

ROS

Sumário

1	Introduçao	2
1.1	Elementos Básicos	2
1.2	Links uteis	2
2	Instalação	2
2.1	Passos da Instalação	2
3	Primeiro projeto	2
4	Nós e topicos	2
5	primeiro Nó	2
5.1	codigo detalhado	3
5.2	explicação terminal	3
6	Nó em python	3
6.1	detalhameento do código	4
7	ROS Publisher e Subscriber em C++ - ParteI	4
7.1	detalhamento counter.cpp	5
7.2	definindo um timer no counter.cpp	6
7.2.1	detalhamento do timercallback	6
8	Publisher e subscriber em c++ partII	7
9	Publisher e subscriber em python part I	8
10	publish e subscriber em python partII	8
11	ROSService server com c++	9
12	ROS service client em c++	10
13	ROS service server em python	11
14	ROS Service Client em Python	12
15	Launch files	12
16	ROS paramater cpp part I	13
16.1	adendos	14
17	ROS parameter cpp partII	15
17.1	checagem da existencia do parametro	15
17.2	deletando parametros	16
17.3	adicionando parametros a um arquivo separado	17
18	ROS parameter com python	17
19	Custom Message	19
20	Custom message com c++	20
21	Custom message em python	25
22	Custom Service	29
23		29

1 Introdução

1.1 Elementos Básicos

- Nós (nodes): executaveis, processos, códigos que realizam alguma atividade, não necessariamente no mesmo sistema
- Tópicos (topics): forma de troca de informações entre os Nós
- Serviços (services): interação entre nós de forma a solicitar uma ação ou um dado como resposta
- Pacotes (packages): forma de organização dos arquivos e codigos no Ros.favorece a modularidade das aplicações do sistema.

1.2 Links uteis

Instalação completa framework site-passo-a-passo site-video
Documentacao acerca do framework wiki ros

Documentacao do que já foi feito no ROS Tutorais ros

Forum para tirar duvidas. ROS Answer

discursões e divulgações do que será feito Discourse ROS

2 Instalação

2.1 Passos da Instalação

video de instalação
link para os passos na wiki ROS

3 Primeiro projeto

Antes de criar um novo pacote, é importante garantir que o ambiente ROS esteja devidamente configurado. Isso é feito utilizando o comando source para carregar as variáveis de ambiente do ROS.
Utilize o comando \$catkin_create_pkg\$ para criar um novo pacote. O comando abaixo cria um pacote chamado \$nome_project\$ com dependências nas bibliotecas roscpp, rospy e \$std_msgs:\$

Comandos no terminal bash linux

```
1 source devel/setup.bash // sempre manter atualizado
2
3 catkin_create_pkg nome_project roscpp rospy std_msgs /criando um package
4 roscpp <diretorio_destino>
```

- CMakeLists.txt
- package.xml
- include/
- *nome_project/*
- src/, msg/

4 Nós e topicos

Podem ser instaciados multiplas vezes

5 primeiro Nó

talker_cpp.cpp

```
1 catkinmake
2 rostopic list
3 rostopic echo /chatter
4
5 0 codigo esta usando o publisher rostopic echo /chatter
6 esse comando mostra o que esta acontecendo dentro do publisher
7
8 rostopic hz /chatter // mostra a frequencia de amostragem
9
10 rostopic info /chatter quem ta publicando e quem ta escrito no no
```

talker_cpp.cpp

```
1 #include "ros/ros.h"
2 #include "std_msgs/String.h"
3
4
5 int main (int args, char **argv){
6     ros::init(args, argv, "talker_cpp");
7
8     ros::NodeHandle nh;
```

```
10         ros::Rate loop_rate(10);
11
12         ros:: Publisher chatter_pub = nh.advertise<std_msgs::String>("chatter", 1000);
13
14         int count = 0;
15         while(ros:: ok()){
16             std::string txt = "Ola ross! contagem:" + std::to_string(count);
17
18             ROS_INFO("%s", txt.c_str());
19
20             std_msgs::String msg;
21             msg.data = txt;
22             chatter_pub.publish(msg);
23
24             //fazer algo
25             ros::spinOnce();
26             loop_rate.sleep();
27             ++count;
28         }
29
30         return 0;
31     }
```

5.1 código detalhado

headers

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

ros/ros.h : Inclui as funcionalidades basicas do ROS, como a inicializacao, criação de nos, e controle de taxa.
std_msgs/String.h: Inclui a definição da mensagem do tipo String, que é usada para enviar textos.

Função main

```
1 int main (int args, char **argv){
2     ros::init(args, argv, "talker_cpp");
```

ros::init(args, argv, "talker_cpp");: Inicializa o nó ROS com o nome "talker_cpp".

Criação de um NodeHandle

```
1     ros::NodeHandle nh;
```

Cria um manipulador de nó, responsável por interagir com o sistema ROS (criando tópicos, assinantes, etc.).

Definição da Taxa de Loop

```
1     ros::Rate loop_rate(10);
```

ros::Rate loop_rate(10);: Define a taxa de repetição do loop para 10 Hz.

Criando o Publicador

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

Cria um publicador no tópico chatter para mensagens do tipo std_msgs::String. O segundo argumento (1000) é o tamanho da fila de mensagens.

5.2 explicação terminal

No exemplo feito em aula, ele rodava quatro terminais ao mesmo tempo, um com o roscore, rosrund my_first_pkg/ ele não acha as bibliotecas do ros e a biblioteca standart message , porém o projeto funciona perfeitamente
cmakelists.txt
Defina o executável talker_cpp e seu arquivo fonte

cmakeLists.txt

```
1
2     add_executable(talker_cpp src/talker_cpp.cpp)\
3
4     # Vincule as bibliotecas necessarias ao executavel\
5     target_link_libraries(talker_cpp ${catkin_LIBRARIES})\
```

6 Nó em python

o Primeiro comando é importante pois ele seta o compilador que vai ser usado

base de um nó em python

```
1     #!/usr/bin/env python3
2
3     import rospy
4
5
6     def talker():
7         rospy.init_node("talker_python")
8
9         loop_rate = rospy.Rate(10)
```

```
10         while not rospy.is_shutdown():
11             loop_rate.sleep()
12
13
14     if __name__ == "__main__":
15         try:
16             talker()
17         except rospy.ROSInterruptException:
18             pass
```

6.1 detalhamento do código

Codigo

```
1      #!/usr/bin/env python3
```

- Shebang (`#`): Informa ao sistema operacional que este script deve ser executado usando o interpretador Python 3. O caminho (`/usr/bin/env python3`) localiza automaticamente o Python 3 no sistema, garantindo que o script seja executado corretamente, independentemente da localizacao do Python 3.

importacao

```
1      import rospy
```

- Importação do módulo rospy: O módulo rospy é a biblioteca ROS para Python, que permite criar nós ROS, publicar/-subscriver em tópicos, usar serviços, etc.

Função talker

```
1      def talker():
2          rospy.init_node("talker_python")
```

- Define uma função chamada talker.
- `rospy.init_node("talker_python")`: Inicializa um nó ROS chamado "talker_python". Todo nó ROS precisa de um nome exclusivo, e esta linha registra o nó no mestre ROS.

loop principal

```
1      loop_rate = rospy.Rate(10)
```

- `rospy.Rate(10)`: Define uma taxa de loop de 10 Hz, ou seja, o loop rodará 10 vezes por segundo. Essa função é usada para controlar a frequência de execução do loop principal.

base de um nó em python

```
1      while not rospy.is_shutdown():
2          loop_rate.sleep()
```

- Loop while: Este loop while continuará rodando até que o ROS seja desligado (por exemplo, ao pressionar Ctrl+C).
- `rospy.is_shutdown()`: Verifica se o ROS foi desligado.
- `loop_rate.sleep()`: Faz o loop dormir o tempo necessário para manter a frequência definida bem `rospy.Rate(10)`. Neste caso, ele dorme por 0,1 segundos para manter a taxa de 10 Hz.

diretiva main

```
1      if __name__ == "__main__":
2          try:
3              talker()
4          except rospy.ROSInterruptException:
5              pass
```

- `if __name__ == "__main__":`: Este bloco verifica se o script está sendo executado diretamente (não importado como um módulo). Se for o caso, ele executa a função talker.
- try-except: O bloco try tenta executar a função talker(). Se uma excecao `rospy.ROSInterruptException` for lançada (geralmente ocorre quando o ROS é interrompido), o bloco except captura essa excecao e ignora-a (o código dentro de pass não faz nada).

7 ROS Publisher e Subscriber em C++ - ParteI

terminal ROS

```
1      rostopic -h
2      rostopic hz <nome_topico> mostra a publicacao do topico com a frequencia estabelecida no
          codigo
3      rosnode list lista os nos disponiveis
4      rostopic list // deve mostrar os topicos em execucao
5      rostopic pub /number std_msgs/Float64 "data: 4.0"
6      rostopic pub /number std_msgs/Float64 "data: 4.0 " -r 2 // iteracao sobre o comando na
          taxa de 2 hz.
7      rostopic -h //lista de comandos associados ao rostopic
8      rostopic info <nome_topico>
9      envia informacoes para o topico en execucao
```

```
1  #include <iostream>
2  #include "ros/ros.h"
3  #include "std_msgs/Float64.h"
4  class Counter{
5  public:
6      Counter(ros::NodeHandle *nh){
7          count = 0;
8
9          num_sub = nh ->subscribe("/number", 10, &Counter::numberCallBack, this);
10     }
11     void numberCallBack(const std_msgs::Float64 &msg){
12         count = count + msg.data;
13         ROS_INFO("CONTAGEM ATUAL:%f", count);
14     }
15 private:
16     double count;
17     ros::Subscriber num_sub;
18 };
19
20 int main(int argc, char **argv){
21     ros::init(argc, argv, "counter_node");
22     ros::NodeHandle nh;
23     Counter counter = Counter( &nh);
24     ros::spin();
25
26     return 0;
```

7.1 detalhamento counter.cpp

- #include "ros/ros.h": Inclui a biblioteca principal do ROS, que fornece as funcionalidades necessárias para criar nós, publicar e subscrever tópicos, etc.
- #include "std_msgs/Float64.h": Inclui a definição do tipo de mensagem std_msgs::Float64, que é usada para enviar dados do tipo double em ROS.

declaração da classe counter

```
1  class Counter {
2  public:
3      Counter(ros::NodeHandle *nh) {
4          count = 0;
5          num_sub = nh->subscribe("/number", 10, &Counter::numberCallBack, this
6          );
7      }
8
9      void numberCallBack(const std_msgs::Float64 &msg) {
10         count = count + msg.data;
11         ROS_INFO("CONTAGEM ATUAL: %f", count);
12     }
13 private:
14     double count;
15     ros::Subscriber num_sub;
16 };
```

