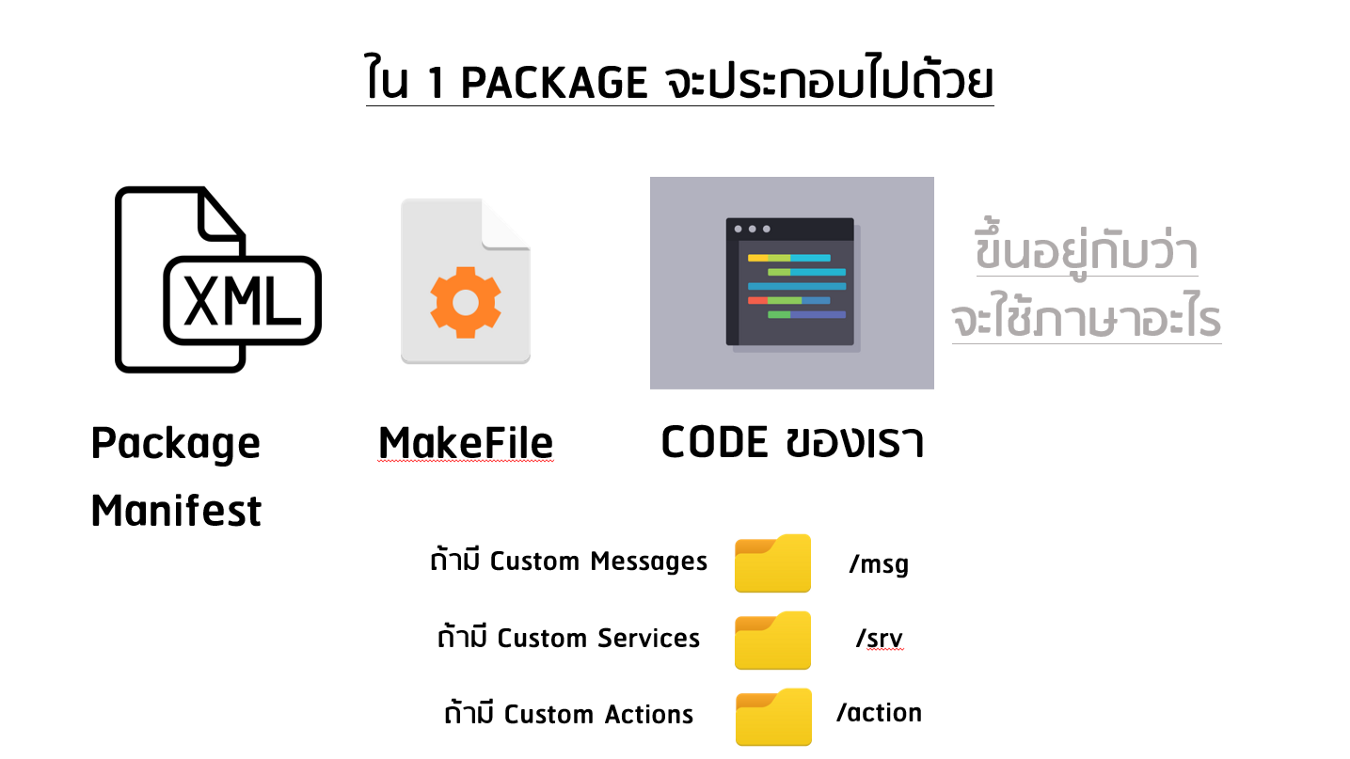
พูดถึงโปรแกรมในหุ่นยนต์

เรามักจะโปรแกรมอะไรต่อมิอะไรเป็นเรื่องๆไป เพื่อที่จะได้แบ่งการทำงานออกให้ชัดเจน แล้วเราจะรวมกลุ่มโปรแกรมหรือ Code ของเรายังไงล่ะ ให้เก็บเป็นสัดเป็นส่วน (และ ให้เราเรียกใช้งานได้ง่ายๆผ่านตัวช่วยต่างๆของ ROS)

โดยการสร้าง Package ใน ROS จะมี Layout องค์ประกอบดังนี้ จะพยายามย่อย และทำ REP ของ ROS ให้ออกมาดูง่ายๆละกันนะคับ :) อ้างอิง — [http://www.ros.org/reps/rep-0128.html#recommended-layout](http://www.ros.org/reps/rep-0128.html?source=post_page---------------------------#recommended-layout)



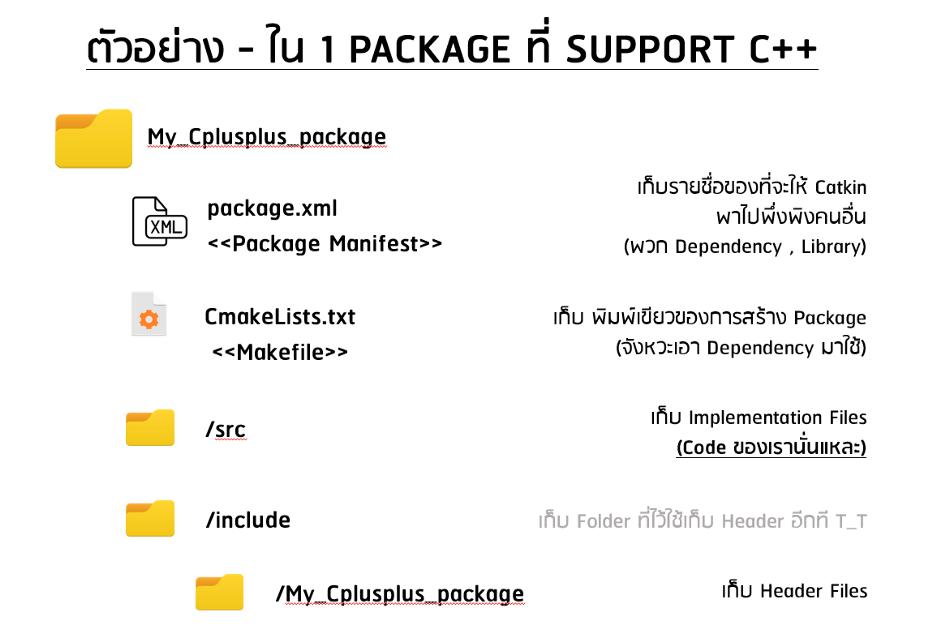
Package Manifest — ตัวที่จะบอกว่า Package นี้ มีรายละเอียดยังไง ต้องการไปพึ่งพิง Code คนอื่นไหม (ตัวช่วย Compile Code , Reference Code ใน Level สูงๆ จะอ่านตรงนี้ และไปอ้างอิงให้)

