Juan Antonio CORRALES RAMON - Laurent LEQUIEVRE

Institut Pascal UMR6602 http://ip.univ-bpclermont.fr/index.php/fr/

Equipe Maccs http://maccs.univ-bpclermont.fr/

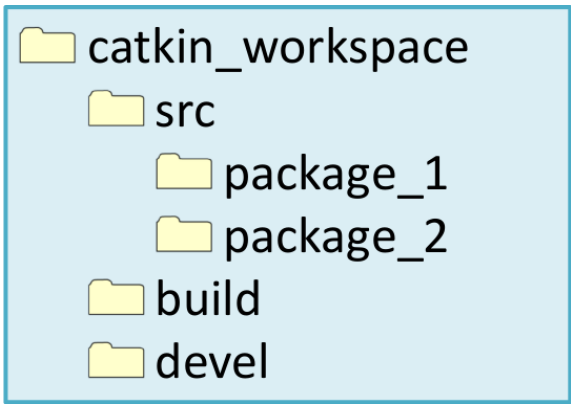
Notions de workspace catkin, de paquets, compilation

Catkin (http://wiki.ros.org/catkin) est l'outil de compilation système de ROS, il permet de compiler tout type de paquets ROS.
- est basé sur l'outil CMAKE
- est multiplate-forme (Ubuntu , Windows, linux embarqué)
- remplace l'ancien outil de compilation 'rosbuild'

Catkin est par défaut inclus dans l'installation de ROS. Mais il peut aussi être installé séparément par les paquets ubuntu via la commande 'apt-get' :

```
sudo apt-get install ros-indigo-ros
```

Catkin nécessite une structure de répertoire particulière :
- chaque projet catkin dispose de son propre 'catkin workspace'
- dans un 'catkin workspace' il peut y avoir plusieurs paquets 'catkin' qui constituent les parties/modules du projet



- Une fois la compilation du 'catkin worspace' terminée, les fichiers temporaires compilés se trouveront dans le dossier 'build' et le résultat dans le dossier 'devel'. Ces dossiers sont automatiquement créés.

Créer un 'catkin worspace' nommé 'my_catkin_ws'

1- Ouvrir une fenêtre terminal et créer un répertoire 'my_catkin_ws' et son sous dossier 'src'.

```
mkdir -p my_catkin_ws/src
```

2- Initialiser le workspace en lançant la commande 'catkin_init_worspace' dans le dossier 'src'

```
cd my_catkin_ws/src
catkin_init_workspace
```

Ceci va créer un fichier 'CMakeLists.txt' qui est en fait un lien vers le fichier 'cmake' ROS '/opt/ros/indigo/share/catkin/cmake/toplevel.cmake'
Ce fichier contenant toute l'architecture système nécessaire à la compilation du 'catkin workspace'.

3- Le worspace est vide, il n'y a pas encore de paquets dedans, mais on peut quand même lance
r un ordre de compilation en utilisant l'outil spécifique 'catkin_make' à la racine du workspace.

```
cd ..
catkin_make
```

Une fois la compilation terminée on peut observer que les dossiers 'build' et 'devel' ont été créés.

4- Rendre visible au système ROS le nouveau workspace créé en lançant le fichier bash 'setup.bash' qui se trouve dans le dossier 'devel'. Cela positionnera dans votre fenêtre terminal des variables d'environnement nécessaires à ROS, dont la variable 'ROS_PACKAGE_PATH'.

```
env | grep ROS_PACKAGE_PATH
ROS_PACKAGE_PATH=/opt/ros/indigo/share:/opt/ros/indigo/stacks

source devel/setup.bash

env | grep ROS_PACKAGE_PATH
ROS_PACKAGE_PATH=/.../my_catkin_ws/src:/opt/ros/indigo/share:/opt/ros/indigo/stacks
```

Créer un paquet ROS dans le workspace

Un paquet 'catkin' ROS est en fait un composant ROS pouvant se compiler et s'exécuter séparément. Il doit être forcément créé dans un 'catkin workspace'. Physiquement parlant, un paquet est un répertoire dans lequel il faut au minimum 2 fichiers, un 'CMakeLists.txt' pour les ordres de compilation, un fichier 'package.xml' contenant les 'metadata' du paquet 'catkin' (nom, auteur, description, dépendances avec d'autres paquets).