- Classe Counter: Define uma classe que gerencia a contagem dos valores recebidos e armazena o estado necessário.
- Construtor Counter(ros::NodeHandle *nh):
 - Este é o construtor da classe, que é chamado quando uma instância da classe Counter é criada.
 - count = 0;: Inicializa a variável count com 0
 - num_sub = nh subscribe("/number", 10, &Counter::numberCallBack, this);:
 - Cria um subscritor (Subscriber) que se inscreve no tópico /number.
 - 10: O segundo argumento é o tamanho da fila de mensagens.
 - &Counter::numberCallBack: Especifica o método que será chamado quando uma nova mensagem for recebida (callback).
 - this: Passa um ponteiro para a instância atual da classe Counter para o método de callback.
- Método numberCallBack
 - void numberCallBack(const std_msgs::Float64 &msg): Método que será chamado sempre que uma nova mensagem for recebida no tópico /number.
 - count = count + msg.data;: Atualiza a variável count somando o valor recebido (msg.data).
 - OS_INFO("CONTAGEM ATUAL: %f", count);: Imprime a contagem atual no terminal, formatando o número com

Função main

```
1  int main(int argc, char **argv) {
2  ros::init(argc, argv, "counter_node");
3  ros::NodeHandle nh;
4  Counter counter = Counter(&nh);
5  ros::spin();
6  return 0;
7  }
```

- `int main(int argc, char **argv):` A função principal que inicia o programa.
- `ros::init(argc, argv, "counter_node");`: Inicializa o nó ROS com o nome "counter_node".
- `ros::NodeHandle nh;`: Cria um objeto NodeHandle, que é necessário para interagir com o sistema ROS.
- `Counter counter = Counter(&nh);`: Cria uma instância da classe Counter, passando o NodeHandle para que o subscriber possa ser criado.
- `ros::spin();`: Entra em um loop que mantém o programa rodando e processando callbacks até que o ROS seja desligado.
- `return 0;`: Retorna 0 ao final da execução do programa, indicando que o programa terminou com sucesso.

7.2 definindo um timer no counter.cpp

counter.cpp - publicando

```
1  #include <iostream>
2  #include "ros/ros.h"
3  #include "std_msgs/Float64.h"
4
5  class Counter{
6  public:
7      Counter(ros::NodeHandle *nh){
8          count = 0;
9          publish_interval = 1;
10
11          num_sub = nh ->subscribe("/number", 10, &Counter::numberCallback, this);
12          count_pub = nh -> advertise < std_msgs::Float64>("/current_count", 10);
13          timer_pub = nh -> createTimer(ros::Duration(publish_interval), &Counter::
14              timerCallback, this);
15
16      }
17      void numberCallback(const std_msgs::Float64 &msg){
18          count = count + msg.data;
19          ROS_INFO("CONTAGEM ATUAL:%f", count);
20
21      }
22
23      void timerCallback(const ros::TimerEvent &event){
24          std_msgs ::Float64 msg;
25          msg.data = count;
26          count_pub.publish(msg);
27
28      }
29
30  private:
31      double count;
32      double publish_interval;
33      ros::Subscriber num_sub;
34      ros:: Publisher count_pub;
35      ros:: Timer timer_pub;
36  };
37
38
39  int main(int argc, char **argv){
40      ros::init(argc, argv, "counter_node");
41      ros:: NodeHandle nh;
42      Counter counter = Counter( &nh);
43      ros::spin();
44
45      return 0;
46
47  }
```

7.2.1 detalhamento do timercallback

- `publish_interval`: Essa variável foi adicionada à classe Counter e é usada para definir o intervalo de tempo (em segundos) entre cada publicação da contagem atualizada. No construtor, ela é inicializada com o valor 1, o que significa que a contagem será publicada a cada segundo.

Criação do Publisher count_pub

```
1  count_pub = nh->advertise<std_msgs::Float64>("/current_count", 10)
```

- `count_pub`: Este é um publisher que foi adicionado à classe Counter. Ele publica mensagens do tipo `std_msgs::Float64` no tópico `/current_count`.
- `/current_count`: Esse é o novo tópico onde a contagem atualizada será publicada. Outros nós podem se inscrever nesse tópico para receber as contagens.
- `10`: Tamanho da fila de mensagens, semelhante ao subscriber.

Criação do Publisher count_pub

```
1  timer_pub = nh->createTimer(ros::Duration(publish_interval), &Counter::
    timerCallback, this);
```

- timer_pub: Um temporizador foi adicionado à classe Counter. Esse temporizador é criado com um intervalo definido por publish_interval (inicialmente 1 segundo) e chama o método timerCallback a cada intervalo.
- ros::Duration(publish_interval): Define o intervalo do temporizador com base na duração especificada em publish_interval.

Criação do Publisher count_pub

```
1         void timerCallback(const ros::TimerEvent &event) {
2             std_msgs::Float64 msg;
3             msg.data = count;
4             count_pub.publish(msg);
5         }
```

- timerCallback: Este método é chamado automaticamente a cada intervalo de tempo definido pelo temporizador.
- msg.data = count;: Cria uma mensagem do tipo std_msgs::Float64 e define seu valor como o valor atual de count.
- ount_pub.publish(msg);: Publica a mensagem no tópico /current_count.

8 Publisher e subcriber em c++ partII

number_publish.cpp

```
1 #include "ros/ros.h"
2 #include "std_msgs/Float64.h"
3
4 class NumberPublisher{
5     public:
6         NumberPublisher(ros::NodeHandle *nh){
7             number = 1.2;
8             publish_interval = 1;
9
10            num_pub = nh->advertise <std_msgs::Float64>("/number", 10);
11            timer_pub = nh -> createTimer (ros::Duration(publish_interval), &NumberPublisher::
                timerCallback, this);
12        }
13
14        void timerCallback(const ros:: TimerEvent &event){
15            std_msgs::Float64 msg;
16            msg.data = number;
17            num_pub.publish(msg);
18        }
19
20        private:
21            double number;
22            double publish_interval;
23            ros::Publisher num_pub;
24            ros::Timer timer_pub;
25 };
26
27
28 int main(int argc, char **argv)
29 {
30     ros::init(argc, argv, "number_publisher");
31     ros::NodeHandle nh;
32     NumberPublisher num_publisher = NumberPublisher(&nh);
33
34     ros::spin();
35     return 0;
36
37 }
```

minimal_node_cpp

```
1 #include "ros/ros.h"
2
3
4 class NodeName{
5
6     // Atualizar
7
8     public:
9         NodeName(ros::NodeHandle *nh){
10
11             // Atualizar
12             //Inicializar Variaveis
13
14         }
15
16     private:
17
18 };
19
20
21 int main(int argc, char **argv)
22 {
23     ros::init(argc, argv, "node_name");
24     ros::NodeHandle nh;
25     NodeName node_name = NodeName(&nh);
26
27     // Atualizar
28
29     ros::spin();
30     return 0;
31
32 }
```

9 Publisher e subscriber em python part I

counter.py

```
1  #!/usr/bin/python3
2  import rospy
3
4  from std_msgs.msg import Float64
5
6  class Counter:
7      def __init__(self) -> None:
8          self.count = 0
9          self.num_sub = rospy.Subscriber("/number", Float64, self.numberCallback, queue_size = 10)
10
11
12      def numberCallback(self, msg):
13          self.count = self.count + msg.data
14          rospy.loginfo("contagem atual" + str(self.count))
15
16
17  if __name__ == "__main__":
18      try:
19          rospy.init_node("counter_node")
20          Counter()
21          rospy.spin()
22      except rospy.ROSInterruptException:
23          pass
```

Ros terminal

```
1  permissao .py
2  chmod +x <codigo_py>.py
3  rostopic pub -r 1 /number std_msgs/Float64 "2.0"
4  roscore //comunicacao com o master
```

counter.py- with publish

```
1  #!/usr/bin/python3
2  import rospy
3
4  from std_msgs.msg import Float64
5
6  class Counter:
7      def __init__(self) -> None:
8          self.count = 0
9          self.publish_interval = 3
10         self.num_sub = rospy.Subscriber("/number", Float64, self.numberCallback, queue_size = 10)
11         self.count_pub = rospy.Publisher("Current_count", Float64, queue_size= 10)
12         self.timer_pub = rospy.Timer(rospy.Duration(), self.TimerCallback)
13
14      def numberCallback(self, msg):
15          self.count = self.count + msg.data
16          rospy.loginfo("contagem atual" + str(self.count))
17      def TimerCallback(self, event):
18          msg = Float64()
19          msg.data = self.count
20          self.count_pub.publish(msg)
21  if __name__ == "__main__":
22      try:
23          rospy.init_node("counter_node")
24          Counter()
25          rospy.spin()
26      except rospy.ROSInterruptException:
27          pass
```

10 publish e subscriber em python partII

number_publish.py

```
1  #!/usr/bin/env python3
2
3  import rospy
4  from std_msgs.msg import Float64
5
6  class NumberPublisher:
7
8      # Mudar
9
10     def __init__(self):
11         self.number = 3.1
12         self.publish_interval = 1.5
13
14         self.num_pub = rospy.Publisher("/number", Float64, queue_size = 10)
15         self.timer_pub = rospy.Timer(rospy.Duration(self.publish_interval), self.timerCallback)
16
17     def timerCallback(self, event):
18         msg = Float64()
19         msg.data = self.number
20         self.num_pub.publish(msg)
```



```
19
20
21 if __name__ == '__main__':
22     try:
23         rospy.init_node("number_publisher_node")           # Mudar
24         NumberPublisher()
25         rospy.spin()
26     except rospy.ROSInterruptException:
27         pass
```

minimal_pyhton_node

```
1  #!/usr/bin/env python3
2
3  import rospy
4
5  class NodeName:                                     # Mudar
6
7      def __init__(self):
8
9
10 if __name__ == '__main__':
11     try:
12         rospy.init_node(node_name)                   # Mudar
13         NodeName()
14         rospy.spin()
15     except rospy.ROSInterruptException:
16         pass
```