CMakeLists.txt — เป็น พิมพ์เขียวของงานนี้ โดยจะเก็บรายละเอียดว่า เรา Run Code ไฟล์ไหนเป็นจุดเริ่มต้นของงาน, Code ต่างๆอ้างอิงกันยังไง , เราอ้างอิง Code Library ของชาวบ้านคนอื่นๆมาใช้กับ Code เราไฟล์ไหนบ้าง   
\* CMakeLists (MakeFile) ตัวนี้สำคัญ \*

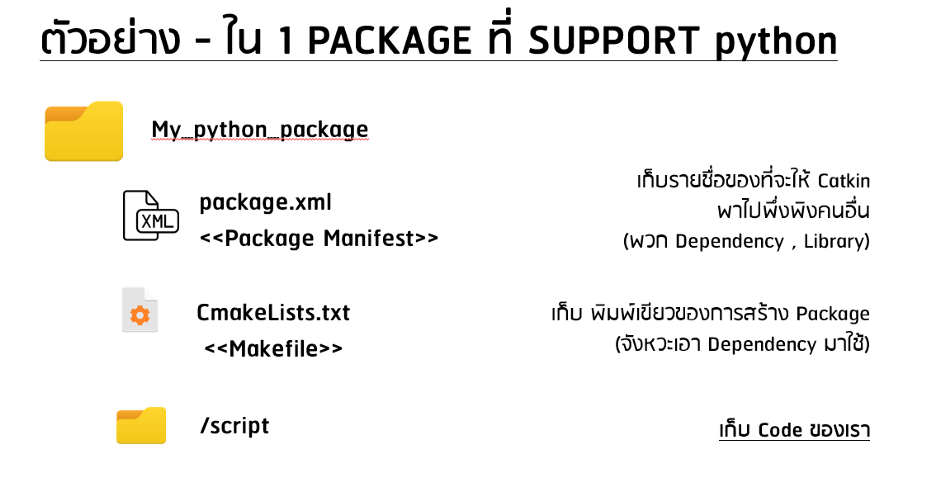
Codeของเรา จะเก็บใน Folder ดังนี้  
(อ้างอิงตาม REP ซึ่งก็คือข้อควรปฏิบัติตามของ ROS  
ซึ่งจะทำตามหรือไม่ทำตามก็ได้ แต่ระวังลำบาก 5555 )   
1. “src” — จะเก็บ Source Code ภาษา C++ ที่เป็นส่วน Implementation   
(ส่วนของ Header จะเก็บแยก)  
2. “include/ชื่อPackage/”- จะเก็บ Header ภาษา C++   
3. “script” — จะเก็บ Code Python   
\* เอ้อ เวลาเขียน Package นึง มันจะมีได้หลาย Node หลายภาษา   
เวลาใช้งานก็แบ่งส่วนดีๆล่ะ \*

Config ต่างๆ   
อันนี้ไม่ซีเรียสมาก แต่ปกติจะสร้าง Folder ชื่อ param , config บ้าง   
แล้วแต่คนชอบ   
[ advance : พวก ตัวแปรที่เป็นแบบ Dynamic Parameter   
(ปรับ ณ เวลาที่ Run Code ได้ ตัวต้นแบบ Config จะอยู่ใน Folder “cfg” ]

ตัวสร้าง Message , Service , Action ของ ROS หรือที่เรียก Message Structure , Message Prototype   
1. ROS Message — จะอยู่ใน Folder “msg”  
2. ROS Service — จะอยู่ใน Folder “srv”   
3. ROS Action — จะอยู่ใน Folder “action”   
โดย Catkin build system จะให้ ROS มาคว้า “หน้าตา” ของ Message Service และ Action จากบริเวณนี้ Auto เลย   
\* ก็อย่างที่เคยบอกไป คือ จริงๆแล้ว จะตั้งชื่ออะไรก็ได้ .. แล้วไปแก้ใน CMakeLists.txt เอาเน้อ \*

