Написание простого publisher и subscriber

1. Написание Узла Издателя (Publisher Node)

В ROS, узел (node) — это термин для обозначения исполняемого файла, который подключён к сети ROS.

1.1 Перейдите в каталог в пакета ros_wokrspace

1.2 Создайте пакет
roscreate-pkg tutorial std_msgs roscpp rospy

1.3 Создайте внутри пакета tutorial файл src/talker.cpp и вставьте в него следующий код:

```
#include "ros/ros.h"
#include "std_msgs/String.h"

#include <sstream>

/**
 * Этот пример демонстрирует простую отправку сообщений через систему ROS.
 */
int main(int argc, char **argv)
{
  /**
   * Функция ros::init() требуется для проверки аргументов argc и argv, чтобы выполнить
   * преобразования или переопределение аргументов ROS, задаваемых через командную строку.
   * Для программного переопределения вы можете использовать другую версию init(), которая
   * принимает переопределение напрямую, но для большинства программ командной сроки, обработка
   * argc и argv - простейший путь реализовать это. Третий аргумент init() - это название узла.
   *
   * Вы должны вызвать одну из версий ros::init() перед использованием любых других
   * частей системы ROS.
   */
  ros::init(argc, argv, "talker");

  /**
   * NodeHandle - главная точка доступа для взаимодействия с системой ROS.
   * Конструктор NodeHandle полностью инициализирует этот узел, а в конце,
   * деструктор NodeHandle завершит работу узла.
   */
```

```
  ros::NodeHandle n;

  /**
   * Функция advertise() определяет для ROS что вы будете публиковать в задаваемую Тему.
   * Это включает в себя вызов Мастер-узла ROS, который сохраняет регистрационные данные
   * кто является издателем, а кто является подписчиком.
   * После завершения advertise(), Мастер-узел ROS извещает всех, кто пытается подписаться
   * на заданную Тему и они должны устанавливать прямое соединение
   * (peer-to-peer connection)  между собой и этим узлом.
   * Функция advertise() возвращает объект Издатель (Publisher), который позволяет Вам
   * публиковать сообщения в Тему, с помощью метода publish(). После того, как все
   * копии возвращённого объекта Издателя (Publisher) будут разрушены, Тема будет автоматически
   * отписана (unadvertised).
   *
   * Второй параметр метода advertise() - это размер очереди сообщений, используемый для
   * публикуемых сообщений. Если сообщения публикуются быстрее, чем они могут быть отправлены,
   * этот параметр задаёт сколько сообщений необходимо сохранять в буфере прежде чем
   * бросать исключение.
   */

  ros::Publisher chatter_pub = n.advertise<std_msgs::String>("chatter", 1000);

  ros::Rate loop_rate(10);

  /**
   * count - счётчик, в котором сохраняется число отправленных сообщений. Используется
   * при создании уникальной строки для каждого сообщения.
   */
  int count = 0;
  while (ros::ok())
  {

    /**
     * Это объект сообщения. Вы сохраняете в него данные, а затем публикуете их.
     */

    std_msgs::String msg;
```

```
    std::stringstream ss;
    ss << "hello world " << count;
    msg.data = ss.str();

    ROS_INFO("%s", msg.data.c_str());

    /**
     * Метод publish() для отправки сообщений. Параметр - это объект сообщения.
     * Тип объекта должен совпадать с типом, который задавался параметром шаблона
     * при вызове advertise<>(), как это сделано в конструкторе выше.
     *
     */

    chatter_pub.publish(msg);

    ros::spinOnce();

    loop_rate.sleep();
    ++count;
  }


  return 0;
}
```

2. Написание Подписчика узла (Subscriber Node)