1- Créer un paquet ROS 'catkin' nommé 'my_first_package' en utilisant la commande 'catkin-create-pkg' dans le dossier 'src' du workspace.

```
cd my_catkin_ws/src
catkin-create-pkg my_first_package roscpp
```

Ici on a ajouté un 3ième paramètre à la commande, permettant de spécifier une dépendance au paquet '**roscpp**' (http://wiki.ros.org/roscpp). Ce paquet permettant de s'interfacer en C++ avec des outils ROS comme des 'topics', 'services', 'paramètres' que l'on étudiera plus tard.

```
ls
CMakeLists.txt  my_first_package
cd my_first_package
ls
CMakeLists.txt  include  package.xml  src
```

Le nouveau paquet créé contient les fichiers 'CMakeLists.txt' et 'package.xml' mais aussi deux dossiers 'src' et 'include' dans lesquels on pourra mettre le code source C++.

2- Ouvrir avec votre éditeur favori le fichier 'package.xml' afin de compléter la partie 'auteur –

description – licence'

```xml
<?xml version="1.0"?>
<package>
  <name>my_first_package</name>
  <version>0.0.0</version>
  <description>The my_first_package package</description>

  <!-- One maintainer tag required, multiple allowed, one person per tag -->
  <!-- Example:  -->
  <!-- <maintainer email="jane.doe@example.com">Jane Doe</maintainer> -->
  <maintainer email="lequievre.lequievre@univ-bpclermont.fr">Laurent LEQUIEVRE</maintainer>


  <!-- One license tag required, multiple allowed, one license per tag -->
  <!-- Commonly used license strings: -->
  <!--   BSD, MIT, Boost Software License, GPLv2, GPLv3, LGPLv2.1, LGPLv3 -->
  <license>BSD</license>

```

Pour en savoir plus sur les types de licences : http://opensource.org/licenses/alphabetical

3- Observer la deuxième partie du fichier 'package.xml' qui contient les informations :

```xml
  <buildtool_depend>catkin</buildtool_depend>
  <build_depend>roscpp</build_depend>
  <run_depend>roscpp</run_depend>


  <!-- The export tag contains other, unspecified, tags -->
  <export>
    <!-- Other tools can request additional information be placed here -->

  </export>
</package>
```

La dépendance au paquet 'roscpp' pour la compilation du paquet 'my_first_package' s'est traduit par l'ajout d'une balise XML '<build_depend>'. De même la balise '<run_depend>' exprime une dépendance pour l'exécution. Enfin la balise '<buildtool_depend>' permet de spécifier le type d'outil pour compiler le paquet, ici 'catkin'.

La balise '<export>' peut servir à indiquer que le paquet est un 'super paquet' (rassemblant plusieurs paquets – '<metapackage/>') ou encore qu'il fournit des 'plugin' (un 'plugin' pour l'outil de visualisation 'rviz' par exemple).

Remarque :
Si vous souhaitez diffuser votre paquet 'catkin' dans la communauté ROS, alors ces informations sont nécessaires.

http://wiki.ros.org/catkin/package.xml

4- Observer le début du fichier 'CMakeLists.txt' du paquet créé :

```
1  cmake_minimum_required(VERSION 2.8.3)
2  project(my_first_package)
3
4  ## Find catkin macros and libraries
5  ## if COMPONENTS list like find_package(catkin REQUIRED COMPONENTS xyz)
6  ## is used, also find other catkin packages
7  find_package(catkin REQUIRED COMPONENTS
8    roscpp
9  )
10
```

La macro 'find_package' permet de définir et de trouver les paquets nécessaires à la compilation du projet. Ici le projet '**my_first_package**' (qui est un paquet ROS) a besoin d'un autre paquet ('**roscpp**') pour être compilé. On a donc besoin de connaître l'emplacement de ses fichiers 'headers' mais aussi de connaître l'emplacement et le nom de ses librairies.

