ATS CMake Documentation

Release latest

Hanwen Wu

May 06, 2016

Contents

This is a project aiming at developing a build tool for ATS. It is based on CMake. Currently, it provides some very useful CMake modules for ATS users to simplify building processes. In the near future, it will support downloading artifacts from a server to help you utilize third party ATS libraries.

The project is hosted on GitHub. Welcome to contribute.

CHAPTER 1

Table of Contents

1.1 Examples

I will try to cover more usage details in the following examples.

1.2 Multiple Executables

This project involves multiple executables in one project. Source files can be found here.

```
SimpleUsage
|   CMakeLists.txt
|   gcd.dats                    # compute GCD using lambda
|   lambda.dats                 # implementation of simple lambda calculas
|   lambda.sats                 # definition of simple lambda calculas
|   lambda_env.dats             # lambda evaluation environment
|   prime.dats                  # compute prime number using lambda
|
\---build                               # out-of-source build dir
        ...
```

What we want are two executables, `gcd` and `prime`.

```
CMAKE_MINIMUM_REQUIRED (VERSION 2.8)
PROJECT (SIMPLE_USAGE C)

FIND_PACKAGE (ATS REQUIRED)
IF (NOT ATS_FOUND)
        MESSAGE (FATAL_ERROR "ATS Not Found!")
ENDIF ()

SET (LAMBDA lambda.sats lambda.dats lambda_env.dats)

ATS_COMPILE (GCD_SRC ${LAMBDA} gcd.dats)
ATS_COMPILE (PRIME_SRC ${LAMBDA} prime.dats)

ADD_EXECUTABLE (gcd ${GCD_SRC})
ADD_EXECUTABLE (prime ${PRIME_SRC})
```

1.3 Local CMake Modules

Sometimes, you may want to use these CMake moduls locally. I will modify the last example with locally loaded ATS modules. Source files can be found here.

```
LocalModule
|   CMakeLists.txt
|   gcd.dats
|   lambda.dats
|   lambda.sats
|   lambda_env.dats
|   prime.dats
|
+---build
\---cmake
        ATSCC.cmake
        FindATS.cmake
```

I put the CMake modules under `${CMAKE_CURRENT_SOURCE_DIR}/cmake`. And the `CMakeLists.txt` should be updated as follows.

```
CMAKE_MINIMUM_REQUIRED (VERSION 2.8)
PROJECT (SIMPLE_USAGE C)

SET (CMAKE_MODULE_PATH ${CMAKE_CURRENT_SOURCE_DIR}/cmake)
INCLUDE (ATSCC)

FIND_PACKAGE (ATS REQUIRED)
IF (NOT ATS_FOUND)
        MESSAGE (FATAL_ERROR "ATS Not Found!")
ENDIF ()

SET (LAMBDA lambda.sats lambda.dats lambda_env.dats)

ATS_COMPILE (GCD_SRC ${LAMBDA} gcd.dats)
ATS_COMPILE (PRIME_SRC ${LAMBDA} prime.dats)

ADD_EXECUTABLE (gcd ${GCD_SRC})
ADD_EXECUTABLE (prime ${PRIME_SRC})
```

1.4 Inside the Code

This page is a reference to all macros/functions in `ATSCC.cmake` and `FindATS.cmake`.

Tips for filename/path

Most of commands/macros in CMake, and most of commands of Lunix require filenames/paths contain **NO** space. So, I assume no space in any of the filenames/paths. If you get errors, first check if there is any space in any filenames/paths, and remove them. It is always good to make a space-free filename/path.

1.5 FindATS Module

1.5.1 FIND_PACKAGE (ATS REQUIRED)

Quick Ref

- **Input** ATSHOME environment variable.
- **Output (CMake variables)** ATS_HOME ATSCC ATSOPT ATSCC_FLAGS ATS_INCLUDE_DIRS ATS_LIBRARIES ATS_VERBOSE: False by default
- **Effects (CMake variables for internal usage)** ATS_INCLUDE_DIR ATS_LIBRARY CMAKE_C_COMPILER

This is a standard CMake FindXXX module. CMake community has a documentation about how to write a FindXXX module, here. You have to write this in your CMakeLists.txt in order to use ATS.

In my FindATS.cmake, I use environment variable ATSHOME to lookup ATS binaries. And if it is found, a series of CMake variables will be set. They are the followings.

ATS_Home: Set to the same value as environment variable ATSHOME.

ATS_INCLUDE_DIR: *For internal usage only.* Set to the include directories of ATS. Its value is ${ATS_HOME}/ccomp/runtime.

ATS_LIBRARY: *For internal usage only.* Set to the link directories of ATS. Its value is ${ATS_HOME}/ccomp/lib.

ATS_INCLUDE_DIRS: Its the same value as ${ATS_INCLUDE_DIR}, but it is for users.

