# SMIL

## C++ User Modules

Matthieu FAESSEL

CMM

February 26, 2015

# Why ?

- Code consolidation, low level functions access.
- Improve code speed (direct memory access, compiler optimizations).
- Allow to wrap functions/objects to any other languages.
- Effort done to ease the addition of supplementary modules:
  - no need to build an external project and add headers, paths, definitions, and link against Smil libraries.
  - no need to know anything about CMake, Swig, Doxygen, ...
  - you just need to copy/rename the Sample Module and write your own C++ code to benefit the Smil framework.
- Benefits:
  - the generated code will be accessible in all available wrapped languages (python, java, octave, ...)
  - the generated module will be part of Smil python/java/... as an add-on
  - unit tests
  - documentation

# How ?

Prerequisites

Create a user module

Write your image functions

Unit Tests

Wrap functions

# Prerequisites

Get the sources
Required tools/libraries
Build Smil

# Prerequisites

## Get the sources

SMIL sources are maintained with Git.

Git clients (preferably use command line clients...):

- **Linux**: apt-get install git
- **Windows**: http://git-scm.com/download/win

SMIL Git repository can be cloned from this address:
`ssh://malte.cmm/home/faessel/gitroot/smil`

**Clone Smil repository:**
**git clone ssh://malte.cmm/home/faessel/gitroot/smil [src]**

Default branch is *master* (to switch to *develop*, use **git checkout develop**)

# Prerequisites

## Required tools/libraries

SMIL has no major dependency: most of the code only requires **CMake** and a **C++ compiler** to be built:

- **Linux:** sudo apt-get install cmake-curses-gui g++
- **Windows:**
  - CMake: http://www.cmake.org/download/
  - MinGW: http://tdm-gcc.tdragon.net/download

However, if you want to use some optional features, you have to satisfy the following requirements:

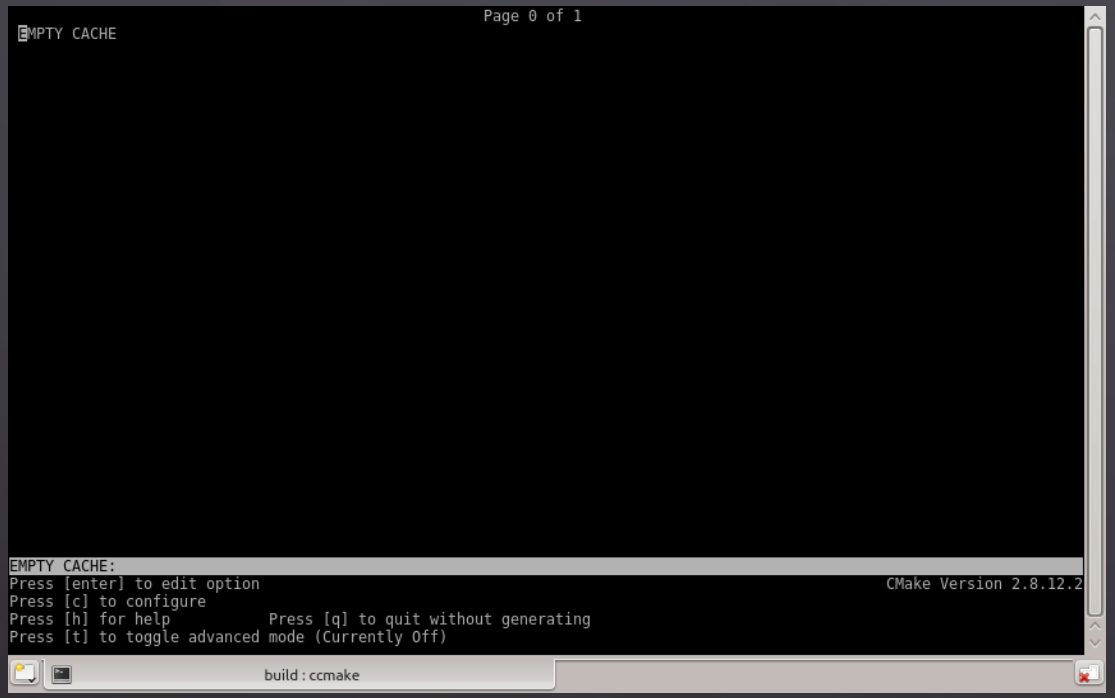
| | Optional Feature | CMake Option | Requirements |
|---|---|---|---|
| I/O | Png files support | USE_PNG | libpng-dev |
| | Jpeg files support | USE_JPEG | libjpeg-dev |
| | Tiff files support | USE_TIFF | libtiff-dev |
| | Http file access | USE_CURL | libcurl4-openssl-dev (or any libcurl-dev) |
| GUI | AA viewer | USE_AALIB | libaa1-dev |
| | Qt Viewer | USE_QT | libqt4-dev |
| | Qwt widgets (histogram, ...) | USE_QWT | libqwt-qt4-dev ($\geq$5) |
| Wrap | * | | swig |
| | Python | WRAP_PYTHON | libpython-dev |
| | Java | WRAP_JAVA | openjdk-jdk |
| | Perl | WRAP_PERL | libperl-dev |
| | Octave | WRAP_OCTAVE | liboctave-dev |
| Misc. | Text drawing | USE_FREETYPE | libfreetype-dev |
| | NumPy support | USE_NUMPY | python-numpy |
| | Doc generation | BUILD_DOC | doxygen |
| | Unit tests | BUILD_TEST | |

For Windows, it is recommended to use the GnuWin32 packages.

