

# *Lecture 4*

## *Getting Started with ITK!*

Methods in Medical Image Analysis - Spring 2024  
16-725 (CMU RI) : BioE 2630 (Pitt)  
Dr. John Galeotti

Based in part on Damion Shelton's slides from 2006



This work by John Galeotti and Damion Shelton, © 2004-2024, was made possible in part by NIH NLM contract# HHSN276201000580P, and is licensed under a [Creative Commons Attribution 3.0 Unported License](http://creativecommons.org/licenses/by/3.0/). To view a copy of this license, visit <http://creativecommons.org/licenses/by/3.0/> or send a letter to Creative Commons, 171 2nd Street, Suite 300, San Francisco, California, 94105, USA. Permissions beyond the scope of this license may be available by emailing [itk@galeotti.net](mailto:itk@galeotti.net). The most recent version of these slides may be accessed online via <http://itk.galeotti.net/>



# Goals for this lecture



- Compile, compile, compile
  - Learn how to use SVN & CMake
  - Build ITK
  - Compile several programs that use ITK
- Find documentation online
- Learn the quirks (if any) of the system you choose to use



# Getting help



- Email your instructor
- Join the insight-users mailing list; instructions are at <http://www.itk.org>

# Assignments

- Collaboration is **encouraged**; unless told otherwise, feel free to discuss assignments with other students
- But... *please* submit your own code - don't copy and paste stuff from friends; see syllabus for details
- More so than other classes, you will be learning techniques that translate directly to the real world - don't cheat yourself

# Grading of assignments

- Grading criteria:
  - Does it accomplish the specified task?
  - Is it well commented? Follow the “6 month rule” - if you leave for 6 months, you should be able to pick up where you left off.
  - Many/most assignments will be divided into sections, with each section pass-fail.
    - We may give opportunities to fix “stupid” problems before final judgment is passed



# Assignments, cont.



- Please interpret due dates as absolute, unless told otherwise
- Really
- We're happy to spend time helping you debug code, but not at 11 pm the day before the assignment is due

# Computer requirements: recommended

- Your own computer is preferable
  - Remote-login to university computers should also work
  - Please be aware that ITK can consume a lot of disk space during the build process (feel free to use BOXDrive or Gdrive)
- Windows 10 or 11 with Visual Studio 2019; Python 3.9-3.11(x64)
  - Suggest Win11
  - Suggest Visual Studio (Code or Enterprise) 2019 and Anaconda 3.11
    - Get [“Microsoft Azure Dev Tools for Teaching”](#) from CMU
    - Newer versions should work, but we haven’t verified
- Mac OS X, X-Code, user-installed Anaconda Python 3.9-3.11, either x64 or M1 (M1 = amd64, includes M2, M3, etc.)
- Recent x64 Linux (e.g. Ubuntu, RHEL, or CentOS), system gcc; user-installed Anaconda Python 3.9-3.11



# What am I using?



- Mac OS X Sonoma:
  - Anaconda Python 3.11 for M1 (M1 is the same as arm64)
  - Latest Xcode



# Alternative usable computer configurations

- Any platform supported by ITK (Mac, Linux, etc.)
- **If there are problems, you will have to work with us to get your code working on one of our machines.**
  - Try having us check your code before it is due.
- If the grader's computer can't run your code, you will have a short (but reasonable) period of time to fix it after he emails you that your code appears broken (along with what errors he got).
  - If you are trying to make things work, but have many things to "fix," then more time may be granted.
  - For final projects, we *may* decide to let you show us your code running on your own machine, on a case-by-case basis.

# What is ITK?

- To clarify, ITK is a source-code/library *toolkit*
  - It doesn't "do" anything
  - You can't "run" it
  - There isn't an itk.exe file
- Typically, you use ITK in conjunction with other toolkits to handle visualization and GUI interaction

# So, what's it good for?

- ITK code is easy to add to existing C++ code
  - Also Python, C#, Java, ...
- It provides a variety of flexible data containers, and ways of processing / analyzing them
- You can do a lot in only a few lines of code
- Once you get used to it, it's easy to use (gasp!)