11 ROSService server com c++

counter.cpp

```
1  #include <iostream>
2  #include "ros/ros.h"
3  #include "std_msgs/Float64.h"
4  #include "std_srvs/Empty.h"
5
6  class Counter{
7  public:
8      Counter(ros::NodeHandle *nh){
9          count = 0;
10         publish_interval = 1;
11
12         num_sub = nh ->subscribe("/number", 10, &Counter::numberCallBack, this);
13         count_pub = nh -> advertise < std_msgs::Float64>("/current_count", 10);
14         timer_pub = nh -> createTimer(ros::Duration(publish_interval), &Counter::timerCallback, this);
15         reset_srv = nh -> advertiseService ("/reset_counter", &Counter::resetServerCallback, this);
16
17     }
18     // callback do subscribe
19     void numberCallBack(const std_msgs::Float64 &msg){
20         count = count + msg.data;
21         ROS_INFO("CONTAGEM ATUAL:%f", count);
22
23     }
24
25     //callback do publish
26     // tempo do ros e nao do computador
27     void timerCallback(const ros::TimerEvent &event){
28         std_msgs ::Float64 msg;
29         msg.data = count;
30         count_pub.publish(msg);
31
32     }
33     bool resetServerCallback(std_srvs::Empty::Request &req, std_srvs::Empty::Response &res){
34         count = 0;
35         ROS_INFO("resetando a contagem");
36         return true;
37     }
38
39
40 private:
41     double count;
42     double publish_interval;
43     ros::Subscriber num_sub;
44     ros:: Publisher count_pub;
45     ros:: Timer timer_pub;
46     ros::ServiceServer reset_srv;
47 };
48
49 int main(int argc, char **argv){
50     ros::init(argc, argv, "counter_node");
51     ros:: NodeHandle nh;
52     Counter counter = Counter( &nh);
```

```
53     ros::spin();
54
55     return 0;
56
57 }
```

number_publisher.cpp

```
1  #include "ros/ros.h"
2
3  #include "std_msgs/Float64.h"
4
5  class NumberPublisher{
6
7      public:
8          NumberPublisher(ros::NodeHandle *nh){
9              number = 1.2;
10             publish_interval = 1;
11
12             num_pub = nh->advertise<std_msgs::Float64>("/number", 10);
13             timer_pub = nh->createTimer(ros::Duration(publish_interval), &NumberPublisher::
                timerCallback, this);
14
15         }
16
17         void timerCallback(const ros::TimerEvent &event){
18             std_msgs::Float64 msg;
19             msg.data = number;
20             num_pub.publish(msg);
21         }
22
23     private:
24         double number;
25         double publish_interval;
26         ros::Publisher num_pub;
27         ros::Timer timer_pub;
28
29 };
30
31
32 int main(int argc, char **argv)
33 {
34     ros::init(argc, argv, "number_publisher");
35     ros::NodeHandle nh;
36     NumberPublisher num_publisher = NumberPublisher(&nh);
37
38     ros::spin();
39     return 0;
40 }
```

ROSService terminal

```
1 rosservice call /reset_counter "{}"
2 rosservice list
3 rosservice info <nome_service>
```

12 ROS service client em c++

number_publisher.cpp

```
1  #include "ros/ros.h"
2  #include "std_msgs/Float64.h"
3  #include "std_srvs/Empty.h"
4
5  class NumberPublisher{
6      public:
7          NumberPublisher(ros::NodeHandle *nh){
8              number = 1.2;
9              publish_interval = 1;
10             reset_interval = 7;
11
12             num_pub = nh->advertise <std_msgs::Float64>("/number", 10);
13             timer_pub = nh -> createTimer (ros::Duration(publish_interval), &NumberPublisher::
                timerCallback, this);
14             timer_reset = nh->createTimer(ros::Duration(reset_interval), &NumberPublisher::
                timeResetCallback, this);
15             client_reset = nh -> serviceClient<std_srvs::Empty>("reset_counter");
16         }
17
18
19         void timerCallback(const ros:: TimerEvent &event){
20             std_msgs::Float64 msg;
21             msg.data = number;
22             num_pub.publish(msg);
23         }
24
25         void timeResetCallback(const ros:: TimerEvent &event){
```

```
26         std_srvs:: Empty srv;
27         ROS_INFO("solicitacao de reset da contagem");
28         client_reset.call(srv);
29
30     }
31 private:
32     double number;
33     double publish_interval;
34     double reset_interval;
35     ros::Publisher num_pub;
36     ros::Timer timer_pub;
37     ros::ServiceClient client_reset;
38     ros::Timer timer_reset;
39
40 };
41
42
43 int main(int argc, char **argv)
44 {
45     ros::init(argc, argv, "number_publisher");
46     ros::NodeHandle nh;
47     NumberPublisher num_publisher = NumberPublisher(&nh);
48
49     ros::spin();
50     return 0;
51 }
52 }
```

Listing:

```
1 roscore
2 rosrun my_project_cpp counter
3 rostopic echo /current_count
4 rosrun my_project_cp
5 rosrun my_project_cpp number_publisher
```

13 ROS service server em python

counter.py

```
1 #!/usr/bin/python3
2 import rospy
3
4 from std_msgs.msg import Float64
5 from std_srvs.srv import Empty, EmptyResponse
6
7 class Counter:
8     def __init__(self) -> None:
9         self.count = 0
10        self.publish_interval = 3
11        self.num_sub = rospy.Subscriber("/number", Float64, self.numberCallback, queue_size = 10)
12        self.count_pub = rospy.Publisher("/current_count", Float64, queue_size= 10)
13        self.timer_pub = rospy.Timer(rospy.Duration(), self.TimerCallback)
14        self.reset_srv =rospy.Service("/reset_counter", Empty, self.resetSrvCallback)
15
16        def numberCallback(self, msg):
17            self.count = self.count + msg.data
18            rospy.loginfo("contagem atual" + str(self.count))
19
20        def TimerCallback(self, event):
21            msg = Float64()
22            msg.data = self.count
23            self.count_pub.publish(msg)
24
25        def resetSrvCallback(self, req):
26            self.count = 0
27            rospy.loginfo("resetando a contagem")
28            return EmptyResponse()
29
30 if __name__ == "__main__":
31     try:
32         rospy.init_node("counter_node")
33         Counter()
34         rospy.spin()
35     except rospy.ROSInterruptException:
36         pass
```

number_publish.py

```
1 #!/usr/bin/env python3
2
3 import rospy
4 from std_msgs.msg import Float64
5
6
7 class NumberPublisher:
8     # Mudar
```

```

9      def __init__(self):
10          self.number = 3.1
11          self.publish_interval = 1.5
12
13
14          self.num_pub = rospy.Publisher("/number", Float64, queue_size = 10)
15          self.timer_pub = rospy.Timer(rospy.Duration(self.publish_interval), self.timerCallback)
16
17      def timerCallback(self, event):
18          msg = Float64()
19          msg.data = self.number
20          self.num_pub.publish(msg)
21
22
23 if __name__ == '__main__':
24     try:
25         rospy.init_node("number_publisher_node")                # Mudar
26         NumberPublisher()
27         rospy.spin()
28     except rospy.ROSInterruptException:
29         pass
```

14 ROS Service Client em Python

number_publish.py

```

1  #!/usr/bin/env python3
2
3  import rospy
4  from std_msgs.msg import Float64
5  from std_srvs.srv import Empty
6
7  class NumberPublisher:                # Mudar
8
9      def __init__(self):
10          self.number = 3.1
11          self.publish_interval = 1.5
12          self.reset_interval = 8
13
14          self.num_pub = rospy.Publisher("/number", Float64, queue_size = 10)
15          self.timer_pub = rospy.Timer(rospy.Duration(self.publish_interval), self.timerCallback)
16          self.client_reset =rospy.ServiceProxy("/reset_counter", Empty)
17          self.timer_reset = rospy.Timer(rospy.Duration(self.reset_interval), self.
              timerResetCallback)
18
19      def timerCallback(self, event):
20          msg = Float64()
21          msg.data = self.number
22          self.num_pub.publish(msg)
23      def timerResetCallback(self, event):
24          rospy.loginfo("solicitando reset da contagem")
25          self.client_reset()
26
27
28
29 if __name__ == '__main__':
30     try:
31         rospy.init_node("number_publisher_node")                # Mudar
32         NumberPublisher()
33         rospy.spin()
34     except rospy.ROSInterruptException:
35         pass
```

15 Launch files

Os arquivos de lançamento (launch files) em ROS (Robot Operating System) são usados para iniciar múltiplos nós (nodes) e definir suas configurações de maneira automatizada e conveniente. Esses arquivos são escritos em XML e têm a extensão .launch. Eles permitem que o usuário especifique a estrutura e os parâmetros necessários para a execução de um sistema robótico complexo

counter_cpp.launch

```

1  <launch>
2      <!-- Define um argumento chamado 'number_topic' com um valor padrao '/number'.
3           Este argumento pode ser substituido na linha de comando ao iniciar o arquivo de
4           lancamento. -->
5
6      <arg name="number_topic" default="/number"/>
7
8      <!-- Inicia um no chamado 'counter_cpp' do pacote 'my_project_cpp', executando o tipo 'counter
9           ',
10          A saida do no e direcionada para a tela ('screen'). -->
11      <node name="counter_cpp" pkg="my_project_cpp" type="counter" output="screen">
12
13          <!-- Redireciona o topico '/current_count' para '/count', alterando a comunicacao entre os
14               nos
15               sem necessidade de modificar o codigo-fonte do no. -->
16          <remap from="/current_count" to="/count"/>
```

```
13         <!-- Redireciona o topico '/number' para o valor do argumento 'number_topic'.
14             Neste caso, sera '/number' a menos que o argumento seja substituido. -->
15         <remap from="/number" to="$(arg number_topic)"/>
16     </node>
17
18     <!-- Inicia outro no chamado 'counter_cpp2' do pacote 'my_project_cpp', tambem executando o
19         tipo 'counter'.
20         A saida deste no tambem e direcionada para a tela ('screen'). -->
21     <node name="counter_cpp2" pkg="my_project_cpp" type="counter" output="screen">
22         <!-- Este no nao possui remapeamentos de topicos definidos. -->
23     </node>
24 </launch>
```

my_project_cpp.launch

```
1 <launch>
2     <!-- Define um argumento chamado 'new_counter' com um valor padrao 'True'.
3         Este argumento pode ser usado para controlar a inclusao condicional de nos ou grupos. -->
4     <arg name="new_counter" value="True"/>
5
6     <!-- Inclui outro arquivo de lancamento localizado no pacote 'my_project_bringup'.
7         O arquivo incluido e 'counter_cpp.launch'. -->
8     <include file="$(find my_project_bringup)/launch/counter_cpp.launch">
9
10         <!-- Passa um argumento para o arquivo de lancamento incluido.
11             Define 'number_topic' com o valor 'new_number' no arquivo incluido. -->
12         <arg name="number_topic" value="new_number"/>
13     </include>
14
15     <!-- Inicia um no chamado 'num_pub_cpp' do pacote 'my_project_cpp', executando o tipo '
16         number_publisher'.
17         A saida do no e direcionada para a tela ('screen'). -->
18     <node name="num_pub_cpp" pkg="my_project_cpp" type="number_publisher" output="screen"/>
19
20     <!-- Agrupa nos condicionalmente baseado no valor do argumento 'new_counter'.
21         O grupo so sera incluido se 'new_counter' for verdadeiro ('True'). -->
22     <group if="$(arg new_counter)">
23
24         <!-- Inicia um no chamado 'num_pub_cpp_2' do pacote 'my_project_cpp', executando o tipo '
25             number_publisher'.
26             A saida deste no tambem e direcionada para a tela ('screen'). -->
27         <node name="num_pub_cpp_2" pkg="my_project_cpp" type="number_publisher" output="screen"/>
28     </group>
29 </launch>
```