# Prerequisites

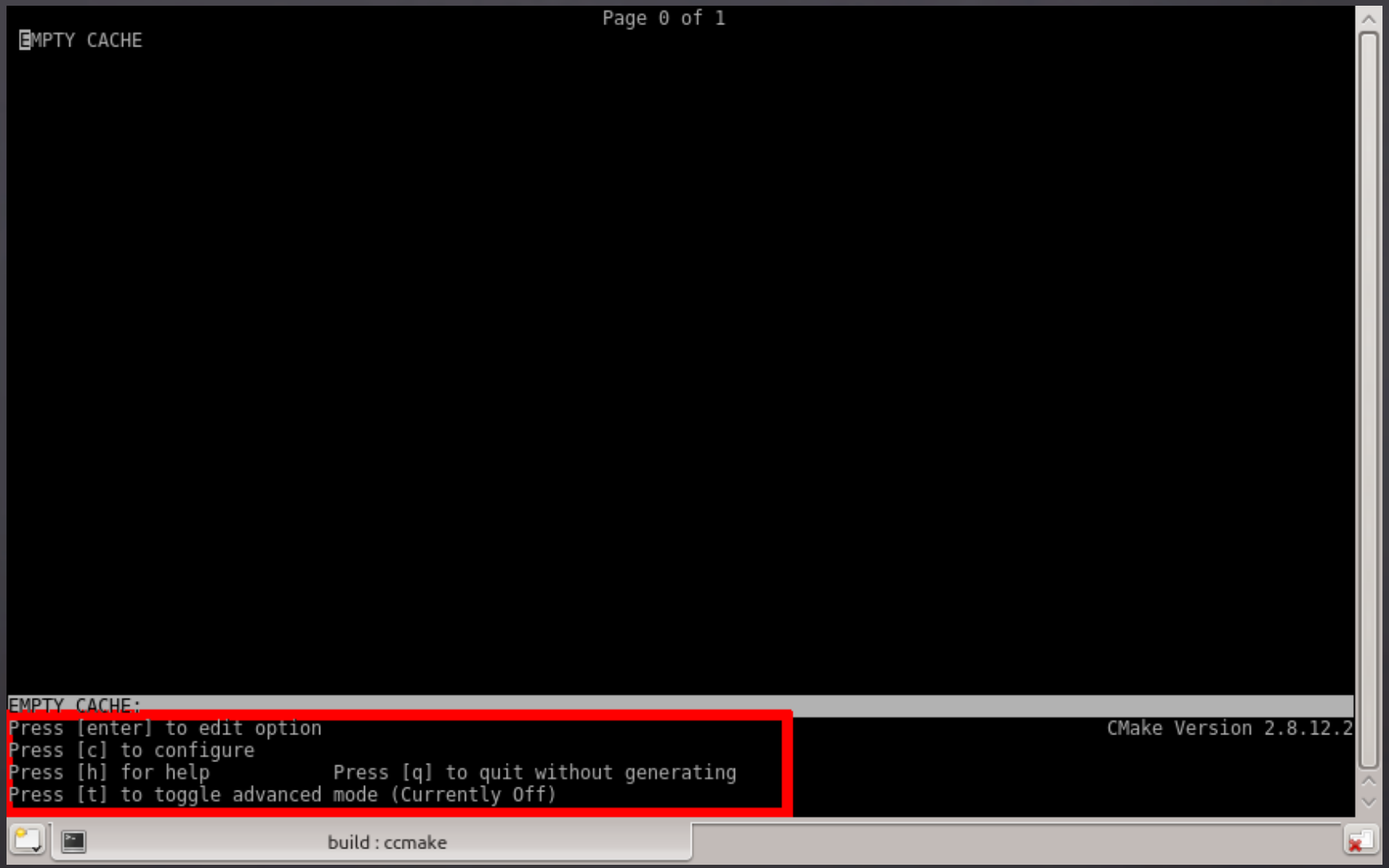
## Build Smil: CMake Linux

```
mkdir build
cd build
ccmake ../path_to_the_sources
```



