Contenido

[Comandos de terminal de ROS 3](#_Toc122172477)

[Creación de los paquetes 3](#_Toc122172478)

[Ejecución de nodos 3](#_Toc122172479)

[Comando para paquetes 3](#_Toc122172480)

[Comandos para los topics 3](#_Toc122172481)

[Comandos para nodos 3](#_Toc122172482)

[Comando para mensajes personalizados 3](#_Toc122172483)

[Comando para servicios 3](#_Toc122172484)

[Comando para acciones 3](#_Toc122172485)

[Comandos de Rospy 4](#_Toc122172486)

[Nodos 4](#_Toc122172487)

[Topics 4](#_Toc122172488)

[Mensajes 4](#_Toc122172489)

[Bucles 4](#_Toc122172490)

[Servicios 4](#_Toc122172491)

[Servidor 4](#_Toc122172492)

[Clientes 4](#_Toc122172493)

[Acciones 5](#_Toc122172494)

[Servidor 5](#_Toc122172495)

[Cliente 5](#_Toc122172496)

[Topics 6](#_Toc122172497)

[Publicador 6](#_Toc122172498)

[Suscriptor 6](#_Toc122172499)

[Mensajes personalizados 7](#_Toc122172500)

[Servicios 8](#_Toc122172501)

[Cliente 8](#_Toc122172502)

[Servidor 8](#_Toc122172503)

[Mensajes personalizados 9](#_Toc122172504)

[Acciones 10](#_Toc122172505)

[Cliente 10](#_Toc122172506)

[Servidor 10](#_Toc122172507)

[Mensajes personalizados 12](#_Toc122172508)

[Comando de C++ 13](#_Toc122172509)

[Nodos 13](#_Toc122172510)

[Topics 13](#_Toc122172511)

[Mensajes 13](#_Toc122172512)

[Bucles 13](#_Toc122172513)

[Servicios 13](#_Toc122172514)

[Cliente 13](#_Toc122172515)

[Servidor 13](#_Toc122172516)

[Acciones 13](#_Toc122172517)

[Cliente 13](#_Toc122172518)

[Servidor 13](#_Toc122172519)

[Compilación de archivos 14](#_Toc122172520)

[Topics 15](#_Toc122172521)

[Publicador 15](#_Toc122172522)

[Suscriptor 15](#_Toc122172523)

[Mensajes personalizados 16](#_Toc122172524)

[Servicios 17](#_Toc122172525)

[Cliente 17](#_Toc122172526)

[Servidor 17](#_Toc122172527)

[Servidor con clases 18](#_Toc122172528)

[Masajes Personalizados 19](#_Toc122172529)

[Acciones 19](#_Toc122172530)

[Clientes 19](#_Toc122172531)

[Servidor 21](#_Toc122172532)

[Servidor con clases 22](#_Toc122172533)

[Mensajes personalizados 23](#_Toc122172534)

# Comandos de terminal de ROS

## Creación de los paquetes

catkin\_create\_package "nombre del paquete" rospyà Crea un paquete con rospy

catkin\_make à Compila todos los paquetes

catkin\_make --only\_pkg\_with\_deps “nombre\_del\_paquete”à Compila un paquete

## Ejecución de nodos

roscore à Inicia el nodo maestro de ROS

rosrun "nombre del paquete" "nombre del archivo" à ejecuta un programa de Ros

## Comando para paquetes

rospack list à Devuelve la lista de paquetes ROS

## Comandos para los topics

rostopic list àPara ver la lista de topics activos

rostopic echo "topic" à Para ver el contenido de un topic

rostopic info "topic" à Para ver la información de un topic

rostopic pub “mensaje” à Se publica un valor en un topic

## Comandos para nodos

rosnode list à Devuelve la lista de nodos activos

rosnode info "nombre del nodo" à Devuelve la información de un nodo

## Comando para mensajes personalizados

rosmsg list à Devuelve la lista de todos los mensajes

rosmsg show "Nombre del mensaje" à Devuelve la información de un mensaje

rosmsg list | grep “Mensaje” à Devuelve la ruta de un mensaje

## Comando para servicios

rosservice list à Devuelve la lista de servicios activos

rosservice info "servicio" à Devuelve la información de un servicio

rosservice call “servicio” “mensaje” à Se le envía una petición a un servicio

rossrv show "paquete/mensaje" à Devuelve la información de los argumentos de un servicio