ATS_LIBRARIES: Its the same value as ${ATS_LIBRARY}, but it is for users.

Note: These two internal variables and two user variables are compliant to CMake naming conventions. Please refer to CMake documentations.

ATS_VERBOSE: False by defalut. If set to ture, it will produce more informations during making process. Please set it only after FIND_PACKAGE (ATS ...). Otherwise, it will be reset to default value inside the FindATS module.

Example

```
FIND_PACKAGE (ATS REQUIRED)
IF (NOT ATS_FOUND)
    MESSAGE (FATAL_ERROR "ATS Not Found!")
ENDIF ()

SET (ATS_VERBOSE True)
```

ATSCC: It is set to the full path of atscc executable.

ATSOPT: It is set to the full path of atsopt executable.

ATSCC_FLAGS: It is set to empty.

CMAKE_C_COMPILER: *For internal usage only.* This is a trick. First, atscc will call atsopt and then gcc to compile the code. Second, atscc includes many useful arguments for gcc so that it

can correctly find all runtime dependencies. Thrid, by setting C compiler to `atscc`, CMake will invoke `atscc` to compile C code, thus utilizing `atscc`'s extra arguments to locate all necessary headers and libraries. You won't need to use this. But I think it's better to let you know this.

Example

```
FIND_PACKAGE (ATS REQUIRED)

IF (NOT ATS_FOUND)
    MESSAGE (FATAL_ERROR "ATS Not Found!")
ENDIF ()
```

Result

If ATS is found, those commands/macros/variables will be avaiable. Otherwise, `ATS Not Found!` will be printed and CMake will terminate.

1.6 ATSCC Module

1.6.1 ATS_INCLUDE (path ...)

This macro will add all paths as directories to look up for `SATS`/`HATS` files. This will result in multiple `IATS` flags for `atsopt`. The paths should be relative to `${CMAKE_CURRENT_LIST_DIR}`, or they are absolute paths. You need at least one path as a parameter.

Example

```
ATS_INCLUDE (SATS HATS /usr/include/ats028/SATS)
```

Result

`${CMAKE_CURRENT_LIST_DIR}/SATS`, `${CMAKE_CURRENT_LIST_DIR}/HATS` and `/usr/include/ats028/SATS` will be added to `atsopt -IATS` flags.

1.6.2 ATS_COMPILE (output src ...)

Quick Ref

- **Input**

  OUTPUT The name of the variable where to store output filenames. It is a list, not a string.

  Source filenames Specify all related files to be compiled. Seperate them using space. Only `DATS` and `SATS` files are needed.

- **Output**

  OUTPUT All fullpaths of C files will be stored in `OUTPUT`.

This macro will compile all sources provided into corresponding C sources, and store all generated C file names into `${output}` for further use. Those file names are **absolute paths**.

The dependencies will be automatically generated. This includes two parts. *First*, all `staload` (for `sats` file) and `#include` (for `hats` file) will be detected using `atsopt --dep1`. *Second*, all generated C files will also be involved in dependencies. For example, if `a.sats` includes `a.hats`, and `a.dats` staload `a.sats`. Then a dependency `a_dats.c -> a_sats.c` will be generated so that if `a.hats` changes, `a_dats.c` will be regenerated.

Example

```
ATS_COMPILE (TEST_SRC SATS/hello.sats DATS/hello.dats DATS/main.dats)
```

Result

All C files compiled from ATS files are stored in `TEST_SRC`. They are `SATS/hello_sats.c`, `DATS/hello_dats.c` and `DATS/main_dats.c`.

Note that there is no need to specify `CATS` files and `HATS` files, since `atsopt` will automatically find them in the paths specified by `ATS_INCLUDE ()`.

Warning: CMake has some really confusing terms, like **list** and **string**. Basically, a list is a single string where inner items are seperated using semicolon, while a string is seperated using spaces. `set (MyString "Hello World")` will give you a string, while `set (MyList Hello World)` will give you a list, which is stored as `Hello;World`. Also, you need to pay attention to quotes. `set (MyString2 "${MyString}")` will be a string, while `set (MyList2 ${MyString})` will be a list, since it will evaluate to `set (MyList2 Hello World)`. You should search "CMake List String" on Google for more information.

1.6.3 ATS_DEPGEN (OUTPUT SRC) (For internal usage only)

Quick Ref

- **Input:** A single source file path.
- **Output:** `${OUTPUT}` will contain space separated dependencies. It is a string, not a list. All dependencies are fullpaths.

It is called by `ATS_COMPILE ()`. It runs `atsopt` to generate ATS dependencies. For example, if `hello.dats` depends on `hello.sats`, it will append the fullpath of `hello.sats` to the output. Later, it will call `ATS_DEPGEN_C ()` to generate C dependencies. Take the above example, it will make `hello_dats.c` depends on `hello_sats.c`. This enables `hello_dats.c` to be regenerated when `hello.sats` is modified.