On aurait pu l'écrire, plus classiquement à la 'cmake', comme ceci :
```
find_package(roscpp)
```

Suite à l'appel du 'find_package', si le paquet à trouver existe sur le système et que l'auteur de celui-ci a respecté des conventions 'cmake' alors des variables d'environnement seront automatiquement positionnées :
<NAME>_FOUND (NAME = nom du paquet) si le paquet est trouvé
<NAME>_INCLUDE_DIRS indique le dossier ou se trouve les 'headers' du paquet
<NAME>_LIBRARIES indique le nom et dossier ou se trouvent les librairies du paquet

Pour respecter ces conventions il faut fournir un fichier nommé '<NAME>Config.cmake' dans le lequel on affecte le contenu des variables '<NAME>_FOUND , <NAME>_INCLUDE_DIRS, <NAME>_LIBRARIES …'. Pour faire cela, il faut bien maîtriser 'cmake'.

Dans le cas de 'roscpp', il y a un fichier nommé 'roscppConfig.cmake' qui se trouve dans le dossier '/opt/ros/indigo/share/roscpp/cmake'.

Le fait d'exprimer que 'roscpp' est un 'catkin component' comme ceci :
```
find_package(catkin REQUIRED COMPONENTS roscpp)
```
permet de positionner automatiquement des variables d'environnement spécifiques à 'catkin' qui commencent par '**catkin_**', qui contiendront toutes les infos cumulées des 'component' spécifiés.

Voici quelques exemples de variables 'catkin' : '**catkin_INCLUDE_DIRS**', '**catkin_LIBRARIES**'.

Un exemple de plusieurs 'catkin component' séparés par une espace :
```
find_package(catkin REQUIRED COMPONENTS roscpp turtlesim std_msgs)
```

Dans cet exemple, la variable 'catkin_LIBRARIES' contiendra toutes les librairies de tous les 'catkin component' spécifiés. Ce qui sera beaucoup plus simple à utiliser par la suite.

http://wiki.ros.org/catkin/CMakeLists.txt

5- Modifier le fichier 'CMakelists.txt' en ajoutant après 'find_package' le code 'cmake' suivant :

```
if(${roscpp_FOUND})
```

```
        message(STATUS "The package roscpp is found !")
        message(STATUS "include dir = ${roscpp_INCLUDE_DIRS}")
        message(STATUS "librairies = ${roscpp_LIBRARIES}")
else(${roscpp_FOUND})
        message(STATUS "The package roscpp is not found !")
endif(${roscpp_FOUND})
```

6- Observer le reste du fichier 'CMakeLists.txt' du paquet créé :

```
107  ############
108  ## Build ##
109  ############
110
111  ## Specify additional locations of header files
112  ## Your package locations should be listed before other locations
113  # include_directories(include)
114  include_directories(
115    ${catkin_INCLUDE_DIRS}
116  )
```

Ici est indiqué, via la macro 'include_directories', d'inclure dans la compilation du paquet les dossiers 'include' de 'catkin' :
catkin_INCLUDE_DIRS = /opt/ros/indigo/include;/usr/include

C'est dans ces dossiers que seront recherchés les fichiers 'headers'.

Ajouter ceci après 'include_directories' :

```
message(STATUS "===== catkin_INCLUDE_DIRS = ${catkin_INCLUDE_DIRS}")
message(STATUS "==== catkin libraries : ${catkin_LIBRARIES}")
```

6bis- Le paquet '**my_first_package**' peut être déclaré lui même comme un paquet 'catkin'. Pour cela il faut utiliser la macro 'catkin_package' dans le fichier 'CMakeLists.txt'.