# Prerequisites

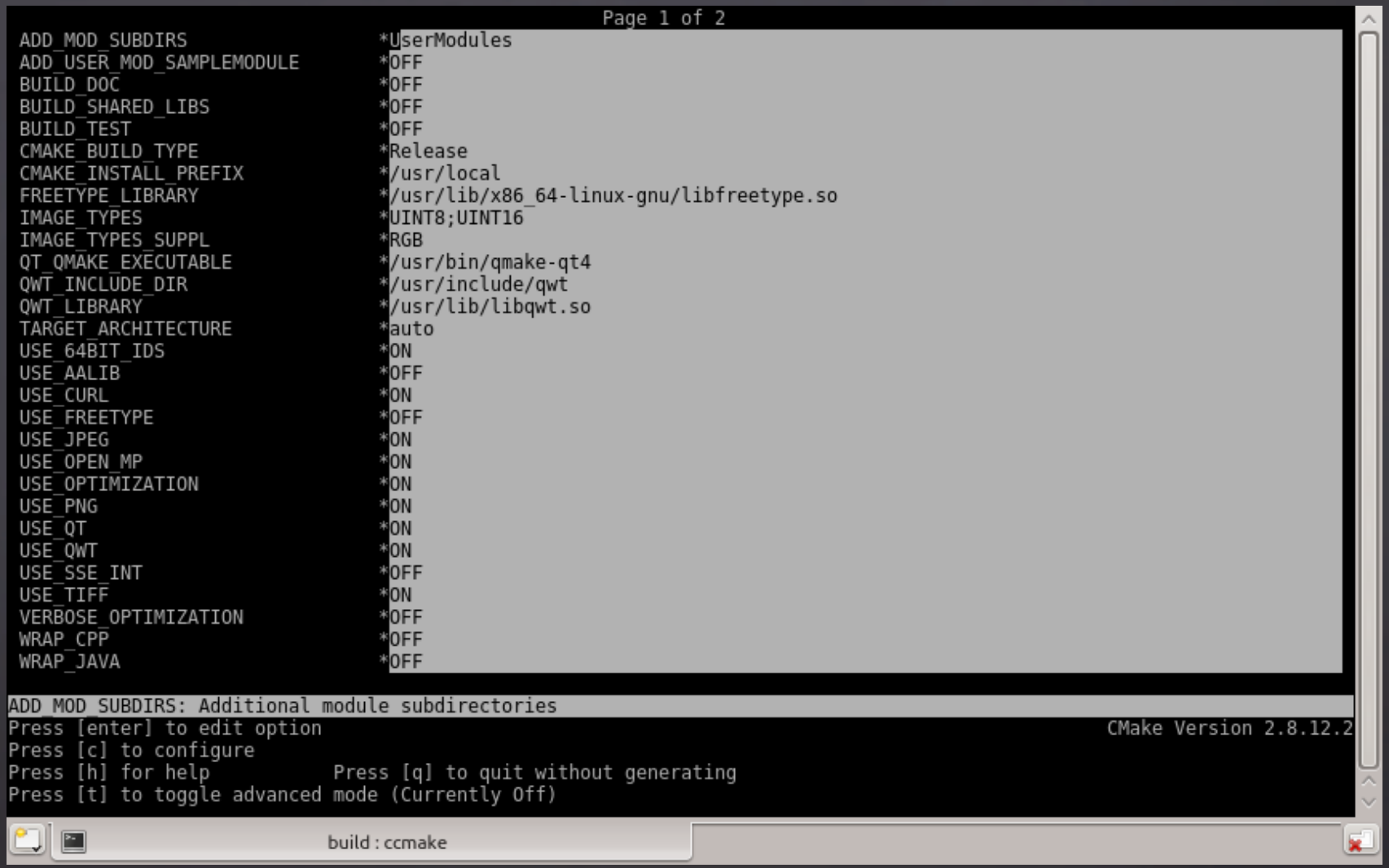
## Build Smil: CMake Linux




Press [c] to configure

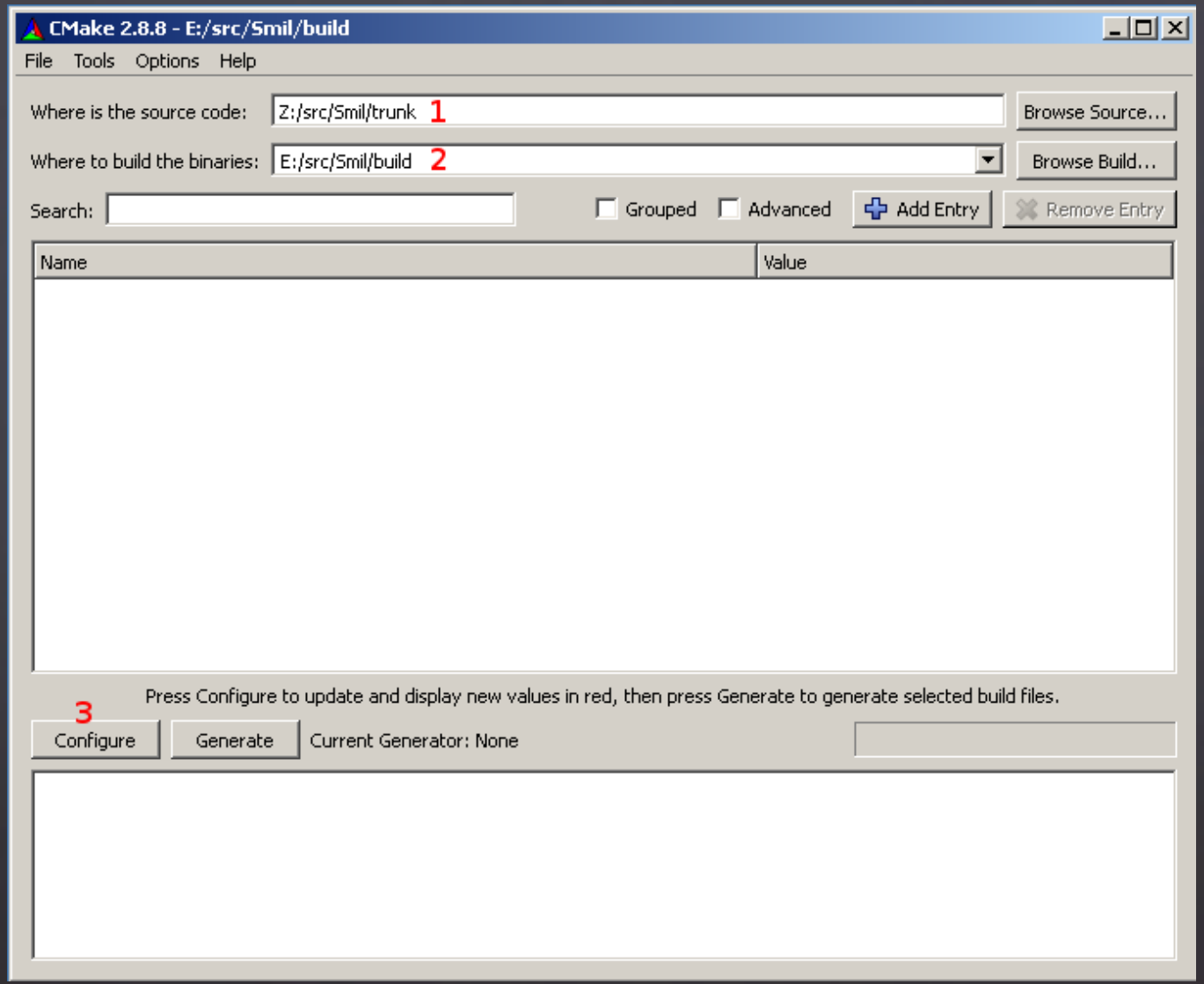
# Prerequisites

## Build Smil: CMake Linux



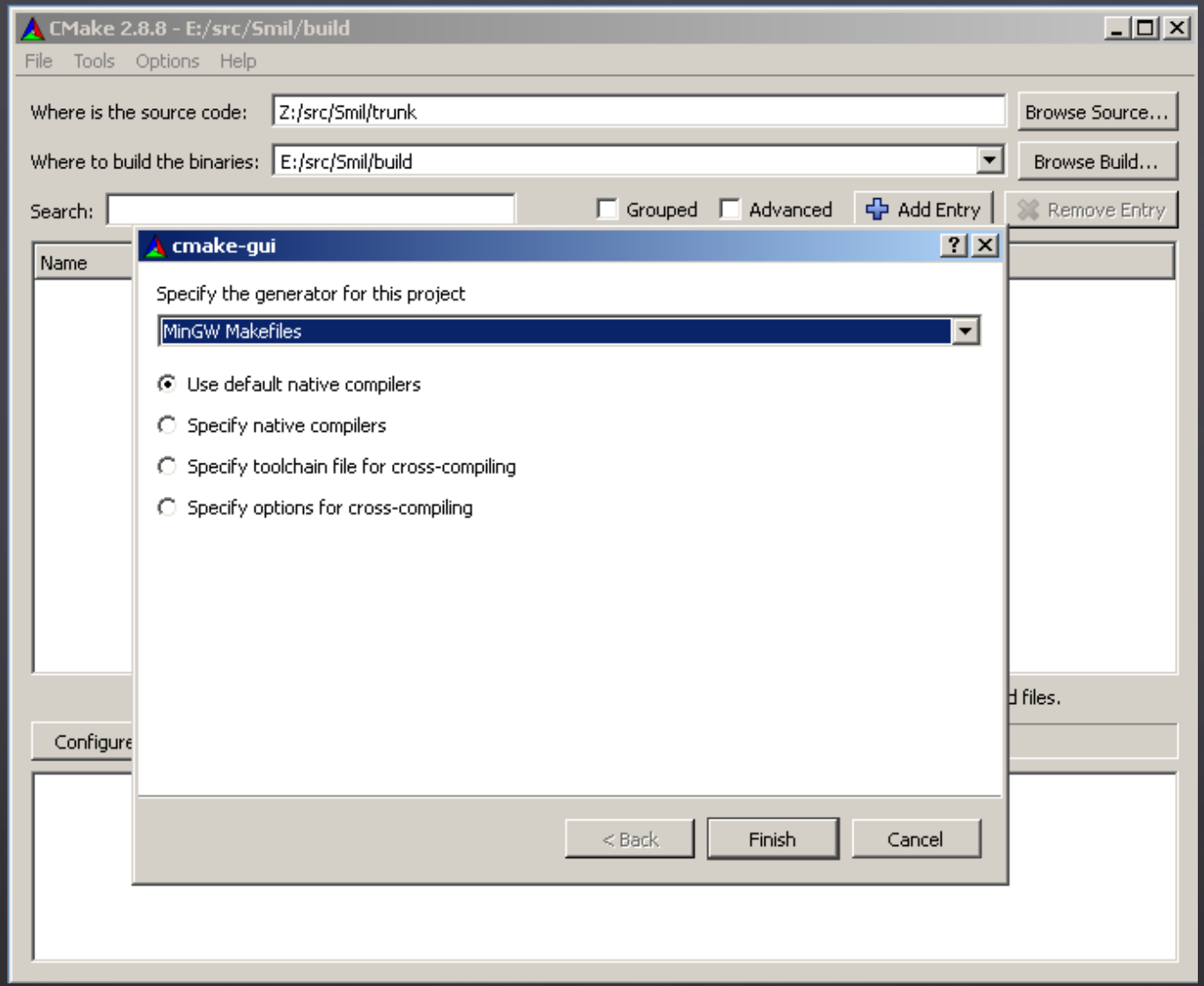