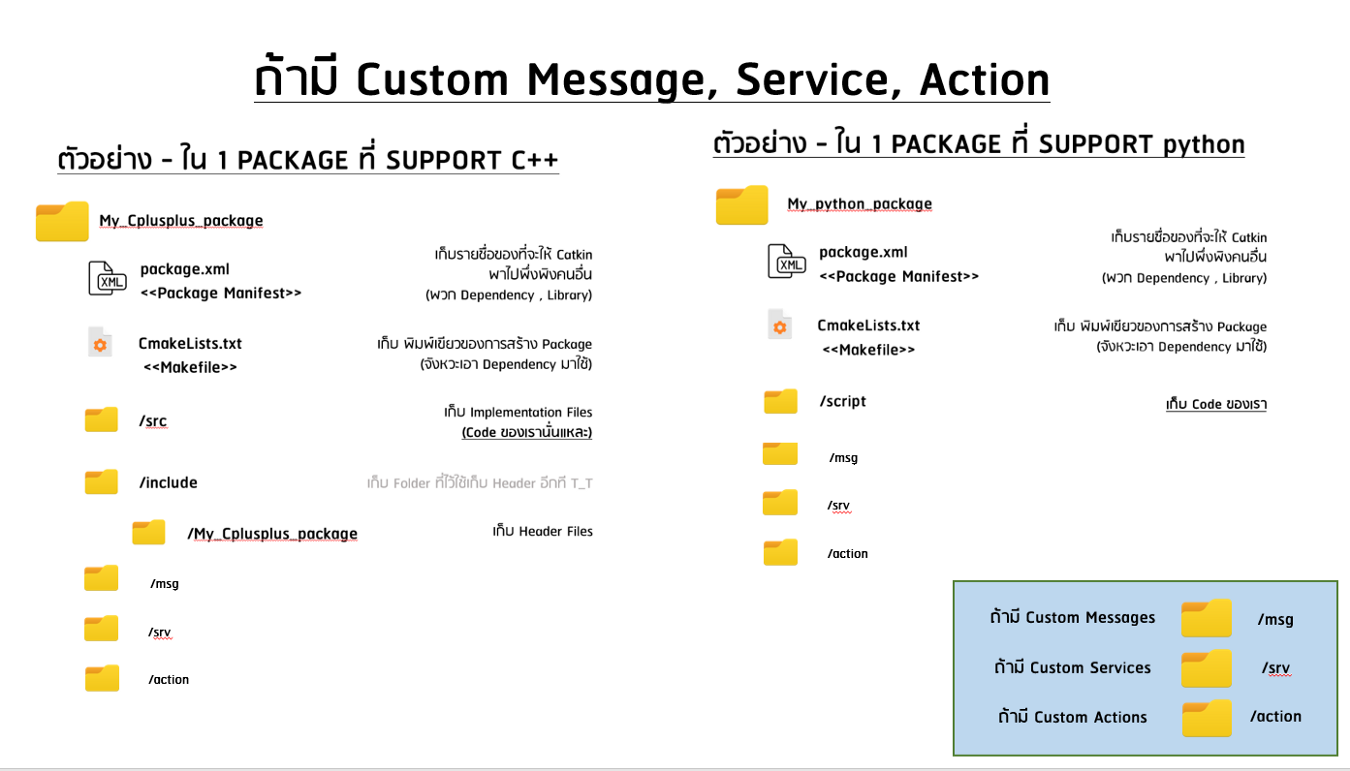


สังเกตว่า ภาษา C++ จะมี ส่วน Implementation file , Header File — เลยต้องมีที่เก็บแยกกัน



Python ไม่มี header เหมือนกับ C++ เลยใช้แค่ Folder ในการเก็บ Code แค่บริเวณเดียว

อยากจะเขียน Message , Service , Action เอง แบบไม่ได้ใช้ของ มาตรฐานที่ติดมากับ ROS ก็จะออกมาประมาณนี้



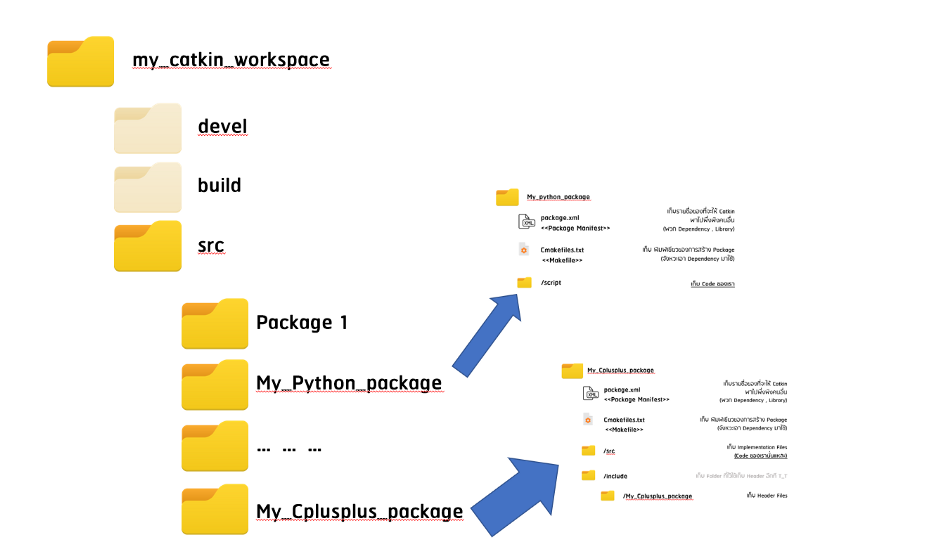
ถ้าอยากจะสร้าง message , service ,action เอง จะต้องมีโฟล์เดอร์เหล่านี้ออกมา + Setting CMakeLists นิดนึง

เอ้อ แล้วมันอยู่กินกันหน้าตายังไงล่ะ ?

ตอนที่แล้วเราพูดถึง Workspace ใช้ไม๊ล๊าา   
ใน 1 Workspace จะมี หลายๆ Packages   
ใน 1 Package จะมีได้หลายๆ Node . . .

. . . ใน 1 Node มักจะทำ 1 หน้าที่ (จริงๆก็ไม่เสมอไปหร๊อก .. )

มันจะอยู่กันหน้าตาประมาณนี้



ตัวอย่างเมื่อมีหลายๆ package ใน workspace เดียวกัน แล้วสร้างมาจากคนละภาษา (อยู่คนละ Package)

มาสร้าง Package กันเถอะ

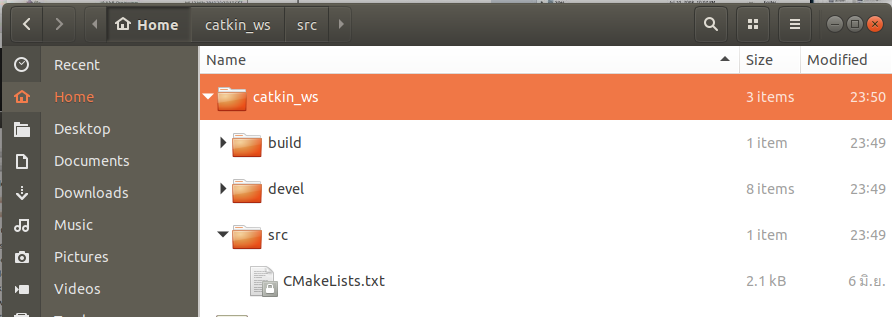
เริ่มแรก เรามาออกแบบกันก่อน

เรากำลังจะทำ Node 2 อัน ที่ทำหน้าที่พูดคนนึง และ รับฟังคนนึง   
Package ชื่อ : chatter\_subscriber\_tutorial  
โดยใช้ Dependencies ดังนี้:   
1. catkin (build system)   
2. roscpp (ROS ภาษา C++)   
3. std\_msgs (Package ข้อความพื้นฐานใน ROS)

