3.7.2. Include modules

CMake modules is a common way to reuse code.

3.7.2.1. Include standard

CMake comes with set of standard modules:

```
cmake_minimum_required(VERSION 2.8)
project(foo NONE)

include(ProcessorCount)

ProcessorCount(N)
message("Number of processors: ${N}")
```

```
[cmake-sources]> rm -rf _builds
[cmake-sources]> cmake -Hinclude-processor-count -B_builds
Number of processors: 4
-- Configuring done
-- Generating done
-- Build files have been written to: /.../cmake-sources/_builds
```

CMake documentation

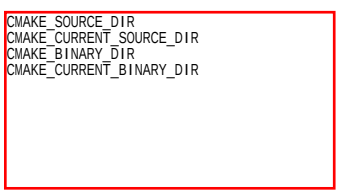
- ProcessorCount

Warning

Do not include `Find*.cmake` modules such way. `Find*.cmake` modules designed to be used via find_package.

3.7.2.2. Include custom

You can modify `CMAKE_MODULE_PATH` variable to add the path with your custom CMake modules:



```
# Top level CMakeLists.txt

cmake_minimum_required(VERSION 2.8)
project(foo NONE)

list(APPEND CMAKE_MODULE_PATH "${CMAKE_CURRENT_LIST_DIR}/modules")

include(MyModule)
```

```
# modules/MyModule.cmake

message("Hello from MyModule!")
```

```
[cmake-sources]> rm -rf _builds
[cmake-sources]> cmake -Hinclude-users -B_builds
Hello from MyModule!
-- Configuring done
-- Generating done
-- Build files have been written to: /.../cmake-sources/_builds
```

CMake documentation

- CMAKE_MODULE_PATH

3.7.2.2.1. Recommendation

To avoid conflicts of your modules with modules from other projects (if they are mixed together by `add_subdirectory`) do “namespace” their names with the project name:

```
cmake_minimum_required(VERSION 2.8)
project(foo)

list(APPEND CMAKE_MODULE_PATH "${CMAKE_CURRENT_LIST_DIR}/cmake/Modules")

include(tool_verifier) # BAD! What if parent project already has 'tool_verifier'?

include(foo_tool_verifier) # Good, includes "./cmake/Modules/foo_tool_verifier.cmake"
```

See also

- OpenCV modules

See also

- Function names
- Cache names

3.7.2.3. Modify correct

Note that the correct way to set this path is to **append** it to existing value:

```
# Top level CMakeLists.txt

cmake_minimum_required(VERSION 2.8)
project(foo NONE)

list(APPEND CMAKE_MODULE_PATH "${CMAKE_CURRENT_LIST_DIR}/modules")

include(ProcessorCount)

ProcessorCount(N)
message("Number of processors: ${N}")
```

For example when user want to use his own modules instead of standard for any reason:

```
# standard/ProcessorCount.cmake

function(ProcessorCount varname)
  message("Force processor count")
  set("${varname}" 16 PARENT_SCOPE)
endfunction()
```

定义附带 参数的 cmake
函数

Works fine:

-H:指定source tree的root路径
-B:指定binary tree的root路径
-D:指定cache变量

```
[cmake-sources]> rm -rf _builds
[cmake-sources]> cmake -Hmodify-path -B_builds "-DCMAKE_MODULE_PATH=`pwd`/modify-path/standard"
Force processor count
Number of processors: 16
-- Configuring done
-- Generating done
-- Build files have been written to: /.../cmake-sources/_builds
```

3.7.2.4. Modify incorrect

就是说，对环境变量的修改，应该是以append的形式进行的，而不是以完全修改的方式操作

It's not correct to set them ignoring current state:

```
# Top level CMakeLists.txt

cmake_minimum_required(VERSION 2.8)
project(foo NONE)

set(CMAKE_MODULE_PATH "${CMAKE_CURRENT_LIST_DIR}/modules") # WRONG!

include(ProcessorCount)

ProcessorCount(N)
message("Number of processors: ${N}")
```