# Prerequisites

## Build Smil: CMake Windows



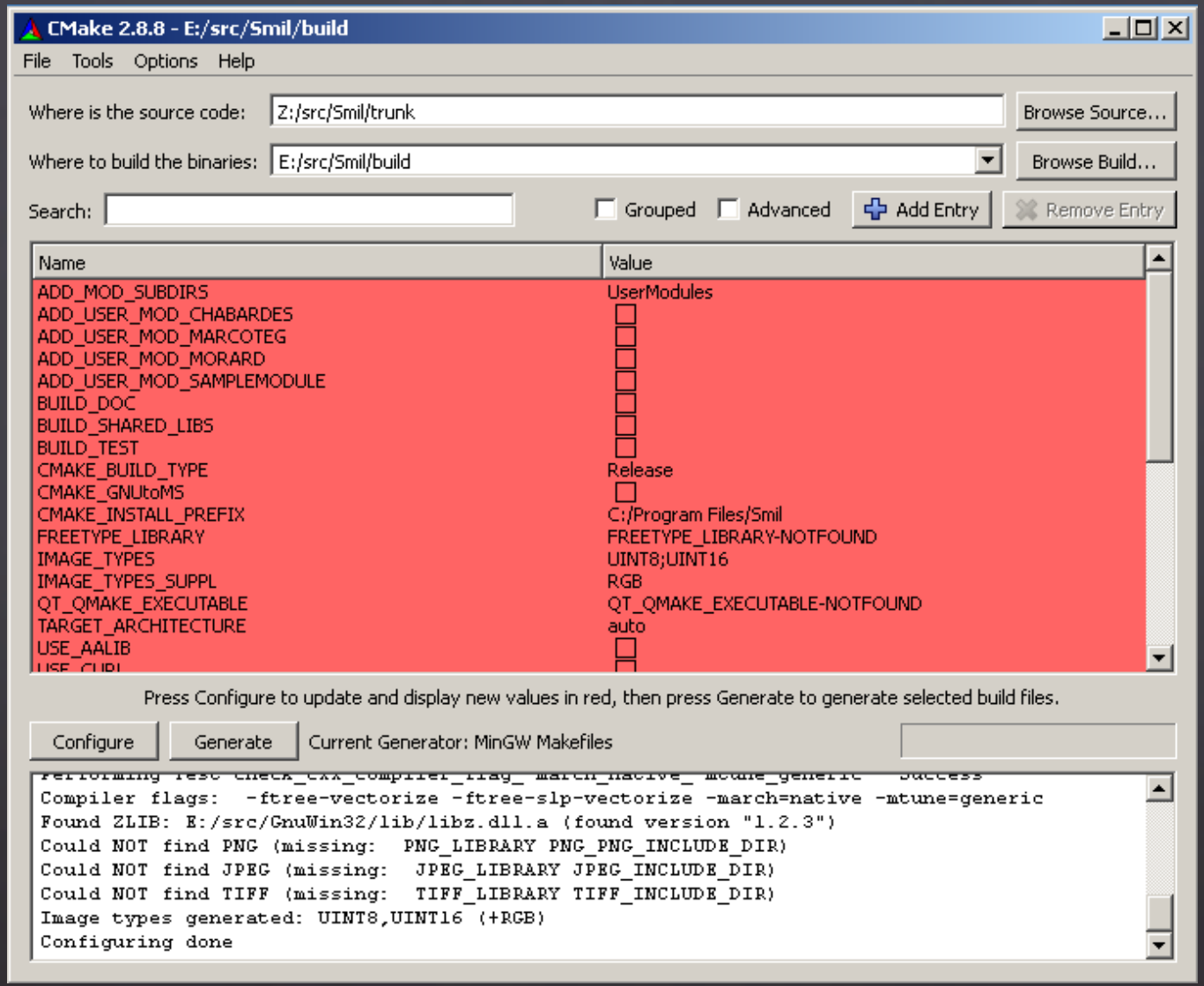
# Prerequisites

## Build Smil: CMake Windows



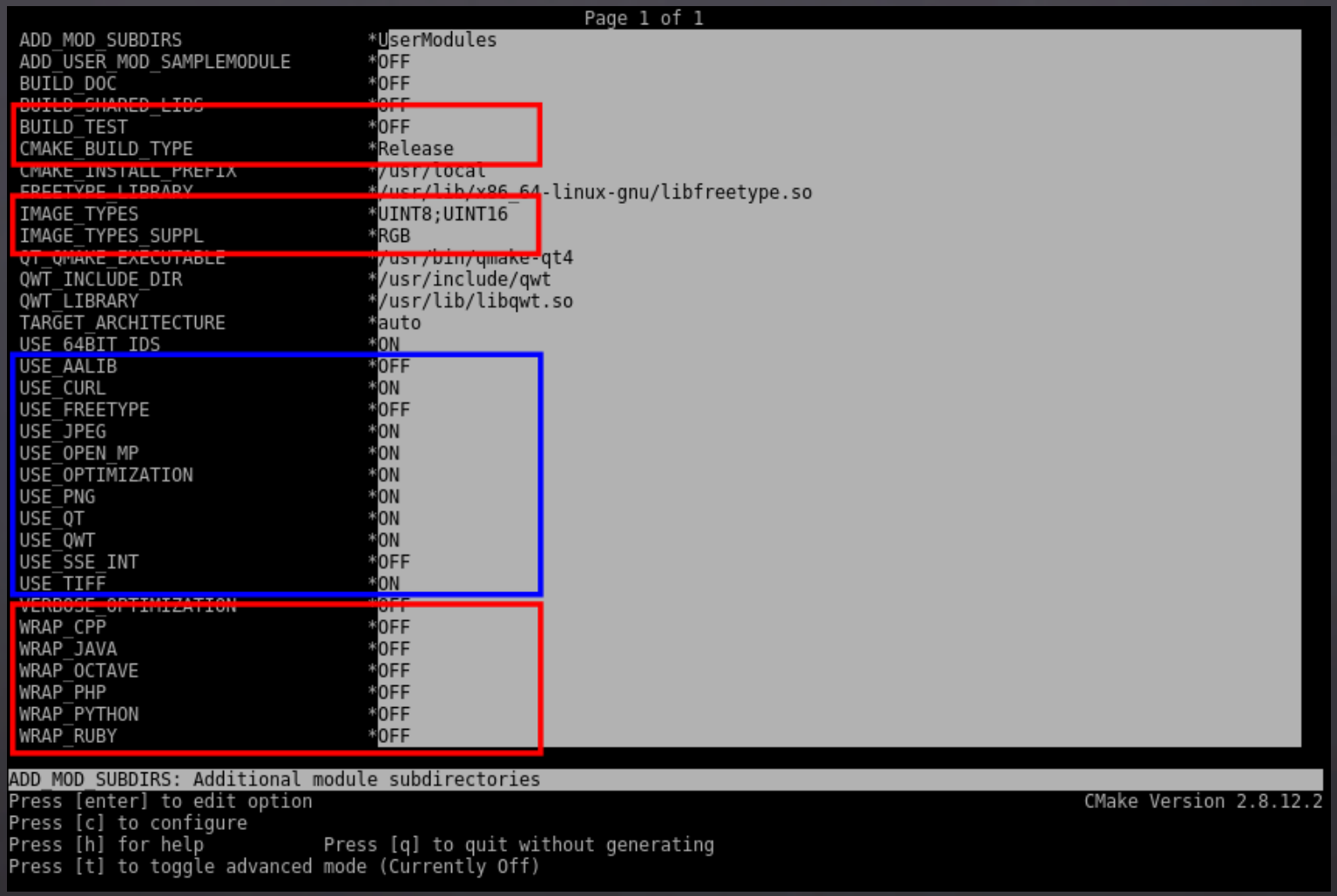
