Cmake and Ccola User Manual

# Introduction

This user manual is intended for users that want to set up and maintain a project structure using CMake and Ccola (CMake Component Layer). This manual does not cover the internals and structure of CMake. Ccola is a Cmake Component layer built on top of CMAKE infrastructure, described further in this document.

CMake is a makefile generator in a broad sense. From a textual description of a project structure given in a hierarchy of CMakeLists.txt text files, it can generate solution and project files for Visual Studio; project files for Eclipse; makefiles for various make tools; and many more. The project structure is described using a CMake-specific script language.

While the CMake script language enables you to specify all sorts of executables, libraries, custom commands, etcetera, doing this manually for each component would be very cumbersome and would lead to inconsistencies. Ccola is the collection of CMake scripts created for the purpose of specifying project structures in our embedded context. While in CMake you would specify something like “for this executable, link to these libraries, compile these source files, use these include paths, use these settings, place the executable on this path for this processor, that path for that processor, etc.”, with the Ccola scripts you would specify something like “for this executable, compile these source files, and be are dependent on these other components”. Therefore, when creating the hierarchy of CMakeLists.txt that describes the project layout, for normal usage only the Ccola macros are used.

# Running CMake

## Separation of Source, Build, and Install directories

## Loading the Visual Studio Solution

## Loading an Eclipse Environment

## Building Projects and Executing Tests from Command Line

On a build server, builds are executed from the command line. Rather than having to find out what options to feed Eclipse or Visual Studio, CMake provides commands to build a complete project. For Eclipse:

cmake –-build C:\Data\Sandbox-Build-STM32F746

For Visual Studio, you have to specify for what configuration (i.e. Debug or RelWithDebInfo) to build:

cmake -–build C:\Data\Sandbox-Build-VisualStudio --config Debug

cmake -–build C:\Data\Sandbox-Build-VisualStudio --config RelWithDebInfo

In order to run all tests in a solution, use the –-target RUN\_TESTS option:

cmake -–build C:\...\...-VisualStudio --config Debug -–target RUN\_TESTS

# Interactions with Visual Studio

## ALL\_BUILD, INSTALL, ZERO\_CHECK Targets

## RUN\_TESTS Target

# Interactions with Eclipse

## Source File Location

## Building the Project

## Debugging and Running

# Project, Package, and Component

Project hierarchies specified with Ccola are structured into a project level, which contains multiple packages, which each contain multiple components.

## Component

A component is a grouping of source files for a specific purpose. That purpose usually is the creation of an executable, or a library. For a list of component types, see section 4. Components can have dependencies on other components. Through such dependencies include paths, settings, libraries, and other entities are communicated between components. For example, given a library lib and an executable exe that is dependent on lib, the include paths of lib are added to the include paths of exe, and generated library of lib is linked to the executable of exe.

Specifying a component is done with the ccola\_component and matching ccola\_end\_component commands. The ccola\_component command takes two parameters: its name and the component type. The ccola\_sources command specifies which source files are used in this component, and the ccola\_dependencies command specifies on which other components this component is dependent. For example:

ccola\_component(application static\_library)

ccola\_sources(

BinaryObject.cpp

BinaryObject.hpp

Compiler.cpp

Compiler.hpp

HexOutput.cpp

HexOutput.hpp

SparseVector.hpp

)

ccola\_dependencies(

infra.util

)

ccola\_end\_component()

## Package

A package is a logical grouping of components. For example, when building a hex\_compiler, the package hex\_compiler contains the components application (a static library which contains all application logic), test (an executable which thoroughly tests the application logic), and main (an executable which instantiates the application logic). The name of the component includes the package name, so the three components are referred to as hex\_compiler.application, hex\_compiler.test, and hex\_compiler.main.

Specifying a package is done with the ccola\_package and matching ccola\_end\_package commands. The ccola\_package command takes a single parameter which specifies the name of the package. For example:

ccola\_package(hex\_compiler)

<component definitions>

ccola\_end\_package()

## Project

The top level CMakeLists.txt file specifies the name of the project. This name is used in the name of the generated Visual Studio solution and the generated Eclipse environment, and should therefore be the name of the actual project for which the source code is created. It is strongly advised to let this be the only place in all source code in which the project name is used, so that when the source base is reused in another project, renaming this project name is the only renaming that has to be done.

