

YTU RACING CPP PUBLISHER, SUBSCRIBER AND LAUNCH FILE REPORT

Name-Surname :Adem Berke Nargul

Department : Autonomous

Date : 29.09.2023

```
catkin_ws > src > pub_cpp > src > G- pub_script.cpp > ...
1  #include <ros/ros.h>
2  #include "ros/init.h"
3  #include <std_msgs/String.h>
4
5  int main(int ac, char **av) {
6
7      ros::init(ac, av, "pub_script", 0);
8      // Yayınlanacak olan mesaj değişkenini tanımlıyoruz.
9      std_msgs::String message;
10     // ros::NodeHandle türünden bir değişken tanımlıyoruz.
11     ros::NodeHandle nodeHandle;
12     // Publisherımızı oluşturuyoruz.
13     // ros::Publisher türünden bir değişken tanımlıyoruz.
14     ros::Publisher publisher;
15
16     // ros::init methodu ile "publisher_script" adında bir node başlatıyoruz.
17
18
19     // Bir rate belirliyoruz. Rate, bir durumun saniyede kaç kere çalışacağını
20     // kontrol etmek için kullanılan bir araçtır. 15 olarak ayarladığım rate 15 Hz
21     // frekansında yayın yapacaktır.
22     ros::Rate rateHz(2);
23
24
25     // Hemen ardından bu publisherı string türünden bir publisher olarak
26     // belirleyip bir konu yayınlamaya ayarlıyoruz. Buradaki 1000 sayısı mesaj
27     // kuyruğunun kapasite büyüklüğünü ifade ediyor. Yani geçici olarak tutulan
28     // maximum mesaj boyutu.
29     publisher = nodeHandle.advertise<std_msgs::String>("Anything", 1000);
30
31     // döngünün koşulu olan "ok()" methodu eğer döngü sonlanmadıysa veya ros
32     // konusu hala yayındaysa "true" döner ve çalışmaya devam eder.
33     while (ros::ok()) {
34         message.data = "Selam Millllettt! 101010101!";
35         publisher.publish(message);
36         rateHz.sleep();
37         ros::spinOnce();
38     }
39 }
```

Publisher Kodu

Bu script için nerdeyse bütün methodları içeren "ros" kütüphanesini, string türünden bir veri akışı sağlayacağımız için ayrıca "std_msgs" kütüphanesini kullanıyoruz.

1. ros::init() : ros kütüphanesinden "init()" metodu. Bu metod ros kütüphanesi ile ilgili yapacağımız bütün işlemleri ve döngüleri(spinOnce()) çalıştırabilmemizi sağlıyor. Bu methodu çağırmazsak publisher çalışmaz.
2. std_msgs::String : std_msgs kütüphanesinden String veri yapısı. Bu veri yapısı, string türünden verileri tutabilmemizi sağlıyor.
3. ros::NodeHandle : ros kütüphanesinden bir NodeHandle sınıf. Ros uygulamalarında yayın yapma, abone olma, hizmet çağrısı yapma ve parametre yönetimi gibi işlemleri gerçekleştirmek için kullanılır.
4. ros::Publisher : ros kütüphanesinden Publisher veri yapısı. Bu veri yapısı, Publisherımızı tutuyor ve içinde publisher ile ilgili bütün özellikleri ve verileri içeriyor.
5. ros::Rate() : ros kütüphanesinden Rate() metodu. Bu metod, ROS'ta zaman tabanlı döngüler oluşturmak için kullanılır ve belirli bir frekansta çalışan döngüler oluşturmak için tasarlanmıştır. Bunu da saniyede 2 kere çalışacak şekilde ayarlıyoruz.

- 6.advertise() : Bu metot ile bir publisher oluşturabiliyoruz. İlk parametresi yayın yapacağı konunun ismi, ikinci parametresi ise yayının kuyruğunun boyutunu belirten sayı. Kuyruk boyutu, yayıncı tarafından hızlı yayınlama ve abone olan düğümlerin yavaş çalışması arasındaki dengeyi sağlamak için kullanılır.
- 7.ros::ok() : Bu metot ros düğümü çalıştığı sürece eğer döngü sonlanmadıysa veya ros::shutdown() çağrılmadıysa "true" değerini döndürür.
- 8.sleep() : Bu metot rate için belirlediğimiz aralığı tamamlayabilmek için gerektiği kadar işlemi uyutuyor.
- 9.ros::spinOnce() : Bu metot ros düğümünün olay döngüsünü bir kez döndürerek ros olaylarına yanıt verme yeteneği sağlar.

