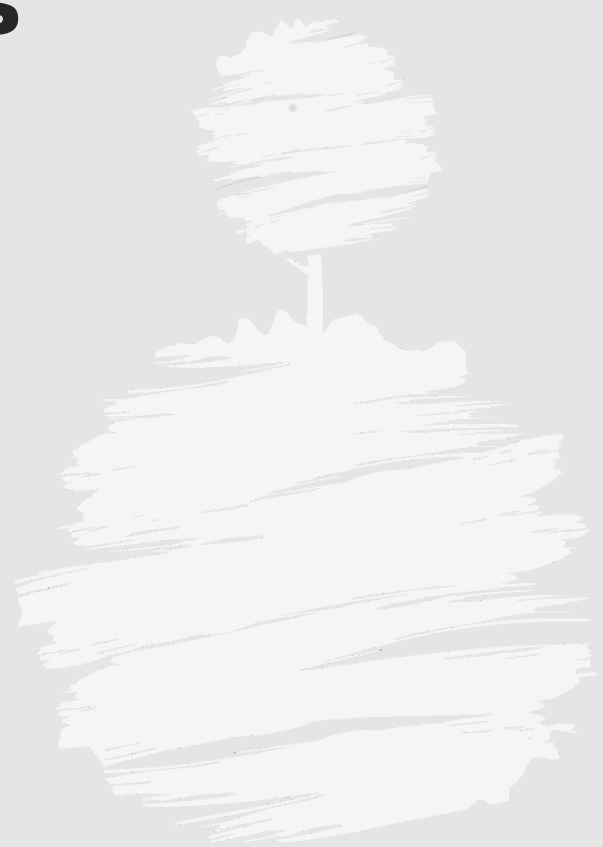
# ROS y C++. Servicios – Mensajes

Plataformas de Software en Robótica



# Crear un mensaje personalizado para un servicio

- La forma de crear un mensaje personalizado para un servicio es la misma que en Python
- Podemos crearlo en el propio paquete en el que lo vamos a usar o tener un paquete con todos nuestros mensajes de servicio al estilo de std_srvs
- Creamos un paquete en el que guardaremos nuestros mensajes personalizados para servicios
  - No son necesarias las dependencias de roscpp ni de std_srvs pero si las agregamos las podemos tener disponibles en caso de que queramos añadir código en el futuro
  - Necesitamos crear una carpeta dentro del paquete con el nombre *srv*
  - Dentro de la carpeta *srv* meteremos nuestros mensajes personalizados de servicios
    - <nombre_mensaje_servicio>.srv

# Crear un mensaje personalizado para un servicio

- Los mensajes para los servicios tienen dos partes
  - Request
  - Response
- Las variables asociadas al Request se separan de las asociadas al Response a través de “- - -”
- Cada una de las partes pueden tener ‘n’ atributos o variables

Ej: srvmsg.srv

```
int32 tiempo
- - -
string msg
bool status
```

# Crear un mensaje personalizado para un servicio

- Para poder tener disponible el mensaje de servicio tenemos que compilar el paquete
- Para compilar el paquete tenemos que modificar el CMakeList.txt y el package.xml
- Modificaciones para el CmakeList.txt
  - Las modificaciones son similares a las que hicimos para los mensajes personalizados en el apartado de Topics

# Crear un mensaje personalizado para un servicio

- Modificación del **CmakeLists.txt**
  - Añadir el paquete message_generation a la sección find_package

```
find_package(catkin REQUIRED COMPONENTS
      roscpp
      std_msgs
      message_generation
)
```

  - Descomentar la sección “add_service_files”

```
 add_service_files(
   FILES
   testsrv.srv
#   Service2.srv
 )
```

# Crear un mensaje personalizado para un servicio

- Modificación del **CmakeLists.txt**
  - Descomentar la sección “add_service_files”

```
generate messages(
  DEPENDENCIES
  std_msgs
)
```

- Modificación del **package.xml**
  - Añadir las siguientes líneas
    - <build_depend>message_generation</build_depend>
    - <exec_depend>message_runtime</exec_depend>