16 ROS paramater cpp part I

terminal ROS

```
1     rosparam set      set parameter
2     rosparam get      get parameter
3     rosparam load     load parameters from file
4     rosparam dump     dump parameters to file
5     rosparam delete   delete parameter
6     rosparam list     list parameter names
```

para setar o parametro no projeto você vai ter que carregar o package individual se não ele nao vai carregar o parametro

a criação do pkg de launch não necessita de pkgs adicionais depois de feito esse processo o pram ele aparce na lista

terminal ROS

```
1 catkin_create_pkg my_project_bringup
2
3 catkin_make --only-pkg-with-deps my_project_cpp
4 roslaunch <package_name> <launch_name>
```

counter.cpp

```
1 #include <iostream>
2 #include "ros/ros.h"
3 #include "std_msgs/Float64.h"
4 #include "std_srvs/Empty.h"
5
6 class Counter {
7 public:
8     Counter(ros::NodeHandle *nh) {
9         count = 0;
10        publish_interval = 1;
11
12        nh->setParam("custom_param", false);
13        if(nh->getParam("initial_count", count)){
14            ROS_INFO("contagem inicia %f", count);
15        }else{
16            ROS_INFO("contagem inicial nao iniciadada");
```

```
17     }
18     nh->param<double>("pub_rate", publish_interval, 1);
19
20     num_sub = nh->subscribe("/number", 10, &Counter::numberCallback, this);
21     count_pub = nh->advertise<std_msgs::Float64>("/current_count", 10);
22     timer_pub = nh->createTimer(ros::Duration(publish_interval), &Counter::timerCallback, this
23     );
24     reset_srv = nh->advertiseService("/reset_counter", &Counter::resetServerCallback, this);
25 }
26
27 // callback do subscribe
28 void numberCallback(const std_msgs::Float64 &msg) {
29     count += msg.data;
30     ROS_INFO("CONTAGEM ATUAL: %f", count);
31 }
32
33 // callback do publish
34 void timerCallback(const ros::TimerEvent &event) {
35     std_msgs::Float64 msg;
36     msg.data = count;
37     count_pub.publish(msg);
38 }
39
40 bool resetServerCallback(std_srvs::Empty::Request &req, std_srvs::Empty::Response &res) {
41     count = 0;
42     ROS_INFO("resetando a contagem");
43     return true;
44 }
45
46 private:
47     double count;
48     double publish_interval;
49     ros::Subscriber num_sub;
50     ros::Publisher count_pub;
51     ros::Timer timer_pub;
52     ros::ServiceServer reset_srv;
53 };
54
55 int main(int argc, char **argv) {
56     ros::init(argc, argv, "counter_node");
57     ros::NodeHandle nh;//deixando claro que questamos trabalhando com o namespace privado do ROS
58     Counter counter(&nh);
59     ros::spin();
60     return 0;
61 }
```

counter_cpp.launch

```
1 <launch>
2   <arg name = "number_topic" default= "/number"/>
3   <param name="initial_count" type = "double" value= "3.1415" />
4   <param name="pub_rate" type = "double" value= "2.5" />
5   <node name = "counter_cpp" pkg = "my_project_cpp" type="counter" output="screen" >
6     <remap from= "/number" to="$(arg number_topic)" />
7   </node>
8 </launch>
```

16.1 adendos

tive muito problema pois tinha definido o nó para namesapces privados, quando busquei o parametro do launch , não achava de jeito nenhum definindo o parametro para o namesapce privado

counter_cpp.launch

```
1 <launch>
2   <arg name="number_topic" default="/number"/>
3   <param name="initial_count" type="double" value="3.1415"/>
4   <node name="counter_cpp" pkg="my_project_cpp" type="counter" output="screen">
5     <remap from="/number" to="$(arg number_topic)"/>
6   </node>
7 </launch>
```

counter.cpp

```
1     #include <iostream>
2     #include "ros/ros.h"
3     #include "std_msgs/Float64.h"
4     #include "std_srvs/Empty.h"
5
6     class Counter {
7     public:
8         Counter(ros::NodeHandle *nh) {
9             count = 0;
10             publish_interval = 1.0;
11
12             nh->setParam("custom_param", false);
13             if (nh->getParam("initial_count", count)) {
14                 ROS_INFO("Contagem inicial: %f", count);
```

```
15         } else {
16             ROS_INFO("Contagem inicial n o iniciada");
17         }
18
19         num_sub = nh->subscribe("/number", 10, &Counter::numberCallBack, this);
20         count_pub = nh->advertise<std_msgs::Float64>("/current_count", 10);
21         timer_pub = nh->createTimer(ros::Duration(publish_interval), &Counter::timerCallback, this
22             );
23         reset_srv = nh->advertiseService("/reset_counter", &Counter::resetServerCallback, this);
24     }
25
26     // Callback do subscribe
27     void numberCallBack(const std_msgs::Float64::ConstPtr& msg) {
28         count += msg->data;
29         ROS_INFO("CONTAGEM ATUAL: %f", count);
30     }
31
32     // Callback do publish
33     void timerCallback(const ros::TimerEvent& event) {
34         std_msgs::Float64 msg;
35         msg.data = count;
36         count_pub.publish(msg);
37     }
38
39     bool resetServerCallback(std_srvs::Empty::Request &req, std_srvs::Empty::Response &res) {
40         count = 0;
41         ROS_INFO("Resetando a contagem");
42         return true;
43     }
44
45 private:
46     double count;
47     double publish_interval;
48     ros::Subscriber num_sub;
49     ros::Publisher count_pub;
50     ros::Timer timer_pub;
51     ros::ServiceServer reset_srv;
52 };
53
54 int main(int argc, char **argv) {
55     ros::init(argc, argv, "counter_node");
56     ros::NodeHandle nh("~"); //definindo namespace padr o como o privado
57     Counter counter(&nh);
58     ros::spin();
59     return 0;
60 }
```

17 ROS parameter cpp partII

17.1 checagem da existencia do parametro

A checagem acontece dentro da classe do nó , e a declaração dentro do launchfile

counter.cpp

```
1 #include <iostream>
2 #include "ros/ros.h"
3 #include "std_msgs/Float64.h"
4 #include "std_srvs/Empty.h"
5
6 class Counter {
7 public:
8     Counter(ros::NodeHandle *nh) {
9         count = 0;
10         publish_interval = 1;
11
12         if(nh->getParam("initial_count", count)){
13             ROS_INFO("contagem incida %f", count);
14         }else{
15             ROS_INFO("contagem inicial nao iniciadada");
16         }
17         nh->param<double>("pub_rate", publish_interval, 1);
18         if(nh->hasParam("pub_rate")||nh->hasParam("initial_count")){
19             ROS_WARN("Parametros customizados");
20             nh->setParam("custom_param", true);
21         }else{
22             nh->setParam("custom_param", false);
23         }
24
25         num_sub = nh->subscribe("/number", 10, &Counter::numberCallBack, this);
26         count_pub = nh->advertise<std_msgs::Float64>("/current_count", 10);
27         timer_pub = nh->createTimer(ros::Duration(publish_interval), &Counter::timerCallback, this
28             );
29         reset_srv = nh->advertiseService("/reset_counter", &Counter::resetServerCallback, this);
30     }
31
32     // callback do subscribe
33     void numberCallBack(const std_msgs::Float64 &msg) {
```

```
33     count += msg.data;
34     ROS_INFO("CONTAGEM ATUAL: %f", count);
35 }
36
37 // callback do publish
38 void timerCallback(const ros::TimerEvent &event) {
39     std_msgs::Float64 msg;
40     msg.data = count;
41     count_pub.publish(msg);
42 }
43
44 bool resetServerCallback(std_srvs::Empty::Request &req, std_srvs::Empty::Response &res) {
45     count = 0;
46     ROS_INFO("resetando a contagem");
47     return true;
48 }
49
50 private:
51     double count;
52     double publish_interval;
53     ros::Subscriber num_sub;
54     ros::Publisher count_pub;
55     ros::Timer timer_pub;
56     ros::ServiceServer reset_srv;
57 };
58
59 int main(int argc, char **argv) {
60     ros::init(argc, argv, "counter_node");
61     ros::NodeHandle nh;//deixando claro que questamos trabalhando com o namespace privado do ROS
62     Counter counter(&nh);
63     ros::spin();
64     return 0;
65 }
```

counter_cpp.launch

```
1 <launch>
2   <arg name = "number_topic" default= "/number"/>
3   <param name="initial_count" type = "double" value= "3.1415" />
4   <param name="pub_rate" type = "double" value= "2.5" />
5   <node name = "counter_cpp" pkg = "my_project_cpp" type="counter" output="screen" >
6     <remap from= "/number" to="$(arg number_topic)" />
7   </node>
8 </launch>
```

number_publisher_cpp.launch

```
1 <launch>
2   <arg name = "number_topic" default= "/number"/>
3   <param name="initial_count" type = "double" value= "3.1415" />
4   <param name="pub_rate" type = "double" value= "2.5" />
5   <node name = "counter_cpp" pkg = "my_project_cpp" type="counter" output="screen" >
6     <remap from= "/number" to="$(arg number_topic)" />
7   </node>
8 </launch>
```