In this case if user want to use custom modules:

```
# standard/ProcessorCount.cmake

function(ProcessorCount varname)
  message("Force processor count")
  set("${varname}" 16 PARENT_SCOPE)
endfunction()
```

They will **not** be loaded:

```
[cmake-sources]> rm -rf _builds
[cmake-sources]> cmake -Hmodify-incorrect -B_builds "-DCMAKE_MODULE_PATH=`pwd`/modify-
incorrect/standard"
Number of processors: 4
-- Configuring done
-- Generating done
-- Build files have been written to: /.../cmake-sources/_builds
```

3.7.2.5. Variables

Information about any kind of listfile can be taken from `CMAKE_CURRENT_LIST_FILE` and `CMAKE_CURRENT_LIST_DIR` variables:

```
# Top-level CMakeLists.txt

cmake_minimum_required(VERSION 2.8)
project(foo NONE)

list(APPEND CMAKE_MODULE_PATH "${CMAKE_CURRENT_LIST_DIR}/cmake")

include(mymodule)
```

CMAKE_CURRENT_LIST_FILE ： 指的是该文件的完全路径，包含文件名

CMAKE_CURRENT_LIST_DIR ： 仅仅指的是路径，不包含文件名的

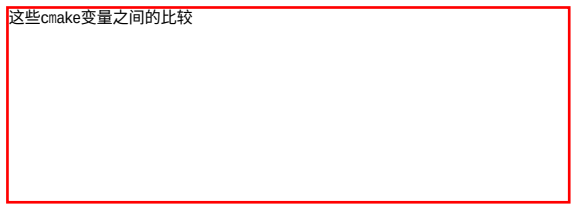
```
# cmake/mymodule.cmake

message("Full path to module: ${CMAKE_CURRENT_LIST_FILE}")
message("Module located in directory: ${CMAKE_CURRENT_LIST_DIR}")
```

```
[cmake-sources]> rm -rf _builds
[cmake-sources]> cmake -Hpath-to-module -B_builds
Full path to module: /.../cmake-sources/path-to-module/cmake/mymodule.cmake
Module located in directory: /.../cmake-sources/path-to-module/cmake
-- Configuring done
-- Generating done
-- Build files have been written to: /.../cmake-sources/_builds
```

3.7.2.5.1. CMAKE_CURRENT_LIST_DIR vs CMAKE_CURRENT_SOURCE_DIR

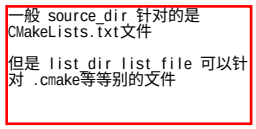
The difference between those two variables is about type of information they provide. `CMAKE_CURRENT_SOURCE_DIR` variable describe **source tree** and should be read as *current source tree directory*. Here is a list of sibling variables describing source/binary trees:

- CMAKE_SOURCE_DIR
- CMAKE_BINARY_DIR
- PROJECT_SOURCE_DIR
- PROJECT_BINARY_DIR
- **CMAKE_CURRENT_SOURCE_DIR**
- CMAKE_CURRENT_BINARY_DIR



The next files **always** exist:

- `${CMAKE_SOURCE_DIR}/CMakeLists.txt`
- `${CMAKE_BINARY_DIR}/CMakeCache.txt`
- `${PROJECT_SOURCE_DIR}/CMakeLists.txt`
- `${CMAKE_CURRENT_SOURCE_DIR}/CMakeLists.txt`



`CMAKE_CURRENT_LIST_DIR` variable describe **current listfile** (it is not necessary `CMakeLists.txt`, it can be `somemodule.cmake`), should be read as *directory of currently processed listfile*, i.e. directory of `CMAKE_CURRENT_LIST_FILE`. Here is another list of sibling variables:

- CMAKE_CURRENT_LIST_FILE
- CMAKE_CURRENT_LIST_LINE
- **CMAKE_CURRENT_LIST_DIR**
- CMAKE_PARENT_LIST_FILE

Stackoverflow

- What is the difference between CMAKE_CURRENT_SOURCE_DIR and CMAKE_CURRENT_LIST_DIR?

