

Make, CMake, Doxygen

Wykład 2

Kontakt

Dr hab. inż. Bartosz Mindur

D11 p. 127 (Kawiory 26A)

Konsultacje: po umówieniu się emailam

email: bartosz.mindur@agh.edu.pl

Temat w każdym email-u: [PREOOP] ...

Nie procesuję i nie odpowiadam na maile wysłane na zły adres lub bez odpowiedniego tagu w tytule!!!

Make - wstęp

- Program **make**
 - Uruchamia komendy niezbędne do kompilacji i linkowania programu(-ów) który chcemy zbudować
 - Make czyta i interpretuje określone przez nas reguły zawarte w pliku reguł
 - Domyślna nazwa pliku reguł to **Makefile**
 - Można oczywiście podać inną ale jest to rzadko stosowane
 - **make** będzie próbował odwołać się do pliku **Makefile** w bieżącym katalogu
 - Jeżeli go nie znajdzie to do pliku makefile (chyba że nie ma rozróżniania wielkości liter w systemie plików)
 - Ewentualnie wskażemy który plik nas interesuje
 - **make -f ścieżka_do_pliku_reguł**
 - Istnieje kilka wersji programu make
 - Np. GNU make, BSD make
 - Różnią się głównie ilością i wartościami predefiniowanych parametrów

Składnia pliku makefile

- Wszystkie linie zaczynające się od znaku # traktowane są jak komentarze
- W pliku make definiuje się komentarze wg następującego formatu
 - `Etykieta1: zaleznosc1 zaleznosc2 ...`
`<TAB> komenda1`
`<TAB> komenda2`
`<TAB> ...`
`Etykieta2: ...`
`<TAB> ...`
`...`
- Dzięki takiemu formatowi możemy definiować dowolną ilość reguł oraz zależności
 - Nie dotyczy to tylko kompilacji i linkowania
 - Można dodać elementy wykonywane przed i po kompilacji lub zupełnie niezależne polecenia
 - **Makefile** i program **make** można stosować także do zupełnie innych celów niż tworzenie oprogramowania
 - Przykład?

Trywialny przykład

- Mamy program w jednym pliku źródłowym
 - Chcemy go zbudować w możliwie najprostszy sposób
 - `program: main.cpp`
`g++ main.cpp -o program`
 - Jedna reguła i jedna zależność
 - Co jest ukryte w tym poleceniu?

Trywialny przykład

- Mamy program w jednym pliku źródłowym
 - Chcemy go zbudować w możliwie najprostszy sposób
 - program: `main.cpp`
`g++ main.cpp -o program`
 - Jedna reguła i jedna zależność
 - Co jest ukryte w tym poleceniu?
 - Przykład 1
 - Nieco lepsze podejście
 - Podzielenie na kompilację i linkowanie wraz z odpowiednimi regułami
 - program: `main.o`
`g++ main.o -o program`
 - `main.o: main.cpp`
`g++ -c main.cpp -o main.o`
 - Przykład 2

Etykiety

- Jeśli nazwa etykiety będzie identyczna jak nazwa pliku, który otrzymamy w wyniku przetwarzania, to wówczas składniki nie będą podlegać ponownemu przetwarzaniu bez potrzeby
 - Oczywiście zależy to od czasu modyfikacji i ich wzajemnej zależności
 - **Jak?**
- Wywołanie programu **make** spowoduje wykonanie komend i zależności z pierwszej napotkanej etykiety w pliku **makefile**
 - Dlatego często na górze podaje się etykietę **all**
 - Czyli:
 - **all: program**
- Jeśli chcemy zrobić coś innego wywołanie powinno wyglądać następująco **make etykieta**
- Istnieje zestaw standardowych etykiet
 - **all, clean, test**
- Etykiety specjalne **.PHONY** (i inne)
- **Przykład 3**

Zmienne

- **make** umożliwia definiowanie zmiennych w sposób
 - **nazwa_zmiennej=wartość**
- Wartość zdefiniowanej wcześniej zmiennej możemy wyłuskać za pomocą operatora **\$ (nazwa_zmiennej)**
- Można też dodawać do zmiennych za pomocą **+=**
- W **make** istnieją predefiniowane zmienne
 - **CC** - nazwa kompilatora C
 - **CXX** - nazwa kompilatora C++
 - **CFLAGS** - flagi kompilacji C
 - **CXXFLAGS** - flagi kompilacji C++
 - **LDFLAGS** - opcje linkera
 - I podobne dla innych kompilatorów
- **Przykład 4**

Zmienne - modyfikacje

- Zmienne już zdefiniowane można automatycznie zmieniać
 - Przydaje się np. w sytuacji kiedy mamy listę plików źródłowych
 - Automatycznie zamienimy np. *.cpp na *.o
 - Format:
 - `$(zmienna:wartość_do_zastąpienia=wartość_zastępująca)`
 - Czyli
 - `SRC=main.cpp`
 - `OBJ=$(SRC:.cpp=.o)`

Zmienne dynamiczne

- Program **make** posiada również zestaw zmiennych o dynamicznych wartościach
 - Wartość tych zmiennych jest zależna od miejsca w kodzie w którym się znajdują i na jakim obiekcie wykonują operacje
 - **\$@** - oznacza pełną nazwę etykiety bieżącej
 - **\$<** - oznacza pierwszy plik z listy bieżących zależności
 - **\$?** - oznacza listę plików z bieżących zależności, które są nieaktualne (ze spacjami pomiędzy nimi)
 - **\$^** - oznacza wszystkie bieżące zależności
 - **\$*** - oznacza odwołanie do dopasowanego wzorca (bez rozszerzenia)
 - Oczywiście są też inne predefiniowane zmienne
- Przykład 5