อยากจะสร้าง Node ที่ทำหน้าที่เป็น

Publisher 1 Node ทำหน้าที่ส่งข้อความลูกเดียวเลย  
Subscriber 1 Node ทำหน้าที่รับข้อความแล้วหาคำตอบ — แล้วโชว์

0. สร้าง Workspace (ถ้าทำตาม Tutorial บทแรกๆจะมี catkin\_ws เป็น Workspace ของเราแล้วนั้นเอง)

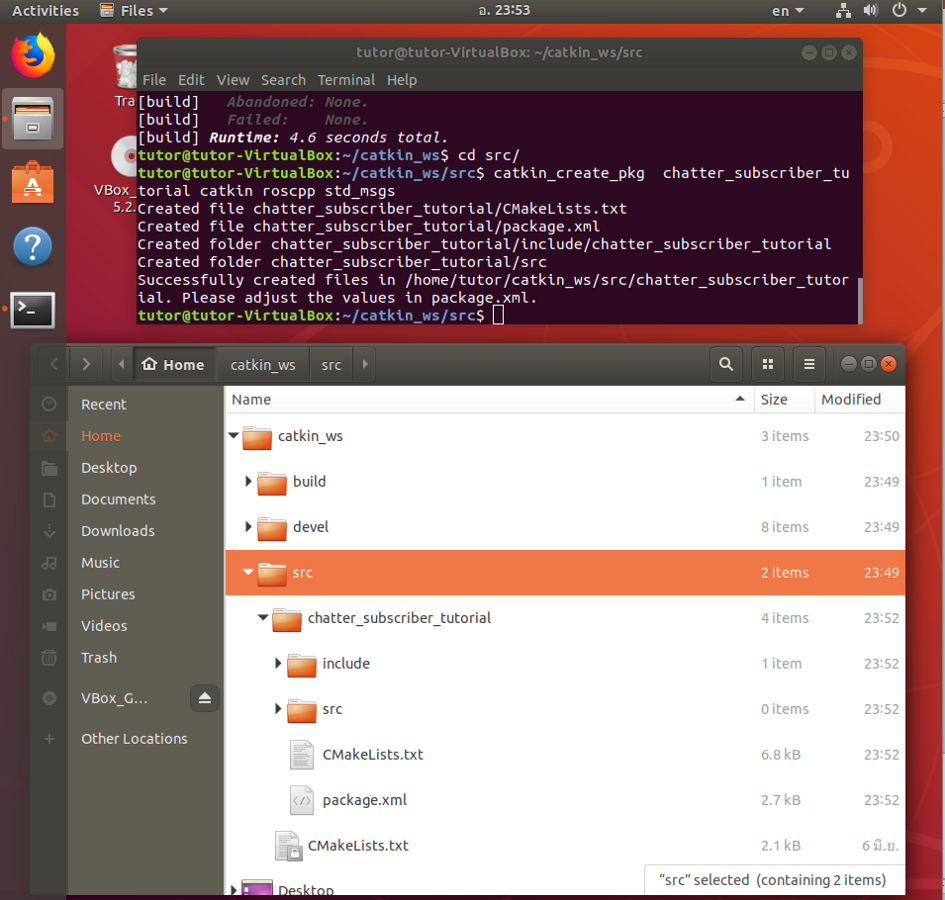


1. ใช้ CLI ของ Catkin ในการสร้าง Package ของ ROS ขึ้นมา ซึ่ง Package จะต้องอยู่ใน src ของ Workspace เน้อ

catkin\_create\_pkg ชื่อ วรรค ตามด้วย dependency หลายๆตัว ที่เว้นวรรค 1 ที

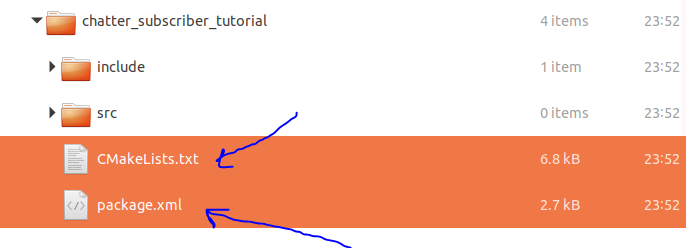
> catkin\_create\_pkg chatter\_subscriber\_tutorial catkin roscpp std\_msgs

เราจะได้ Folder ที่มีชื่อว่า chatter\_subscriber\_tutorial ขึ้นมา   
พอเป็นเข้าไปด้านใน จะพบว่ามี Folder มารอดังนี้



2. วางโครงสร้างของ Code C++

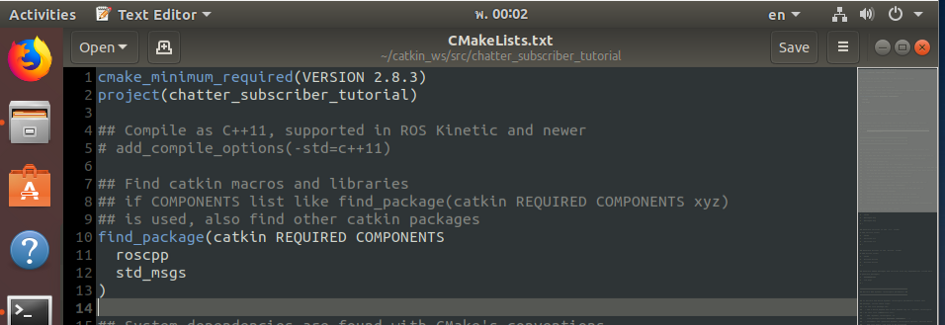
เราจะต้องอธิบาย Package เราก่อนว่า ประกอบด้วยอะไร และไปใช้งานใครบ้าง



สองไฟล์นี้คือตัวเริ่มก่อโครงสร้างของ Package เรา ว่าจะมี Node อะไรบ้าง และ Node นั้นๆ ใช้ Library อะไรบ้าง มี header ไฟล์อะไรบ้าง

เริ่มที่ file package.xml (จะพบว่ามีมาให้แล้วจากคำสั่งในข้อ 1)  
โดยปกติ ถ้าใช้ CLI (ไอ้คำสั่ง catkin\_create\_pkg) สร้าง package มาแล้ว ไฟล์นี้จะรวมของให้เรียบร้อยแล้ว — แปลว่าไม่ต้องแก้อะไรก็ได้ นั่นเอง