# What we assume you can do

- Understand C++ and/or Python syntax
  - Standard flow control such as for, do, calling functions, etc.
  - Classes
  - Inheritance
    - For C++: Pointers, dereferencing, passing by reference
- Work comfortably in the operating system of your choice, using the compiler or Python environment of your choice, running programs from the command line (i.e. terminal)



# You may have not...



- Written C++ code that builds on multiple platforms
- Used cross-platform make software
  - (CMake or Jam, for example)
- Designed software using a data-flow architecture, worried about smart pointers, etc.

# Cross platform (C++) development

- ITK builds on a large combination of operating systems and platforms
- For C++, each compiler has it's own input format: Makefiles, workspaces, etc.
- Q: How can you possibly coordinate builds on different platforms?

# The answer: CMake

- Cross platform tool to manage the build process
- Simplifies the build process
- Auto-configuration
- Easy access to external libraries
- Used by several other open source projects



[www.cmake.org](http://www.cmake.org)

A screenshot of the CMake Wiki homepage. The browser address bar shows 'http://www.cmake.org/Wiki/CMake'. The page title is 'CMake'. Below the title is the CMake logo and a welcome message: 'Welcome to CMake, the cross-platform, open-source make system. CMake is used to control the software compilation process using simple platform and compiler independent configuration files. CMake generates native makefiles and workspaces that can be used in the compiler environment of your choice. CMake is quite sophisticated: it is possible to support complex environments requiring system configuration, pre-processor generation, code generation, and template instantiation. You will find here not only documentation for CMake, but also for CPack and CTest.' Below this is a 'Contents [hide]' section with a tree view of the site structure. Under '1 CMake', there are sub-sections for '1.1 Primary Resources - Look here first!', '1.2 Development Topics', '1.3 Tutorials' (with sub-sections 1.3.1 Basic Introductions, 1.3.2 Specific Topics, 1.3.3 Recipes), '1.4 Converters from other buildsystems to CMake' (with sub-sections 1.4.1 automake/autotools/autoconf, 1.4.2 qmake, 1.4.3 Visual Studio, 1.4.4 Basic CMakeLists.txt from-scratch-generator), '1.5 Success Stories', and '1.6 More Topics'. Under '2 CTest', there are '2.1 Tutorials', '2.2 More Information', and '2.3 More Topics'. Under '3 CDash' and '4 CPack', there are '4.1 Tutorials' and '4.2 Recipes'. Below the contents is another 'CMake' section with a sub-section 'Primary Resources - Look here first!' containing a list of links: 'Where can I download CMake?', 'CMake Documentation', 'Useful CMake Variables', 'FAQ (Frequently asked questions)', 'CMake Mailing List (for searchable archives see CMake FAQ)', 'CMake 2.6 Notes', and 'Getting Started With CMake Screencasts @PlayControl.net'. Below that is a 'Development Topics' section with a list of links: 'Cross compiling', 'RPATH handling', 'Assembler Support', 'Editors/IDEs with CMake syntax support', 'Docs for Specific Project Generators (Eclipse, KDevelop3, CodeBlocks, Makefile)', 'Contributed macros', 'Module Maintainers', 'Platform Dependent Information', 'Documentation for previous releases', and 'Matrix for checking backwards-compatibility of current features'.

# CMake is:

- Required to build native (C++) ITK
- Cross-platform project generator
- Often simpler than particular environments
- Text as input
- Project file as output:

<b>Windows</b>	Visual Studio Solution
<b>UNIX</b>	Makefile or Ninja
<b>Mac OS X</b>	Xcode project or Ninja or Makefile



# How CMake runs

- Write a **CMakeLists.txt** file describing your project in CMake's language
- Run CMake to generate an appropriate makefile/project/workspace for your compiler
- Compile as you normally would

## How CMake runs, cont.

- This is not unlike the configure-make process you may be familiar with from various Unix systems
- But... it works with many compilers
- CMakeLists.txt files are easy to perform revision control on

# CMakeLists.txt syntax

- Comment lines indicated with #
- Look at examples in ITK
- Simple example:

```
cmake_minimum_required(VERSION 3.28)  
# Make sure the user's CMake is recent enough  
# CMake policies sometimes change, so it is  
# recommended that new projects require the  
# latest CMake version  
  
#Give this project a name:  
PROJECT(cool_demo)  
  
#The command-line executable "demo"  
#is built from "demo_code.cxx" and  
#must be linked with the ITK libraries  
ADD_EXECUTABLE(demo demo_code.cxx)  
TARGET_LINK_LIBRARIES(demo ${ITK_LIBRARIES})
```