Wzorce reguł

- Wzorce pozwalają bardziej ogólnie podejść do budowy reguł
- Zdecydowanie upraszcza pisanie reguł dla wielu zależności
 - Ma kolosalne znaczenie przy dużej liczbie plików źródłowych
 - `$(OBJ) : $(SRC)`
`$(CXX) $(CXXFLAGS) $(DEP_FLAGS) -c $*.cpp -o $@`
- Wewnętrznie zdefiniowane reguły w make
 - `%.o: %.cpp`
`<TAB> $(CXX) $(CXXFLAGS) -c $< -o $@`
 - W miejsce znaku `%` jest podstawiana nazwa pliku
- Przykład 6

Generowanie zależności

- Kompilator g++ potrafi wygenerować zależności dla każdego kompilowanego pliku
 - Służy do tego opcja **-MMD**, która dodaje zależności w szczególności od plików nagłówkowych
 - Ale nie uwzględnia plików systemowych (takich które z założenia się nie zmieniają)
 - Opcja **-MP** służy do generowania reguł **.PHONY**
 - Pozwala nam to w łatwy sposób uzależnić rekompilację plików **cpp** jeśli zajdą jakiekolwiek zmiany w nagłówkach
- Aby wykorzystać wygenerowane zależności należy użyć w **makefile** dyrektywy **-include**
 - Np. **-include \$(DEP)**
 - Przykład 7

Makefile - zajęcia

- `LABNR=${shell date +%F}`
- `GROUP=$(OOP_GROUP)`
- `LOGINNAME=$(shell whoami)`
- `NAME=$(shell finger $(LOGINNAME) | head -1 | sed -r 's/.*Name: //g' | gawk '{printf("%s_%s", $$1, $$2)}')`
- `SUBJECT_NAME=$(shell echo $(NAME) | sed -e 's/_/ /')`
- `SRC=$(wildcard *.cpp)`
- `APP=$(NAME)`
- `ZIP=$(GROUP)_$(APP)_$(LABNR).zip`
- `SUBJ=[CPP] [$(GROUP)] $(SUBJECT_NAME) $(LABNR)`
- `SUBJ_NDST=$(SUBJ) NDST`
- `CXX=g++`
- `CXXFLAGS+=-c -Wall -g -MMD -MP $(GXX_FLAGS)`
- `LDFLAGS+=-lboost_thread -lm`
- `INCLUDE+=/usr/include/`
- `OBJ=$(SRC:.cpp=.o)`
- `DEP=$(patsubst %.o,%.d,$(OBJ))`
- `all: $(APP)`
- ...

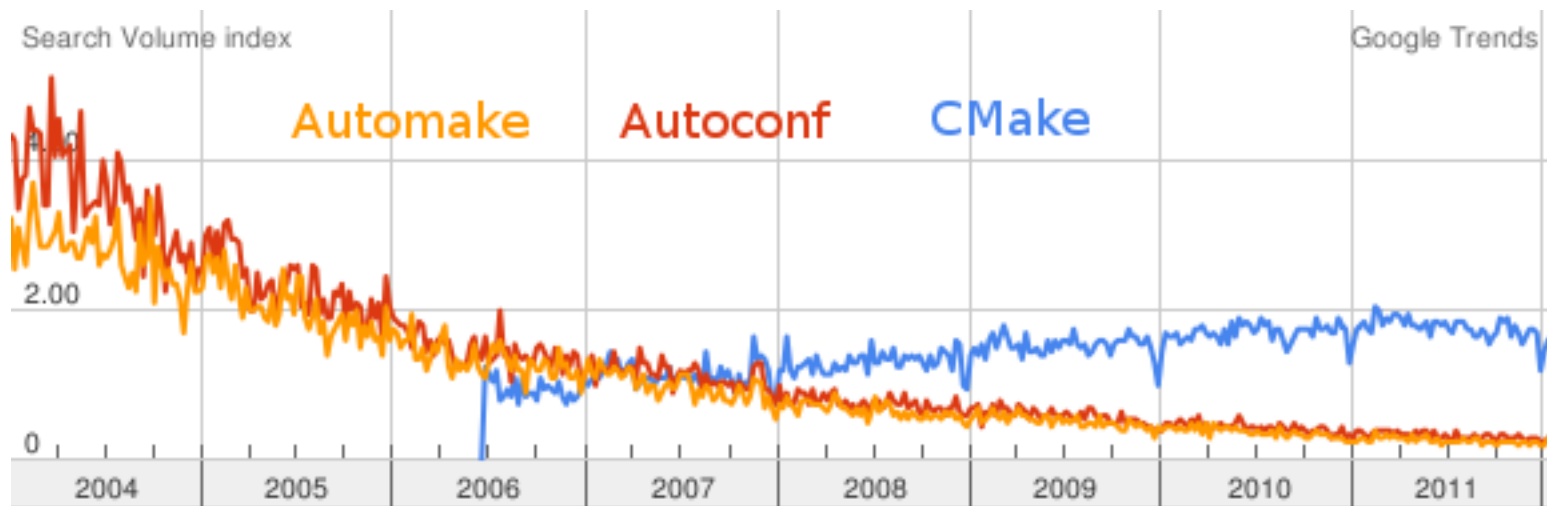
Make - uwagi

- Bardzo wygodne narzędzie do kompilacji programów, dokumentów itp.
- Należy pamiętać o odpowiednich zależnościach
- Można wykorzystywać również do
 - Pre-procesowanie
 - Post-procesowania
 - I wiele innych ...
- Pokazane przykłady to tylko wstęp
 - Istnieje wiele innych funkcji i możliwości make niepokazanych tutaj
 - Warunki, funkcje