หลังจากนั้นมาต่อที่ CMakelists.txt (ซึ่งคำสั่งก็ช่วย Generate ให้แล้วเช่นกัน)



หน้าตา File CMakeLists.txt

ในส่วนที่เรียกว่า find\_package นั้นจะเป็นการไปหา package ที่เราเคยลงไว้ในเครื่อง เช่น roscpp , std\_msgs (ลงมาพร้อม ROS)

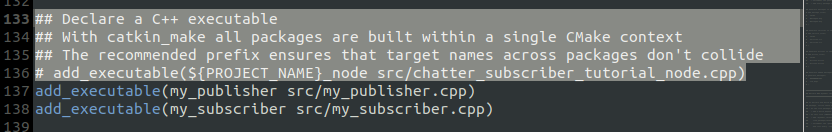
[advance user : ถ้าลง opencv , eigen lib หรือ library นอก เช่น dynamixel-sdk หรือ บอร์ดขับ Motor ใดๆ แม้กระทั่ง Serial ถ้าจะเอามาเขียนใน Codeก็ต้องมากำหนดตรงนี้ เพื่อสร้าง Scope ให้ catkin build system รู้จักพวกของนอก Package (dependency) เหล่านี้] ณ จุดนี้ เรายังไม่ต้องกำหนดอะไรเพิ่มมาก :)

นอกจากบอกว่า เราจะสร้าง Node กี่อัน แล้ว Run อันไหนได้บ้างเราจะบอกว่า Package นี้มี Code ที่รันได้ จะสร้างจาก file 2 อันนี้นะ จะต้องเติม 3 จุด ตำแหน่งในการเติม ให้ลองมองๆหาตัวอย่างใน File CMakeLists.txt ที่เขียนแบบนี้ดู เราจะเติม 3 คำสั่ง คือ add\_executable , add\_dependencies, target\_link\_library

เขียนตามลำดับ(บนลงล่าง)นะเห้ยไม่งั้น Compile ไม่ผ่านหาว่าไม่เตือนนะจ๊ะ

add\_executable จะเป็นการบอกว่า มีอะไรที่ run ได้บ้าง   
“คือ มี MAIN Function นั่นแหละ”   
และ ตั้งชื่อมันว่าอะไร ด้วยการใช้คำสั่งดังนี้

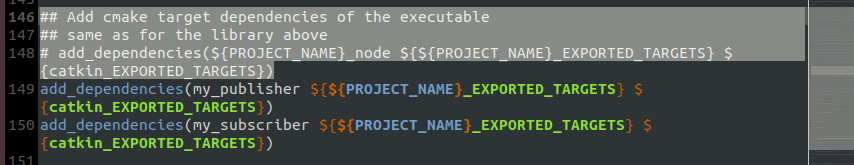
> add\_executable(ตั้งชื่อexecutable วรรค ตามด้วย ไฟล์ Node ที่มี Main)  
> add\_executable(node\_funny src/hallo.src)



มี Node ชื่อ my\_publisher — ให้ใช้ File ใน Package folder src/my\_publisher.cpp เป็นตัวตั้งต้น

add\_dependencies จะเป็นการบอกว่า Executable นั้นๆ   
ไปใช้ Dependency ที่วางรอตรงไหนบ้าง   
(ซึ่งปกติมักจะไปถามเอาจาก catkin ด้วย การพิมพ์ catkin\_EXPORTED\_TARGETS)

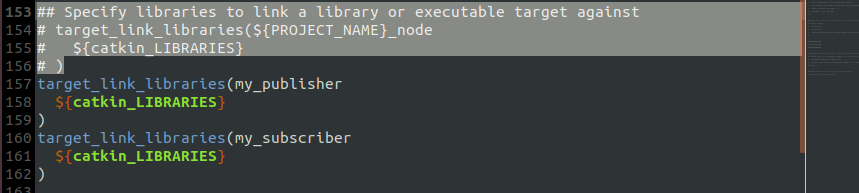
> add\_dependencies(ชื่อexecutable export-targetต่างๆ)  
> add\_dependencies(my\_publisher ${${PROJECT\_NAME}\_EXPORTED\_TARGETS} ${catkin\_EXPORTED\_TARGETS})



add dependency , include ต่างๆ เข้ากับ executable

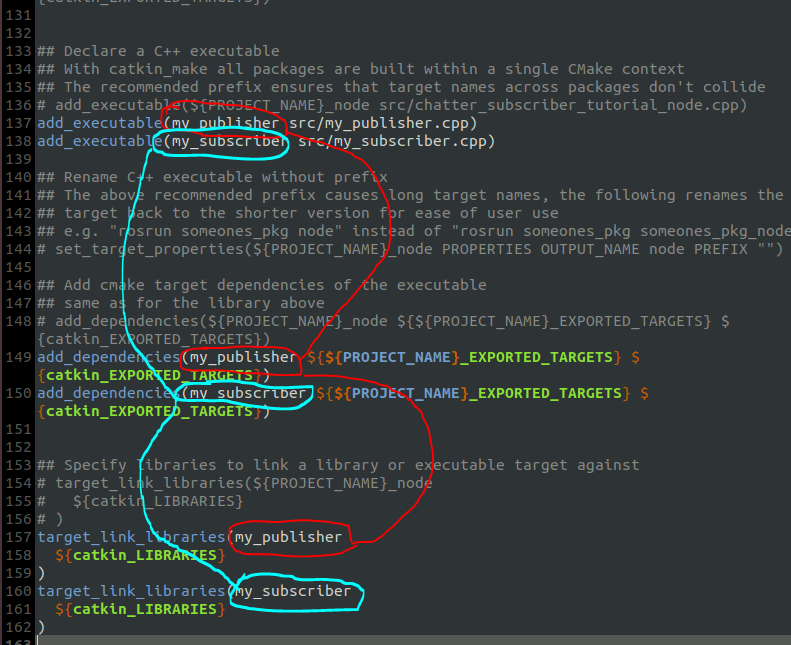
target\_link\_libraries(ชื่อexecutable วรรคตามด้วย Libraries ต่างๆ)   
ไว้จะมาเจาะกันลึกๆคราวหน้าเนาะ ตอนนี้ ให้ใส่ ${catkin\_LIBRARIES} ตามไปก่อน เพื่อใช้ Lib ที่ Catkin รวบรวมมาให้ (ได้แก่พวกของ ROS นั่นเอง)

