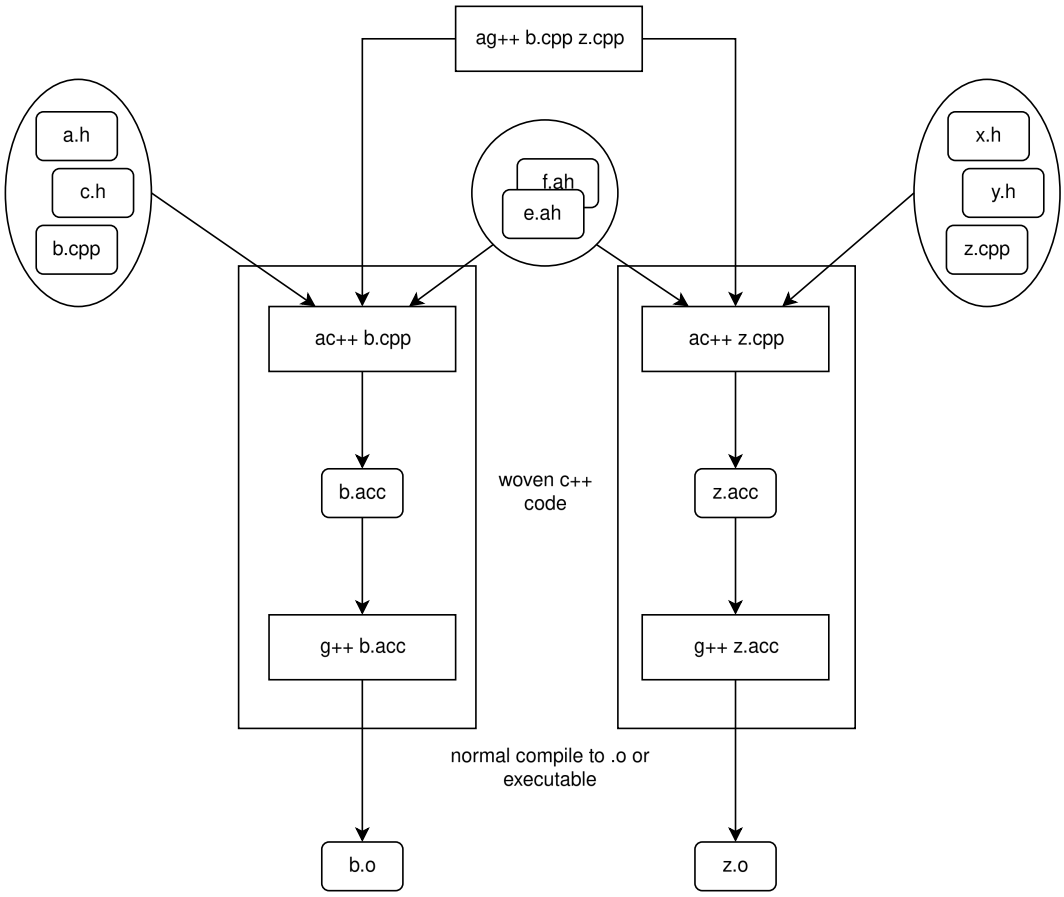

Figure 1: Weaving and Compiling in ag++

2 Invocation

The usage of ag++ is mainly influenced by the usage of the GNU g++ compiler and the synopsis is like:

```
ag++ [options] [files...]
```

Let's say, you want to **compile** a single file (here: main.cc) with g++, you have to run

```
g++ -c main.cc
```

in order to generate an object file.

To **weave** and **compile** a single file you simply invoke

```
ag++ -c main.cc
```

the same way like you did before with g++.

2.1 Options

All available options are summed up in the options table (see table 1). The column labeled with *AC++* shows if the option is taken over from ac++ by ag++ ('X'), not supported by ac++ ('–') or modified by ag++ ('!'). All options which are taken over, are not described in this document. Consult the AC++ Compiler Manual instead. Options which are not listed in the option table are accounted as g++ options. Some g++ options can not be automatically handled correctly by the options parser of ag++. So all g++ options starting with -p, -a, -d and -r (e.g. -pipe, -ansi, -dletters ,-remap) have to be written between --Xcompiler (see 2.1.12) and --Xweaver (see 2.1.13). If such options passed to ag++ without using --Xcompiler they will be interpreted a ag++/ac++ options ; e.g. -pipe will be interpreted as -p "ipe".