3.7.2.5.2. Example

Assume we have external CMake module that calculates SHA1 of CMakeLists.txt and save it with some custom info to `sha1` file in current binary directory:

```
# External module: mymodule.cmake

file(READ "${CMAKE_CURRENT_LIST_DIR}/info/message.txt" _mymodule_message)
file(SHA1 "${CMAKE_CURRENT_SOURCE_DIR}/CMakeLists.txt" _mymodule_cmakelists_sha1)
file(
    WRITE
    "${CMAKE_CURRENT_BINARY_DIR}/sha1"
    "${_mymodule_message}\nsha1(CMakeLists.txt) = ${_mymodule_cmakelists_sha1}\n"
)
```

`mymodule.cmake` use some resource. Resource `info/message.txt` is a file with content:

```
Message from external module
```

To read this resource we must use `CMAKE_CURRENT_LIST_DIR` because file located **in same external directory** as module:

```
# External module: mymodule.cmake

file(READ "${CMAKE_CURRENT_LIST_DIR}/info/message.txt" _mymodule_message)
file(SHA1 "${CMAKE_CURRENT_SOURCE_DIR}/CMakeLists.txt" _mymodule_cmakelists_sha1)
file(
    WRITE
    "${CMAKE_CURRENT_BINARY_DIR}/sha1"
    "${_mymodule_message}\nsha1(CMakeLists.txt) = ${_mymodule_cmakelists_sha1}\n"
)
```

To read CMakeLists.txt we must use `CMAKE_CURRENT_SOURCE_DIR` because CMakeLists.txt located **in source directory**:

```
# External module: mymodule.cmake

file(READ "${CMAKE_CURRENT_LIST_DIR}/info/message.txt" _mymodule_message)
file(SHA1 "${CMAKE_CURRENT_SOURCE_DIR}/CMakeLists.txt" _mymodule_cmakelists_sha1)
file(
    WRITE
    "${CMAKE_CURRENT_BINARY_DIR}/sha1"
    "${_mymodule_message}\nsha1(CMakeLists.txt) = ${_mymodule_cmakelists_sha1}\n"
)
```

Subdirectory `boo` use those module:

```
# boo/CMakeLists.txt

message("Processing boo/CMakeList.txt")

add_subdirectory(baz)
add_subdirectory(bar)

include(mymodule)
```

```
[cmake-sources]> rm -rf _builds
[cmake-sources]> cmake -Hwith-external-module/example -B_builds -DCMAKE_MODULE_PATH=`pwd`/with-
external-module/external
Top level CMakeLists.txt
Processing foo/CMakeList.txt
Processing boo/CMakeList.txt
Processing boo/baz/CMakeLists.txt
Processing boo/bar/CMakeLists.txt
-- Configuring done
-- Generating done
-- Build files have been written to: /.../cmake-sources/_builds
```

