

# Elementos de ROS

Clase 5  
Ing. Alexander López

## Repasso clase 4

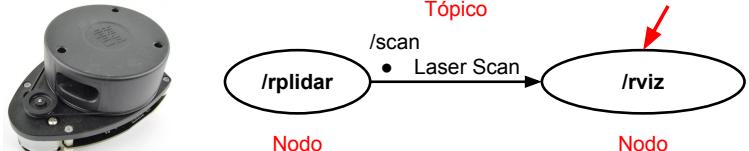
- IA - Ros Graph
- Roscore
- Workspace
- Tecla TAB
- Ros packages
- Rosbash
- Rosrun
- Names
- Namespaces
- Remapping
- Roslaunch

## Tópicos

### ¿Que es?

- Es un nombre para un flujo de mensajes con un tipo definido.
- Estos implementan un publicación/suscriptor como mecanismo de comunicación
- Establece la comunicación entre nodos.

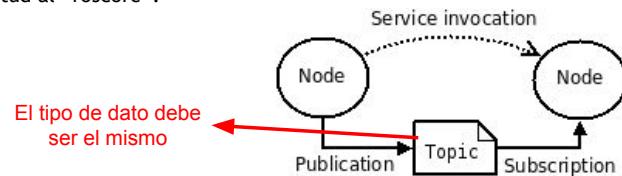
#### Ejemplo:



## Tópicos

### ¿Como se establece?

- Primero se anuncia que tópico y tipo de data se va a enviar, el “roscore” registra esta información.
- Nodos pueden empezar a publicar data en el tópico.
- Otros nodos pueden empezar a recibir messages suscrito haciendo una solicitud al “roscore”.



## Tópicos

Ventajas:

- Sintaxis definida.
- Mecanismos de comunicación ya implementados.
- Establecer comunicación entre nodos.
- Verificación de data correctamente recibida.

## Publicando un tópico

```
#!/usr/bin/env python
import rospy
from std_msgs.msg import Int32
rospy.init_node('topic_publisher')
pub = rospy.Publisher('counter', Int32)
rate = rospy.Rate(2)
count = 0
while not rospy.is_shutdown():
    pub.publish(count)
    count += 1
    rate.sleep()
```

Conocido como  
shebag

Le permite al sistema  
operativo saber que es  
un script de Python

## Publicando un tópico

```
#!/usr/bin/env python
import rospy
from std_msgs.msg import Int32
rospy.init_node('topic_publisher')
pub = rospy.Publisher('counter', Int32)
rate = rospy.Rate(2)
count = 0
while not rospy.is_shutdown():
    pub.publish(count)
    count += 1
    rate.sleep()
```

Importa las funciones  
básicas que se van a  
utilizar

## Publicando un tópico

```
#!/usr/bin/env python
import rospy
from std_msgs.msg import Int32
rospy.init_node('topic_publisher')
pub = rospy.Publisher('counter', Int32)
rate = rospy.Rate(2)
count = 0
while not rospy.is_shutdown():
    pub.publish(count)
    count += 1
    rate.sleep()
```

Importa la definición  
del mensaje

El tipo de mensaje a  
usar es enteros de  
32-bits

## Publicando un tópico

- Se debe agregar la siguiente dependencia en el archivo “package.xml”.
- Sin esta el nodo no podría encontrar la definición de mensajes.

```
<buildtool_depend>catkin</buildtool_depend>
  <build_depend>message_generation</build_depend>
  <build_depend>rostest</build_depend>
  <build_depend>std_msgs</build_depend>

  <run_depend>message_runtime</run_depend>
  <run_depend>rospy</run_depend>
  <run_depend>std_msgs</run_depend>
```

Ejemplo, lista de dependencia de rospy\_tutorials

## Publicando un tópico

```
#!/usr/bin/env python
import rospy
from std_msgs.msg import Int32
rospy.init_node('topic_publisher')
pub = rospy.Publisher('counter',
Int32)
rate = rospy.Rate(2)
count = 0
while not rospy.is_shutdown():
    pub.publish(count)
    count += 1
    rate.sleep()
```

Establecemos el ratio al cual vamos a publicar  
En Hertz

## Publicando un tópico

```
#!/usr/bin/env python
import rospy
from std_msgs.msg import Int32
rospy.init_node('topic_publisher') ← Iniciando el nodo
pub = rospy.Publisher('counter', Int32, queue_size=10) ← Se publica un tópico de nombre counter y con un tipo de dato Int32
rate = rospy.Rate(2)
count = 0
while not rospy.is_shutdown():
    pub.publish(count)
    count += 1
    rate.sleep() ← Esta información se envía al roscore
```

## Publicando un tópico

```
#!/usr/bin/env python
import rospy
from std_msgs.msg import Int32
rospy.init_node('topic_publisher')
pub = rospy.Publisher('counter',
Int32)
rate = rospy.Rate(2)
count = 0
while not rospy.is_shutdown():
    pub.publish(count)
    count += 1
    rate.sleep()
```

Se Inicializa la variable count en 0

Función is\_shutdown()  
True: Se activa cuando se cierra el nodo

## Publicando un tópico

```
#!/usr/bin/env python
import rospy
from std_msgs.msg import Int32
rospy.init_node('topic_publisher')
pub = rospy.Publisher('counter',
Int32)
rate = rospy.Rate(2)
count = 0
while not rospy.is_shutdown():
    pub.publish(count)
    count += 1
    rate.sleep()
```

Publicar la variable count  
Se aumenta la cuenta en una unidad  
Esperar 30 segundos para que el programa pueda continuar

¿Chequear si todo esta bien?