2.1 Создайте файл src/listener.cpp в пакете tutorial и вставьте в него следующий код:

```
#include "ros/ros.h"
#include "std_msgs/String.h"

/**
 * Этот пример демонстрирует простое получение сообщений в системе ROS.
 */

void chatterCallback(const std_msgs::String::ConstPtr& msg)
{
  ROS_INFO("I heard: [%s]", msg->data.c_str());
}

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

  /**
   * Функция ros::init() требуется для проверки аргументов argc и argv, чтобы выполнить
```

```
   * преобразования или переопределение аргументов ROS, задаваемых через командную строку.
   * Для программного переопределения вы можете использовать другую версию init(), которая
   * принимает переопределение напрямую, но для большинства программ командной сроки, обработка
   * argc и argv - простейший путь реализовать это. Третий аргумент init() - это название узла.
   *
   * Вы должны вызвать одну из версий ros::init() перед использованием любых других
   * частей системы ROS.
   */

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

  /**
   * NodeHandle - главная точка доступа для взаимодействия с системой ROS.
   * Конструктор NodeHandle полностью инициализирует этот узел, а в конце,
   * деструктор NodeHandle завершит работу узла.
   */

  ros::NodeHandle n;

  /**
   * subscribe() сообщает ROS, что вы хотите получать сообщения
   * на заданную тему. Это приводит к запросу Мастер-узла ROS,
   * который содержит регистрационные данные о том, кто публикуте и
   * кто получает сообщения. Сообщения передаются в функцию обратного вызова, здесь
   * она называется chatterCallback. subscribe() возвращает объект подписчика(Subscriber), который вы
   * должны держать, пока вы не захотите отказаться от подписки. Когда все копии объекта подписки
   * выходят из области видимости, то обратный вызов будет автоматически отписан от
   * этой темы.
   *
   * Второй параметр subscribe() указывает размер очереди сообщений.
   * Если сообщения прибывают быстрее, чем они обрабатываются, это
   * число указывает количество сообщений, которые будут сохраняться
   * в буфере прежде чем удалять самые старые.
   */

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

  /**
   * ros::spin() будет входить в цикл, прокачки обратных вызовов. В этой версии, все
   * обратные вызовы будут вызвана из этой нити (основноой).
   * ros::spin() будет завершён после нажатия Ctrl-C, или отключения узла от мастера.
   */

  ros::spin();

  return 0;
}
```

3. Сборка ваших узлов

roscreate-pkg создаст Makefile по-умолчанию и CMakeLists.txt для вашего пакета.

CMakeList.txt должен выглядеть примерно так:

```
cmake_minimum_required(VERSION 2.4.6)
include($ENV{ROS_ROOT}/core/rosbuild/rosbuild.cmake)

# Set the build type.  Options are:
#  Coverage       : w/ debug symbols, w/o optimization, w/ code-coverage
#  Debug          : w/ debug symbols, w/o optimization
#  Release        : w/o debug symbols, w/ optimization
#  RelWithDebInfo : w/ debug symbols, w/ optimization
#  MinSizeRel     : w/o debug symbols, w/ optimization, stripped binaries
#set(ROS_BUILD_TYPE RelWithDebInfo)

rosbuild_init()

#set the default path for built executables to the "bin" directory
set(EXECUTABLE_OUTPUT_PATH ${PROJECT_SOURCE_DIR}/bin)
#set the default path for built libraries to the "lib" directory
set(LIBRARY_OUTPUT_PATH ${PROJECT_SOURCE_DIR}/lib)

#uncomment if you have defined messages
#rosbuild_genmsg()
#uncomment if you have defined services
#rosbuild_gensrv()

#common commands for building c++ executables and libraries
#rosbuild_add_library(${PROJECT_NAME} src/example.cpp)
#target_link_libraries(${PROJECT_NAME} another_library)
#rosbuild_add_boost_directories()
#rosbuild_link_boost(${PROJECT_NAME} thread)
#rosbuild_add_executable(example examples/example.cpp)
#target_link_libraries(example ${PROJECT_NAME})
```

Добавьте в конец файла следующее:

```
rosbuild_add_executable(talker src/talker.cpp)
rosbuild_add_executable(listener src/listener.cpp)
```

http://wiki.ros.org/ROS/Tutorials/WritingPublisherSubscriber(c++)