# Prerequisites

## Build Smil: CMake Windows



# Prerequisites

## Build Smil: CMake Options

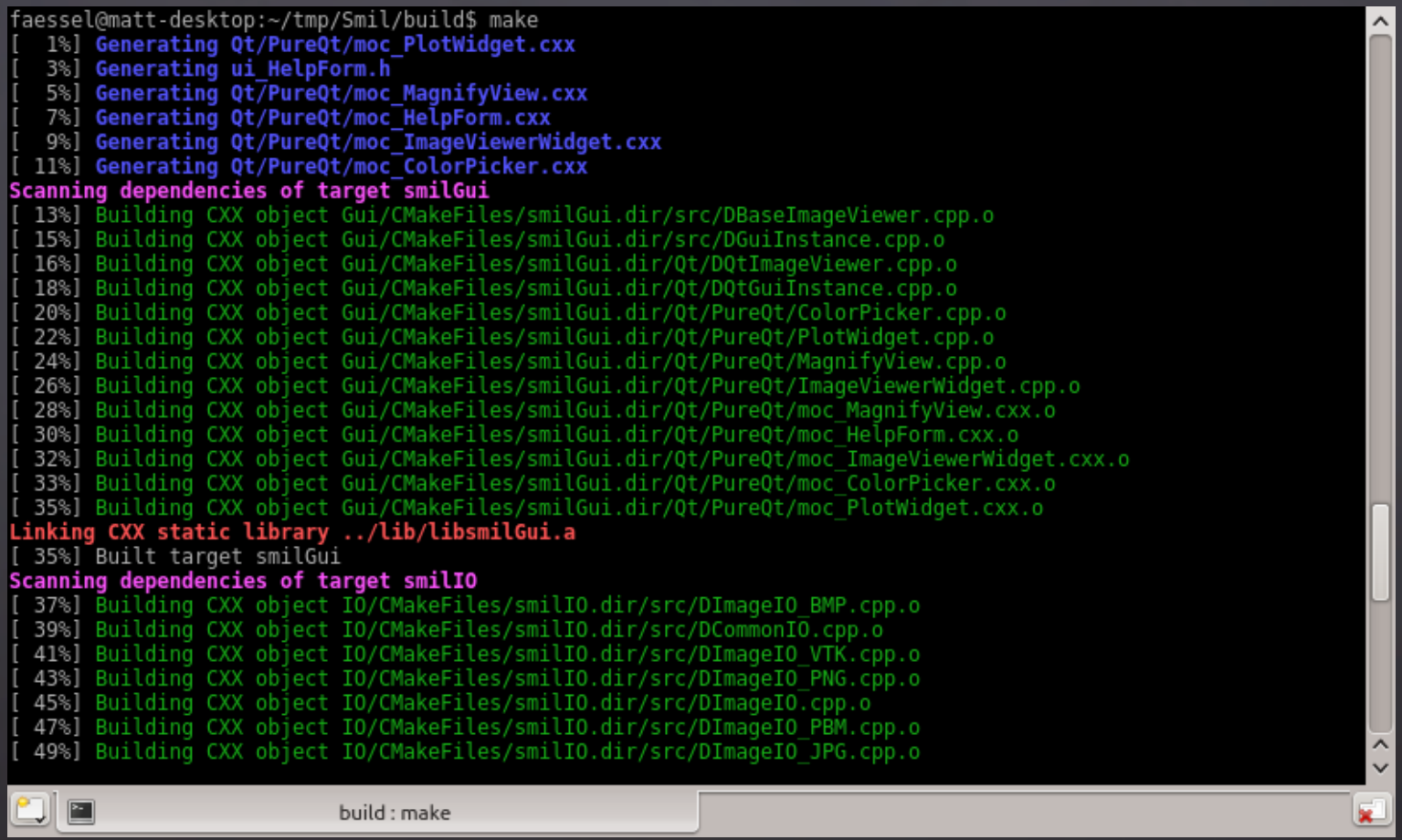



Change options/press configure [c] until everything is OK, then generate [g].

