Ecole d'ingénieurs et d'architectes de Fribourg
Hochschule für Technik und Architektur Freiburg

GRID & Cloud Computing Group

POP-C++

CMake support

Author:
Valentin Clément

Date: August 24, 2011
Revision: 1.0

Contents

1 Introduction

This document introduce the POP-C++ support for CMake to the POP-C++ developer. This is a user manual and nothing is related to the implementation of this support in this document.

1.1 CMake introduction

The following introduction is from the CMake project website[1].

CMake is an extensible, open-source system that manages the build process in an operating system and in a compiler-independent manner. Unlike many cross-platform systems, CMake is designed to be used in conjunction with the native build environment. Simple configuration files placed in each source directory (called CMakeLists.txt files) are used to generate standard build files (e.g., makefiles on Unix and projects/workspaces in Windows MSVC) which are used in the usual way. CMake can generate a native build environment that will compile source code, create libraries, generate wrappers and build executables in arbitrary combinations. CMake supports in-place and out-of-place builds, and can therefore support multiple builds from a single source tree. CMake also supports static and dynamic library builds. Another nice feature of CMake is that it generates a cache file that is designed to be used with a graphical editor. For example, when CMake runs, it locates include files, libraries, and executables, and may encounter optional build directives. This information is gathered into the cache, which may be changed by the user prior to the generation of the native build files.

1.2 POP-C++ support in CMake

As POP-C++ has a specific build process, the need of new command in cmake was obvious. These commands are separated in two sub-groups:

- Compiling (ADD_POPC_MAIN, ADD_POPC_OBJECT)
- Object map generation (ADD_POPC_OBJMAP)

These commands are detailed in Chapter 2. POP-C++ in CMake is referred as POPCPP.

This document is structured as follow:

- Chapter 2 details the three new commands added in CMake.
- Chapter 3 gives some instructions to the developers about the specificity of this language support.
- Chapter 4 explains how to install the support.
- Chapter 5 gives two examples of CMakeLists.txt file with POP-C++.

2 CMake new instruction for POP-C++

2.1 ADD_POPC_MAIN

This instruction is used to compile a POP-C++ main. Its signature is as follow:

```
ADD_POPC_MAIN(target source1 [source2 ...] [FLAGS flag1 [flag2 ...]]
[DEP dependency1 [dependency2]])
```

This command will define a new target executed in the build process. Flags can be overwritten with the FLAGS option. The default flag is "-o". Dependencies to this target can be added with the option DEP.

2.2 ADD_POPC_OBJECT

This instruction is used to compile a POP-C++ object. Its signature is as follow:

```
ADD_POPC_OBJECT([target] source1 [source2 ...] [FLAGS flag1 [flag2 ...]]
[DEP dependency1 [dependency2]] [NOBROKER|NOPARALLEL] )
```

This command will define a new target executed in the build process. Flags can be overwritten with the FLAGS option. The default flags are "-object -o". Dependencies to this target can be added with the option DEP. The option NOBROKER will compile the target with the following flags: "-parclass-nobroker -c". The target name will be set automatically with the name of the first source file. For example, if the first source file is named source1.ph, the target will be named nobroker-source1.ph. The option NOPARALLEL is used to compile non-parallel object. The flags used for this option are "-c". The target will also be named with the first source file. For example, if the first source file is named source1.ph, the target will be named object-source1.ph.

2.3 ADD_POPC_OBJMAP

This instruction is used to generate the POP-C++ object map. Its signature is as follow:

```
ADD_POPC_OBJMAP(obj1 [obj2 ...] [OBJECTMAP name] [NOT_APPEND])
```

This command will defined a new target executed in the build process to generate an object map. The option OBJECTMAP can be specified to change the name of the generated object map file. The default name is "obj.map". The option NOT_APPEND will erase any object map file with the same name in the current folder.

3 Developer tips

3.1 CMake version

The POP-C++ support for CMake has been developed with the version 2.8 of CMake. There is no guarantee this will work with an older version.

3.2 Target order

As CMake process target in a special order, there is no guarantee the compilation process will follow the command flow. To manage the order of the compilation process, please use the dependencies.

4 Installation

To install the POP-C++ support for CMake, cmake must be installed on the operating system. In the distribution, there is a **cmake** directory. This directory has a **CMakeLists.txt** file. To install, just run the cmake make command in this directory as follow:

```
user@machine:~/popc2.0/cmake$ cmake .
```

5 Example of CMakeLists.txt files

5.1 Simple CMake file

The following code shows a simple CMake file to compile an application with one parallel object and one main. Line 1 is set to defined the minimum version of CMake to be used. POP-C++ support for CMake use the version 2.8 or later.
On line 2, the project name is defined and the option NONE is set to avoid any other programming language. Line 3 enable the programming language POP-C++.

The lines 5 and 6 will create two targets for the compilation of the application. The line 7 will create a target to generate the object map file.

```
1: CMAKE_MINIMUM_REQUIRED(VERSION 2.8)
2: PROJECT(simple-project NONE)
3: ENABLE_LANGUAGE(POPCPP)
4:
5: ADD_POPC_OBJECT(integer.obj integer.cc integer.ph)
6: ADD_POPC_MAIN(main main.cc integer.cc integer.ph)
7: ADD_POPC_OBJMAP(integer.obj NOT_APPEND)
```

5.2 Advanced CMake file

In this example, the order of compilation is important. Using dependencies allows the developer to control the compilation process. In the line 5, the option NOBROKER is used. The created target will be named nobroker-Barrier.ph. This target is a dependency of the "main" compilation. The target nobroker-Barrier.ph will generate Barrier.o and Barrier.stub.o. These files are used in the next command.

```
1: CMAKE_MINIMUM_REQUIRED(VERSION 2.8)
2: PROJECT(test-barrier NONE)
3: ENABLE_LANGUAGE(POPCPP)
4:
5: ADD_POPC_OBJECT(Barrier.ph Barrier.cc NOBROKER)
6: ADD_POPC_MAIN(main-barrier Barrier.stub.o worker.ph worker.cc main.cc
DEP nobroker-Barrier.ph)
7: ADD_POPC_OBJECT(Cworker.obj Barrier.stub.o worker.cc worker.ph
DEP main-barrier)
8: ADD_POPC_OBJECT(Barrier.obj Barrier.ph Barrier.cc DEP Cworker.obj)
9: ADD_POPC_OBJMAP(Cworker.obj Barrier.obj NOT_APPEND)
```

For more example of CMakeLists.txt files, please refers to the POP-C++ test suite with CMake support in the distribution (In the following directory: *POPC_DISTRIB*/test_cmake/)

6 References

[1] Kitware Software, *CMake project introduction*, http://www.cmake.org/cmake/project/about.html