# Full Example of CMakeLists.txt

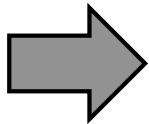
```
CMAKE_MINIMUM_REQUIRED(VERSION 3.28)  
  
# specify the C++ standard; ITK5.4 uses  
  C++17  
set(CMAKE_CXX_STANDARD 17)  
set(CMAKE_CXX_STANDARD_REQUIRED True)  
  
PROJECT(cool_demo)  
  
# Find ITK:  
FIND_PACKAGE(ITK REQUIRED)  
INCLUDE(${ITK_USE_FILE})  
  
ADD_EXECUTABLE(demo demo_code.cxx)  
TARGET_LINK_LIBRARIES(demo ${ITK_LIBRARIES})
```

# Steps to get started with C++ ITK & SimpleITK

- Pay Attention
- This is important for *everyone* for HW2
- If you are compiling C++ SimpleITK, you do *not* need to compile ITK separately
  - Compiling C++ SimpleITK using the “SuperBuild” approach automatically downloads and compiles ITK for us—we’ll explain more shortly
- I suggest you make a directory to hold all your source code and build trees for this class, e.g. c:\bmia
  - Windows WARNING: Do not use long directory names on Windows, or place the directory in your Documents folder, or else the total path names will become too long for Windows and building (Simple)ITK will fail.

# Step 0 – Don't panic!

- There is *substantial* documentation on everything I'm going to present here, and vastly more about things that we will never cover in this course
- <https://simpleitk.readthedocs.io/>
- <https://simpleitk.readthedocs.io/en/master/faq.html>
- ITK C++ API: <https://itk.org/Doxygen/html/index.html>
- Download a copy of the *ITK Software Guide*
  - <https://itk.org/ItkSoftwareGuide.pdf>



# Step 1 - Install CMake

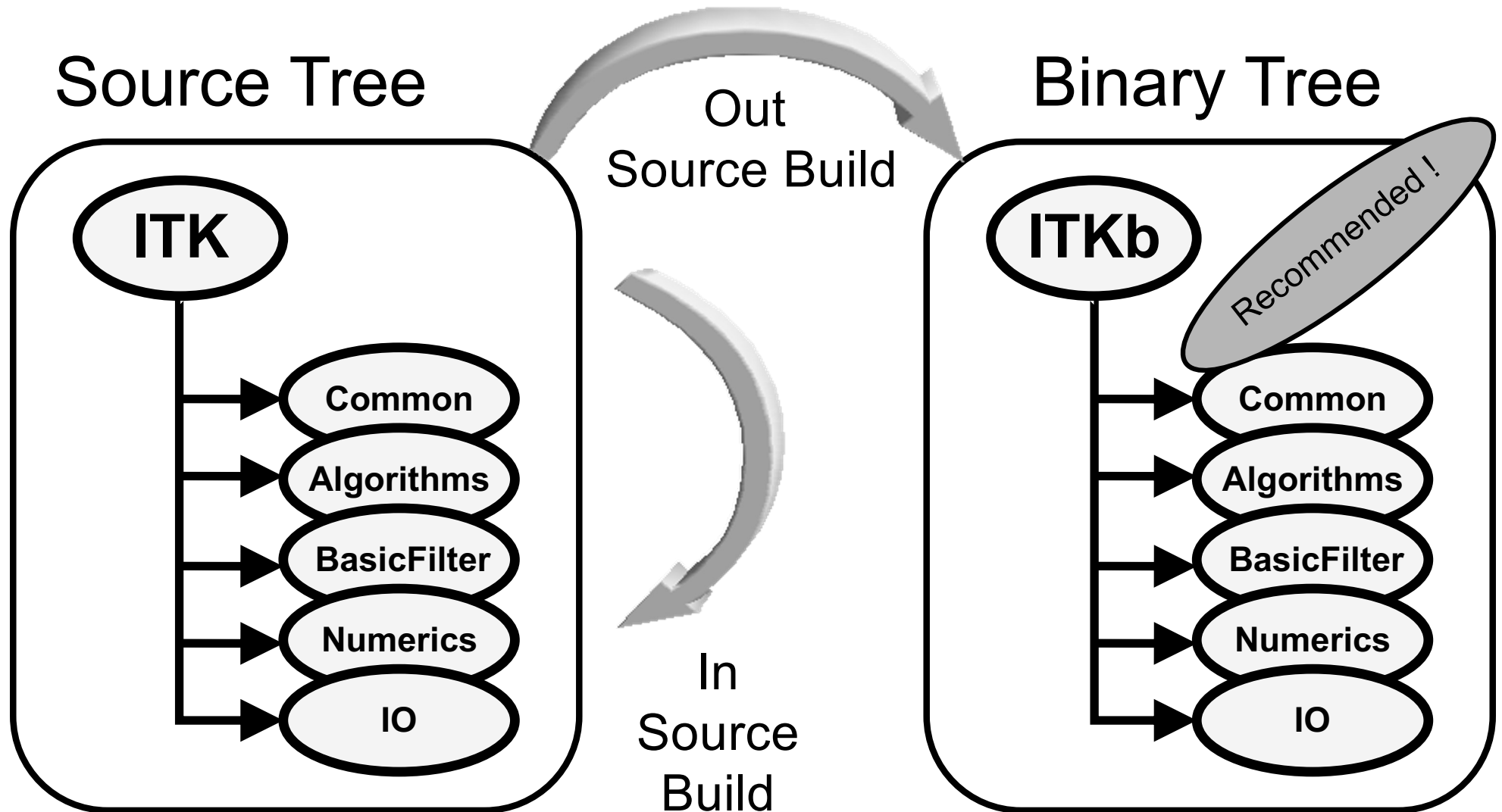
- Check if CMake  $\geq 3.18$  is already installed on your computer.
- If not, ...
- Download and install the latest stable binary distribution of CMake from:
  - <http://www.cmake.org/>