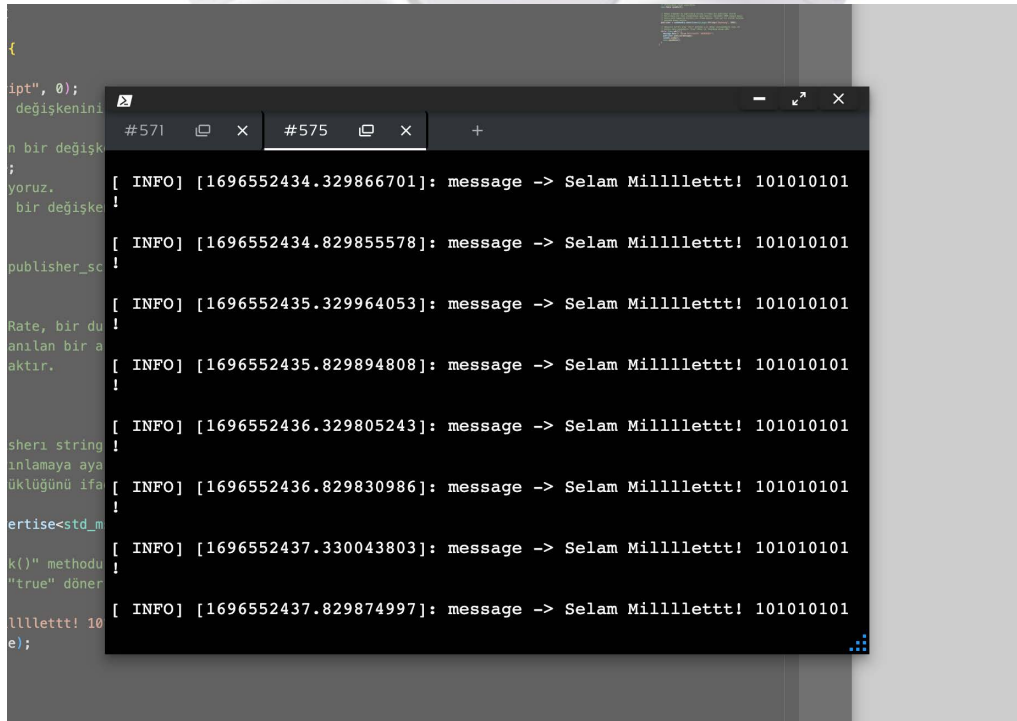
```
catkin_ws > src > sub_cpp > src > G+ sub_script.cpp
1  #include "ros/init.h"
2  #include "ros/node_handle.h"
3  #include "ros/spinner.h"
4  #include "ros/subscriber.h"
5  #include <ros/ros.h>
6  #include <std_msgs/String.h>
7
8  void handle(const std_msgs::String::ConstPtr &message) {
9      ROS_INFO("message -> %s\n", message->data.c_str());
10 }
11
12 int main(int ac, char **av) {
13     ros::init(ac, av, "sub_script", 0);
14     ros::Subscriber aboneyim_abone;
15     ros::NodeHandle nodeHandle;
16
17     aboneyim_abone = nodeHandle.subscribe("Anything", 1000, handle);
18
19     ros::spin();
20 }
```

1. ros::Subscriber : Bu metot ros::Subscriber türünden bir veri yapısı tanımlamanızı sağlıyor. Bu veri yapısı, Subscriber tutuyor ve içinde subscriber ile ilgili bütün özellikleri ve verileri içeriyor.
2. ROS_INFO : Ros düğümünün çalışması sırasında hata ayıklama (debugging) amacıyla kullanılır ve programcılara bilgi verir.

```
catkin_ws > src > pub_cpp > launch > <> general.launch
1 <launch>
2   <node name="Anything2" pkg="sub_cpp" type="sub_script" output="screen">
3     <!-- Enter arguments and parameters-->
4   </node>
5   <node name="Anything" pkg="pub_cpp" type="pub_script" output="screen">
6     <!-- Enter arguments and parameters-->
7   </node>
8 </launch>
```

Launch File : Ros'ta birçok düğümü (node) başlatmak ve bunları yönetmek için kullanılan XML formatındaki bir dosyadır. Launch dosyaları, birden fazla ROS düğümünü, parametreleri, argümanları ve diğer konfigürasyonları başlatmak için kullanılır. Bu dosyalar, ROS uygulamalarını daha düzenli ve yönetilebilir hale getirmeye yardımcı olur.

Bu kod, subscriber ve publisherı aynı anda başlatmak için yazılmış bir launch file örneği.



```
{
  ipt", 0);
  değişkenini
n bir değişke
;
yoruz.
bir değişke
publisher_sc
Rate, bir du
anılan bir a
aktır.
sherı string
ınlamaya aya
üklüğünü ifa
ertise<std_m
k()" methodu
"true" döner
llllettt! 10
e);
```

```
#571 x #575 x +
[ INFO] [1696552434.329866701]: message -> Selam Millllettt! 101010101
[ INFO] [1696552434.829855578]: message -> Selam Millllettt! 101010101
[ INFO] [1696552435.329964053]: message -> Selam Millllettt! 101010101
[ INFO] [1696552435.829894808]: message -> Selam Millllettt! 101010101
[ INFO] [1696552436.329805243]: message -> Selam Millllettt! 101010101
[ INFO] [1696552436.829830986]: message -> Selam Millllettt! 101010101
[ INFO] [1696552437.330043803]: message -> Selam Millllettt! 101010101
[ INFO] [1696552437.829874997]: message -> Selam Millllettt! 101010101
```