## Comando para acciones

rostopic list à Devuelve la lista de topics y las acciones

\* "acción"/cancel à Cancela la acción

\* "acción"/feedback à Muestra el feedback

\* "acción" goal à Muestra el objetivo

\* "acción"/result à Muestra el resultado

\* "acción"/status à Muestra el estado de la acción

rostopic pub "nombre acción/acción" à Ejecuta la acción

rostopic info "acción" à Devuelve la información de la acción

rosmsg list | grep "acción" à Muestra la lista de mensajes de las acciones

\* "acción"/MensajeAction à

\* "acción"/MensajeActionFeedback à

\* "acción"/MensajeActionGoal à

\* "acción"/MensajeActionResult à

\* "acción"/MensajeFeedback àMensaje de feedback de una acción

\* "acción"/MensajeGoal à Mensaje de petición de una acción

\* "acción"/MensajeResult àMensaje de resultado de acción

rosmsg show "acción"/Mensaje\_\_\_ “ à Para devolver información del mensaje

#### Python

# Comandos de Rospy

## Nodos

rospy.init\_node(" ") à Creación de un nodo

## Topics

“nombre” = Publisher("nombre", Tipo de mensaje, queue\_size=10) à Creación de un publicador

“nombre”.publish(mensaje) àPublica el mensaje en el topic

rospy.Subscriber("nombre", Tipo de mensaje, callback) à Creación de un suscriptor

## Mensajes

Tipos de mensajes básicos : Int32,Int64, String …

“nombre” = Int32()à Creación de un mensaje

“nombre”.”variable del mensaje” = “valor” à Asignación de un valor al mensaje

## Bucles

rospy.spin()àBucle infinito

rospy.is\_shutdown()à Se usa en un While not para finalizar el bucle con **Control+C**

“nombre”=rospy.Rate(“frecuencia”)à Crea una frecuencia para un bucle

“nombre”.sleep()à Pausa para tener la frecuencia del bucle

## Servicios

### Servidor

“nombre”= rospy.Service(Nombre, Tipo de mensaje, callback )

### Clientes

rospy.wait\_for\_services("nombre") à Espera a que el servidor este activo

“nombre\_servicio”= rospy.ServiceProxy("/servicio", Tipo de mensaje) à Creación de un cliente para un servicio

“nombre” = “nombre\_servicio”(“petición”) à Envía la petición al servicio

## Acciones

### Servidor

“nombre”= actionlib.SimpleActionServer("nombre",”mensaje”,self.goal\_callback,False)à Crea el servidor

“nombre”.start() à Inicia el servidor

self.”nombre”.is\_preempt\_requested() à Comprueba si hay una cancelación anticipada

self.”nombre”.set\_preempted() à Establece el estado como cancelado anticipadamente

self.”nombre”.publish\_feedback(self.\_\_retroalimentacion) à Envia el feedback

self.”nombre”.set\_succeeded(self.\_\_resultado) à Envia el resultado

self.”nombre”.set\_aborted() à Establece el cestado como abortado

### Cliente

“nombre”= actionlib.SimpleActionClient (“nombre”,Tipo de mensaje) àCrea un cliente

“nombre”.wait\_for\_server() à Espera por el servidor de acción

“nombre”.send\_goal(“mensaje”,done\_cb=“callback\_resultado”,feedback\_cb=callback\_feedback)àenvia el mensaje

“nombre”.cancel\_goal() à Cancela el servidor anticipadamente

Status = “nombre”.get\_status() à Para obtener el estado de la acción con el siguiente código: 0 = pendiente , 1 = ejecutándose , 2= finalizado , 3= warning, 4 = error , 5 = cancelado

“nombre”.wait\_for\_result() à Espera al que el servidor termine de ejecutar la acción

# Topics

## Publicador

#!/usr/bin/env python

import rospy

from std\_msgs.msg import Int32

#Creamos el nodo principal

rospy.init\_node("Primer nodo")

#Creamos el Publicador

pub = rospy.Publisher("laser", Int32, queue\_size=10)

#Creamos el mensaje

a = Int32()

contador = 0

#Creamos la frecuencia del bucle

rate = rospy.Rate(10)

while not rospy.is\_shutdown():

    contador += 1

    a.data = contador

#Publicamos el mensaje

    pub.publish(a)