> target\_link\_libraries(my\_publisher ${catkin\_LIBRARIES})  
> target\_link\_libraries(my\_subscriber ${catkin\_LIBRARIES})



Link Library เข้ากับ Executable

ซึ่งตอนจบจะเป็นแบบนี้



ความสำคัญตอนการกำหนด Executable , ใส่ Dependency path , และ Link Library ต่างๆ ของ Node หนึ่งใดๆ

อันนี้แบบไฟล์เปล่าๆ ไม่มี comment จาก catkin

cmake\_minimum\_required(VERSION 2.8.3)

project(chatter\_subscriber\_tutorial)

find\_package(catkin REQUIRED COMPONENTS

roscpp

std\_msgs

)

catkin\_package(

# INCLUDE\_DIRS include

# LIBRARIES chatter\_subscriber\_tutorial

# CATKIN\_DEPENDS roscpp std\_msgs

# DEPENDS system\_lib

)

include\_directories(

# include

${catkin\_INCLUDE\_DIRS}

)

add\_executable(my\_publisher src/my\_publisher.cpp)

add\_executable(my\_subscriber src/my\_subscriber.cpp)

add\_dependencies(my\_publisher ${${PROJECT\_NAME}\_EXPORTED\_TARGETS} ${catkin\_EXPORTED\_TARGETS})

add\_dependencies(my\_subscriber ${${PROJECT\_NAME}\_EXPORTED\_TARGETS} ${catkin\_EXPORTED\_TARGETS})

target\_link\_libraries(my\_publisher

${catkin\_LIBRARIES}

)

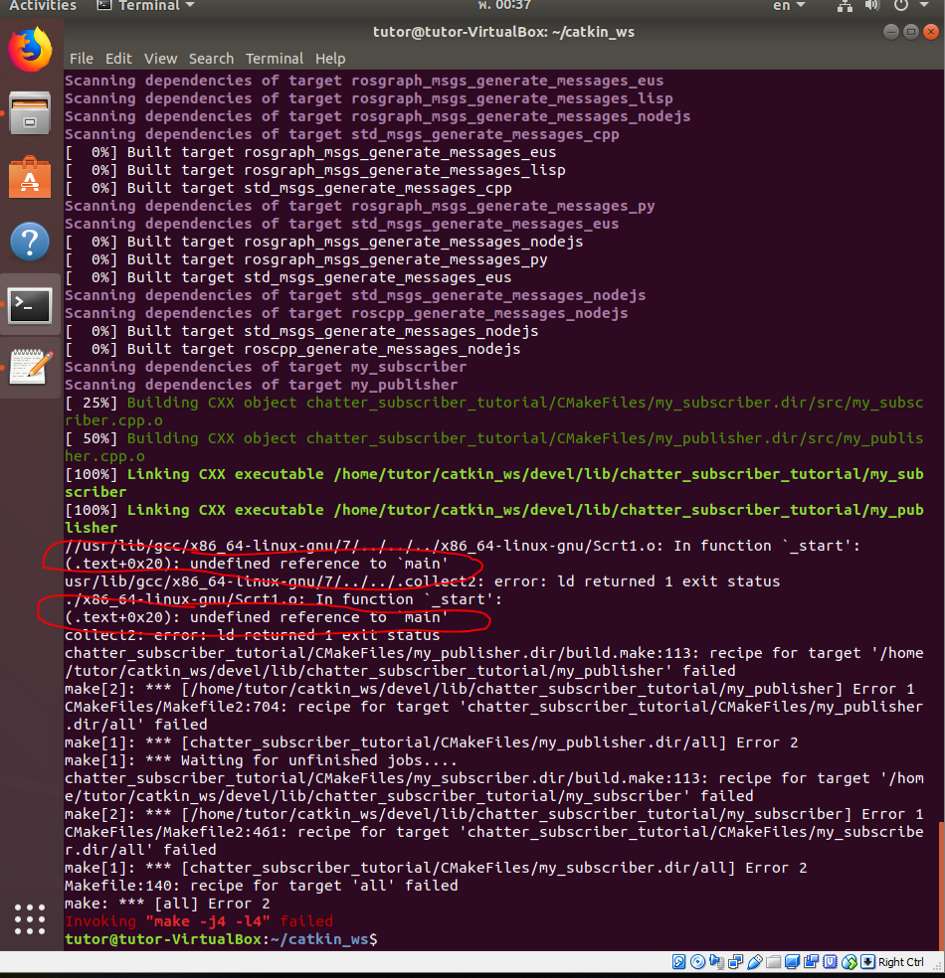
target\_link\_libraries(my\_subscriber

${catkin\_LIBRARIES}

)

3. กด Build ได้เลย==>พัง 555555

โดนดักอะดิ มันยังไม่มี code เลย แม้แต่ main ยังไม่มีเลย จะ Run ได้ยังไง !!



Code C++ ที่ไม่มี Main — — — จะรันได้ไง๊ 555555555555

ก่อนอื่น ลองเอา Code ROS มา Run ให้ได้ก่อนนะะ !!!

1. สร้างตัวส่งข้อความ “Publisher”

เอ … ระหว่างเรา Code จะลอง Compile และ Run ยังไงดีล่ะ ?