2.1.1 --gen_config

Just create a parser configuration and quit afterwards. The argument of the -o option specifies the name of the file. In any other case (no --gen_config and/or no -o option) a configuration file with the name 'puma.config' will be generated in the directory where ag++ was invoked.

2.1.2 --weave_only

Generate only woven source code files. With -o option and one file the generated output is named after the argument of the -o option.

Option	AC++	Description
--gen_config	–	Only generate Puma configuration file
--weave_only	–	Weave only
-c	!	Compile only
--keep_woven	–	Keep woven source code files
--c_compiler <arg>	–	Path to C++ compiler
--ac_compiler <arg>	–	Path to AspectC++ compiler
--config_command <arg>	–	Specify command which prints information about compiler
--Xcompiler	–	In case of doupt account following options as g++ options.
--Xweaver	–	In case of doupt account following options as ac++ options.
--deps <arg>	–	Path to an aspect dependency file (combine with -M...)
-p|--path <arg>	!	Defines a project directory
-d|--dest <arg>	X	Specifies a target directory for saving
-v|--verbose <arg>	!	Level of verbosity (0-9)
-o|--output <arg>	X	Name of the output file
--include_files	X	Generate manipulated header files (short version -i is not supported)
-a|--aspect_header <arg>	!	Name of aspect header file or 0
-r|--repository <arg>	X	Name of the project repository
--expr <arg>	X	Pointcut expression to match in repository
--config <arg>	!	Parser configuration file
--no_line	X	Disable generation of #line directives
-k|--keywords	X	Allow AspectC++ keywords in normal project files
--real-instances	X	Let ac++ perform a full template analysis
--problem...	X	Enable back-end compiler problem workaround
--no_problem...	X	Disable back-end compiler problem workaround
--warn...	X	Show a specific ac++ warning that is suppressed by default
--no_warn...	X	Suppress a specific ac++ warning
-I <arg>	X	Include file search path
-D <name>[=<value>]	X	Macro definitions
-U <name>	X	Undefine a macro
--include <arg>	X	Forced include

Table 1: ag++ Compiler Option Summary

2.1.3 --path

This options differs only slightly from the `--path` option of `ac++`. In `ac++` it is mandatory to specify a project path, whereby `ag++` the current working directory is used as project path by default. Especially for larger projects it is NOT wise to rely on the default project path, as weaving take a lot of time. See the AspectC++ Compiler Manual for a more detailed description of this option.

2.1.4 -c

Like the `-c` option of `g++`, this options effects the creation of object files of one or more source files.

2.1.5 -v|--verbose [<arg>]

Set the level of verbosity.

2.1.6 --aspect-header <arg>

This option differs from meaning in `ac++` only if dependencies are generated (see 2.2).

2.1.7 --keep_woven

Don't remove intermediate woven files.

2.1.8 --c_compiler

Specify path to GNU C++ compiler˙ The default is `g++`.

2.1.9 --ac_compiler

Specify path to AspectC++ compiler˙ By default ag++ assumes, that the ac++ executable is located in the same directory like itself.

2.1.10 --config_command

Specify the command which prints information about the compiler. This information is necessary for generating the parser (puma) configuration file. The default value is "`<compiler> <compiler options> -E -dM -v -x c++ <an empty file>`".

2.1.11 --config <arg>

Path to a puma configuration file. If this option is available the configuration file will not be generated automatically.

2.1.12 --Xcompiler

`ac++` and `ag++` options that might interfere with `g++` options are not recognized after using `--Xcompiler` in the argument list of an `ag++` invocation.

2.1.13 --Xweaver

Enable the recognition of those `ac++` and `ag++` options which previously have been disabled by the usage of –`Xcompiler`.