Cmake - Cross-platform Make

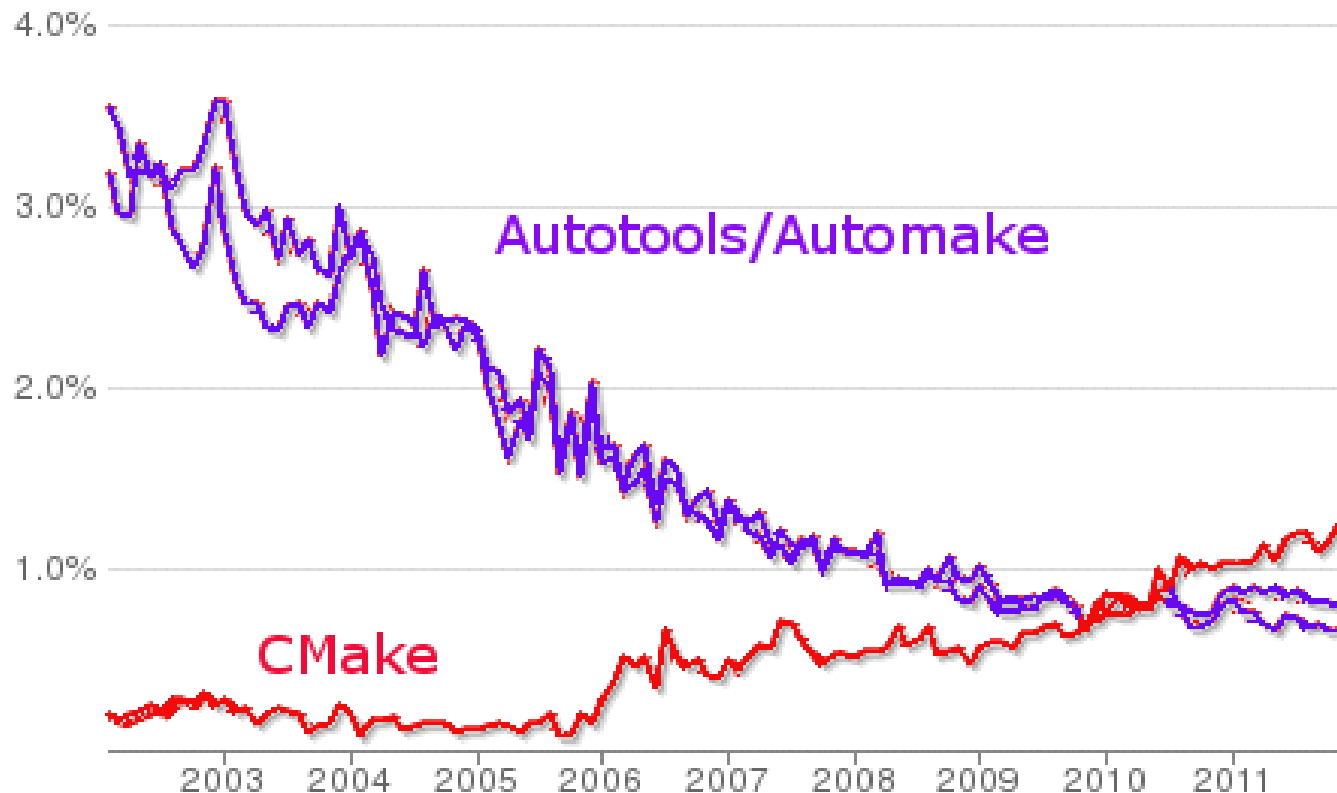
- CMake to wieloplatformowe narzędzie do automatycznego zarządzania procesem kompilacji programu
 - Charakteryzuje się niezależnością od platformy sprzętowej oraz użytego kompilatora
 - Potrafi także zarządzać regułami cross-kompilacji
- Tworzy reguły umożliwiające budowanie aplikacji w danym środowisku
 - Linux - makefile
 - Win - projekt Visual Studio
- Zawiera pakiety umożliwiające wyszukiwanie standardowych bibliotek
 - Np. boost, gsl, Qt, itp.