FILE : my\_publisher.cpp

#include "ros/ros.h"

#include "std\_msgs/String.h"

#include <sstream>

int main(int argc, char \*\*argv)

{

ros::init(argc, argv, "talker");

ros::NodeHandle n;

ros::Publisher chatter\_pub = n.advertise<std\_msgs::String>("chatter", 1000);

ros::Rate loop\_rate(10);

int count = 0;

while (ros::ok())

{

std\_msgs::String msg;

std::stringstream ss;

ss << "hello world " << count;

msg.data = ss.str();

ROS\_INFO("%s", msg.data.c\_str());

chatter\_pub.publish(msg);

ros::spinOnce();

loop\_rate.sleep();

++count;

}

return 0;

}

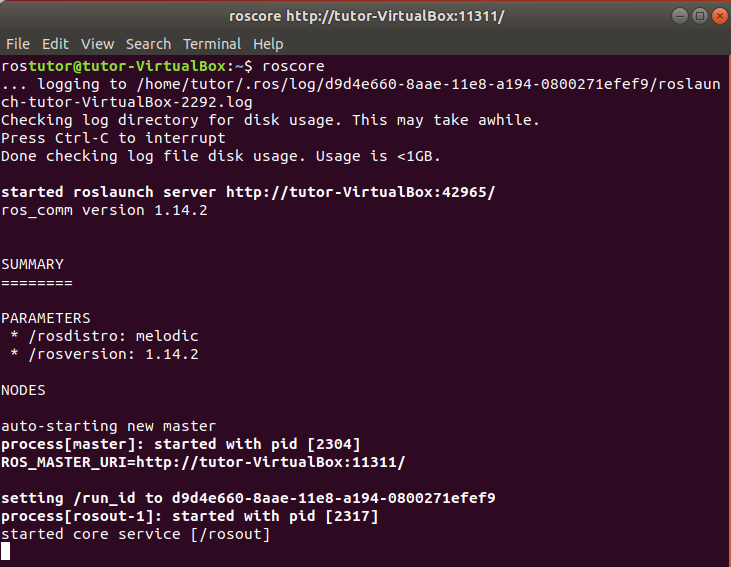
Code ส่วน Publisher — จะปล่อยข้อความ Hello world ตามด้วยเลขที่ไม่ซ้ำกัน 10 คำต่อวิเลย ผ่านทางช่องที่กำหนดไว้ “chatter”

หลังจาก Code เสร็จแล้ว ก็ Compile โลดด

>cd ~/catkin\_ws/   
>catkin\_make

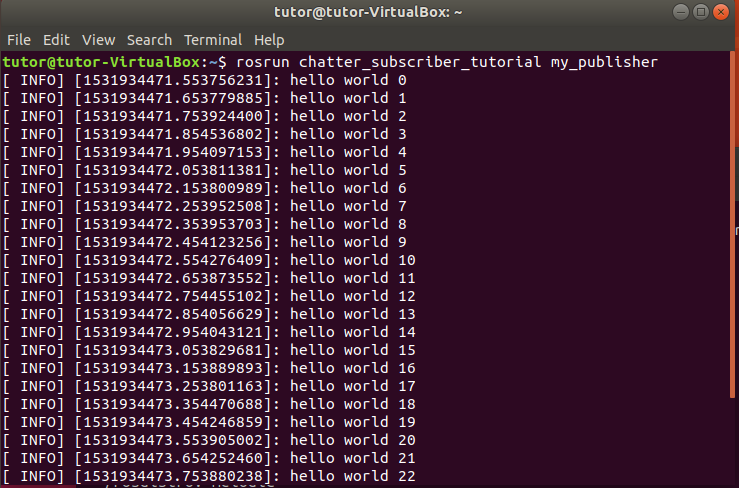
เราจะ RUN ระบบ ROS จะต้องมี Core ที่ทำหน้าที่เป็นหัวใจของระบบ

> roscore



หลังจากนั้นจึง Run NODE ของเราได้

>rosrun ชื่อpackage *ชื่อexecutableที่กำหนดในCMakeLists.txt*  
>rosrun chatter\_subscriber\_tutorial *my\_publisher*



ตะโกน Hello world เข้าไปในความว่างเปล่าาาาา ~~ ไม่มี Subscriber มาคอยฟังเลย

*เอ้อ ปิดยังไงอะ*

กด Ctrl + C นะครับ เหมือนการกด Copy เลย เป็นปุ่มในตำนานที่เอาไว้หยุดโปรแกรมที่ทำงานใน Terminal ครับ !

อ้ะๆๆ มาต่อ เหลืออีก Node นึงที่ต้องทำ คือ ตัวส่งข้อมูล

2. สร้างตัวรับข้อความ “Subscriber”

FILE : my\_subscriber.cpp

#include "ros/ros.h"

#include "std\_msgs/String.h"

void chatterCallback(const std\_msgs::String::ConstPtr& msg)

{

ROS\_INFO("I heard: [%s]", msg->data.c\_str());

}

int main(int argc, char \*\*argv)

{

ros::init(argc, argv, "listener");

ros::NodeHandle n;

ros::Subscriber sub = n.subscribe("chatter", 1000, chatterCallback);

ros::spin();

return 0;

}

Code ส่วน Subscriber — จะรับข้อความที่ Publisher ปล่อยออกมาในช่องที่กำหนดไว้(chatter) แล้วเอามาแสดง

ลองเอามา Run พร้อมๆกันดู — ปิดทุก Node ก่อน แล้วมา Run ใหม่กัน

เริ่มจาก เปิดระบบ ROS   
>roscore

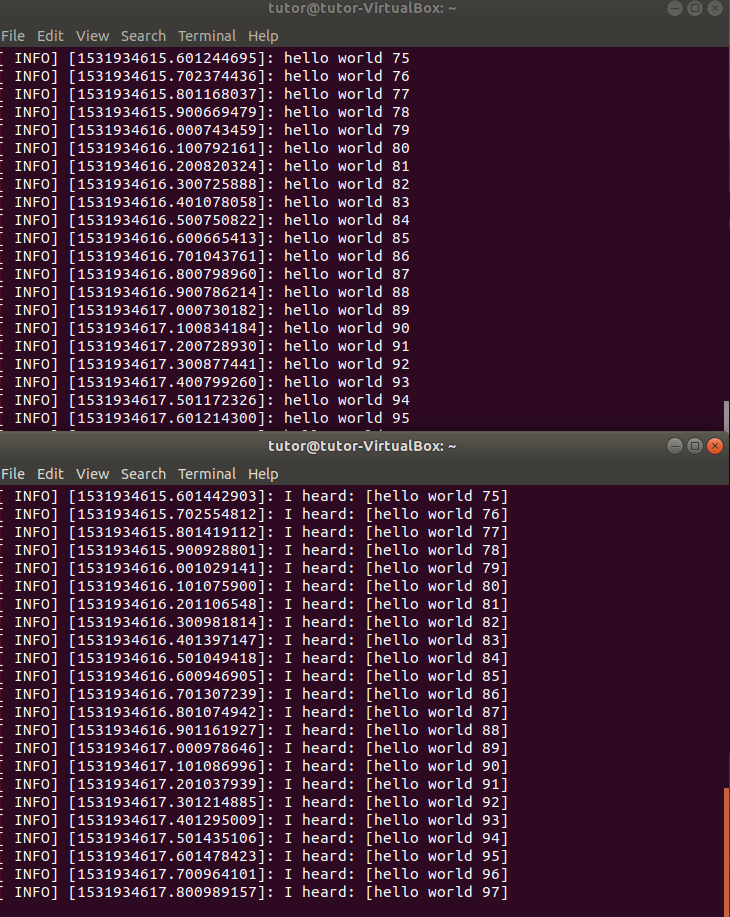
ตามมาด้วยการเปิด Subscriber (ตัวฟัง/รับข้อมูล)