2.1.14 --deps <arg>

This option is helpful to limit the effect of aspect header changes on incremental builds. The path to an aspect dependency file, which must exist, is passed as an argument. It only makes sense to use this option in combination with the generation of dependency information (see Section 2.2). An aspect dependency file consists of lines, where each line is either empty or defines an aspect dependency rule. The syntax of an aspect dependency rule is as follows:

<aspect-header-pattern> '=>' <source-code-pattern>

Each line may end with a comment that starts with `'#'`. A pattern is written in double quotes (`'"'`) and is matched against the canonical (absolute) path of source code files. `'/'` is used as the path separator. The left pattern is matched against all aspect header paths and the right against ordinary source code of the project. With this file the programmer of an aspect header can specify the source code files that are affected by the aspect.

```
# Example aspect dependency rules
"A1.ah" => "C.*"
"D.ah" =>"/rootdir/**/dir/*.c++" # abc
"deptest/?la.ah" => "y.c*"
"dep*/b*.ah" => "x.h"
"Slice*.ah" => ""
```

The first rule, for instance, specifies that a change in "`A1.ah`" (or an included header file) shall trigger a rebuild of all translation units that depend on "`C.*`", e.g. "`C.cc`" or "`C.h`". Attention: If the rule is not consistent with the aspects that

are implemented in “A1.ah”, rebuilds will produce invalid programs. Use this feature with care! If there no rule for an aspect header in an aspect dependency file, *all* translation units will depend on it. This is safe, but any change will trigger a *full rebuild*. A path pattern may contain the following wildcard symbols:

Symbol	Meaning
*	0 or more arbitrary characters in a path element
?	1 arbitrary character in a path element
**/	0 or more arbitrary directory names

If a pattern does not begin with `'/'`, the pattern is automatically prefixed by `'/**/'`. As as consequence a filename, such as `"A1.ah"`, is matched in any directory of the project. The character '\' is used as an escape character, e.g. `"name-with-\*.txt"`.

2.2 Generating dependency information

To produce dependency files just pass the `-M, -MM`, `-MD` or `-MMD` to `ag++`. Dependency files generated by `ag++` are slighty different from dependency files created by `g++`, as they contain dependencies to aspect header files. If the `--aspect-header` option is provided, only the header file specified as option argument is considered when building the dependency file; otherwise all aspect header files within the whole project path are considered.

If one or more aspect dependency files are provided, the dependency rules described in Section 2.1.14 are applied. If there is no such file or if there is no rule defined for an aspect header, all translation units will depend on it.

2.3 Examples

- `ag++ --help`
  Displays all options with a short description.

- `ag++ -o test Test.cc main.cc`
  Weave, compile and link the source files `Test.cc` and `main.cc`. The created executable will be named '`test`'.

- `ag++ --gen_config`
  Create a puma configuration file named `puma.config` within the current working directory.

- `ag++ --gen_config -o my.config`
  Create a puma configuration file named `my.config`.

- `ag++ --path src --include_files --dest gen/includes`
  Generate modified include files out of all include files found below `src` directory and store them under '`gen/includes`'.
- `ag++ -M -MFmain.d main.cc`
  Generate dependency file `main.d` from source file `main.cc` .
- `ag++ -MD -MFmain.d -c -o main.o main.cc`
  Generate dependency file `main.d` and object file `main.o` from source file `main.cc` .
- `ag++ -p ../aspects -p .  --Xcompiler`
  This string could be used to substitute `g++` in a simple `make` environment.

3 Special Use-Cases

3.1 Debugging with .acc Files

A temporary .acc file containing all source code and the woven aspect headers is generated when compiling (See Figure 1). This file is then used by `g++` (or by another specified backend compiler) to compile to .o object files or an .elf executable. The temporary .acc-file can be saved and be used for debugging with the `--keep_woven` or `--weave_only` parameters. Because the code is woven automatically, the resulting .acc file is not optimized for readability. The following listing shows an example of woven code:

```
#line 6 "examples/helloworld/hello.h"

#line 6 "examples/helloworld/hello.h"
void hello()
#line 234 "main.acc"
{
  typedef TJP__Z5hellov_1< void, void, void, void (),
AC::TLE > __TJP;
    __TJP tjp;
  tjp._result = 0;
  AC::invoke_CSVProfiler_GenericProfiler__a0_around<__TJP> (&tjp);

}
__attribute__((always_inline)) inline void __exec_old_hello()
#line 6 "examples/helloworld/hello.h"

#line 6 "examples/helloworld/hello.h"
```

```
{
  std::cout << "Hello" << std::endl;
}
```