CMake - porównanie do innych narzędzi



<http://www.google.com/trends>

CMake - porównanie do innych narzędzi

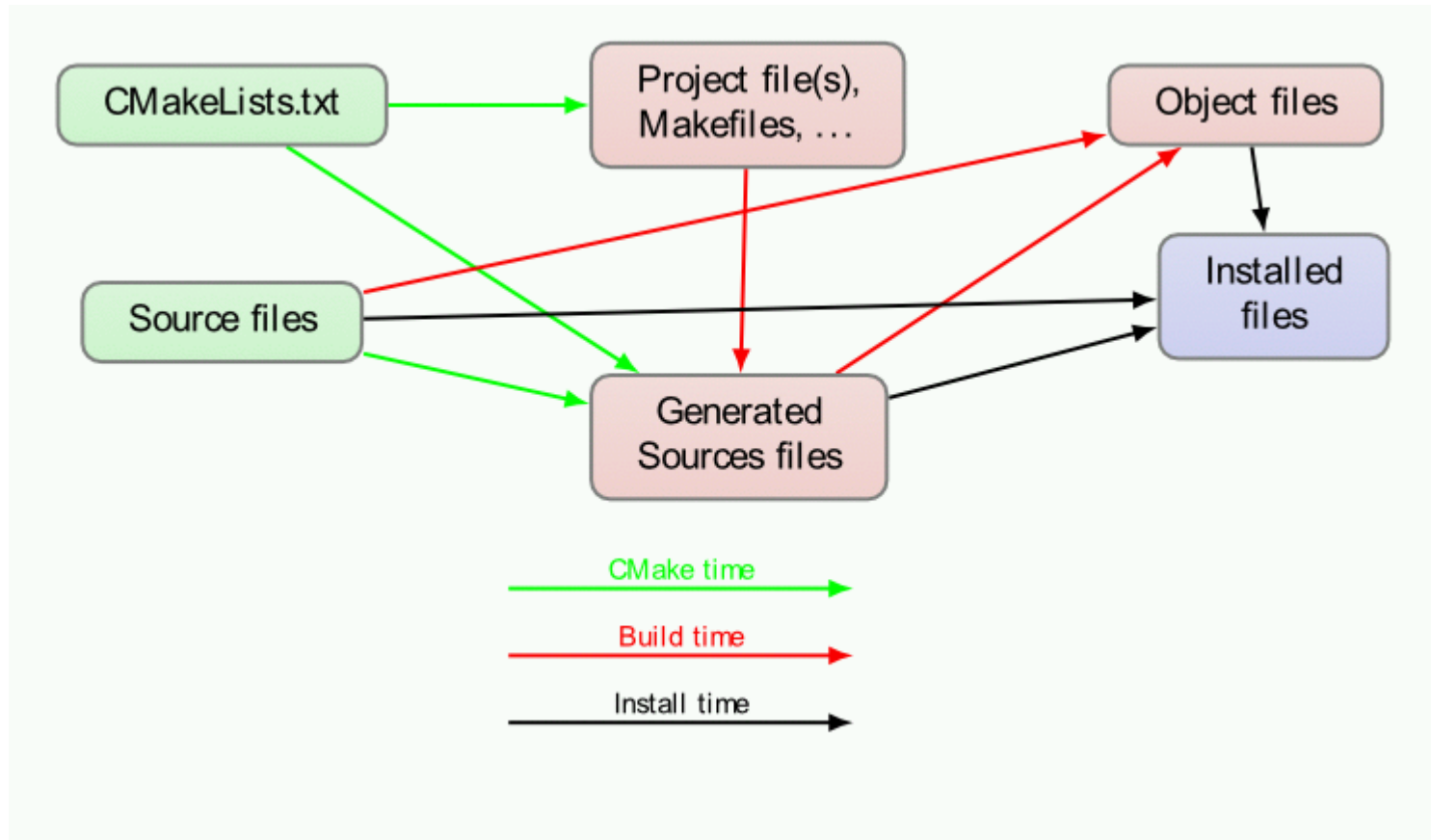


<https://www.openhub.net/languages/compare>

Sposób budowania

- Generalnie jesteśmy przyzwyczajeni do budowania aplikacji w miejscu gdzie są źródła
 - **make** właśnie tak robi (a raczej my)
- **CMake** umożliwia budowanie źródeł „w miejscu”
 - Niezalecane, ewentualnie niewygodne
 - Cmake tworzy zdecydowanie więcej plików pośrednich niż make
- **CMake** wspiera budowanie poza obszarem źródeł
 - Ang. out-of-source
 - Odseparowanie źródeł od plików pośrednich i wynikowych
 - Kasowanie niepotrzebnych rzeczy jest banalnie proste
 - Łatwe budowanie różnych wersji
 - Jak i po co?

Przeptyw w CMake



CMake komendy i zmienne

■ `cmake --help-command-list`

- ❑ `add_executable`
- ❑ `add_library`
- ❑ `install`
- ❑ `set`
- ❑ `find`
- ❑ ...

■ `cmake --help-variable-list`

- ❑ `CMAKE_BUILD_TYPE`
- ❑ `CMAKE_INSTALL_PREFIX`
- ❑ ...

■ Przykład 1

Zmienne

- Zmienne można zmieniać z poziomu skryptu
 - ❑ `set(CMAKE_INSTALL_PREFIX ./)`
 - ❑ `set(CMAKE_BUILD_TYPE Release)`
- Ciekawsze jest dynamiczne zmienianie w momencie wywołania **CMake**
 - ❑ `cmake -DCMAKE_BUILD_TYPE=Release ../`
 - ❑ `cmake -DCMAKE_INSTALL_PREFIX=./ ../`
 - ❑ `cmake -DCMAKE_BUILD_TYPE=Release -DCMAKE_INSTALL_PREFIX=./ ../`
- W ten sposób mamy kontrolę co, gdzie i jak jest budowane
- Przykład 2 i 3

CTest

- Automat pozwalający uruchamiać test i analizować ich rezultaty
 - `enable_testing()`
 - `add_test(TestName1 Tutorial)`
- Bardzo przydatne i szybkie narzędzie do określania poprawności wykonania aplikacji w różnych konfiguracjach
 - `$ make test`
 - `Running tests...`
 - `Test project /home/emv/OOP/cmake/p03/_b`
 - `Start 1: TestName1`
 - `1/3 Test #1: TestName1 Passed 0.00 sec`
 - `Start 2: TestName2`
 - `2/3 Test #2: TestName2 Passed 0.00 sec`
 - `Start 3: TestName3`
 - `3/3 Test #3: TestName3***Failed 0.00 sec`
 - `67% tests passed, 1 tests failed out of 3`
 - `Total Test time (real) = 0.01 sec`
 - `The following tests FAILED:`
 - `3 - TestName3 (Failed)`
 - `Errors while running CTest`
 - `make: *** [test] Error 8`
- Przykład 4

CTest - publikacja wyników (CDash)

Login

All Dashboards

CMake

Dashboard

Calendar

Previous

Current

Project

Friday, October 14 2016 05:14:14

No file changed as of **Thursday, October 13 2016 - 21:00 EDT**

22 minutes ago: 412 tests not run on Linux-Gentoo-Sparc32-GCC-4.9

22 minutes ago: 1 test failed on Linux-Gentoo-Sparc32-GCC-4.9


22 minutes ago: 5 errors introduced on Linux-Gentoo-Sparc32-GCC-4.9

26 minutes ago: 1 test failed on Win32-cygwin


40 minutes ago: 1 test failed on AIX-7.1_IBM-12.1

See full feed










Style

Site	Build Name	Update	Configure		Build		Test			Build Time ▼
		Files	Error	Warn	Error	Warn	Not Run	Fail ▼	Pass	
dashmacmini5.kitware	 KWStyle	45	0	0	0	0	0	0	0	8 hours ago

Scan Build

Site	Build Name	Update	Configure		Build		Test			Build Time ▼
		Files	Error	Warn	Error	Warn	Not Run	Fail ▼	Pass	
elysium-linux.kitware	 Linux-clang-scanbuild	0	0	0	0	0	0	0	0	5 hours ago