# Prerequisites

## Build Smil: make

make



# Prerequisites

## Build Smil: Build results

build/lib/

```
lib/
├── libsmilBase.a
├── libsmilBaseJava.so
├── libsmilCore.a
├── libsmilCoreJava.so
├── libsmilGui.a
├── libsmilGuiJava.so
├── libsmilIO.a
├── libsmilIOJava.so
├── libsmilJava.so
├── libsmilMorpho.a
├── libsmilMorphoJava.so
├── libsmilRGB.a
├── _smilBasePython.so
├── _smilCorePython.so
├── _smilGuiPython.so
├── _smilIOPython.so
├── smilJava
├── smil_Java.java
├── smil_JavaJNI.java
├── _smilMorphoPython.so
├── smilPython
├── smilPython.pth
└── _smil_Python.so
```

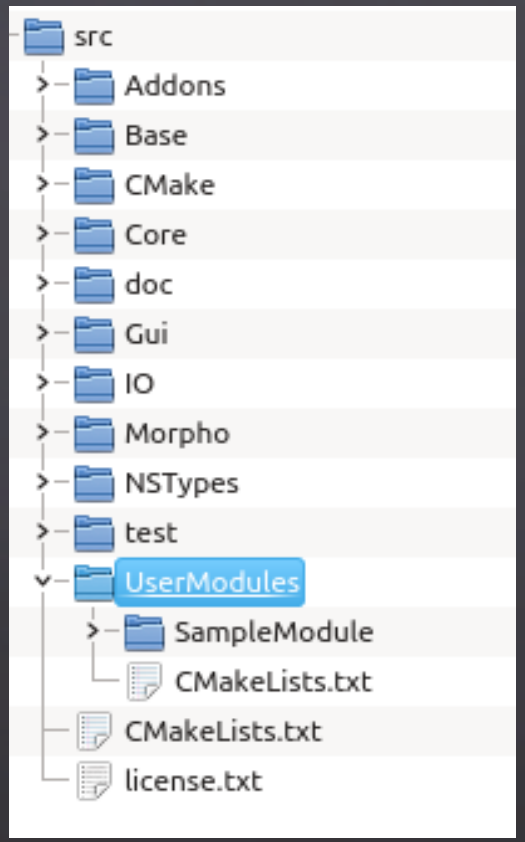
build/bin/

```
bin/
├── bench_adv_arithmetic
├── bench_base
├── bench_base_arithmetic
├── bench_blob_measures
├── bench_convolution
├── bench_distance
├── bench_hierar_queue
├── bench_matrix
├── bench_max_tree
├── bench_measures
├── bench_morpho_base
├── bench_morpho_label
├── bench_parallel_hierar_queue
├── test_arith
├── test_base_RGB
├── test_convolution
├── test_cpuid
├── test_draw
├── test_filters
├── test_global
├── test_globalMB
├── test_graph
├── test_hierar_queue
├── test_histogram
├── test_IO
├── test_IO_VTK
├── test_label_measures
├── test_matrix
```

# Create a user module

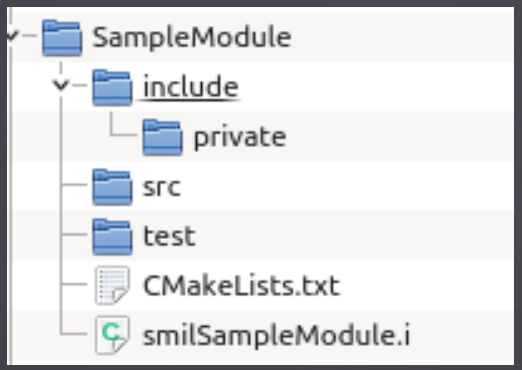
## Where ?

In the **UserModules** directory: CMake scans all its sub-directories and creates a module for each (that has the required structure/files)



# Create a user module

## What is the required structure ?




- Files (root of the module dir):
  - **CMakeLists.txt** (required)
  - **smil*NameOfTheModule*.i** (optional, needed only for wrap)
- Sub-forlders:
  - **include**: declarations files (*.h)
  - **include/private**: templates (*.hpp)
  - **src**: definition files (*.cpp)
  - **test**: test source files (test_[...].cpp)

# Create a user module

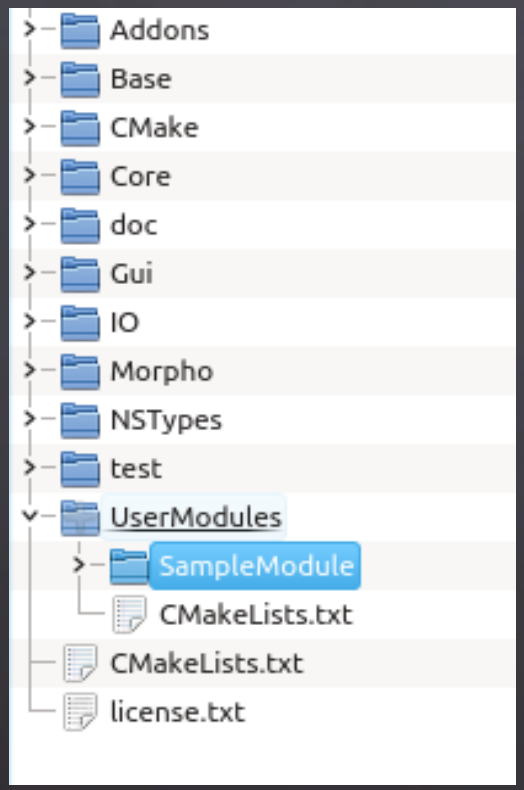
## The ~~lazy~~ easy way

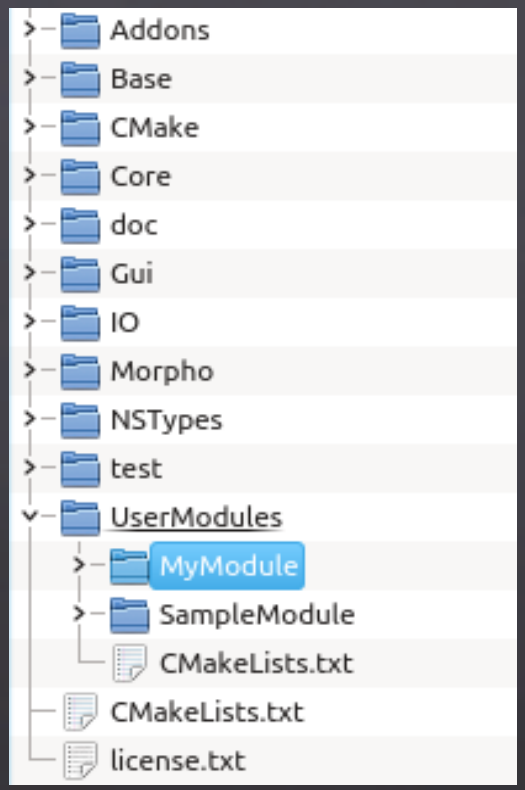
1. Copy the existing **SampleModule**
2. Rename files
3. Modify files