3.2 Optimization by Configuration Reuse

`ag++` generates a temporary configuration file that is deleted after compilation. To optimize compilation time, the configuration file can be generated once using `ag++ --gen_config` and then be reused with `ag++ --config puma.config`[1]. The generated configuration file stores the system-specific information necessary to weave .cpp, .h, and .ah files into an .acc file. This information includes architecture of the target system, paths to system includes, system macros, and sizes of builtin data types (see Figure 4 for an example). The generated configuration file can be edited if needed, although this should not be necessary for normal usage.

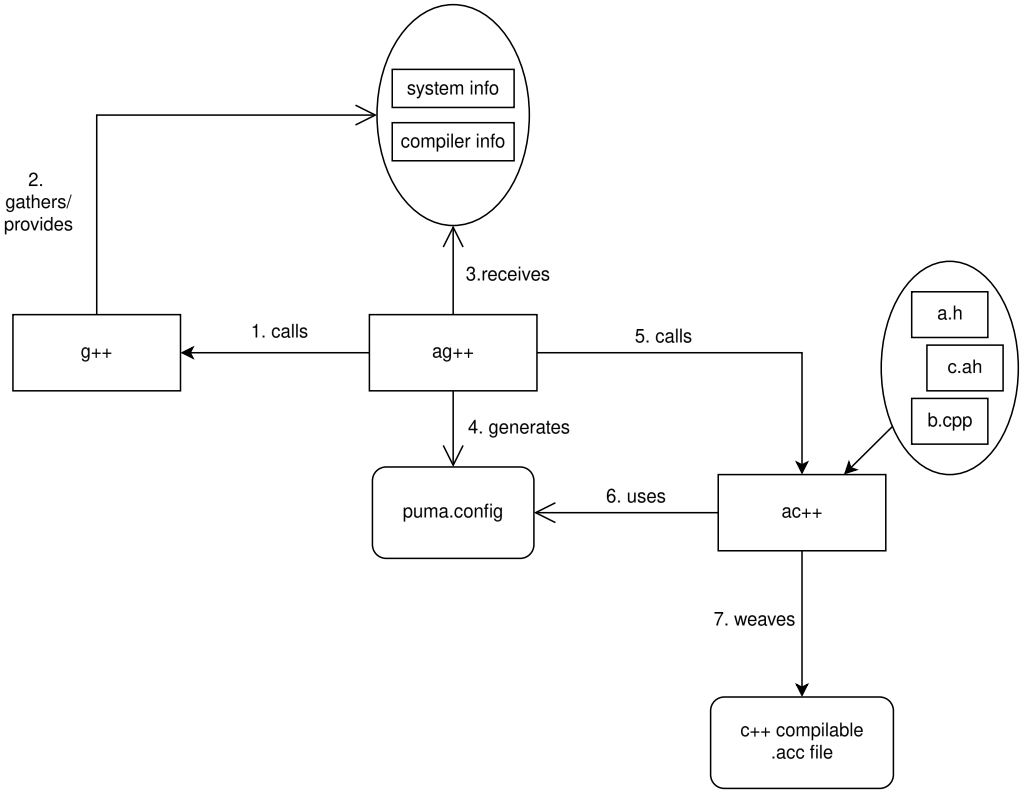

Figure 2: Generation and usage of puma.config

[1] Historically a source code analysis and transformation framework called Puma was used to generate the configuration file. It it is no longer used in ag++, but it is still the default config name. The config name can be changed using the `-o` paramter)

```
--size-type "long unsigned int"
-D "__SIZE_WIDTH__=64"
-D "__SSE2_MATH__=1"
-D "__SSE2__=1"
-D "__SSE_MATH__=1"
-D "__SSE__=1"
-D "__SSP_STRONG__=3"
-D "__STDCPP_DEFAULT_NEW_ALIGNMENT__=16"
-D "__STDCPP_THREADS__=1"
-D "__STDC_HOSTED__=1"
-D "__STDC_IEC_559_COMPLEX__=1"
-D "__STDC_IEC_559__=1"
-D "__STDC_IEC_60559_BFP__=201404L"
-D "__STDC_IEC_60559_COMPLEX__=201404L"
-D "__STDC_ISO_10646__=201706L"
-D "__STDC_UTF_16__=1"
-D "__STDC_UTF_32__=1"
-D "__STDC__=1"
-D "__UINT16_C(c)=c"
-D "__UINT16_MAX__=0xffff"
-D "__UINT16_TYPE__=short unsigned int"
-D "__UINT32_C(c)=c ## U"
-D "__UINT32_MAX__=0xffffffffU"
-D "__UINT32_TYPE__=unsigned int"
-D "__UINT64_C(c)=c ## UL"
-D "__UINT64_MAX__=0xffffffffffffffffUL"
-D "__UINT64_TYPE__=long unsigned int"
-D "__UINT8_C(c)=c"
-D "__UINT8_MAX__=0xff"
-D "__UINT8_TYPE__=unsigned char"
-D "__UINTMAX_C(c)=c ## UL"
-D "__cpp_template_auto=201606L"
-D "__cpp_template_template_args=201611L"
-D "__cpp_threadsafe_static_init=200806L"
-D "__cpp_unicode_characters=201411L"
-D "__cpp_unicode_literals=200710L"
-D "__cpp_user_defined_literals=200809L"
-D "__cpp_variable_templates=201304L"
-D "__cpp_variadic_templates=200704L"
-D "__cpp_variadic_using=201611L"
-D "__gnu_linux__=1"
-D "__k8=1"