#Esperamos para que se cumpla el tiempo del bucle

    rate.sleep()

## Suscriptor

#!/usr/bin/env python

import rospy

from std\_msgs.msg import Int32

def callback(msg):

    """Funcion que se ejecuta cuando se recibe un mensaje   """

    print (msg)

#Creamos el nodo

rospy.init\_node('lectura')

#Nos suscribimos al topic laser

rospy.Subscriber('nombre', Int32, callback)

#Creamos el bucle

rospy.spin()

## Mensajes personalizados

1. Crear una carpeta msg en el paquete
2. Generamos un archivo “nombre”.msg y añadimos las variables del mensaje
3. Modificar el archivo CMakeLists.txt
4. En la función **find\_package ()** dentro de ella escribimos "**message\_generation**" y también añadimos los **paquetes** que use el mensaje creado (En líneas diferentes)
5. En la función **add\_message\_files ()** la descomentamos y añadimos el **nombre del archivo .msg** que creamos (En líneas diferentes)
6. En la función **generate\_messages ()** la descomentamos y **añadimos el nombre de los paquetes** que usamos en el mensaje creado
7. En la función **catkin\_package ()** la descomentamos y en la **línea de CATKIN\_DEPENDS añadimos rospy y message\_rutine,**  pero dejamos comentado la línea del INCLUDE
8. Comprobamos si la función **include\_directories()** **esta descomentada y si ${catkin\_INCLUDE\_DIRS} esta dentro** de ella,

4.**Modificar package.xml**

1) En el final modificamos el archivo para que siga el siguiente esquema

**<build\_depend> rospy </build\_depend>**

**<build\_depend> message\_generation </build\_depend>**

**<build\_export\_depend> rospy </build\_export\_depend>**

**<build\_export\_depend> message\_runtine </build\_export\_depend>**

**<exec\_depend> rospy </exec\_depend>**

**<exec\_depend> message\_runtine </exec\_depend>**

5. **Compilar el paquete** ejecutando **catkin\_make --only\_pkg\_with\_deps “nombre\_del\_paquete”** dentro de la carpeta catkin\_ws

# Servicios

## Cliente

#!/usr/bin/env python

import rospy

from paquete.srv import Mensaje, MensajeRequest

#Crea el nodo principal

rospy.init\_node('cliente\_servicio')

#Esperamos a que el servicio este disponible

rospy.wait\_for\_service('/nombre\_servicio')

#Se crea la conexion con el servicio

servicio = rospy.ServiceProxy("/servicio", "nombre del servicio")

#Creamos la petición del servicio

peticion = MensajeRequest()

peticion.dato = "dato"

#Enviamos la petición al servicio

resultado = servicio(peticion)

#Se imprime el resultado

print(resultado)

## Servidor

#!/usr/bin/env python

import rospy

from std\_srvs.srv import Empty, EmptyResponse

def mi\_callback(request):

    """Funcion que sera ejecutada cuando se llame al servicio mi\_servicio"""

    print("Peticion recibida")

    return (EmptyResponse) #Retorno un mensaje vacio

#Inicializa el nodo principal

rospy.init\_node('servidor')

mi\_servicio = rospy.Service("mi\_servicio", Empty, mi\_callback) #Crea el servicio. Argumentos: Nombre del servicio, Tipo de mensaje, Funcion que se ejecuta cuando se recibe una peticion

#Mantiene el nodo activo

rospy.spin()

## Mensajes personalizados

1. Crear una carpeta srv en el paquete del servidor del servicio
2. Creamos y editamos el fichero "Nombre".srv con el siguiente esquema

#Petición

int32 dato

---

#Respuesta

bool éxito

1. Modificamos el CMLists.txt para que se compile el servicio
2. En la función **find package()**, la descomentamos y **añadimos message\_generation** y añadimos las **dependencias** del archivo
3. En la función **add\_service\_files** **añadimos el nombre del archivo** de mensaje
4. En la función **generate\_messages**, la descomentamos y **añadimos las dependencias del archivo**
5. En la función **catkin\_package** **añadimos rospy y message\_runtime en la línea de CATKIN\_DEPENDS**

1. **Modificar package.xml**

1.En el final modificamos el archivo para que siga el siguiente esquema.