## Publicando un tópico

Para verificar si todo está bien hay que seguir los siguiente pasos:

- Abrir un terminal e iniciar el “roscore”.
- Abrir otro terminal e inscribir:

\$ rostopic list

```
alex@alex-VirtualBox: ~
alex@alex-VirtualBox: ~$ rostopic list
/rossout
/rossout_agg
alex@alex-VirtualBox: ~$
```

\$ rostopic -h

```
alex@alex-VirtualBox: ~
alex@alex-VirtualBox: ~$ rostopic -h
[rostopic] is a command-line tool for printing information about ROS Topics.

Commands:
  rostopic bw      display bandwidth used by topic
  rostopic delay   display delay of topic from timestamp in header
  rostopic echo    print messages to screen
  rostopic find    find topics by type
  rostopic hz      display current rate of topic
  rostopic info    print information about active topic
  rostopic list    list active topics
  rostopic pub    publish data to topic
  rostopic type   print topic type

Type rostopic <command> -h for more detailed usage, e.g. 'rostopic echo -h'
```

## Publicando un tópico

¿Chequear si todo esta bien?

Ahora se inicia el tópico creado en el terminal que tengamos disponible.

\$ chmod u+x topic\_publisher.py

Da los permisos para ejecutar el archivo, hay que ubicarse en la carpeta del archivo

\$ rostopic list

```
alex@alex-VirtualBox: ~
alex@alex-VirtualBox: ~$ rostopic list
/counter
/rossout
/rossout_agg
alex@alex-VirtualBox: ~$
```

\$ rostopic echo counter -n 5

```
alex@alex-VirtualBox: ~
alex@alex-VirtualBox: ~$ rostopic echo counter -n 5
subscribed to [/counter]
data: 154
...
data: 155
...
data: 156
...
data: 157
...
data: 158
...
alex@alex-VirtualBox: ~$
```

Más información

## Publicando un tópico

Aquí se puede verificar ratio de transmisión de datos.

\$ rostopic hz counter

```
alex@alex-VirtualBox: ~
alex@alex-VirtualBox: ~$ rostopic hz counter
subscribed to [/counter]
average rate: 1.999
min: 0.500s max: 0.500s std dev: 0.00000s window: 2
average rate: 2.000
min: 0.500s max: 0.500s std dev: 0.00031s window: 4
average rate: 2.000
min: 0.500s max: 0.500s std dev: 0.00032s window: 6
^Caverage rate: 2.000
min: 0.500s max: 0.500s std dev: 0.00033s window: 8
alex@alex-VirtualBox: ~$
```

Verificar las conexiones entre los tópicos, puerto usado en la conexión TCP

\$ rostopic info counter

```
alex@alex-VirtualBox: ~
alex@alex-VirtualBox: ~$ rostopic info counter
Type: std_msgs/Int32
Publishers:
  * /topic_publisher (http://alex-VirtualBox:42379/)
Subscribers: None

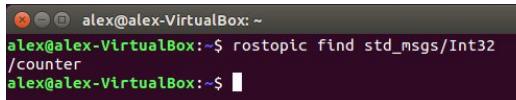
alex@alex-VirtualBox: ~$
```

## Publicando un tópico

Más información

Se puede los tópicos publicados que transmiten ciertos tipos de mensajes.

```
$ rostopic find std_msgs/Int32
```



```
alex@alex-VirtualBox: ~
alex@alex-VirtualBox: $ rostopic find std_msgs/Int32
/counter
alex@alex-VirtualBox: $
```

## Suscribiendo a un tópico

```
#!/usr/bin/env python
import rospy
from std_msgs.msg import Int32
```

shebag

Importa las funciones a utilizar

```
def callback(msg):
    print msg.data

rospy.init_node('topic_subscriber')
sub = rospy.Subscriber('counter', Int32, callback)
rospy.spin()
```

## Suscribiendo a un tópico

```
#!/usr/bin/env python
import rospy
from std_msgs.msg import Int32
def callback(msg):
    print msg.data
rospy.init_node('topic_subscriber')
sub = rospy.Subscriber('counter', Int32, callback)
rospy.spin()
```

Se define una función que imprima en la consola el mensaje que llega del tópico

## Suscribiendo a un tópico

```
#!/usr/bin/env python
import rospy
from std_msgs.msg import Int32
def callback(msg):
    print msg.data
rospy.init_node('topic_subscriber')
sub = rospy.Subscriber('counter', Int32, callback)
rospy.spin()
```

Se inicia un nodo y se especifica su nombre

El suscriptor especifica el nombre del tópico, el tipo de dato y la función "callback".