Nightly Expected

Site	Build Name	Update	Configure		Build		Test			Build Time ▼
		Files	Error	Warn	Error	Warn ▼	Not Run	Fail ▼	Pass	
mug.neocisinc	 Win32-ninja-ci12-Debug	45	0	0	0	0	0	3 ¹ ₁	420 ¹ ₁	3 hours ago
trinsic.kitware	 vs14-64-ninja	45	0	0	0	0	0	3 ¹ ₁	416 ¹ ₃	7 hours ago
dash2win64.kitware	 vs10-32-ninja	45	0	0	0	0	0	2 ¹ ₂	416 ¹ ₂	2 hours ago
hythloth.kitware	 nightly-binaries	45	0	0	0	0	0	2 ¹ ₂	2 ¹ ₂	4 hours ago
dash2win64.kitware	Win32-cygwin	45	0	0	0	0	0	1	420	1 hour ago
gcc111.fstfrance.org	 AIX-7.1_IBM-12.1	45	0	0	0	0	0	1	421	1 hour ago
vesper.kitware	 ici-64-ninja	45	0	0	0	0	0	1 ¹ ₁	353 ¹ ₁	3 hours ago
gcc111.fstfrance.org	 AIX-7.1_GCC-4.8.1	45	0	0	0	0	0	1 ¹ ₁	427 ¹ ₁	3 hours ago
dash2win64.kitware	 Windows-ici-11.1-32	45	0	0	0	0	0	1 ¹ ₁	425 ¹ ₁	4 hours ago
trinsic.kitware	 ici-64-ninja	45	0	0	0	0	0	1 ¹ ₁	357 ¹ ₁	6 hours ago

First

Previous

1

2

3

4







5

Next

Last

Items per page 10 ▼

Nightly Master

Site	Build Name	Update	Configure		Build		Test			Build Time ▼
		Files	Error	Warn	Error	Warn	Not Run	Fail ▼	Pass	
dash20.kitware	 master-vs9-32	20	0	0	0	0	0	1	421	3 hours ago
dashmacmini2.kitware.com	 master-MacOSX10.4.11i386-Xcode	0	0	0	0	0	0	1 ¹ ₁	405 ¹ ₁	5 hours ago
dashcyg1.kitware	master-Win32Cygwin-gnu	20	0	0	0	0	0	0	425	2 hours ago
faraway	 master-Linux-106-U3.13.0x86_64-gcc	20	0	0	0	0	0	0	467	4 hours ago
mvstdash1	 master-bcc32-7.11-bormake	20	0	0	0	0	0	0	342	4 hours ago
dash22.kitware	 master-Win32-x86-ci-gmake	20	0	0	0	0	0	0	422	5 hours ago
mvstdash1	 master-bcc32-7.01-bormake	20	0	0	0	0	0	0	342	5 hours ago

CMake - przykład

```
■ cmake_minimum_required (VERSION 2.8)
■ project (ggssLibs)

■ #flagi
■ set (CMAKE_MODULE_PATH ${CMAKE_MODULE_PATH} ${CMAKE_SOURCE_DIR}/CMakeFiles) #for .cmake files
■ set (PROJECT_INCLUDE_DIRECTORY ${PROJECT_SOURCE_DIR}/include)
■ set (PROJECT_LINK_DIRECTORY ${PROJECT_SOURCE_DIR}/lib)
■ if (UNIX)
■     set (CMAKE_CXX_FLAGS "-std=c++11 -Wall -Wno-reorder")
■ endif()

■ include(FindLibraries) # boost, gsl, caen
■ include(CheckPlatform) # sets WINDOWS or LINUX flag
■ include(ParseDirectory) # parse_directory

■ set(PROJECTS
■     logLib
■     xmlLib
■     utilsLib
■     handleLib
■     ThreadLib
■     fifoLib
■     FitLib
■     OrtecMcbLib
■     CaenHVLib
■     ggssLib
■     usbrmLib
■     CaenN1470Lib
■     mcaLib
■     daemonLib)
```


CMake - uwagi

- Jak każde narzędzie wymaga trochę nauki
- Jednak może warto ze względu na wieloplatformowość
 - Istotne z punktu widzenia budowania aplikacji np. Win i Linux z tych samych kodów źródłowych
 - Trzeba nauczyć się jednego narzędzia, które już zna większość dostępnych na poszczególnych platformach kompilatorów, itp.
- Możliwość łatwego budowania wielu wersji
 - Do „debugowania”
 - Do „profilowania”
 - Produkcyjnych