**<build\_depend> rospy </build\_depend>**

**<build\_depend> message\_generation </build\_depend>**

**<build\_export\_depend> rospy </build\_export\_depend>**

**<build\_export\_depend> message\_runtine </build\_export\_depend>**

**<exec\_depend> rospy </exec\_depend>**

**<exec\_depend> message\_runtine </exec\_depend>**

1. **Compilamos** el paquete usando **catkin\_make --only\_pkg\_with\_deps “nombre\_del\_paquete”** dentro de la carpeta catkin\_ws

# Acciones

## Cliente

#!/usr/bin/env python3

import rospy

import actionlib

from action\_server.msg import MensajeAction, MensajeActionGoal, MensajeActionResult , MensajeActionFeedback

def mi\_funcion(feedback):

    """Funcion que se ejecuta cuando el servidor envia feedback"""

    print(feedback)

def acabado(result):

 """Funcion que se ejecuta cuando el servidor termina de ejecutarla accion"""

    print("El resultado es: ", result.resultado)

#Inciamos el nodo de accion\_cliente

rospy.init\_node("accion\_cliente")

#Creamos el cliente de accion

cliente = actionlib.SimpleActionClient("accion\_servidor",MensajeAction)

#Esperamos al servidor de acciones

cliente.wait\_for\_server()

#Creamos el mensaje goal

goal = MensajeGoal()

goal.ciclos = 10

#Enviamos el mensaje goal

cliente.send\_goal(goal,done\_cb = acabado, feedback\_cb=mi\_funcion)

rospy.spin()

## Servidor

#!/usr/bin/env python3

import rospy

import actionlib

from servidor\_action.msg import MensajeAction, MensajeActionResult, MensajeActionFeedback

class accion\_simple(object):

    #Creamos dos variables privadas que seran mensajes

    \_\_retroalimentacion = MensajeFeedback()

    \_\_resultado = MensajeResult()

    def \_\_init\_\_(self):

        #Creamos el servidor

self.\_\_act\_serv=actionlib.SimpleActionServer("accion\_servidor",MensajeAction, self.goal\_callback,False)

        #Iniciamos el servidor

        self.\_\_act\_serv.start()

    def goal\_callback(self,datos):

        """Funcion que se ejecuta cuando se pide una accion al servidor

        RETURN: MensajeResult"""

        #Creamos la variable con los datos que llegan a la accion

        Datos = datos.goal

        #Definimos la variable del estado de la accion

        succes = False

        #Definimos una variable par el estado de cancelado

        cancelado = False

        #Creamos un bucle

        while condicion :

            #<--> Acciones <-->

            #Comprobamos que no hay una cancelacion anticipateda

            if self.\_\_act\_serv.is\_preempt\_requested():

            #Fijamos el estado de la accion como cancelada anticipatedamente

                cancelado = True

                self.\_\_act\_serv.set\_preempted()

                break

            #Definimos el valor del feedback

            self.\_\_retroalimentacion.feedback = 1

            #Publicamos el valor del feedback

            self.\_\_act\_serv.publish\_feedback(self.\_\_retroalimentacion)

        #Comprobamos si el proceso esta finalizado

        if cancelado== False :

            if succes:

                #Definimos el valor del resultado

                self.\_\_resultado.resultado = 1

                #Ponemos el estado como finalizado y devolvemos el resultado

                self.\_\_act\_serv.set\_succeeded(self.\_\_resultado)

            else :

 #Definimos el estado como abortado pero si no fue cancelada anticipatedamente

                self.\_\_act\_serv.set\_aborted()

#FIXME:Modulo principal

rospy.init\_node("accion\_servidor")#Inciamos el nodo de accion\_servidor

accion\_simple() #Creamos la clase

rospy.spin()

## Mensajes personalizados

1. Creamos la carpeta action
2. Creamos el archivo “nombre”.action y añadimos los valores del mensaje

#Goal

---

#Resultado

---

#Feedback

1. Modificamos el CMakelists.txt
2. En la función **find\_package REQUIRED COMPONETS** añadimos **actionlib\_msgs** y **las librerias** de que depende el mensaje
3. En la función **add\_action\_files** añadimos los archivos "nombre".action
4. En la función **generate\_messages** añadimos **actionlib\_msgs** y las **librerías** que depende el mensaje
5. Comprobamos la función **catkin\_package** esta descomentada y **añadimos rospy** y dejamos INCLUDE comentado
6. Comprobamos que la función **include\_directories** esta descomentada
7. Modificaciones del package.xml
8. Añadimos las siguientes línea