## Step 2 - Install (Simple)ITK

- Note: If you are going to compile SimpleITK (we are), then do *not* install ITK directly
  - SimpleITK's "SuperBuild" will compile & install ITK for you
  - ...Following a procedure similar to that for ITK
  - <https://simpleitk.readthedocs.io/en/master/building.html>
- Create a directory to hold the source and build for both SimpleITK and ITK
- Open a command prompt and go to the new directory
- Download the latest version of SimpleITK using git:
  - **git clone <https://github.com/SimpleITK/SimpleITK.git>**



# In source vs. out source builds



# Why use two trees?

- Keeps your C++ source and binary code separate
- Minimizes the amount of damage you can do to your SVN tree
- We suggest that you build SimpleITK in a new folder you create named SITKBin
- **WARNING:** Because ITK is downloaded automatically by building SimpleITK, both the source and the build directories for ITK will be found inside the SITKBin directory

# Where Will Everything Be?

- After building the SimpleITK SuperBuild as instructed, your important stuff will be here:
  - SimpleITK Source:
    - **.\SimpleITK**
  - SimpleITK Build:
    - **.\SITKBin**
  - SimpleITK C++ Libraries (one or both of these two):
    - **.\SITKBin\lib{\Release OR \Debug may be present}**
    - **.\SITKBin\SimpleITK-Build\lib{\Release OR \Debug}**
  - ITK Source:
    - **.\SITKBin\ITK**
  - ITK C++ Build:
    - **.\SITKBin\ITK-build**
  - ITK C++ Libraries:
    - **.\SITKBin\ITK-build\lib{\Release OR \Debug may be present}**