Specifying a project is only done in the top level CMakeLists.txt, and uses the ccola\_project and matching ccola\_end\_project commands and two lines of boilerplate code:

cmake\_minimum\_required(VERSION 3.2.3)

include("${CCOLA\_DIR}/ccola.inc")

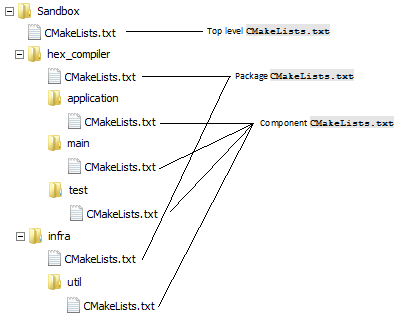
ccola\_project(Sandbox)

<package definitions>

ccola\_end\_project()

## Creating a Hierarchy using Subdirectories

While it is possible to cram everything into one CMakeLists.txt, the structure of packages and components is better represented when each component is placed in its own directory, and all components of a single package are grouped into a package directory. The directory structure of the examples in the previous sections would therefore look like this:



A CMakeLists.txt can include another CMakeLists.txt by using the ccola\_subdirectories command and specifying the names of the directories in which nested CMakeLists.txt files reside. The top level CMakeLists.txt file would therefore look like this:

cmake\_minimum\_required(VERSION 3.2.3)

include("${CCOLA\_DIR}/ccola.inc")

ccola\_project(Sandbox)

ccola\_subdirectories(

infra

hex\_compiler

)

ccola\_end\_project()

# Specifying Platforms

Not all code compiles and runs on every platform. For example, the component that implements hal functionality for the stm32fxxx family only needs to be compiled when compiling for the stm32fxxx family. In order to specify different options for different platforms, a platform specifier is used to include or exclude those options. For example, when specifying that component main must only be built for windows, use this command:

ccola\_component(main win: executable other: noncompile)

When compiling for windows, the win: executable other: noncompile clause reduces to executable. When not compiling for windows, it reduces to noncompile.

This construction can also be used when adding subdirectories, source files, or dependencies. For example, the FreeRTOS component has different source files for a Cortex M0, M3, M4, and M7. It therefore specifies its source files using a platform specifier:

ccola\_sources(

...

cortex-m0:

Source/portable/GCC/ARM\_CM0/port.c

Source/portable/GCC/ARM\_CM0/portmacro.h

cortex-m3:

Source/portable/GCC/ARM\_CM3/port.c

Source/portable/GCC/ARM\_CM3/portmacro.h

cortex-m4:

Source/portable/GCC/ARM\_CM4F/port.c

Source/portable/GCC/ARM\_CM4F/portmacro.h

cortex-m7:

Source/portable/GCC/ARM\_CM7/r0p1/port.c

Source/portable/GCC/ARM\_CM7/r0p1/portmacro.h

)

# Component Types

## Executable

# Other Commands

## Ccola\_definitions and ccola\_inheritable\_definitions

ccola\_inheritable\_definitions(\_VARIADIC\_MAX=10)

ccola\_definitions(GTEST\_HAS\_SEH=0 GTEST\_HAS\_EXCEPTIONS=0)

## Ccola\_include\_directories and ccola\_inheritable\_include\_directories

ccola\_inheritable\_include\_directories(include)

ccola\_include\_directories(.)

## Ccola\_linker\_scripts and ccola\_default\_linker\_scripts

## Ccola\_programmer

With ccola\_programmer, a programmer other than the default programmer for a certain platform can be specified. This influences the launch configuration generated for Eclipse. For example, the default programmer for the STM32F407 is the ST-Link programmer. In order to generate a launch configuration that uses the J-Link programmer, use this command:

ccola\_programmer(j-link)

# Known Problems

## Adding a New Component

When a new component is added and a build is executed, CMake is invoked to re-generate the project structure. However, while changes to existing projects are detected by Visual Studio, changes to the solution file are not detected.