Dans notre cas on indique une simple dépendance au paquet 'roscpp' :

```
catkin_package(
  CATKIN_DEPENDS roscpp
)
```

L'idée générale étant d'exporter l'interface, l'environnement nécessaire, à l'utilisation du paquet 'my_first_package'.

Modifier le fichier 'CMakeLists.txt' en conséquence.

Voici un autre exemple plus détaillé de 'CMakeLists.txt' :

```
cmake_minimum_required(VERSION 2.8.3)
project(foo)

find_package(Boost REQUIRED
  COMPONENTS
  system
  thread
```

```
)

find_package(PythonLibs REQUIRED)
find_package(OpenGL REQUIRED)

find_package(catkin REQUIRED
  COMPONENTS
  rosconsole
  roscpp
)

include_directories(
  include
  ${catkin_INCLUDE_DIRS}
  ${OPENGL_INCLUDE_DIR}
  ${PYTHON_INCLUDE_PATH}
)

catkin_package(
  INCLUDE_DIRS include ${OPENGL_INCLUDE_DIR}
  LIBRARIES foo ${OPENGL_LIBRARIES}
  CATKIN_DEPENDS roscpp
  DEPENDS Boost
)
```

Ici on précise via la macro 'catkin_package' :
→ que les 'headers' du paquet se trouvent dans le dossier 'include' et le dossier 'include' de 'OpenGL'.
→ que le paquet a une dépendance avec un autre paquet 'catkin' 'roscpp'.
→ que le paquet a une dépendance avec le paquet (pas 'catkin') 'Boost'.
→ que le paquet nécessite les librairies 'foo' et les librairies de 'OpenGL'.

7- Compiler le workspace et observer l'affichage des 'message cmake'.

```
cd my_catkin_ws
catkin_make
```

```
-- ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
-- ~~  traversing 1 packages in topological order:
-- ~~  - my_first_package
-- ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
-- +++ processing catkin package: 'my_first_package'
-- ==> add_subdirectory(my_first_package)
-- The package roscpp is found !
-- include dir = /opt/ros/indigo/include;/usr/include
-- librairies = /opt/ros/indigo/lib/libroscpp.so;/usr/lib/x86_64-linux-gnu/libbo
```

Il est à noter que 'catkin' indique les paquets à compiler du workspace et lance leur compilation.

8- Utiliser 'roscd' pour changer de dossier.

'roscd' fait partie de 'rosbash' (http://wiki.ros.org/rosbash) contenant les outils nécessaires pour utiliser ros sous 'bash'.
'roscd' suivi du nom du paquet permet de se positionner directement dans le dossier du paquet.

```
cd my_catkin_ws
roscd my_first_package
```

```
…/my_catkin_ws/src/my_first_package$
```

9- Utiliser 'rospack' (http://wiki.ros.org/rospack) pour obtenir des infos sur un paquet

'rospack' fait partie des 'ROS package management tool' et peut être utilisé avec différents paramètres.

→ trouver le chemin absolu ou se trouve le paquet :

```
rospack find my_first_package
…/my_catkin_ws/src/my_first_package
```

Pour faire cela 'rospack' recherche dans le dossier contenu dans la variable d'environnement 'ROS_ROOT' mais aussi dans tous les dossiers contenus dans la variable 'ROS_PACKAGE_PATH'.

→ afficher les dépendances (paquets) de premier niveau d'un paquet :

```
rospack depends1 my_first_package
roscpp
```

Pour faire cela, 'rospack' consulte le contenu du fichier 'package.xml' du paquet 'my_first_package'.

→ afficher toutes les dépendances (paquets) d'un paquet :

```
rospack depends my_first_package
cpp_common
rostime
...
roscpp
```

Ici 'rospack' applique une consultation récursive. Le paquet 'my_first_package' dépend de 'roscpp' qui lui même dépend de 'cpp_common message_runtime …' et ainsi de suite ...

→ afficher tous les paquets installés dans le système

```
rospack list
```