**<build\_depend>actionlib</build\_depend>**

**<build\_depend>actionlib\_msgs</build\_depend>**

**<build\_export\_depend>actionlib</build\_export\_depend>**

**<build\_export\_depend>actionlib\_msgs</build\_export\_depend>**

**<exec\_depend>actionlib</exec\_depend>**

**<exec\_depend>actionlib\_msgs</exec\_depend>**

1. **Compilamos** el paquete usando **catkin\_make --only\_pkg\_with\_deps “nombre\_del\_paquete”** dentro de la carpeta catkin\_ws

#### C++

# Comando de C++

## Nodos

ros::init(argc, argv ,”nombre”) à Crea el nodo

ros::NodeHandle nh;à Crea el objeto NodeHandle

## Topics

ros:: Publisher “nombre” = nh.advertise<”Tipo de mensaje”>(“nombre”, “tamaño de la cola”); à Crea un publicador

ros::Subscriber “nombre” = nh.subscribe("topic", “tamaño cola”, “Callback”); à Crea un suscriptor

## Mensajes

“carpeta de mensaje”::”Tipo de mensaje” “nombre variable” à Crea un mensaje

“nombre”. “campo” à asigna un valor al campo

## Bucles

ros::spin() à Crea un bucle infinita

ros::Rate loop\_rate(“frecuencia”) à crea un bucle con frecuencia

ros::ok() à Comprueba en un bucle si se cancelo el proceso

ros::MultiThreadedSpinner spinner(n); à crea un n numero de hilos

## Servicios

### Cliente

ros::ServiceClient nombre = nh.serviceClient<Mensaje >("nombre del servicio");à Crea un cliente

client = waitForExistence(); àEspera que el servicio este disponibles

client.call(“Mensaje”) à Se envia la petición( se añade en un if)

### Servidor

ros::ServiceServer “nombre” = nh.advertiseService<MensajeRequest , MensajeResponse >("Servidor", callback); à Crea un servidor

## Acciones

### Cliente

actionlib::SimpleActionClient<Mensaje> “nombre” (“Nombre\_servicio", true);à Crea un cliente

cliente->waitForServer(); à Espera que la actividad este actividad

cliente->sendGoal(“peticion”, doneCB , activeCB, feedbackCB); à Envia el mensaje

bool rdo=cliente->waitForResult(ros::Duration(30)); à Espera un tiempo por el resultado

cliente->cancelAllGoals() 🡪 Cancela la acción anticipadamente

### Servidor

actionlib::SimpleActionServer<MensajeAcction>servidor(nh,"Servidor",cbaccion ,false)àCrea un servidor

servidor.start();à Inicial el servidor de acción

servidor->isPreemptRequested() à Comprueba que no se cancela

servidor->publishFeedback(feedback);à Envia el Feedback

servidor->setSucceeded(result); à Envia el resultado y pone el estado como resuelto

servidor->setPreempted(result); à Envia el resultado y pone el estado como cancelado

# Compilación de archivos

Para compilar un archivo de roscpp tenemos que modificar el Cmakelist.txt en el apartado Build

Modificaciones:

1. Descomentamos la función add\_executable

**Resultado**: add\_executable(“nombre binario” src/”nombre archivo”.cpp)

1. Descomentamos la función add\_dependencies

**Resultado**: add\_dependencies(“nombre binario” ${${PROJECT\_NAME}\_EXPORTED\_TARGETS} ${catkin\_EXPORTED\_TARGETS})

1. Descomentamos la función target\_link\_libraries

**Resultado**: target\_link\_libraries(“nombre binario”

${catkin\_LIBRARIES}

)

1. Compilamos con el comando **catkin\_make --only\_pkg\_with\_deps “nombre\_del\_paquete”**
2. **Lanzamos** el binario con **rosrun “nombre paquete” “nombre binario”**

# Topics

## Publicador

#include <ros/ros.h>

#include <std\_msgs/Int32.h>

#include <iostream>

using namespace std;

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

    //Creamos el nodo principal del programa

    ros::init(argc, argv, "Nodo publicador");

    ros::NodeHandle nh;