1.6.4 ATS_DEPGEN_C (DEP) (For internal usage only)

Quick Ref

- **Input:** All dependencies for a source file.
- **Output:** C dependencies will be appended.

It is called by `ATS_DEPGEN ()`. For example, if we have `1.sats <- 2.sats`, then we add `1_sats.c <- 2_sats.c`.

This is useful when `1.sats` inludes a `HATS` file. When the HATS file updates, `1.sats` is not changed, but `1_sats.c` is changed. And since `2.sats` depends on `1.sats` and it is not changed, `2_sats.c` is not recompiled. However, it should be recompiled since the actual meaning of `1.sats` has been changed. Thus, we need to append C dependencies.

1.7 Useful CMake Commands

These are useful CMake commands. They are parts of CMake, not my project. But I think you will need them everywhere. If you need detail information, please refer to CMake offical documents.

Note: You can always check out latest usage of CMake here. Every commands are listed and documented.

1.7.1 `TARGET_LINK_LIBRARIES (target libs...)`

It will link those libraries to a specific target listed in the *same* CMake list files. Those library names could be confusing sometime. If you want to link a library file `libzlog.so.2`, you may try `zlog` or `libzlog` as parameters to `TARGET_LINK_LIBRARIES`.

1.7.2 `ADD_EXECUTABLE (output src ...)`

It will produce the binary output from all the source files.

1.8 Contacts

If you find something wrong, please email me at `hwwu AT bu DOT edu`.

CHAPTER 2

Features

- It uses CMake, which is cross platform.
- Automatic dependency resolving. This is especially useful, and is first supported by this project.
- Easy to use.

CHAPTER 3

Quick Start

- Install CMake. You can download them from CMake Website.

  Note: Version 2.8.3+ required, since `ATS-CMake` uses `CMAKE_CURRENT_LIST_DIR` variable

- Install ATS from ATS Website.

  Note: You need to setup environment variables `ATSHOME` and `PATH` properly. `ATS-CMake` use them to locate your currently available ATS binaries. For example:

```
export ATSHOME=/cs/coursedata/cs320/ATS029
export ATSHOMERELOC=ATS-0.2.9
export PATH=$PATH:$ATSHOME/bin
```

- Download this project from GitHub. Particularly, `FindATS.cmake` and `ATSCC.cmake`.
- Copy those CMake modules into CMake module dir.

  Note: Normally, the module dir is `/usr/share/cmake-x.x.x/Modules`. You can find more information at CMake Website.

- Start using it!

CHAPTER 4

Hello World

Suppose you have a small project containing `hello.sats`, `hello.dats` and `main.dats`. Then, you need to write a `CMakeLists.txt` like the following

```
CMAKE_MINIMUM_REQUIRED (VERSION 2.8)

#Specify project name as HELLOWORLD, and project language as C. Yes, it is C
PROJECT (HELLOWORLD C)

#Actually, this makes CMake to find ATSCC.cmake using FindATS.cmake
FIND_PACKAGE (ATS REQUIRED)

#The ATS_FOUND is automatically set by FindATS.cmake module
IF (NOT ATS_FOUND)
    MESSAGE (FATAL_ERROR "ATS Not Found!")
ENDIF ()

#ATS_COMIPLE is the core of this project. To put it simple,
#you just specify related SATS/DATS files here, and use a variable
#like TEST_SRC to store the outputs. ATS_COMPILE will analyze their
#dependencies, compile them into C files, and store those C file
#names into TEST_SRC

#You can use ATS_INCLUDE to add search paths for the compiler to
#find proper SATS/HATS files.
ATS_COMPILE (TEST_SRC
    hello.sats
    hello.dats
    main.dats)

#It generate the final file "test" using all the C files in TEST_SRC.
#And this is a standard CMake command
ADD_EXECUTABLE (test ${TEST_SRC})
```

After you have a correct `CMakeLists.txt`, we just need to invoke `cmake`. But please make sure that you have a correct project layout.

```
HelloWorld
|   CMakeLists.txt
|   hello.dats
|   hello.sats
|   main.dats
|
\---build
```

```
        ...
```

Note: I suggest using *out-of-source* build, which makes everything clean, especially when you want to delete all temp files. See here for more information. I use a `./build` dir for this purpose.

Now, go to `./build` and invoke `cmake`. It will generate a `makefile` for you under `./build`. You can invoke `make` now, to build the project as usual, and congratulations! The output binary will be under `./build`

```
>>> cd ./build
>>> cmake ..
...
>>> make
...
```

Note: We use `cmake ..` because the present working directory is `./build`, while the `CMakeLists.txt` is in the parent directory. Therefore, it is `cmake ..` instead of `cmake ..` Pay attention.

CHAPTER 5

What's Next

In the followings, I will try to cover more use cases, and then look into what's happening in the CMake modules, so that you can better use them, and even help me develop it.