# Configuring a SimpleITK Build: Easy Start

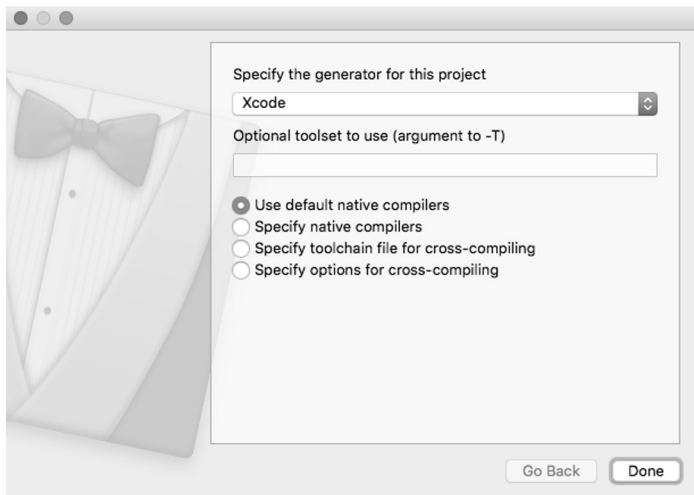


- Run CMake
- Select the SOURCE directory:
  - `.\SimpleITK\SuperBuild`
- Select the BINARY directory:
  - `.\SITKBin`
- Press the “Configure” button

# CMake: Choosing a Compiler

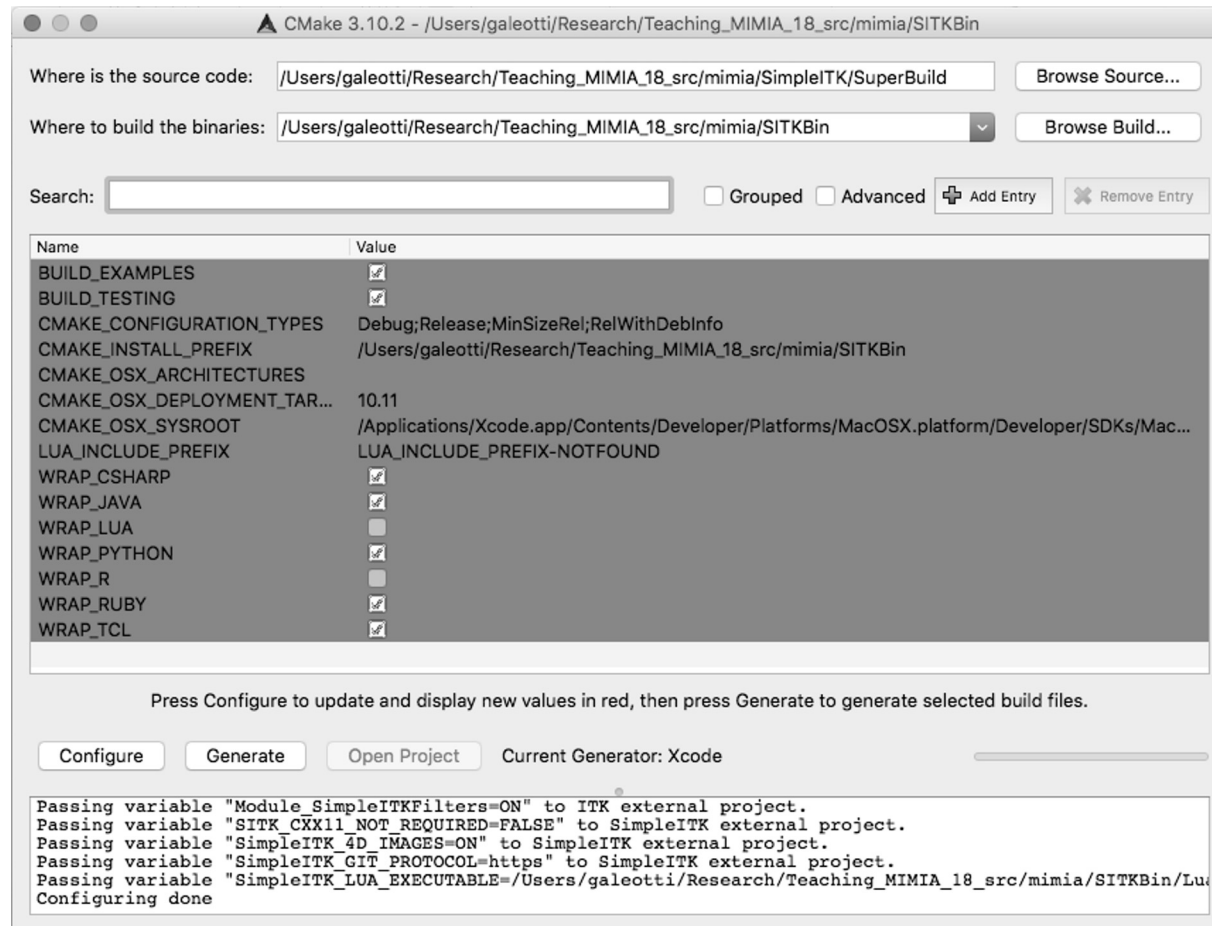
CMake may ask to choose which compiler (“generator”) you want to use for compiling:

On my OS X system, here are the choices:



- On Windows, choose your version of **Visual Studio**
- On OS X, choose **Xcode** (recommended) or “Unix Makefiles”
- On Linux, choose an appropriate Makefiles or Ninja option

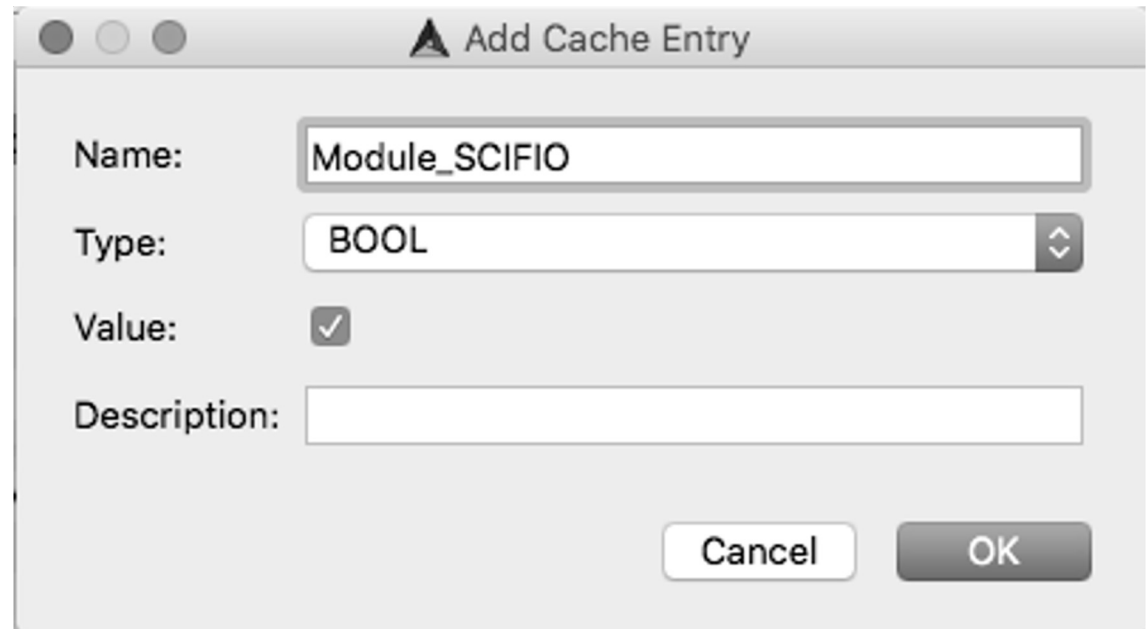
# Configure - Easy Start, cont.



If you get a message about needing Java, just click ok.

# Configure - Easy Start, cont.

- Disable each of these:
  - BUILD\_EXAMPLES
  - BUILD\_TESTING
  - WRAP\_\*, e.g. WRAP\_JAVA, WRAP\_LUA, etc.
- To read special microscopy-format images
  - Click the “Add Entry” button
  - Fill in the dialog as shown



The screenshot shows a dialog box titled "Add Cache Entry". It contains the following fields and controls:

- Name:** A text input field containing "Module\_SCIFIO".
- Type:** A dropdown menu with "BOOL" selected.
- Value:** A checkbox that is checked.
- Description:** An empty text input field.
- Buttons:** "Cancel" and "OK" buttons at the bottom right.