    //Creamos el Publicador

    ros::Publisher pub = nh.advertise<std\_msgs::Int32>(contador, 1);

    //Creacion del mensaje

    std\_msgs::Int32 cont;

    cont.data = 0;

    //Creacion de la frecuencia de publicacion

    ros::Rate loop\_rate(1);

    //Bucle de publicacion

    while (ros::ok()){

        pub.publish(cont);

        cont.data++;

        loop\_rate.sleep();

    }

    return 0;

}

## Suscriptor

#include <ros/ros.h>

#include <std\_msgs/Int32.h>

void Callback(const std\_msgs::Int32::ConstPtr& msg)

{

    /\*Funcion que se ejecuta cuando llega un mensaje por el topic subcriptor\*/

    ROS\_INFO("I heard: [%d]", msg->data);

}

int mian(int argc, char \*\*argv){

    //Creamos el nodo principal

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

    ros::NodeHandle nh;

    //Creamos el subcriptor

    ros::Subscriber sub = nh.subscribe("topic", 1000, Callback);

    //Creacion del bucle infinito

    ros::spin();

    return 0;

}

## Mensajes personalizados

1. Creamos la carpeta msg
2. Creamos el archivo “nombre”.msg y añadimos los valores del mensaje
3. Modificamos el CMakelists.txt
4. En la función **find\_package REQUIRED COMPONETS** añadimos **message\_generation**
5. En la función **add\_message\_files** añadimos los archivos "nombre".msg
6. En la función **generate\_messages** añadimos las **librerías** que depende el mensaje
7. Comprobamos la función **catkin\_package** esta descomentada y **añadimos cpp y message\_runtime**
8. Modificaciones del package.xml
9. Añadimos las siguientes líneas al final

**<build\_depend>message\_generation</build\_depend>**

**<exec\_depend>message\_runtime</exec\_depend>**

1. **Compilamos** el paquete usando **catkin\_make --only\_pkg\_with\_deps “nombre\_del\_paquete”** dentro de la carpeta catkin\_ws

Para poder usarla en un paquete al generar el paquete es necesario añadirlo

catkin\_create\_pkg “nombre paquete ” roscpp std\_msgs “nombre del paquete”

# Servicios

## Cliente

#include "ros/ros.h"

#include "Servicio/Prueba.h"

#include "iostream"

using namespace std;

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

    //Creamos el nodo principal

    ros::int\_node("Cliente", argc, argv);

    ros::NodeHandle nh;

    //Creacion del cliente de servicios

    ros::ServiceClient client = nh.serviceClient<std\_msgs::Int32>("Servidor");

    //Espremos para que el servidor este listo

    client = waitForExistence();

    ROS\_INFO("Servidor listo");

    //Creamos el mensaje

    Servicio::Prueba peticion;

    peticion.request.valor\_peticion = 5;

    //Llamamos al servicio

    if(client.call(msg)){

        //Si el servicio se ejecuto correctamente

        ROS\_INFO("El servicio finalizo correctamente");

        //Imprimimos el resultado

        ROS\_INFO("El resultado es: %d", msg.response.valor\_respuesta);

        return 0

    }

    else{

        ROS\_ERROR("Fallo al llamar al servicio");

        return 1;

    }

}

## Servidor

#include <ros/ros.h>

#include <Servicio/Prueba.h>

#include <iostream>

using namespace std;

bool callback (Servicio::PruebaRequest &req, Servicio::PruebaResponse &res){

    /\*Funcion que se ejecuta cuando llega una peticion al servidor\*/

    //Ejemplo

    ROS\_INFO("Recibido: %s", req.mensaje.c\_str());

    //Enviamos la respuesta

    res.valores\_respuesta = 0

    return true;

}

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

    //Creacion del nodo principal

    ros::init\_node("Servidor", argc, argv);

    ros::NodeHandle nh;

    //Creacion del servidor de servicios

    ros::ServiceServer server = nh.advertiseService<Servicio::PruebaRequest , Servicio:PruebaResponse >("Servidor", callback);

    //Creamos varios hilos de escucha

    ros::MultiThreadedSpinner spinner(4);

    spinner.spin();

}

## Servidor con clases

#include <ros/ros.h>

#include <Prueba/MensajeServicio.h>

class Servidor{

    /\*Clase de un servidor de servicios\*/

    private:

        //Creacion de variables privadas

        ros::NodeHandle nh;

        ros::ServiceServer server;

  bool callback(Preueba::MensajeServicioRequest &req, Prueba::MensajeServicioResponse &res){

        /\*Función callback del servicio\*/

        ROS\_INFO("Recibido: %s", req.mensaje.c\_str());

        //Enviamos la respuesta

        res.valor\_respuesta = 0;

        return true;

    }

    public:

        //Creamos el constructor

        Servidor(){

           server = nh.advertiseService("servicio", &Servidor::callback, this);

        }

}

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

    //Creo el nodo principal

    ros::init(argc, argv, "servidor\_con\_clase");

    //Creo el objeto del servicio

    Servidor servidor;

    ros::spin();

    return 0;

}

## Masajes Personalizados

1. **Creamos una carpeta srv**
2. Creamos un **archivo "nombre".srv** y **lo modificamos** con el siguiente esquema

#Peticion

---

#Respuesta

1. **Modificamos el CmakeLsit.txt**
   1. En la **función** **find\_package** añadimos **message\_generation** y añadimos las dependencias del mensaje
   2. En la **función add\_service\_files** **añadimos** el nombre del archivo **“nombre”.srv**
   3. **Descomentamos** la función **generate\_messages**
2. **Modificamos el package.xml** añadimos las siguientes líneas:

<build\_depend>message\_generation</build\_depend>

<exec\_depend>message\_runtime</exec\_depend>

1. **Compilamos el paquete**

# Acciones

## Clientes

//tenemos que añadir en el paquete actionlib

#include <ros/ros.h>

#include <Prueba/MensajeAccionAction.h>

#include <actionlib/client/simple\_action\_client.h>

#include <iostream>

using namespace std;

//Definicion de variables globales

actionlib::SimpleActionClient<Prueba::MensajeAccionAction> \*cliente=NULL;

void doneCB(const actionlib::SimpleClientGoalState& state, const Prueba::MensajeAccionResultConstPtr& result){

    /\*Funcion que se ejecuta cuando se termina la accion\*/

    ROS\_INFO("Terminado y el resultado es: %d", result->resultado.resultado);

}

void feedbackCB(const Prueba::MensajeAccionFeedbackConstPtr& feedback){

    /\*Funcion que se ejecuta cuando se envia feedback\*/

    ROS\_INFO("Progreso: %d", feedback->progreso);

    //Calcelacion de la accion

    if (feedback->progreso == 50){

        ROS\_INFO("Cancelando accion");

        cliente->cancelAllGoals();

    }

}

void activeCB(){

    /\*Funcion que se ejecuta cuando se activa la accion\*/

    ROS\_INFO("Accion activa");

}

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

    //Creamos el nodo principal

    ros::init(argc, argv, "nodo\_cliente");

    ros::NodeHandle nh;

    //Creamos el cliente

    cliente =new actionlib::SimpleActionClient<Prueba::MensajeAccionAction> ("Servidor\_accion", true);

    //Esperamos a que el servidor se inicie

    cliente->waitForServer();

    ROS\_INFO("Servidor\_accion conectado");

    //Creamos el mensaje

    Prueba::MensajeAccionGoal goal;

    goal.numero = 5;

    // FIXME: (Opcion 1)

    // Enviamos el mensaje

    cliente->sendGoal(goal, doneCB , activeCB, feedbackCB);

    //Esperamos a que el servidor nos devuelva el resultado

    bool rdo=cliente->waitForResult(ros::Duration(30));

    if(rdo){

        ROS\_INFO("El estado del servidor es: %s", cliente->getState().toString().c\_str());

        ROS\_INFO("El resultado es: %d", cliente->getResult()->resultado);

    }

    else{

        ROS\_INFO("El servidor no ha devuelto el resultado");

    }

    //FIXME: (Opcion 2)

//Esperamos a que el servidor nos devuelva el resultado

    action::SimpleClientGoalState estado = cliente->getState();

    while (estado == actionlib::SimpleClientGoalState::PENDING or estado == actionlib::SimpleClientGoalState::ACTIVE){

        ROS\_INFO("El estado del servidor es: %s", estado->toString().c\_str());

        estado = cliente->getState();

    }

    ROS\_INFO("El resultado es: %d", estado.getResult()->result)

    return 0;