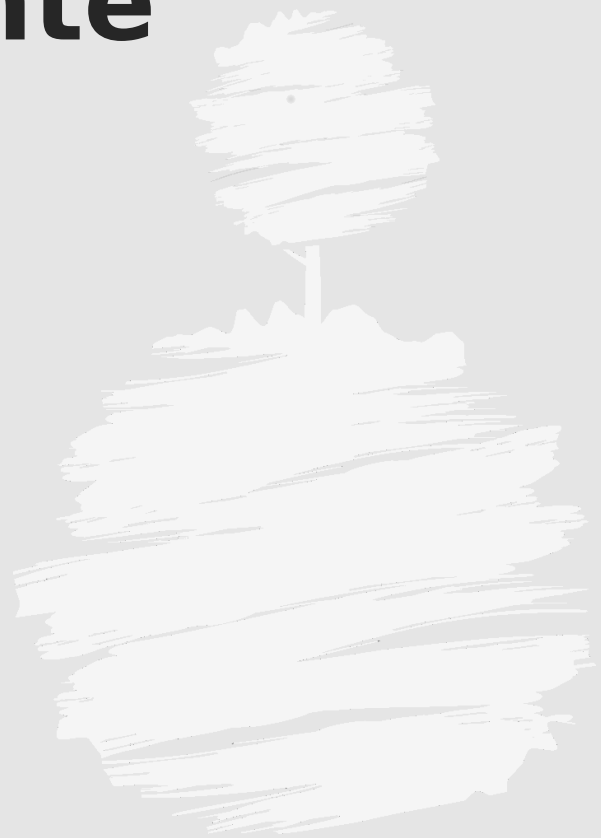
# Crear un mensaje personalizado para un servicio

- Para emplear el paquete de mensajes en otro proyecto podemos simplemente agregarlo como una dependencia al crear el nuevo paquete
  - catkin_create_pkg <nuevo_pkg> roscpp  <my_own_srv_msgs_pkg>

- Las modificaciones específicas  realizadas en el *CMakeList.txt* y el *package.xml* para emplear los mensajes personalizados **solo se hacen en el paquete donde se crean los mensajes de servicio**. En el paquete que se utilizan no es necesario

# ROS y C++. Servicios – Cliente

Plataformas de Software en Robótica



# Crear un cliente para llamar a un servicio

- El proceso es similar a la creación de un publicador de un topic
- Tenemos que crear un paquete importando las dependencias necesarios
  - Roscpp
  - <paquete_mensaje_servicio> (ej. std_srvs)

# Crear un cliente para llamar a un servicio

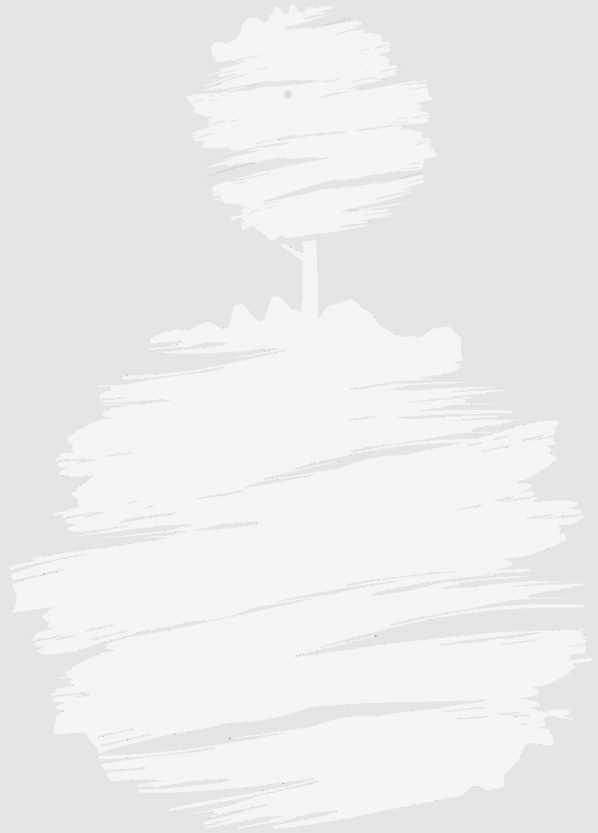
- Crear un fichero .cpp con el código fuente para llamar al servicio
- Puntos importantes
  - Incluir el paquete que contenga los mensajes de servicio
  - Crear el cliente de servicio (ServiceClient) a través del NodeHandle
  - Gestionar la respuesta del servicio (mensaje response), una vez se finaliza la llamada
- Cambiar el CMakeList para poder compilar el código fuente
    - add_executable
    - add_dependencies
    - target_link_libraries
- Crear la carpeta launch (opcional)
- Compilar
- Lanzar el nodo con roslaunch o rosrun (incluir roscore si no está activo)