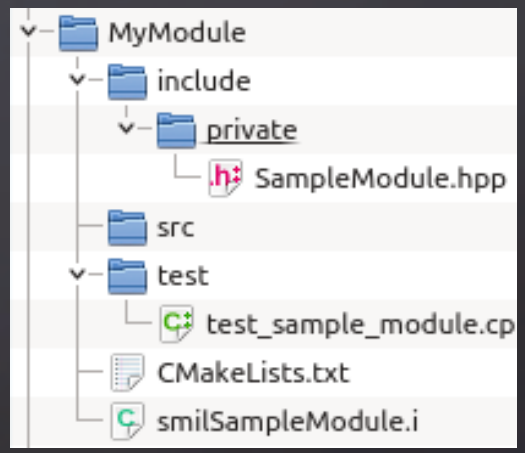
# Create a user module

## 1 - Copy the SampleModule

```
cd src/UserModules
cp -r SampleModule MyModule
```



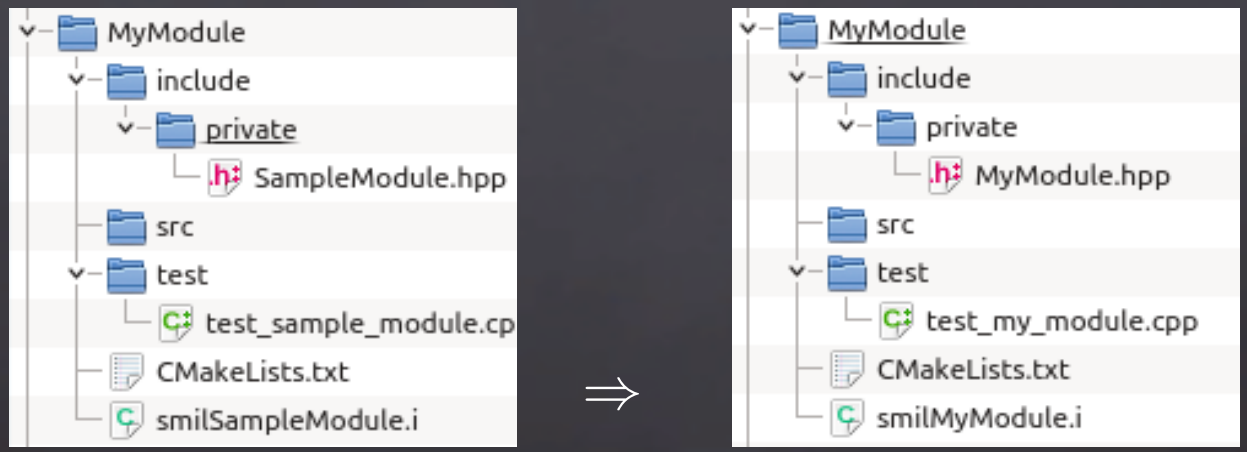
# Create a user module

## II - Rename files

In the created folder (*MyModule*), **rename**:

- **smilSampleModule.i** ⇒ **smilMyModule.i** (same as folder name)

- include/private/SampleModule.hpp ⇒include/private/MyModule.hpp (or whatever)
- test/test_sample_module.cpp ⇒test/test_my_module.cpp (or whatever)



# Create a user module

## III - Modify files

- CMakeLists.txt

```
27
28  SET(MODULE_NAME MyModule)
29  SET(MODULE_DEPS smilCore smilGui)
30
31  LIST(APPEND MODULE_DEPS)
32
33  ADD_SMIL_LIBRARY(${MODULE_NAME} ${MODULE_DEPS})
34  ADD_SMIL_TESTS(${MODULE_NAME} ${MODULE_DEPS})
35
```

- smilMyModule.i

```
30  %include smilCommon.i
31
32  SMIL_MODULE(smilMyModule)
33
34  %import smilCore.i
35  %import smilMorpho.i
36
37  %{
38  /* Includes needed header(s)/definitions in the wrapped code */
39  #include "MyModule.hpp"
40  %}
41
42  %include "MyModule.hpp"
43
44  TEMPLATE_WRAP_FUNC(samplePixelFunction)
45  TEMPLATE_WRAP_FUNC(sampleMorphoFunction)
46
```

- MyModule.hpp

```
29
30  #ifndef _MY_MODULE_HPP
31  #define _MY_MODULE_HPP
32
```

- test_my_module.cpp

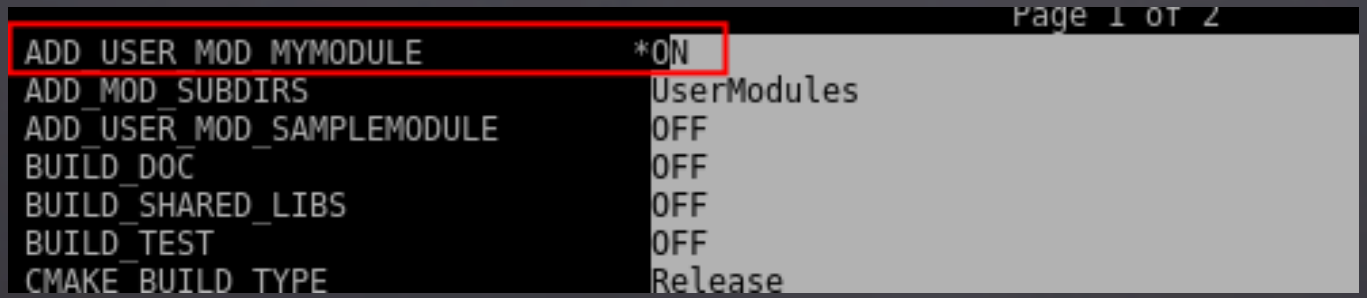
```
23
24  #include "MyModule.hpp"
25
```

# Create a user module

## Activate the module in CMake and check compilation

```
cd build
ccmake ../path_to_the_sources    (or ccmake .)
```

Set option **ADD_USER_MOD_MYMODULE** to **ON**




configure [c], generate [g] and build (make)

The new Module should be properly compiled.