}

## Servidor

#include <ros/ros.h>

#include <actionlib/server/simple\_action\_server.h>

#include <Prueba/MensajeAccionAction.h>

#include <iostream>

using namespace std;

//Definicion de variables globales

actionlib::SimpleActionServer<Prueba::MensajeAccionAction> \*servidor=NULL;

void cbaccion(const Prueba::MensajeAccionGoalConstPtr &goal){

    /\*Funcion que se ejecuta cuando se pide una accion al servidor \*/

    //Creamos los mensajes

    Prueba::MensajeAccionFeedback feedback;

    Prueba::MensajeAccionResult result;

    bool estado = true;

    //Declaramos velocidad de ciclos

     ros::Rate rate(1);

    //FIXME: Accion (Ejemplo)

    for(int i=0; i<goal->numero; i++){

        //Comprobamos si se ha cancelado la accion

        if (servidor->isPreemptRequested()){

            ROS\_INFO("El cliente ha cancelado la accion");

            estado = false

            break;

        }

        //Enviamos el feedback

        feedback.progreso = 0;

        servidor->publishFeedback(feedback);

rate.sleep();

    }

    //Enviamos el resultado

    result.resultado = 0;

        //Comprobamos que no se ha cancelado la accion

    if (estado == true){

        servidor->setSucceeded(result);

    }

    else{

        servidor->setPreempted(result);

    }

}

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

    //Creamos el nodo principal

    ros::init(argc, argv, "nodo\_servidor");

    ros::NodeHandle nh;

    //Creamos el servidor

    servidor =new actionlib::SimpleActionServer<Prueba::MensajeAccionAction> (nh, "Servidor\_accion",cbaccion ,false);

    servidor.start();

    ROS\_INFO("Servidor\_accion iniciado");

    ros::spin();

return 0;

}

## Servidor con clases

#include <ros/ros.h>

#include <actionlib/server/simple\_action\_server.h>

#include <Prueba/MensajeAccionAction.h>

#include <string>

#include <iostream>

using namespace std;

class Servidor{

    private:

        //Definicion de las varaibles privadas

        ros::NodeHandle nh;

        actionlib::SimpleActionServer<Prueba::MensajeAccionAction> server;

        Prueba::MensajeAccionFeedback feedback;

        Prueba::MensajeAccionResult result;

        ros::Subscriber sub = nh.subscribe("topic", 1000, &Servidor::Callback, this);

        string accion\_name;

        void cbServidor(const Prueba::MensajeAccionGoalConstPtr &goal){

            /\*Funcion que se ejecuta cuando se pide una accion al servidor \*/

            //Declaramos velocidad de ciclos

            ros::Rate rate(1);

            // FIXME: Accion (Ejemplo)

            for (int i = 0; i < goal->numero; i++){

                // Comprobamos si se ha cancelado la accion

                if (server.isPreemptRequested()){

                    ROS\_INFO("El cliente ha cancelado la accion");

                    estado = false;

                    break;

                }

                // Enviamos el feedback

                feedback.progreso = 0;

                server.publishFeedback(feedback);

                rate.sleep();

            }

            // Enviamos el resultado

            result.resultado = 0;

            if (estado == true){

                server.setSucceeded(result);

            }

            else{

                server.setPreempted(reuslt);

            }

        }

    public:

        //Declaramos el contructor

        Servidor(string name): server(nh, name, boost::bind(&Servidor::cbServidor, this , \_1) ,false ),accion\_name(name){

            //Inicializamos el servidor

            servidor.start();

        }

}

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

    //Creamos el nodo principal

    ros::init(argc, argv, "Servidor\_clase");

    //Creamos el objeto servidor

    Servidor servidor("Servidor\_accion");

    ros::spin();

    return 0;

}

## Mensajes personalizados

1. Primero en el paquete añadimos el paquete actionlib\_msgs
2. Creamos la carpeta action
3. Creamos el archivo "MensajeAction.action" y modificamos el archivo con el siguiente contenido:

#Goal

---

#Result

---

#Feedback

1. Modificamos el CMakeList.txt:
   1. Descomentamos add\_action\_files() y añadimos el nombre de los archivos que queremos compilar
   2. Descomentamos el generate\_messages()
   3. Descomentamos el catkin\_package()

1. Compilamos el paquete con catkin\_make