17.2 deletando parametros

counter.cpp

```
1 #include <iostream>
2 #include "ros/ros.h"
3 #include "std_msgs/Float64.h"
4 #include "std_srvs/Empty.h"
5
6 class Counter {
7 public:
8     Counter(ros::NodeHandle *nh) {
9         count = 0;
10        publish_interval = 1;
11
12        if(nh->getParam("initial_count", count)){
13            ROS_INFO("contagem inicia %f", count);
14        }else{
15            ROS_INFO("contagem inicial nao iniciadada");
16        }
17        nh->param<double>("pub_rate", publish_interval, 1);
18        if(nh->hasParam("pub_rate")||nh->hasParam("initial_count")){
19            ROS_WARN("Parametros customizados");
20            nh->setParam("custom_param", true);
21        }else{
22            nh->setParam("custom_param", false);
23        }
24        if(nh->hasParam("pub_rate")){
25            ROS_INFO("deletando parametro");
26            nh->deleteParam("pub_rate");
27            nh->deleteParam("initial_count");
```



```
28     }
29
30     num_sub = nh->subscribe("/number", 10, &Counter::numberCallBack, this);
31     count_pub = nh->advertise<std_msgs::Float64>("/current_count", 10);
32     timer_pub = nh->createTimer(ros::Duration(publish_interval), &Counter::timerCallback, this
    );
33     reset_srv = nh->advertiseService("/reset_counter", &Counter::resetServerCallback, this);
34 }
35
36 // callback do subscribe
37 void numberCallBack(const std_msgs::Float64 &msg) {
38     count += msg.data;
39     ROS_INFO("CONTAGEM ATUAL: %f", count);
40 }
41
42 // callback do publish
43 void timerCallback(const ros::TimerEvent &event) {
44     std_msgs::Float64 msg;
45     msg.data = count;
46     count_pub.publish(msg);
47 }
48
49 bool resetServerCallback(std_srvs::Empty::Request &req, std_srvs::Empty::Response &res) {
50     count = 0;
51     ROS_INFO("resetando a contagem");
52     return true;
53 }
54
55 private:
56     double count;
57     double publish_interval;
58     ros::Subscriber num_sub;
59     ros::Publisher count_pub;
60     ros::Timer timer_pub;
61     ros::ServiceServer reset_srv;
62 };
63
64 int main(int argc, char **argv) {
65     ros::init(argc, argv, "counter_node");
66     ros::NodeHandle nh;//deixando claro que estamos trabalhando com o namespace privado do ROS
67     Counter counter(&nh);
68     ros::spin();
69     return 0;
70 }
```

counter_cpp.launch

```
1 <launch>
2   <arg name = "number_topic" default= "/number"/>
3   <param name="initial_count" type = "double" value= "3.1415" />
4   <param name="pub_rate" type = "double" value= "2.5" />
5   <param name="delete_param" type = "bool" value= "True" />
6   <node name = "counter_cpp" pkg = "my_project_cpp" type="counter" output="screen" >
7     <remap from= "/number" to="$(arg number_topic)" />
8   </node>
9 </launch>
```

17.3 adicionando parametros a um arquivo separado

counter_cpp.launch

```
1 <launch>
2   <arg name = "number_topic" default= "/number"/>
3   <!--param name="initial_count" type = "double" value= "3.1415" />
4   <param name="pub_rate" type = "double" value= "2.5" /-->
5   <param name="delete_param" type = "bool" value= "True" />
6
7   <rosptram command="load" file = "$(my_project_bringup)/config/my_project.yaml"/>
8
9   <node name = "counter_cpp" pkg = "my_project_cpp" type="counter" output="screen" >
10     <remap from= "/number" to="$(arg number_topic)" />
11   </node>
12 </launch>
```

my_project.yaml

```
1 initial_count: 5
2 pub_rate: 0.1
```

18 ROS parameter com python

prestar sempre atenção no namespace dos parametros.

counter.py

```
1 #!/usr/bin/env python3
```

```
2 import rospy
3 from std_msgs.msg import Float64
4 from std_srvs.srv import Empty, EmptyResponse
5
6 class Counter:
7
8     def __init__(self):
9         self.count = rospy.get_param("initial_count", 0)
10        self.publish_interval = rospy.get_param("~pub_rate", 3)
11
12
13        if(rospy.has_param("~pub_rate") or rospy.has_param("initial_count")):
14            rospy.loginfo("parametros customizados")
15            rospy.set_param("~custom_param", True)
16        else:
17            rospy.set_param("~custom_param", False)
18
19        if(rospy.has_param("~delete_param")):
20            rospy.loginfo("deletando parametros")
21            if(rospy.has_param("~pub_rate")):
22                rospy.delete_param("~pub_rate")
23            if(rospy.has_param("initial_count")):
24                rospy.delete_param("initial_count")
25
26
27        self.num_sub = rospy.Subscriber("/number", Float64, self.numberCallback, queue_size = 10)
28        self.count_pub = rospy.Publisher("/current_count", Float64, queue_size = 10)
29        self.timer_pub = rospy.Timer(rospy.Duration(self.publish_interval), self.timerCallback)
30        self.reset_srv = rospy.Service('reset_counter', Empty, self.resetSrvCallback)
31
32    def numberCallback(self, msg):
33        self.count = self.count + msg.data
34        rospy.loginfo("Contagem Atual " + str(self.count))
35
36
37    def timerCallback(self, event):
38        msg = Float64()
39        msg.data = self.count
40        self.count_pub.publish(msg)
41    def resetSrvCallback(self, req):
42        self.count = 0
43        rospy.loginfo("resetando a contagem")
44        return EmptyResponse()
45
46 if __name__ == '__main__':
47     try:
48         rospy.init_node("counter_node")
49         Counter()
50         rospy.spin()
51     except rospy.ROSInterruptException:
52         pass
```

counter_py.launch

```
1 <launch>
2   <arg name="number_topic" default="/number"/>
3   <param name="initial_count" type="double" value="-10.0"/>
4   <node name="counter_py" pkg="my_project_py" type="counter.py" output="screen" >
5       <param name="pub_rate" type="double" value="0.1"/>
6       <param name="~delete_param" type="bool" value="false"/>
7       <remap from="/number" to="$(arg number_topic)"/>
8   </node>
9
10   <node name="counter_py2" pkg="my_project_py" type="counter.py" output="screen" >
11       <param name="pub_rate" type="double" value="0.1"/>
12       <remap from="/number" to="$(arg number_topic)"/>
13   </node>
14
15   <!--node name="num_pub" pkg="my_project_py" type="counter.py" output="screen" /
16       -->
17
18 </launch>
```

number_publisher.py

```
1 #!/usr/bin/env python3
2 import rospy
3 from std_msgs.msg import Float64
4 from std_srvs.srv import Empty
5
6 class NumberPublisher:
7
8     def __init__(self):
9         self.number = rospy.get_param("num", 3.1)
10        self.publish_interval = rospy.get_param("pub_interval", 1.5)
11        self.reset_interval = rospy.get_param("reset", 8)
12
13
14        self.num_pub = rospy.Publisher("/number", Float64, queue_size = 10)
```

```
14         self.timer_pub = rospy.Timer(rospy.Duration(self.publish_interval), self.timerCallback)
15         self.timer_reset = rospy.Timer(rospy.Duration(self.reset_interval), self.
16             timerResetCallback)
17         self.client_reset = rospy.ServiceProxy("reset_counter", Empty)
18
19     def timerCallback(self, event):
20         msg = Float64()
21         msg.data = self.number
22         self.num_pub.publish(msg)
23
24     def timerResetCallback(self, event):
25         rospy.loginfo("solicitacao de reset")
26         self.client_reset()
27
28
29 if __name__ == '__main__':
30     try:
31         rospy.init_node("number_publisher_node")
32         NumberPublisher()
33         rospy.spin()
34     except rospy.ROSInterruptException:
35         pass
```

my_project_py.launch

```
1 <launch>
2
3     <include file = "$(find my_project_bringup)/launch/counter_py.launch">
4         <arg name = "number_topic" value = "new_number"/>
5
6     </include>
7
8     <node name= "num_pub_py" pkg = "my_project_py" type="number_publisher.py" output = "screen"/>
9     <param name= "num" type = "double" value = "4.5"/>
10    <param name= "pub_interval" type = "double" value= "2.5"/>
11    <param name= "reset" type= "double" value = "2"/>
12
13 </launch>
```

19 Custom Message

comando do terminal para o pacote de std msg

```
1 catkin_create_pkg my_project_msg std_msgs message_generation
```

configuração do cmake list interno do pkg de custom message

my_project_msg/CMakeLists.txt

```
1 cmake_minimum_required(VERSION 3.0.2)
2 project(my_project_msg)
3
4 ## Compile as C++11, supported in ROS Kinetic and newer
5 # add_compile_options(-std=c++11)
6
7 ## Find catkin macros and libraries
8 ## if COMPONENTS list like find_package(catkin REQUIRED COMPONENTS xyz)
9 ## is used, also find other catkin packages
10 find_package(catkin REQUIRED COMPONENTS
11     message_generation
12     std_msgs
13 )
14
15 add_message_files(
16     FILES
17     CounterHistory.msg
18 )
19
20 generate_messages(
21     DEPENDENCIES
22     std_msgs
23 )
24
25
26 catkin_package(
27     CATKIN_DEPENDS message_generation std_msgs
28 )
29
```

foi adicionado <exec_depend>message_generation</exec_depend> pois o ROS tem dois packages com o mesmo nome

src/my_project_msg/package.xml

```
1 <?xml version="1.0"?>
2 <package format="2">
3     <name>my_project_msg</name>
4     <version>0.0.0</version>
```

```
5 <description>The my_project_msg package</description>
6
7 <!-- One maintainer tag required, multiple allowed, one person per tag -->
8 <!-- Example: -->
9 <!-- <maintainer email="jane.doe@example.com">Jane Doe</maintainer> -->
10 <maintainer email="marcos@todo.todo">marcos</maintainer>
11
12
13 <!-- One license tag required, multiple allowed, one license per tag -->
14 <!-- Commonly used license strings: -->
15 <!-- BSD, MIT, Boost Software License, GPLv2, GPLv3, LGPLv2.1, LGPLv3 -->
16 <license>TODO</license>
17
18
19 <!-- Url tags are optional, but multiple are allowed, one per tag -->
20 <!-- Optional attribute type can be: website, bugtracker, or repository -->
21 <!-- Example: -->
22 <!-- <url type="website">http://wiki.ros.org/my_project_msg</url> -->
23
24
25 <!-- Author tags are optional, multiple are allowed, one per tag -->
26 <!-- Authors do not have to be maintainers, but could be -->
27 <!-- Example: -->
28 <!-- <author email="jane.doe@example.com">Jane Doe</author> -->
29
30
31 <!-- The *depend tags are used to specify dependencies -->
32 <!-- Dependencies can be catkin packages or system dependencies -->
33 <!-- Examples: -->
34 <!-- Use depend as a shortcut for packages that are both build and exec dependencies -->
35 <!-- <depend>roscpp</depend> -->
36 <!-- Note that this is equivalent to the following: -->
37 <!-- <build_depend>roscpp</build_depend> -->
38 <!-- <exec_depend>roscpp</exec_depend> -->
39 <!-- Use build_depend for packages you need at compile time: -->
40 <!-- <build_depend>message_generation</build_depend> -->
41 <!-- Use build_export_depend for packages you need in order to build against this package: -->
42 <!-- <build_export_depend>message_generation</build_export_depend> -->
43 <!-- Use buildtool_depend for build tool packages: -->
44 <!-- <buildtool_depend>catkin</buildtool_depend> -->
45 <!-- Use exec_depend for packages you need at runtime: -->
46 <!-- <exec_depend>message_runtime</exec_depend> -->
47 <!-- Use test_depend for packages you need only for testing: -->
48 <!-- <test_depend>gtest</test_depend> -->
49 <!-- Use doc_depend for packages you need only for building documentation: -->
50 <!-- <doc_depend>doxygen</doc_depend> -->
51 <buildtool_depend>catkin</buildtool_depend>
52 <build_depend>message_generation</build_depend>
53 <build_depend>std_msgs</build_depend>
54 <build_export_depend>std_msgs</build_export_depend>
55 <exec_depend>std_msgs</exec_depend>
56 <exec_depend>message_generation</exec_depend>
57
58
59 <!-- The export tag contains other, unspecified, tags -->
60 <export>
61 <!-- Other tools can request additional information be placed here -->
62
63 </export>
64 </package>
```

src/my_project_msg/msg/CounterHistory.msg

```
1 float64 current_value
2 float64 last_value
```