# Crear un cliente para llamar a un servicio

```cpp
#include<ros/ros.h>
#include<prueba_msg_srv/Timer.h>
#include<iostream>

using namespace std;


int main(int argc, char** argv){
    ros::init(argc, argv, "cliente_srv");// Inicializa el nodo
    ros::NodeHandle nh;
    ros::ServiceClient srv_cliente=nh.serviceClient<prueba_msg_srv::Timer>("timer_service");/* Crea el cliente para el servicio.
    Es una plantilla por lo que se necesita especificar el tipo de mensaje que soporta el servicio*/

    ROS_INFO("Esperando al servicio...");
    srv_cliente.waitForExistence();//Espera a que el servicio esté levantado
    //ros::service::waitForService("timer_service") //Espera a que el servicio esté levantado
    prueba_msg_srv::Timer msg_time; //Mensaje para invocar el servicio . Tiene un campo request y otro response
    cout<<"Introduce el numero de segundos: ";
    cin>>msg_time.request.time;//Cubrimos los campos asociados al request

    if(srv_cliente.call(msg_time)){//llamamos al servicio y se queda bloqueado hasta que se resuelve la llamada
        //Si el servicio se resuelve con True
        ROS_INFO("El servicio finalizo correctamente");
        //Una vez resuelto el servicio los campos response quedan cubiertos (los cubre el servidor del servicio)
        ROS_INFO_STREAM("Estado: "<<((msg_time.response.status)?"OK":"KO")<<". Resultado: "<<msg_time.response.resultado);
        return 0;

    }else{
        ROS_INFO("El servicio finalizo con problemas");
        return 1;
    }
```

# ROS y C++. Servicios – Servidor

Plataformas de Software en Robótica



# Crear un servidor de servicio

- Similar a la creación de un suscriptor en los Topics
- Creamos un paquete con las dependencias necesarias
  - roscpp
  - <paquete_mensajes_servicio> (ej. std_srvs)

# Crear un servidor deservicio

- Crear un fichero .cpp con el código fuente
- Puntos importantes
  - Incluir  el paquete que contenga los mensajes de servicio
  - Crear el servidor de servicio con el NodeHandle (ServiceServer)
  - Callback con la acción a realizar por el servicio
- Cambiar el CMakeList para poder compilar el código fuente
    - add_executable
    - add_dependencies
    - target_link_libraries
- Crear la carpeta launch (opcional)
- Compilar
- Lanzar el nodo con roslaunch o rosrun (incluir roscore si no está activo)

# Crear un servidor de servicio

```cpp
#include<ros/ros.h>
#include<prueba_msg_srv/Timer.h>

bool cbTimerService(prueba_msg_srv::TimerRequest &req, prueba_msg_srv::TimerResponse &res){
    /*!!!!! Fijaos que se reciben referencias NO constantes. Esto significa que podemos
    cambiarlas dentro de la función y le afecta "fuera". En los servicios es la forma de
    devolver información*/
    ROS_INFO("Accediendo al servicio");
    ros::Rate loop_rate(1);

    for (int i=0; i<req.time;i++){
        ROS_INFO_STREAM("Waiting..."<<i);
        loop_rate.sleep();
    }

    res.status=true;//modificamos la referencia del mensaje response
    res.resultado="Todo correcto";//modificamos la referencia del mensaje response
    return true;/*Esto es la devolución del servicio. Nos sirve para indicar si se pudo finalizar
    correctamente el servicio. No tiene nada que ver con la lógica que queremos implementar en el
    servicio. Esto es lo que permite que la condicional (IF-ELSE) del cliente funcione*/
}

int main(int argc, char** argv){
    ros::init(argc, argv, "servidor_srv");
    ros::NodeHandle nh;

    ros::ServiceServer srv_server=nh.advertiseService<prueba_msg_srv::TimerRequest,prueba_msg_srv::TimerResponse>("timer_service", cbTimerService);
    ROS_INFO("Publicando el servicio");
    ros::spin();

    return 0;
}
```

# Lanzar los nodos con el fichero launch

- Ejemplo del fichero *launch* que lanzaría el servicio

```
<launch>
    <node pkg='prueba_servicio' type='servidor_node' name ='servidor_nodo' output='screen'/>
</launch>
```

- Ejemplo del fichero *launch* lanzando el cliente y servicio al mismo tiempo

```
<launch>
    <include file="$(find prueba_servicio)/launch/servidor_servicio.launch"/>
    <node pkg='prueba_servicio' type='cliente_node' name ='cliente_servicio' output='screen'/>
</launch>
```