# Write your image functions

## Base pixel function

(SampleModule.hpp)

```
38      // Sample inv function
39      template <class T>
40      RES_T samplePixelFunction(const Image<T> &imIn, Image<T> &imOut)
41      {
42          ASSERT_ALLOCATED(&imIn)
43          ASSERT_SAME_SIZE(&imIn, &imOut)
44
45          ImageFreezer freeze(imOut);
46
47          typename Image<T>::lineType pixelsIn = imIn.getPixels();
48          typename Image<T>::lineType pixelsOut = imOut.getPixels();
49
50          for (size_t i=0;i<imIn.getPixelCount();i++)
51            pixelsOut[i] = ImDtTypes<T>::max() - pixelsIn[i];
52
53          return RES_OK;
54      }
55
```

# Write your image functions

## Base neighbor function

(SampleModule.hpp)

```
57      // Sample Morpho functor
58      template <class T>
59      struct SampleMorphoFunctor: public MorphImageFunctionBase<T>
60      {
61          virtual inline void processPixel(size_t pointOffset, vector<int> &dOffsets)
62          {
63              double pixSum = 0;
64
65              for (vector<int>::iterator it=dOffsets.begin();it!=dOffsets.end();it++)
66                pixSum += this->pixelsIn[ pointOffset + *it ];
67
68              this->pixelsOut[ pointOffset ] = T( pixSum / dOffsets.size() );
69
70          }
71      };
72
73      template <class T>
74      RES_T sampleMorphoFunction(const Image<T> &imIn, Image<T> &imOut, const StrElt &se=DEFAULT_SE)
75      {
76          SampleMorphoFunctor<T> func;
77          return func(imIn, imOut, se);
78      }
```

# Write your image functions

## Other handy functions. Ex: flooding functors

(SampleModule.hpp)

```
82
83      // Sample (silly) Flooding functor
84      template <class T, class labelT>
85      struct SampleFloodingFunctor: public BaseFlooding<T, labelT>
86      {
87          virtual RES_T initialize(const Image<T> &imIn, Image<labelT> &imLbl, const StrElt &se)
88          {
89               BaseFlooding<T, labelT>::initialize(imIn, imLbl, se);
90               this->imgLbl->updatesEnabled = true; // Enable image repaint
91          }
92
93          virtual inline void processPixel(const size_t &curOffset)
94          {
95              this->lblPixels[ curOffset ] = UINT16(255);
96
97              BaseFlooding<T, labelT>::processPixel(curOffset);
98
99              this->imgLbl->modified(); // Trigger image repaint
100             Gui::processEvents();  // Refresh display
101
102             usleep(1000);
103         }
104     };
105
106     template <class T>
107     RES_T sampleFloodingFunction(const Image<T> &imIn, Image<T> &imOut, const StrElt &se=DEFAULT_SE)
108     {
109         SampleFloodingFunctor<T, T> func;
110         fill(imOut, T(0));
111         size_t pixNbr = imOut.getPixelCount();
112         // Put two random markers
113         imOut.setPixel(pixNbr/3, 1);
114         imOut.setPixel(pixNbr*2/3, 2);
115         return func.flood(imIn, imOut, imOut, se);
116     }
117
```

# Unit Tests

Must be in the **test** forlder and have the **test_** prefix.

Ex: test_sample_module.cpp

```
28
29  class Test_SampleModule : public TestCase
30  {
31    virtual void run()
32    {
33        UINT8 vec1[16] = {
34          1,   2,   3,   4,
35          5,   6,   7,   8,
36          9,  10,  11,  12,
37          13,  14,  15,  16,
38        };
39
40        Image_UINT8 im1(4,4), im2(im1);
41        im1 << vec1;
42
43        UINT8 vecTruth[16] = {
44          254, 253, 252, 251,
45          250, 249, 248, 247,
46          246, 245, 244, 243,
47          242, 241, 240, 239,
48        };
49
50        Image_UINT8 imTruth(im1);
51        imTruth << vecTruth;
52
53        samplePixelFunction(im1, im2);
54
55        TEST_ASSERT(im2==imTruth)
56
57        if (retVal!=RES_OK)
58        {
59            im2.printSelf(1);
60            imTruth.printSelf(1);
61        }
62
63    }
64  };
65
66  int main(int argc, char *argv[])
67  {
68        TestSuite ts;
69
70        ADD_TEST(ts, Test_SampleModule);
71
72        return ts.run();
73  }
```

# Wrap functions

Defined in the **smil*NameOfTheModule*.i** file.

Ex: smilMyModule.i

```
29
30  %include smilCommon.i
31
32  SMIL_MODULE(smilMyModule)
33
34  %import smilCore.i
35  %import smilMorpho.i
36
37  %{
38  /* Includes needed header(s)/definitions in the wrapped code */
39  #include "MyModule.hpp"
40  %}
41
42  %include "MyModule.hpp"
43
44  TEMPLATE_WRAP_FUNC(samplePixelFunction)
45  TEMPLATE_WRAP_FUNC(sampleMorphoFunction)
46  TEMPLATE_WRAP_FUNC(sampleFloodingFunction)
47
48
```

Or, with a non-template function (the function must be defined in a *.cpp file):

```
%{
/* Includes needed header(s)/definitions in the wrapped code */

#include "non_template_header.h" // ( or add directly the declaration of the function here )

%}

// Tell swig to export it:

void myNonTemplateFunction(Image<UINT8> &imIn, Image<UINT8> &imOut);
```

# Wrap functions

## Exported functions in other languages

### Octave

```
smilOctave
smilCoreOctave
smilGuiOctave
smilIOOctave
smilBaseOctave
smilMorphoOctave
smilMyModuleOctave


im1 = Image_UINT8("http://cmm.ensmp.fr/~faessel/smil/images/lena.p
im2 = Image_UINT8(im1)
sampleMorphoFunction(im1, im2, hSE(5))

im1.show()
im2.show()
Gui.execLoop()
```

### Python

```
from smilPython import *

im1 = Image("http://cmm.ensmp.fr/~faessel/smil/images/lena.png")
im2 = Image(im1)
showAll()

sampleFloodingFunction(im1, im2)
```

### Java

```
import smilJava.*;

class JavaClass
{

    protected void javaFunction()
    {
        private Image_UINT8 im1 = new Image_UINT8();
        private Image_UINT8 im2 = new Image_UINT8();

        sampleFloodingFunction(im1, im2);
    }
}
```

PHP, Ruby, ...

# Thanks

http://smil.cmm.mines-paristech.fr/