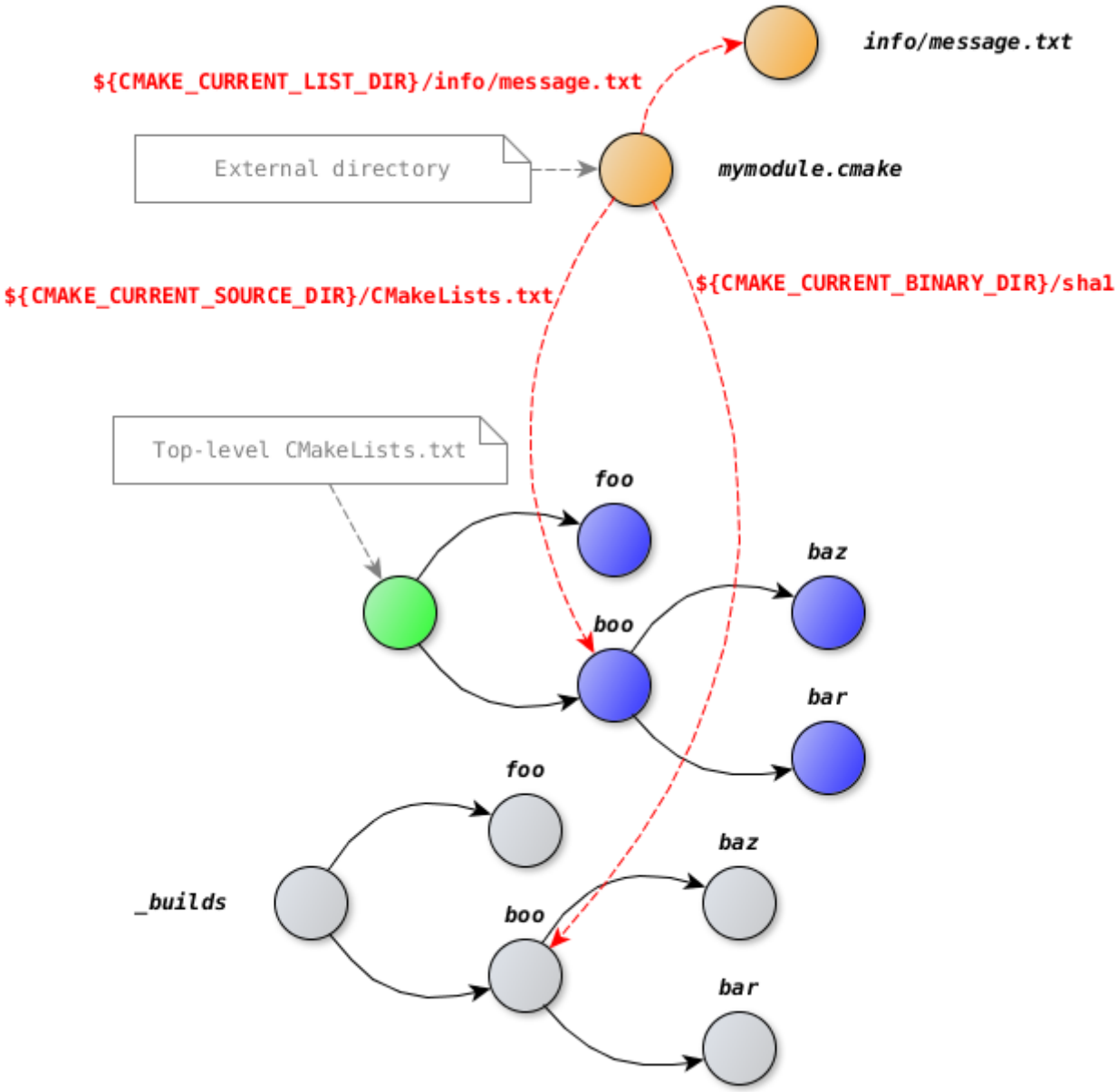
Check `sha1` file created by module:

```
[cmake-sources]> cat _builds/boo/sha1
Message from external module

sha1(CMakeLists.txt) = 9f0ceda4ca514a074589fc7591aad0635b6565eb
```

Verify value manually:

```
[cmake-sources]> openssl sha1 with-external-module/example/boo/CMakeLists.txt
SHA1(with-external-module/example/boo/CMakeLists.txt)= 9f0ceda4ca514a074589fc7591aad0635b6565eb
```

This diagram will make everything clear:



3.7.2.5.3. Recommendation

Instead of keeping in head all this information you can remember just two variables:

- `CMAKE_CURRENT_LIST_DIR`
- `CMAKE_CURRENT_BINARY_DIR`

CMAKE_CURRENT_LIST_DIR : 表示当前正在执行的函数所在的文件的绝对文件名；不是该函数定义的地方

CMAKE_CURRENT_BINARY_DIR : 表示对一些手工生成的文件的存储

Note that in function `CMAKE_CURRENT_LIST_DIR` variable is set to the directory where function **used**, not where function **defined** (see function for details).