>rosrun chatter\_subscriber\_tutorial my\_subscriber

หลังจากนั้น เปิด Publisher (ตัวส่งข้อมูล)

>rosrun chatter\_subscriber\_tutorial *my\_publisher*

**ผลลัพธ์ !!!!**



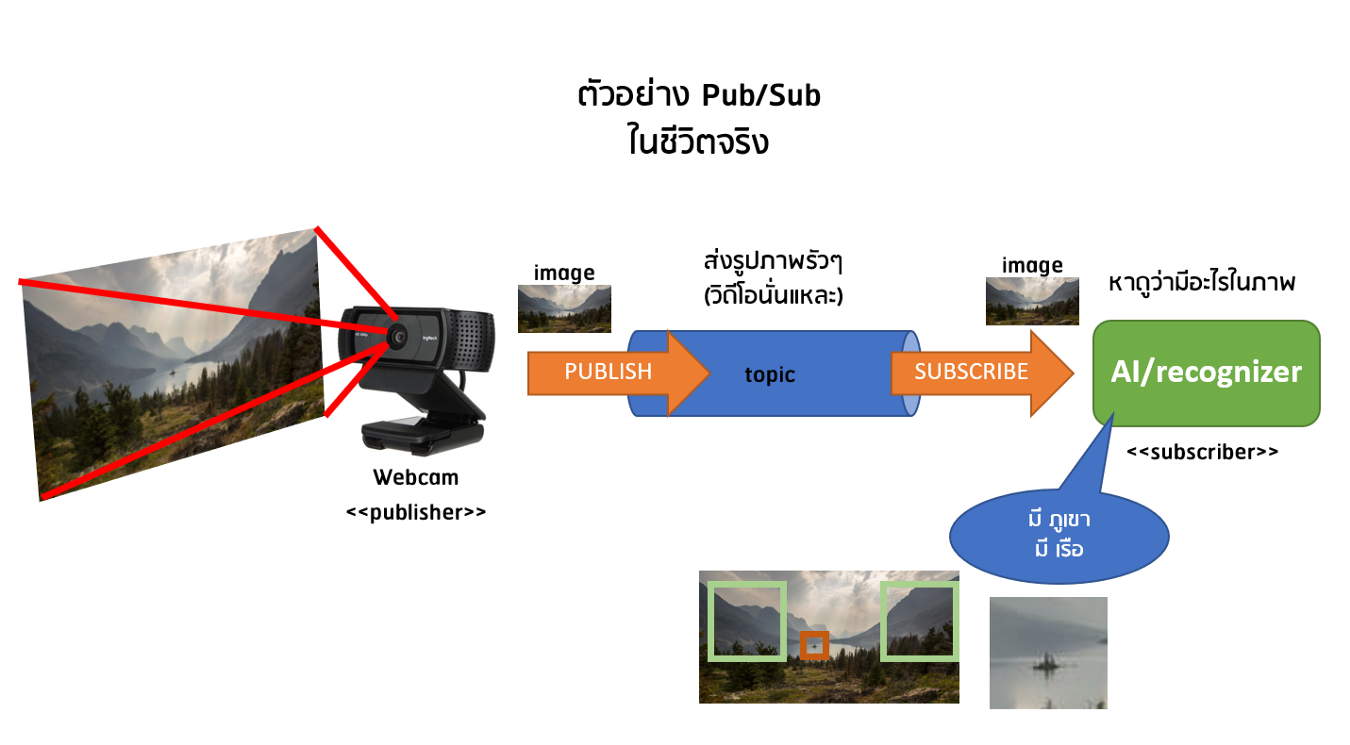
ตัวส่งกับตัวรับมันจะคุยกันได้แล้ว !!!

**มันเกิดอะไรขึ้นบ้าง ?**

ไว้ยกไป Chapter ต่อไปละกัน !! ต้องอธิบายเยอะอยู่ — บรรทัดต่อบรรทัด

ไว้จะทำเป็น Tutorial แบบ Practical Node implementation ละกันนะ คงเป็น Part 5 แหละ เพราะอ่านมาตรงนี้น่าจะเหนื่อย (โคตรๆ) แล้ว :)

**Hello ไปๆมาๆ แล้วมันจะกลายเป็นโปรแกรมหุ่นยนต์ยังไงฟระ ?**



ถ้า Publisher เป็นคนสร้างข้อมูลขึ้นมา

และ Subscriber เป็นคนที่คอยรับข้อมูลต่างๆ (เข้าไปประมวลผล)

เราจะแบ่งหมวดของ Node ต่างๆ เป็นประเภทคร่าวๆได้ดังนี้

Producer — ผลิตข้อมูลนำเข้าทุกอย่าง

Logic-รับข้อมูล ประมวลผล และอาจส่งต่อให้ Logic อีกที

Actuate-โต้ตอบกับสิ่งแวดล้อม ไม่ว่าจะการแสดงผล (Visual Display) , หรือทางกายภาพ (Physical Interaction)

ซึ่งจะสอดคล้องกับหลักการทำงานพื้นฐานของหุ่นยนต์เลยแหละ !!!