-D "__k8__=1"
-D "__linux=1"
-D "__linux__=1"
-D "__pic__=2"
-D "__pie__=2"
-D "__unix=1"
-D "__unix__=1"
-D "__x86_64=1"
-D "__x86_64__=1"
-D "linux=1"
-D "unix=1"
--target x86_64-linux-gnu
--gnu 13.3.0
--isystem "/usr/include/c++/13"
--isystem "/usr/include/x86_64-linux-gnu/c++/13"
--isystem "/usr/include/c++/13/backward"
--isystem "/usr/lib/gcc/x86_64-linux-gnu/13/include"
--isystem "/usr/local/include"
--isystem "/usr/include/x86_64-linux-gnu"
--isystem "/usr/include"
```

Figure 3: Example content of puma.config

3.3 Cross Compilation

By default, `ag++` invokes the host `g++` compiler to generate the executable of woven code and the parser configuration file (see Section 3.2). However, you can also make `ag++` use a *cross compiler*. You can, for example, compile an program to ARM microcontrollers using the GNU ARM embedded toolchain with the following command:

```
ag++ --c_compiler arm-none-eabi-g++ example.cpp
```

The `ag++` option `--c_compiler` specifies that the `arm-none-eabi-g++` compiler (if installed) shall be used to generate the executable and the parser configuration file.

Note that since `ac++` is internally using a clang parser in the weaving process, only targets supported by the open source version of clang work with `ac++`. To find out if your target is supported by `ac++` you can install clang and print the target list with the following command:

```
clang -print-targets
```

Besides the listed CPU types `ac++` also supports the Infineon AURIX Tricore platform (`tricore-g++`).

3.4 Solving Option Ambiguities

Since `ag++` executes both weaver and compiler, options that are passed to `ag++` can have ambiguous meanings. For example you might want to run `ag++ -pipe` to execute `g++` with the `-pipe` option, but `ac++ -p "ipe"` will be executed instead, since both variants match your input and the matching weaver option is applied first. To clarify which tool your given option is intended for, you can use `--Xcompiler` (see 2.1.12) and `--Xweaver` (see 2.1.13) in `ag++`.

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

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 be 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. When designing a Makefile you should keep that in mind and include `--Xcompiler` per default.