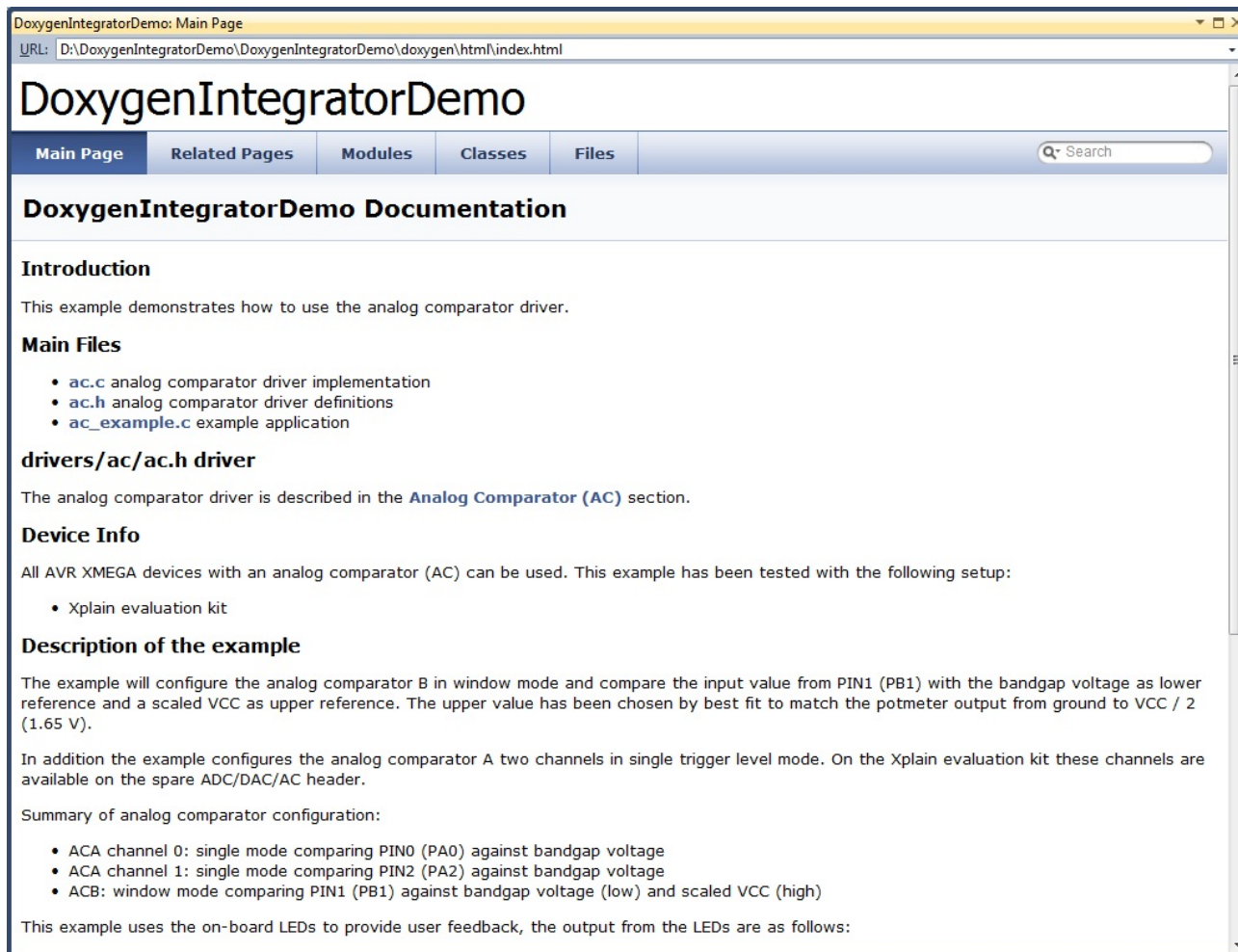
Dokumentacja kodu

- Dokumentowanie kodu jest kluczową kwestią podczas pisania programów
 - Często jednak ta część jest zaniedbywana lub w ogóle jej nie ma
- Pisanie dokumentacji samej w sobie jest bardzo trudne i niewdzięczne
 - Dlatego powstały systemy to ułatwiające
 - JavaDoc
 - Doxygen
- Dzięki odpowiednim narzędziom pisanie dokumentacji jest łatwiejsze
 - W szczególności kiedy piszemy ją bezpośrednio w kodzie
 - Dokumentacja znajduje się w obrębie dokumentowanego kodu
 - Pliku, klasy, funkcji, itp.
 - Umożliwia to w miarę łatwe utrzymywanie dokumentacji w zgodzie z istniejącym kodem