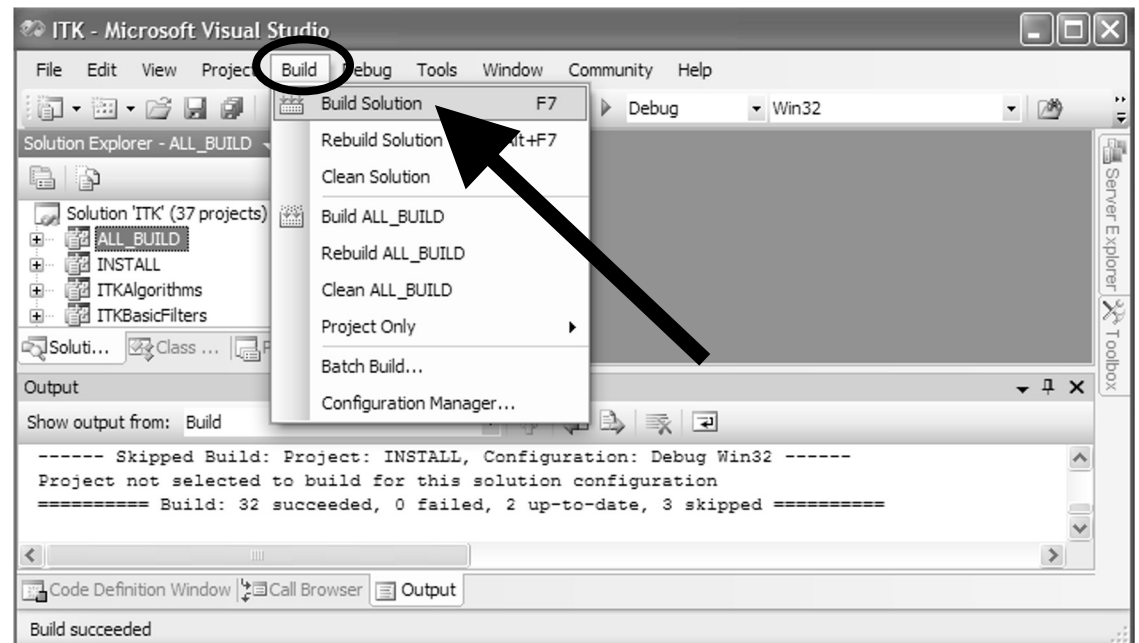
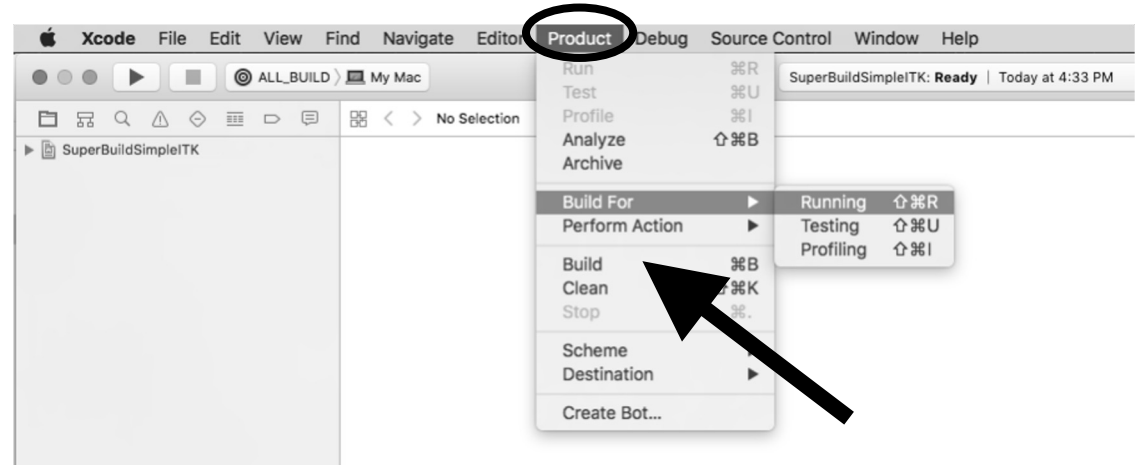
# Configuring and Generating

- After you change an option or options you will need to “configure” CMake again
- If the generate button (“OK” under Windows) is not presented, you definitely need to click the “configure” button again
- If any of the options are highlighted in red, you need to click the “configure” button again
- When done, click either “Generate” or “OK”
  - Generating is usually very fast