20 Custom message com c++

src/my_project_cpp/CMakeLists.txt

```
1 cmake_minimum_required(VERSION 3.0.2)
2 project(my_project_cpp)
3
4 ## Compile as C++11, supported in ROS Kinetic and newer
5 # add_compile_options(-std=c++11)
6
7 ## Find catkin macros and libraries
8 ## if COMPONENTS list like find_package(catkin REQUIRED COMPONENTS xyz)
9 ## is used, also find other catkin packages
10 find_package(catkin REQUIRED COMPONENTS
11   roscpp
12   std_msgs
13   my_project_msg
14 )
15
16 ## System dependencies are found with CMake's conventions
17 # find_package(Boost REQUIRED COMPONENTS system)
```

```
18
19
20 ## Uncomment this if the package has a setup.py. This macro ensures
21 ## modules and global scripts declared therein get installed
22 ## See http://ros.org/doc/api/catkin/html/user_guide/setup_dot_py.html
23 # catkin_python_setup()
24
25 #####
26 ## Declare ROS messages, services and actions ##
27 #####
28
29 ## To declare and build messages, services or actions from within this
30 ## package, follow these steps:
31 ## * Let MSG_DEP_SET be the set of packages whose message types you use in
32 ##   your messages/services/actions (e.g. std_msgs, actionlib_msgs, ...).
33 ## * In the file package.xml:
34 ##   * add a build_depend tag for "message_generation"
35 ##   * add a build_depend and a exec_depend tag for each package in MSG_DEP_SET
36 ##   * If MSG_DEP_SET isn't empty the following dependency has been pulled in
37 ##     but can be declared for certainty nonetheless:
38 ##     * add a exec_depend tag for "message_runtime"
39 ## * In this file (CMakeLists.txt):
40 ##   * add "message_generation" and every package in MSG_DEP_SET to
41 ##     find_package(catkin REQUIRED COMPONENTS ...)
42 ##   * add "message_runtime" and every package in MSG_DEP_SET to
43 ##     catkin_package(CATKIN_DEPENDS ...)
44 ##   * uncomment the add_*_files sections below as needed
45 ##     and list every .msg/.srv/.action file to be processed
46 ##   * uncomment the generate_messages entry below
47 ##   * add every package in MSG_DEP_SET to generate_messages(DEPENDENCIES ...)
48
49 ## Generate messages in the 'msg' folder
50 # add_message_files(
51 #   FILES
52 #   Message1.msg
53 #   Message2.msg
54 # )
55
56 ## Generate services in the 'srv' folder
57 # add_service_files(
58 #   FILES
59 #   Service1.srv
60 #   Service2.srv
61 # )
62
63 ## Generate actions in the 'action' folder
64 # add_action_files(
65 #   FILES
66 #   Action1.action
67 #   Action2.action
68 # )
69
70 ## Generate added messages and services with any dependencies listed here
71 # generate_messages(
72 #   DEPENDENCIES
73 #   std_msgs
74 # )
75
76 #####
77 ## Declare ROS dynamic reconfigure parameters ##
78 #####
79
80 ## To declare and build dynamic reconfigure parameters within this
81 ## package, follow these steps:
82 ## * In the file package.xml:
83 ##   * add a build_depend and a exec_depend tag for "dynamic_reconfigure"
84 ## * In this file (CMakeLists.txt):
85 ##   * add "dynamic_reconfigure" to
86 ##     find_package(catkin REQUIRED COMPONENTS ...)
87 ##   * uncomment the "generate_dynamic_reconfigure_options" section below
88 ##     and list every .cfg file to be processed
89
90 ## Generate dynamic reconfigure parameters in the 'cfg' folder
91 # generate_dynamic_reconfigure_options(
92 #   cfg/DynReconf1.cfg
93 #   cfg/DynReconf2.cfg
94 # )
95
96 #####
97 ## catkin specific configuration ##
98 #####
99 ## The catkin_package macro generates cmake config files for your package
100 ## Declare things to be passed to dependent projects
101 ## INCLUDE_DIRS: uncomment this if your package contains header files
102 ## LIBRARIES: libraries you create in this project that dependent projects also need
103 ## CATKIN_DEPENDS: catkin_packages dependent projects also need
104 ## DEPENDS: system dependencies of this project that dependent projects also need
105 catkin_package(
106 #   INCLUDE_DIRS include
```

```
107 # LIBRARIES my_project_cpp
108 # CATKIN_DEPENDS roscpp std_msgs
109 # DEPENDS system_lib
110 )
111
112 #####
113 ## Build ##
114 #####
115
116 ## Specify additional locations of header files
117 ## Your package locations should be listed before other locations
118 include_directories(
119 # include
120   ${catkin_INCLUDE_DIRS}
121 )
122
123 ## Declare a C++ library
124 # add_library(${PROJECT_NAME}
125 #   src/${PROJECT_NAME}/my_project_cpp.cpp
126 # )
127
128 ## Add cmake target dependencies of the library
129 ## as an example, code may need to be generated before libraries
130 ## either from message generation or dynamic reconfigure
131 # add_dependencies(${PROJECT_NAME} ${${PROJECT_NAME}_EXPORTED_TARGETS} ${catkin_EXPORTED_TARGETS})
132
133 ## Declare a C++ executable
134 ## With catkin_make all packages are built within a single CMake context
135 ## The recommended prefix ensures that target names across packages don't collide
136 # add_executable(${PROJECT_NAME}_node src/my_project_cpp_node.cpp)
137
138 ## Rename C++ executable without prefix
139 ## The above recommended prefix causes long target names, the following renames the
140 ## target back to the shorter version for ease of user use
141 ## e.g. "roslaunch someones_pkg node" instead of "roslaunch someones_pkg someones_pkg_node"
142 # set_target_properties(${PROJECT_NAME}_node PROPERTIES OUTPUT_NAME node PREFIX "")
143
144 ## Add cmake target dependencies of the executable
145 ## same as for the library above
146 # add_dependencies(${PROJECT_NAME}_node ${${PROJECT_NAME}_EXPORTED_TARGETS} ${
147   catkin_EXPORTED_TARGETS})
148
149 ## Specify libraries to link a library or executable target against
150 # target_link_libraries(${PROJECT_NAME}_node
151 #   ${catkin_LIBRARIES}
152 # )
153
154 #####
155 ## Install ##
156 #####
157
158 # all install targets should use catkin DESTINATION variables
159 # See http://ros.org/doc/api/catkin/html/adv\_user\_guide/variables.html
160
161 ## Mark executable scripts (Python etc.) for installation
162 ## in contrast to setup.py, you can choose the destination
163 # catkin_install_python(PROGRAMS
164 #   scripts/my_python_script
165 #   DESTINATION ${CATKIN_PACKAGE_BIN_DESTINATION}
166 # )
167
168 ## Mark executables for installation
169 ## See http://docs.ros.org/melodic/api/catkin/html/howto/format1/building\_executables.html
170 # install(TARGETS ${PROJECT_NAME}_node
171 #   RUNTIME DESTINATION ${CATKIN_PACKAGE_BIN_DESTINATION}
172 # )
173
174 ## Mark libraries for installation
175 ## See http://docs.ros.org/melodic/api/catkin/html/howto/format1/building\_libraries.html
176 # install(TARGETS ${PROJECT_NAME}
177 #   ARCHIVE DESTINATION ${CATKIN_PACKAGE_LIB_DESTINATION}
178 #   LIBRARY DESTINATION ${CATKIN_PACKAGE_LIB_DESTINATION}
179 #   RUNTIME DESTINATION ${CATKIN_GLOBAL_BIN_DESTINATION}
180 # )
181
182 ## Mark cpp header files for installation
183 # install(DIRECTORY include/${PROJECT_NAME}/
184 #   DESTINATION ${CATKIN_PACKAGE_INCLUDE_DESTINATION}
185 #   FILES_MATCHING PATTERN "*.h"
186 #   PATTERN ".svn" EXCLUDE
187 # )
188
189 ## Mark other files for installation (e.g. launch and bag files, etc.)
190 # install(FILES
191 #   # myfile1
192 #   # myfile2
193 #   DESTINATION ${CATKIN_PACKAGE_SHARE_DESTINATION}
194 # )
```

```
195 #####
196 ## Testing ##
197 #####
198
199 ## Add gtest based cpp test target and link libraries
200 # catkin_add_gtest(${PROJECT_NAME}-test test/test_my_project_cpp.cpp)
201 # if(TARGET ${PROJECT_NAME}-test)
202 #   target_link_libraries(${PROJECT_NAME}-test ${PROJECT_NAME})
203 # endif()
204
205 ## Add folders to be run by python nosetests
206 # catkin_add_nosetests(test)
207 add_executable(counter src/counter.cpp)
208 target_link_libraries(counter ${catkin_LIBRARIES})
209
210 add_executable(number_publisher src/number_publisher.cpp)
211 target_link_libraries(number_publisher ${catkin_LIBRARIES})
```