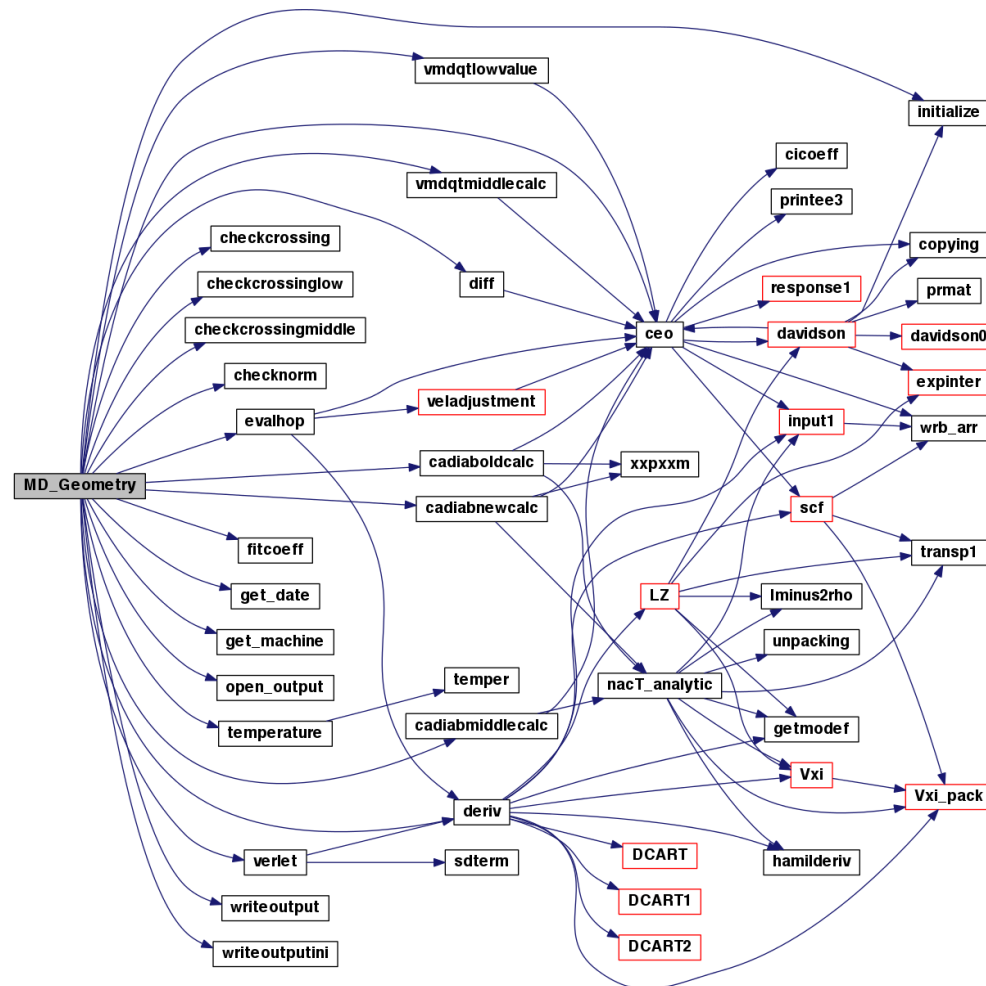
Doxygen - możliwości

- Doxygen jest dojrzałym systemem umożliwiającym tworzenie dokumentacji do kodów napisanych w różnych językach programowania
 - C++, C, C#, Objective-C, Java, Perl, Python, IDL, VHDL, Fortran, Tcl and PHP
 - Jest darmowy oparty na licencji GNU
- Potrafi generować dokumentację w różnych formatach
 - HyperText Markup Language (HTML), Microsoft Compiled HTML Help (CHM), Rich Text Format (RTF), Portable Document Format (PDF), LaTeX, PostScript, man pages

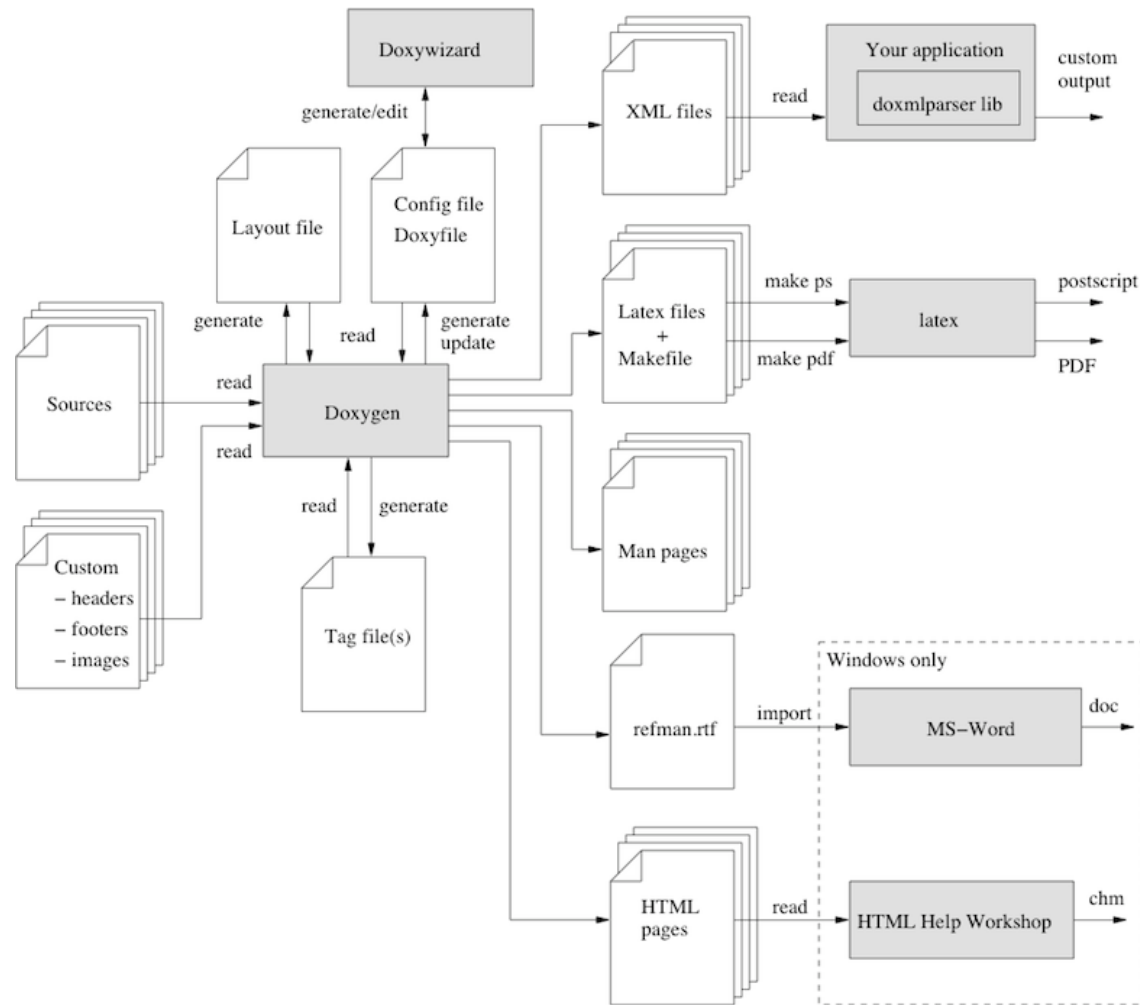
Doxygen - przykładowy html



Doxygen -diagram (dot)



Doxygen - schemat działania



Doxygen - plik konfiguracyjny

- Plik konfiguracyjny
 - `doxygen -g config_name`
 - `doxywizard`
 - Graficznie narzędzie
- Plik konfiguracyjny ma format zbliżony do prostego **Makefile** i zawiera tagi
 - `TAGNAME = VALUE or
TAGNAME = VALUE1 VALUE2 ...`
 - Większość tagów może mieć wartości domyślne
 - Na początek wystarczy wybrać odpowiedni format dokumentacji
 - `GENERATE_HTML` = YES
 - Reszta po generacji pliku jest w nim opisana

Doxygen - dokumentowanie kodu

- Jeżeli **EXTRACT_ALL** jest ustawione na **NO** to **doxygen** wygeneruje dokumentację tylko dla udokumentowanych fragmentów kodu
- Jest kilka sposobów zaznaczania dokumentacji
 - Styl JavaDoc
 - **/****
 - *** ... text ...**
 - ***/**
 - Styl Qt
 - **/*!**
 - *** ... text ...**
 - ***/**
 - W obu przypadkach pośrednie ***** są opcjonalne