# Build ITK

- You are now ready to actually compile C++ SimpleITK and ITK
- In Cmake, click “Open Project”
  - This will open up Visual Studio or Xcode, etc. for your SimpleITK project
- Visual Studio:
  - Build→Build Solution
- Xcode:
  - Product→Build
- It will probably take somewhere between 30-300 minutes, but your time may vary a lot



# Verify the Build

SimpleITK Libraries will be found in:

SITK\_BINARY / { SimpleITK-Build / } lib / { Debug, Release }

ITK Libraries will be found in:

SITK\_BINARY / ITK-build / bin / { Debug, Release }



# Alternative Approach: Building with Makefiles and gcc



- **WARNING:** Do not follow this alternative approach if you already followed the previous CMake instructions.
- Order of operations is the same
- Differences
  - Can only configure for a debug OR a release build
  - Run the ccmake executable, with **one** of these:
    - -DCMAKE\_BUILD\_TYPE=DEBUG
    - -DCMAKE\_BUILD\_TYPE=RELEASE
  - ccmake uses a curses TUI, “identical” to the GUI
  - Run make instead of Visual Studio
  - Think of CMake as replacing the “./configure” step you may be used to

# Alternative Approach: Building with Makefiles and gcc

Start in the directory *containing* SimpleITK

```
mkdir SITKBin  
cd SITKBin  
ccmake -DCMAKE_BUILD_TYPE=RELEASE  
        -Module_SCIFIO=ON  
        ../SimpleITK/SuperBuild
```

Edit CMake options

Reconfigure if needed

```
make
```



# Now what?



- At this point, you have a bunch of source code and a bunch of compiled libraries
- As mentioned earlier, you don't yet have anything executable

# Reminder: What Will Be Where

- SimpleITK Source:
  - **.\SimpleITK**
- SimpleITK Build:
  - **.\SITKBin**
- SimpleITK C++ Libraries (one or both of these two):
  - **.\SITKBin\lib{\Release OR \Debug may be present}**
  - **.\SITKBin\SimpleITK-Build\lib{\Release OR \Debug}**
  
- ITK Source:
  - **.\SITKBin\ITK**
- ITK C++ Build:
  - **.\SITKBin\ITK-build**
- ITK C++ Libraries:
  - **.\SITKBin\ITK-build\lib{\Release OR \Debug may be present}**

# Building an application

- ITK comes with a simple application you can build in order to test the ITK libraries “out of source” (i.e. not built inside ITK)
- It can be found in:  
**SITKBin/ITK/Examples/Installation**

# How to build HelloWorld

- Copy & rename the *Installation* directory somewhere outside of the Insight directory
- Run CMake on **HelloWorld**
  - Remember the source/binary distinction and use **HelloWorldBin** as your build location
- CMake should automatically find ITK
  - if not, edit the **ITK\_DIR** option



# How to build HelloWorld, cont.

- Once CMake is happy, generate the Makefile/project for your compiler
- Build HelloWorld
- Give it a try

# More examples

- You can turn on ITK's *Examples* option in CMake, which will build all of the examples for you
- Or... you can copy the examples out-of-source and build them like you did HelloWorld
- These examples link into ITK Software Guide; read the chapter, poke the code and see what happens...



# C++ Workflow thoughts



You should get used to the idea of:

1. Writing some code
2. Writing a CMakeLists.txt file
3. Running CMake
4. Building your code
5. Rinse, repeat

## An aside: how to use ITK with existing C/C++ applications

- Your existing app may not use CMake
- In this case, you need to link to the ITK libraries explicitly and include the appropriate source directories
- This isn't hard, but it may take some trial and error to discover everything you need
- You don't need to worry about this in the context of this class



# ITK Documentation



- Most of the ITK documentation is generated automatically from source comments using Doxygen
- Please familiarize yourself with the various means of navigating the Doxygen documentation online, e.g. selecting a Module and then selecting a Class
- Don't forget the URLs on slide 22 ("Step 0").

# More Great News! ITK in Python

- You can now install a (slightly) simplified version of *full* ITK in Python:
  - **`conda install -c conda-forge itk`**
- Great news if you need some of ITK's more advanced functionality, but only use Python
- Examples: <https://discourse.itk.org/t/itk-5-0-beta-1-pythonic-interface/1271>