src/my_project_cpp/package.xml

```
1 <?xml version="1.0"?>
2 <package format="2">
3   <name>my_project_cpp</name>
4   <version>0.0.0</version>
5   <description>The my_project_cpp package</description>
6
7   <!-- One maintainer tag required, multiple allowed, one person per tag -->
8   <!-- Example: -->
9   <!-- <maintainer email="jane.doe@example.com">Jane Doe</maintainer> -->
10  <maintainer email="marcos@todo.todo">marcos</maintainer>
11
12
13  <!-- One license tag required, multiple allowed, one license per tag -->
14  <!-- Commonly used license strings: -->
15  <!--   BSD, MIT, Boost Software License, GPLv2, GPLv3, LGPLv2.1, LGPLv3 -->
16  <license>TODO</license>
17
18
19  <!-- Url tags are optional, but multiple are allowed, one per tag -->
20  <!-- Optional attribute type can be: website, bugtracker, or repository -->
21  <!-- Example: -->
22  <!-- <url type="website">http://wiki.ros.org/my_project_cpp</url> -->
23
24
25  <!-- Author tags are optional, multiple are allowed, one per tag -->
26  <!-- Authors do not have to be maintainers, but could be -->
27  <!-- Example: -->
28  <!-- <author email="jane.doe@example.com">Jane Doe</author> -->
29
30
31  <!-- The *depend tags are used to specify dependencies -->
32  <!-- Dependencies can be catkin packages or system dependencies -->
33  <!-- Examples: -->
34  <!-- Use depend as a shortcut for packages that are both build and exec dependencies -->
35  <!--   <depend>roscpp</depend> -->
36  <!-- Note that this is equivalent to the following: -->
37  <!--   <build_depend>roscpp</build_depend> -->
38  <!--   <exec_depend>roscpp</exec_depend> -->
39  <!-- Use build_depend for packages you need at compile time: -->
40  <!--   <build_depend>message_generation</build_depend> -->
41  <!-- Use build_export_depend for packages you need in order to build against this package: -->
42  <!--   <build_export_depend>message_generation</build_export_depend> -->
43  <!-- Use buildtool_depend for build tool packages: -->
44  <!--   <buildtool_depend>catkin</buildtool_depend> -->
45  <!-- Use exec_depend for packages you need at runtime: -->
46  <!--   <exec_depend>message_runtime</exec_depend> -->
47  <!-- Use test_depend for packages you need only for testing: -->
48  <!--   <test_depend>gtest</test_depend> -->
49  <!-- Use doc_depend for packages you need only for building documentation: -->
50  <!--   <doc_depend>doxygen</doc_depend> -->
51  <buildtool_depend>catkin</buildtool_depend>
52  <build_depend>roscpp</build_depend>
53  <build_depend>std_msgs</build_depend>
54  <build_depend>my_project_msg</build_depend>
55  <build_export_depend>roscpp</build_export_depend>
56  <build_export_depend>std_msgs</build_export_depend>
57  <build_export_depend>my_project_msg</build_export_depend>
58  <exec_depend>roscpp</exec_depend>
59  <exec_depend>std_msgs</exec_depend>
60  <exec_depend>my_project_msg</exec_depend>
61
62
63  <!-- The export tag contains other, unspecified, tags -->
64  <export>
65    <!-- Other tools can request additional information be placed here -->
66
67  </export>
68 </package>
```

No momento do curso ele nao entrava na funcao de callback portanto nao atualizava os parametros da funcao, no terminal.

src/my_project_cpp/src/counter.cpp

```
1 #include "ros/ros.h"
2
3 // Mensagens
4 #include "std_msgs/Float64.h"
5 #include "std_srvs/Empty.h"
6 #include "my_project_msg/CounterHistory.h"
7
8
9 class Counter{
10
11     public:
12         Counter(ros::NodeHandle *nh){
13             count = 0;
14             publish_interval = 1;
15
16             if (nh->getParam("initial_count", count) ){
17                 ROS_INFO("Contagem inicial em %f", count);
18             }
19             else{
20                 ROS_INFO("Contagem inicial nao definido");
21             }
22             nh->param<double>("pub_rate", publish_interval, 1);
23
24             if (nh->hasParam("pub_rate") || nh->hasParam("initial_count") ){
25                 ROS_WARN("Parametros customizados");
26                 nh->setParam("custom_param", true);
27             }
28             else{
29                 nh->setParam("custom_param", false);
30             }
31
32             if (nh->hasParam("delete_param")){
33                 ROS_ERROR("Deletando Parametros");
34                 nh->deleteParam("pub_rate");
35                 nh->deleteParam("initial_count");
36             }
37
38             last_count = count;
39             cycles = 0;
40
41             num_sub = nh->subscribe("/number", 10, &Counter::numberCallback, this);
42             count_pub = nh->advertise<std_msgs::Float64>("/current_count", 10);
43
44             timer_pub = nh->createTimer(ros::Duration(publish_interval), &Counter::timerCallback, this);
45             reset_srv = nh->advertiseService("/reset_counter", &Counter::resetSrvCallback, this);
46
47             history_pub = nh->advertise<my_project_msg::CounterHistory>("/history_counter", 10);
48         }
49
50         void numberCallback(const std_msgs::Float64 &msg){
51             ROS_INFO("Callback chamado com valor: %f", msg.data);
52             cycles++;
53             last_count = count;
54             count += msg.data;
55             ROS_INFO("Contagem Atual %f", count);
56         }
57
58         void timerCallback(const ros::TimerEvent &event){
59             ROS_INFO("Timer chamado");
60             std_msgs::Float64 msg;
61             msg.data = count;
62             count_pub.publish(msg);
63
64             my_project_msg::CounterHistory history_msg;
65             history_msg.current_value = count;
66             history_msg.last_value = last_count;
67             history_msg.cycles = cycles;
68             history_pub.publish(history_msg);
69         }
70
71         bool resetSrvCallback(std_srvs::Empty::Request &req, std_srvs::Empty::Response &res){
72             count = 0;
73             ROS_INFO("Resetando a contagem.");
74             return true;
75         }
76
77     private:
78         double count;
79         double publish_interval;
80         double last_count;
81         int cycles;
82
83         ros::Publisher history_pub;
84         ros::Subscriber num_sub;
```



```
86         ros::Publisher count_pub;
87         ros::Timer timer_pub;
88         ros::ServiceServer reset_srv;
89         ros::ServiceServer check_greater_srv;
90     };
91
92
93     int main(int argc, char **argv){
94
95         ros::init(argc, argv, "counter_node");
96         ros::NodeHandle nh;
97         Counter counter = Counter(&nh);
98         ros::spin();
99
100         return 0;
101     }
```