Doxygen - format dokumentacji

- Jeśli ktoś woli komentowanie każdej linii, a nie całych bloków możliwe są jeszcze dwie opcje
 - `///`
 - `/// ... text ...`
 - `///`
- lub
 - `//!`
 - `//!... text ...`
 - `//!`
- W tych przypadkach pusta linia kończy dokumentację w danym miejscu
- Jeszcze inna wersja
 - `/**`
 - `* ... text`
 - `*/`
- lub
 - `////`
 - `/// ... text ...`
 - `////`

Doxygen - format dokumentacji

- W przypadku dokumentowania składników czasami warto wstawić dokumentację nie przed a po
 - `int var; /**< Detailed description after the member */`
 - `int var; //!< Detailed description after the member`
 - `//!<`
 - `int var; ///< Detailed description after the member`
 - `///<`
 - `int var; //!< Brief description after the member`
 - `int var; ///< Brief description after the member`
- Dla funkcji można użyć komendy `@param` lub wstawić dokumentację typu inline
 - `void foo(int v /**< [in] docs for input parameter v. */);`
 - `[in], [out], [in,out]` - pokazują kierunek

Doxygen - prosty przykład

```
/*! A test class */  
  
class Test  
{  
public:  
    /** An enum type.  
     * The documentation block cannot be put after the enum!  
     */  
    enum EnumType  
    {  
        int EVal1,    /**< enum value 1 */  
        int EVal2     /**< enum value 2 */  
    };  
    void member();    /**< a member function.  
  
protected:  
    int value;        /**< an integer value */  
};
```

AfterDocs

Main Page | **Classes** | Files

Class List | Class Index | Class Members

Public Types | Public Member Functions | Protected Attributes | Use of all members

Test Class Reference

#include <afterdoc.h>

Public Types

enum EnumType { EVal1, EVal2 }
An enum type. More...

Public Member Functions

void member()
a member function.

Protected Attributes

int value

Detailed Description

A test class

Member Enumeration Documentation

enum Test::EnumType

An enum type.
The documentation block cannot be put after the enum!

Enumerator	
EVal1	enum value 1
EVal2	enum value 2


Member Data Documentation

int Test::value protected

an integer value

The documentation for this class was generated from the following file:

- afterdoc.h

Generated on Sat Jan 4 2015 13:23:12 for AfterDocs by  1.8.9.1

Doxygen - markdown

- **Markdown** - język znaczników przeznaczony do formatowania tekstu
 - ❑ Został stworzony w celu jak najbardziej uproszczenia tworzenia i formatowania tekstu
 - ❑ **Nagłówek pierwszego poziomu**
=====
 - ❑ **Nagłówek drugiego poziomu**

 - ❑ [Polska Wikipedia – Wolna Encyklopedia] (<http://pl.wikipedia.org>)
 - ❑ ! (<http://upload.wikimedia.org/wikipedia/commons/thumb/1/1b/Mars.jpg/160px-Mars.jpg>)

Markdown - Przykład

text using Markdown syntax	the corresponding HTML produced by a Markdown processor	the text viewed in a browser
<pre>Heading ===== Sub-heading ----- ### Another deeper heading Paragraphs are separated by a blank line. Let 2 spaces at the end of a line to do a line break Text attributes <i>italic</i>, bold, `monospace`, ~~strikethrough~~ . A [link](http://example.com). <<< No space between] and (>>> Shopping list: * apples * oranges * pears Numbered list: 1. apples 2. oranges 3. pears The rain---not the reign---in Spain.</pre>	<pre><h1>Heading</h1> <h2>Sub-heading</h2> <h3>Another deeper heading</h3> <p>Paragraphs are separated by a blank line.</p> <p>Let 2 spaces at the end of a line to do a
 line break</p> <p>Text attributes italic, bold, <code>monospace</code>, <s>strikethrough</s>.</p> <p>A link.</p> <p>Shopping list:</p> apples oranges pears <p>Numbered list:</p> apples oranges pears <p>The rain&mdash;not the reign&mdash;in Spain.</p></pre>	<p>Heading</p> <hr/> <p>Sub-heading</p> <p>Another deeper heading</p> <p>Paragraphs are separated by a blank line.</p> <p>Let 2 spaces at the end of a line to do a line break</p> <p>Text attributes <i>italic</i>, bold, <code>monospace</code>, strikethrough.</p> <p>A link http://example.com.</p> <p>Shopping list:</p> <ul style="list-style-type: none">• apples• oranges• pears <p>Numbered list:</p> <ol style="list-style-type: none">1. apples2. oranges3. pears <p>The rain—not the reign—in Spain.</p>

Doxygen

- Doxygen jest naprawdę wygodnym i przydatnym narzędziem
- Potrafi wygenerować także ładnie sformatowany tekst wprowadzający do projektu
 - Tzw. „Main page”
- Potrafi generować graficzną reprezentację zależności klas
 - Pozwala nam bardzo szybko zorientować się w projekcie
- Generuje także indeksy i spisy
 - Plików, klas, funkcji
- Należy jednak pamiętać, że nawet takie narzędzie nie zastąpi nam pisanie komentarzy

Inne uwagi ogólne

- Wygodne środowisko do pracy nad kodami źródłowym, itp. jest niezbędne
 - Po pierwsze dobry edytor
 - Notepad++ (Windows)
 - Sublime Text (Windows + Linux)
 - VisualStudioCode (Windows + Linux)
 - Emacs, vi (Windows + Linux)
 - Gedit, Kate wraz z wtyczkami (Linux)
 - QtCreator - całe środowisko (ale proszę nie używać na zajęciach)
 - ...
 - Sprawne operowanie znanymi już narzędziami (konsolowymi)
 - Kurs „Podstawy systemów operacyjnych UNIX ”
- W całym kursie bardzo istotne jest bycie „na bieżąco” z tym co się dzieje na wykładach i zajęciach ćwiczeniowych