Use `CMAKE_CURRENT_BINARY_DIR` for storing manually generated files.

⚠ Warning

Do not use `CMAKE_CURRENT_BINARY_DIR` for figuring out the full path to objects that was build by native tool, e.g. using `${CMAKE_CURRENT_BINARY_DIR}/foo.exe` is a bad idea since for Linux executable will be named `${CMAKE_CURRENT_BINARY_DIR}/foo` and for multi-configuration

generators it will be like `${CMAKE_CURRENT_BINARY_DIR}/Debug/foo.exe` and really should be determined on a build step instead of generate step. In such cases generator expressions is helpful. For example $<TARGET_FILE:tgt>.

Make sure you **totally** understand what each variable means in other scenarios:

- `CMAKE_SOURCE_DIR` / `CMAKE_BINARY_DIR` these variables point to the root of the source/binary trees. If your project will be added to another project as a subproject by `add_subdirectory`, the locations like `${CMAKE_SOURCE_DIR}/my-resource.txt` will point to `<top-level>/my-resource.txt` instead of `<my-project>/my-resource.txt`
- `PROJECT_SOURCE_DIR` / `PROJECT_BINARY_DIR` these variables are better then previous but still have kind of a global nature. You should change all paths related to `PROJECT_SOURCE_DIR` if you decide to move declaration of your project or decide to detach some part of the code and add new `project` command in the middle of the source tree. Consider using extra variable with clean separate purpose for such job `set(FOO_MY_RESOURCES "${CMAKE_CURRENT_LIST_DIR}/resources")` instead of referring to `${PROJECT_SOURCE_DIR}/resources`.
- `CMAKE_CURRENT_SOURCE_DIR` this is a directory with `CMakeLists.txt`. If you're using this variable internally you can substitute is with `CMAKE_CURRENT_LIST_DIR`. In case you're creating module for external usage consider moving all functionality to `function`.

PROJECT_SOURCE_DIR 也是具有一定的全局属性，当一个cmake项目中，存在多个PROJECT的时候，所以做好的办法就是使用CMAKE_CURRENT_LIST_DIR 来进行代替

当可以使用CMAKE_CURRENT_LIST_DIR代替CMAKE_CURRENT_SOURCEC_DIR的时候，最好进行替换

With this recommendation previous example can be rewritten in next way:

```
# External module: mymodule.cmake

# This is not a part of the function so 'CMAKE_CURRENT_LIST_DIR' is the path
# to the directory with 'mymodule.cmake'.
set(MYMODULE_PATH_TO_INFO "${CMAKE_CURRENT_LIST_DIR}/info/message.txt")

function(mymodule)
  # When we are inside function 'CMAKE_CURRENT_LIST_DIR' is the path to the
  # caller, i.e. path to directory with CMakeLists.txt in our case.
  file(SHA1 "${CMAKE_CURRENT_LIST_DIR}/CMakeLists.txt" sha1)

  file(READ "${MYMODULE_PATH_TO_INFO}" msg)
  file(
      WRITE
      "${CMAKE_CURRENT_BINARY_DIR}/sha1"
      "${msg}\nsha1(CMakeLists.txt) = ${sha1}\n"
  )
endfunction()
```

Note

As you may notice we don't have to use `_long_variable` names since function has it's own scope.

And call `mymodule` function instead of including module:

```
# boo/CMakeLists.txt

message("Processing boo/CMakeList.txt")

add_subdirectory(baz)
add_subdirectory(bar)

mymodule()
```

Effect is the same:

```
[cmake-sources]> cat _builds/boo/sha1
Message from external module
sha1(CMakeLists.txt) = 36bcbf5f2f23995661ca4e6349e781160910b71f

[cmake-sources]> openssl sha1 with-external-module-good/example/boo/CMakeLists.txt
SHA1(with-external-module-good/example/boo/CMakeLists.txt)=
36bcbf5f2f23995661ca4e6349e781160910b71f
```