21 Custom message em python

O codigo nao entra no callback, portanto não atualiza os parametros

src/my_project_py/scripts/counter.py

```
1  #!/usr/bin/env python3
2  import rospy
3  from std_msgs.msg import Float64
4  from std_srvs.srv import Empty, EmptyResponse
5  from my_project_msg.msg import CounterHistory
6
7  class Counter:
8
9      def __init__(self):
10         self.count = rospy.get_param("initial_count", 0)
11         self.publish_interval = rospy.get_param("~pub_rate", 3)
12
13
14         if(rospy.has_param("~pub_rate") or rospy.has_param("initial_count")):
15             rospy.loginfo("parametros customizados")
16             rospy.set_param("~custom_param", True)
17         else:
18             rospy.set_param("~custom_param", False)
19
20         if(rospy.has_param("~delete_param")):
21             rospy.loginfo("deletando parametros")
22             if(rospy.has_param("~pub_rate")):
23                 rospy.delete_param("~pub_rate")
24             if(rospy.has_param("initial_count")):
25                 rospy.delete_param("initial_count")
26         self.last_count = self.count
27         self.cycles = 0
28
29         self.num_sub = rospy.Subscriber("/number", Float64, self.numberCallback, queue_size = 10)
30         self.count_pub = rospy.Publisher("/current_count", Float64, queue_size = 10)
31         self.timer_pub = rospy.Timer(rospy.Duration(self.publish_interval), self.timerCallback)
32         self.reset_srv = rospy.Service('reset_counter', Empty, self.resetSrvCallback)
33         self.history_pub = rospy.Publisher("/history_count", CounterHistory, queue_size = 10)
34
35     def numberCallback(self, msg):
36         self.cycles = self.cycles + 1
37         self.last_count = self.count
38         self.count = self.count + msg.data
39         rospy.loginfo("Contagem Atual " + str(self.count))
40
41
42     def timerCallback(self, event):
43         msg = Float64()
44         msg.data = self.count
45         self.count_pub.publish(msg)
46         history_msg = CounterHistory()
47         history_msg.current_value = self.count
48         history_msg.last_value = self.last_count
49         history_msg.cycles = self.cycles
50         self.history_pub.publish(history_msg)
51
52     def resetSrvCallback(self, req):
53         self.count = 0
54         rospy.loginfo("resetando a contagem")
55         return EmptyResponse()
56
57 if __name__ == '__main__':
58     try:
59         rospy.init_node("counter_node")
60         Counter()
61         rospy.spin()
62     except rospy.ROSInterruptException:
63         pass
```

```
1 <?xml version="1.0"?>
2 <package format="2">
3   <name>my_project_py</name>
4   <version>0.0.0</version>
5   <description>The my_project_py package</description>
6
7   <!-- One maintainer tag required, multiple allowed, one person per tag -->
8   <!-- Example: -->
9   <!-- <maintainer email="jane.doe@example.com">Jane Doe</maintainer> -->
10  <maintainer email="marcos@todo.todo">marcos</maintainer>
11
12
13  <!-- One license tag required, multiple allowed, one license per tag -->
14  <!-- Commonly used license strings: -->
15  <!--   BSD, MIT, Boost Software License, GPLv2, GPLv3, LGPLv2.1, LGPLv3 -->
16  <license>TODO</license>
17
18
19  <!-- Url tags are optional, but multiple are allowed, one per tag -->
20  <!-- Optional attribute type can be: website, bugtracker, or repository -->
21  <!-- Example: -->
22  <!-- <url type="website">http://wiki.ros.org/my_project_py</url> -->
23
24
25  <!-- Author tags are optional, multiple are allowed, one per tag -->
26  <!-- Authors do not have to be maintainers, but could be -->
27  <!-- Example: -->
28  <!-- <author email="jane.doe@example.com">Jane Doe</author> -->
29
30
31  <!-- The *depend tags are used to specify dependencies -->
32  <!-- Dependencies can be catkin packages or system dependencies -->
33  <!-- Examples: -->
34  <!-- Use depend as a shortcut for packages that are both build and exec dependencies -->
35  <!--   <depend>roscpp</depend> -->
36  <!--   Note that this is equivalent to the following: -->
37  <!--   <build_depend>roscpp</build_depend> -->
38  <!--   <exec_depend>roscpp</exec_depend> -->
39  <!-- Use build_depend for packages you need at compile time: -->
40  <!--   <build_depend>message_generation</build_depend> -->
41  <!-- Use build_export_depend for packages you need in order to build against this package: -->
42  <!--   <build_export_depend>message_generation</build_export_depend> -->
43  <!-- Use buildtool_depend for build tool packages: -->
44  <!--   <buildtool_depend>catkin</buildtool_depend> -->
45  <!-- Use exec_depend for packages you need at runtime: -->
46  <!--   <exec_depend>message_runtime</exec_depend> -->
47  <!-- Use test_depend for packages you need only for testing: -->
48  <!--   <test_depend>gtest</test_depend> -->
49  <!-- Use doc_depend for packages you need only for building documentation: -->
50  <!--   <doc_depend>doxygen</doc_depend> -->
51  <buildtool_depend>catkin</buildtool_depend>
52  <build_depend>rospy</build_depend>
53  <build_depend>std_msgs</build_depend>
54  <build_depend>my_project_msg</build_depend>
55  <build_export_depend>rospy</build_export_depend>
56  <build_export_depend>std_msgs</build_export_depend>
57  <build_export_depend>my_project_msg</build_export_depend>
58  <exec_depend>rospy</exec_depend>
59  <exec_depend>std_msgs</exec_depend>
60  <exec_depend>my_project_msg</exec_depend>
61
62
63
64  <!-- The export tag contains other, unspecified, tags -->
65  <export>
66    <!-- Other tools can request additional information be placed here -->
67
68  </export>
69 </package>
```

```
1 cmake_minimum_required(VERSION 3.0.2)
2 project(my_project_py)
3
4 ## Compile as C++11, supported in ROS Kinetic and newer
5 # add_compile_options(-std=c++11)
6
7 ## Find catkin macros and libraries
8 ## if COMPONENTS list like find_package(catkin REQUIRED COMPONENTS xyz)
9 ## is used, also find other catkin packages
10 find_package(catkin REQUIRED COMPONENTS
11   rospy
12   std_msgs
13   my_project_msg
14 )
15
16 ## System dependencies are found with CMake's conventions
```

```
17 # find_package(Boost REQUIRED COMPONENTS system)
18
19
20 ## Uncomment this if the package has a setup.py. This macro ensures
21 ## modules and global scripts declared therein get installed
22 ## See http://ros.org/doc/api/catkin/html/user_guide/setup_dot_py.html
23 # catkin_python_setup()
24
25 #####
26 ## Declare ROS messages, services and actions ##
27 #####
28
29 ## To declare and build messages, services or actions from within this
30 ## package, follow these steps:
31 ## * Let MSG_DEP_SET be the set of packages whose message types you use in
32 ##   your messages/services/actions (e.g. std_msgs, actionlib_msgs, ...).
33 ## * In the file package.xml:
34 ##   * add a build_depend tag for "message_generation"
35 ##   * add a build_depend and a exec_depend tag for each package in MSG_DEP_SET
36 ##   * If MSG_DEP_SET isn't empty the following dependency has been pulled in
37 ##     but can be declared for certainty nonetheless:
38 ##     * add a exec_depend tag for "message_runtime"
39 ## * In this file (CMakeLists.txt):
40 ##   * add "message_generation" and every package in MSG_DEP_SET to
41 ##     find_package(catkin REQUIRED COMPONENTS ...)
42 ##   * add "message_runtime" and every package in MSG_DEP_SET to
43 ##     catkin_package(CATKIN_DEPENDS ...)
44 ##   * uncomment the add_*_files sections below as needed
45 ##     and list every .msg/.srv/.action file to be processed
46 ##   * uncomment the generate_messages entry below
47 ##   * add every package in MSG_DEP_SET to generate_messages(DEPENDENCIES ...)
48
49 ## Generate messages in the 'msg' folder
50 # add_message_files(
51 #   FILES
52 #   Message1.msg
53 #   Message2.msg
54 # )
55
56 ## Generate services in the 'srv' folder
57 # add_service_files(
58 #   FILES
59 #   Service1.srv
60 #   Service2.srv
61 # )
62
63 ## Generate actions in the 'action' folder
64 # add_action_files(
65 #   FILES
66 #   Action1.action
67 #   Action2.action
68 # )
69
70 ## Generate added messages and services with any dependencies listed here
71 # generate_messages(
72 #   DEPENDENCIES
73 #   std_msgs
74 # )
75
76 #####
77 ## Declare ROS dynamic reconfigure parameters ##
78 #####
79
80 ## To declare and build dynamic reconfigure parameters within this
81 ## package, follow these steps:
82 ## * In the file package.xml:
83 ##   * add a build_depend and a exec_depend tag for "dynamic_reconfigure"
84 ## * In this file (CMakeLists.txt):
85 ##   * add "dynamic_reconfigure" to
86 ##     find_package(catkin REQUIRED COMPONENTS ...)
87 ##   * uncomment the "generate_dynamic_reconfigure_options" section below
88 ##     and list every .cfg file to be processed
89
90 ## Generate dynamic reconfigure parameters in the 'cfg' folder
91 # generate_dynamic_reconfigure_options(
92 #   cfg/DynReconf1.cfg
93 #   cfg/DynReconf2.cfg
94 # )
95
96 #####
97 ## catkin specific configuration ##
98 #####
99 ## The catkin_package macro generates cmake config files for your package
100 ## Declare things to be passed to dependent projects
101 ## INCLUDE_DIRS: uncomment this if your package contains header files
102 ## LIBRARIES: libraries you create in this project that dependent projects also need
103 ## CATKIN_DEPENDS: catkin_packages dependent projects also need
104 ## DEPENDS: system dependencies of this project that dependent projects also need
105 catkin_package(
```

```
106 # INCLUDE_DIRS include
107 # LIBRARIES my_project_py
108 # CATKIN_DEPENDS rospy std_msgs
109 # DEPENDS system_lib
110 )
111
112 #####
113 ## Build ##
114 #####
115
116 ## Specify additional locations of header files
117 ## Your package locations should be listed before other locations
118 include_directories(
119 # include
120   ${catkin_INCLUDE_DIRS}
121 )
122
123 ## Declare a C++ library
124 # add_library(${PROJECT_NAME}
125 #   src/${PROJECT_NAME}/my_project_py.cpp
126 # )
127
128 ## Add cmake target dependencies of the library
129 ## as an example, code may need to be generated before libraries
130 ## either from message generation or dynamic reconfigure
131 # add_dependencies(${PROJECT_NAME} ${${PROJECT_NAME}_EXPORTED_TARGETS} ${catkin_EXPORTED_TARGETS})
132
133 ## Declare a C++ executable
134 ## With catkin_make all packages are built within a single CMake context
135 ## The recommended prefix ensures that target names across packages don't collide
136 # add_executable(${PROJECT_NAME}_node src/my_project_py_node.cpp)
137
138 ## Rename C++ executable without prefix
139 ## The above recommended prefix causes long target names, the following renames the
140 ## target back to the shorter version for ease of user use
141 ## e.g. "roslaunch someones_pkg node" instead of "roslaunch someones_pkg someones_pkg_node"
142 # set_target_properties(${PROJECT_NAME}_node PROPERTIES OUTPUT_NAME node PREFIX "")
143
144 ## Add cmake target dependencies of the executable
145 ## same as for the library above
146 # add_dependencies(${PROJECT_NAME}_node ${${PROJECT_NAME}_EXPORTED_TARGETS} ${
147   catkin_EXPORTED_TARGETS})
148
149 ## Specify libraries to link a library or executable target against
150 # target_link_libraries(${PROJECT_NAME}_node
151 #   ${catkin_LIBRARIES}
152 # )
153
154 #####
155 ## Install ##
156 #####
157
158 # all install targets should use catkin DESTINATION variables
159 # See http://ros.org/doc/api/catkin/html/adv_user_guide/variables.html
160
161 ## Mark executable scripts (Python etc.) for installation
162 ## in contrast to setup.py, you can choose the destination
163 # catkin_install_python(PROGRAMS
164 #   scripts/my_python_script
165 #   DESTINATION ${CATKIN_PACKAGE_BIN_DESTINATION}
166 # )
167
168 ## Mark executables for installation
169 ## See http://docs.ros.org/melodic/api/catkin/html/howto/format1/building_executables.html
170 # install(TARGETS ${PROJECT_NAME}_node
171 #   RUNTIME DESTINATION ${CATKIN_PACKAGE_BIN_DESTINATION}
172 # )
173
174 ## Mark libraries for installation
175 ## See http://docs.ros.org/melodic/api/catkin/html/howto/format1/building_libraries.html
176 # install(TARGETS ${PROJECT_NAME}
177 #   ARCHIVE DESTINATION ${CATKIN_PACKAGE_LIB_DESTINATION}
178 #   LIBRARY DESTINATION ${CATKIN_PACKAGE_LIB_DESTINATION}
179 #   RUNTIME DESTINATION ${CATKIN_GLOBAL_BIN_DESTINATION}
180 # )
181
182 ## Mark cpp header files for installation
183 # install(DIRECTORY include/${PROJECT_NAME}/
184 #   DESTINATION ${CATKIN_PACKAGE_INCLUDE_DESTINATION}
185 #   FILES_MATCHING PATTERN "*.h"
186 #   PATTERN ".svn" EXCLUDE
187 # )
188
189 ## Mark other files for installation (e.g. launch and bag files, etc.)
190 # install(FILES
191 #   # myfile1
192 #   # myfile2
193 #   DESTINATION ${CATKIN_PACKAGE_SHARE_DESTINATION}
194 # )
```

```
194 #####
195 ## Testing ##
196 #####
197
198
199 ## Add gtest based cpp test target and link libraries
200 # catkin_add_gtest(${PROJECT_NAME}-test test/test_my_project_py.cpp)
201 # if(TARGET ${PROJECT_NAME}-test)
202 #   target_link_libraries(${PROJECT_NAME}-test ${PROJECT_NAME})
203 # endif()
204
205 ## Add folders to be run by python nosetests
206 # catkin_add_nosetests(test)
```

22 Custom Service

O diretorio para o service é criado dentro do pkg de msg pelo serviço se assemelhar. Adiciona-se o service no cmake

my_project_msg/srv/CheckNumber.srv

```
1 float64 number
2 ---
3 bool result
```

my_project_msg/CMakeLists.txt)project(my_project_msg)

```
1
2 ## Compile as C++11, supported in ROS Kinetic and newer
3 # add_compile_options(-std=c++11)
4
5 \#\# Find catkin macros and libraries
6 ## if COMPONENTS list like find_package(catkin REQUIRED COMPONENTS xyz)
7 ## is used, also find other catkin packages
8 find_package(catkin REQUIRED COMPONENTS
9   message_generation
10   std_msgs
11
12 )
13 add_message_files(
14   FILES
15   CounterHistory.msg
16
17 )
18 add_service_files(
19   FILES
20   CheckNumber.srv
21
22 )
23
24 generate_messages(
25   DEPENDENCIES
26   std_msgs
27
28 )
29
30 catkin_package(
31   CATKIN_DEPENDS message_generation std_msgs
32
33 )
```

terminal bash

```
1 rossrv package my_project_msg
2 rossrv show my_project_msg/CheckNumber.srv
```

23

src/my_project_py/CMakeLists.txt