## Suscribiendo a un tópico

```
#!/usr/bin/env python
import rospy
from std_msgs.msg import Int32

def callback(msg):
    print msg.data

rospy.init_node('topic_subscriber')
sub = rospy.Subscriber('counter', Int32, callback)
rospy.spin()
```

Si no hay necesidad de controlarlo, basta llamar a "rospy.spin()", ya que "callbacks" se ejecutan en sus propios subprocesos.

Dar control del nodo a ROS y sale cuando el nodo está listo para apagarse.

## Suscribiendo a un tópico

¿Chequear si todo esta bien?

- Primero se verifica si el nodo publicador sigue corriendo.
- Luego se inicia en otro terminal el nodo suscriptor.

```
alex@alex-VirtualBox: ~
alex@alex-VirtualBox: ~$ rosrun new_package topic_subscriber.py
600
601
602
603
604
605
^Calex@alex-VirtualBox: ~
```

- Si se tiene este resultado, se está corriendo el primer sistema de ROS.  
Esto se tiene que Escalabilizar!

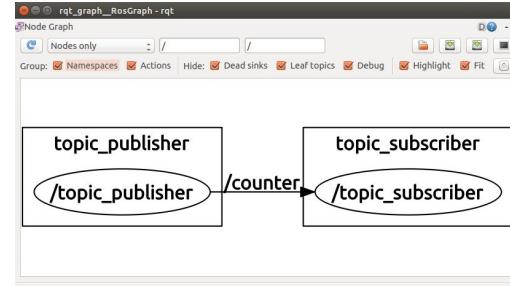
## Suscribiendo a un tópico

```
alex@alex-VirtualBox: ~
alex@alex-VirtualBox: ~$ rosrun new_package topic_subscriber.py
[rosrun] Couldn't find executable named topic_subscriber.py below /home/alex/catkin_ws/src/new_package
[rosrun] Found the following, but they're either not files,
[rosrun] or not executable:
[rosrun] /home/alex/catkin_ws/src/new_package/src/topic_subscriber.py
alex@alex-VirtualBox: ~
```

¿Que puede estar pasando?

## Suscribiendo a un tópico

\$ rqt\_graph



## Suscribiendo a un tópico

- Se puede publicar mensajes en el tópico a conveniencia de usuario.

```
$ rostopic pub counter std_msgs/Int32 1000000
```

```
$ rostopic info counter
```

```
alex@alex-VirtualBox:~$ alex@alex-VirtualBox:~$ rostopic info counter
type: std_msgs/int32
Publishers:
* /topic_publisher (http://alex-VirtualBox:42379/)
Subscribers:
* /topic_subscriber (http://alex-VirtualBox:41602/)

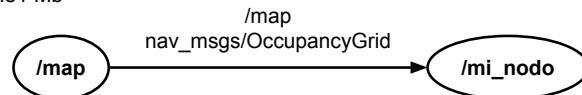
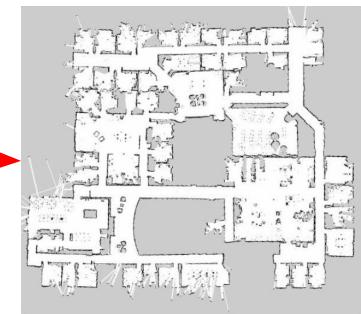
alex@alex-VirtualBox:~$
```

```
alex@alex-VirtualBox:~$ alex@alex-VirtualBox:~$ rostopic pub counter std_msgs/Int32 1000000
publishing and latching message. Press ctrl-C to terminate
906
907
908
909
910
911
912
913
914
915
916
917
918
919
920
921
922
923
1000000
924
```

## Latched Topics

Mapa generado por  
sensor Láser 2D

Mapa de 200x200 metros o  
2000x2000 pixeles.  
Un byte por canal rbg o b/n.  
Por imagen = 2000\*2000\*1  
Total = 4000000 bytes  
Total = 3.81 Mb



## Latched Topics

- Ofrecen una simple solución a este problema.
- Si un tópico es marcado como “latched” cuando se declara, los suscriptores obtienen automáticamente el último mensaje enviado cuando se suscriben al tema, no se tiene que esperar por uno nuevo.

```
pub = rospy.Publisher('map', nav_msgs/OccupancyGrid, latched=True)
```

## Tipos de mensajes

- Ros ofrece una amplia variedad de mensajes con formato ya creados.
- Los std\_msgs package define los de tipo primitivo.
- Un tópico se define por el tipo de mensaje transmite o recibe.
- Ros tiene comandos para verificar el estado o obtener información de los mensajes con los que se trabajan.

```
$ rosmsg package std_msgs
```

## Tipos de mensajes

ROS type	Serialization	C++ type	Python type	Notes
bool	Unsigned 8-bit integer	uint8_t	bool	
int8	Signed 8-bit integer	int8_t	int	
uint8	Unsigned 8-bit integer	uint8_t	int	uint8[] is treated as a string in Python
int16	Signed 16-bit integer	int16_t	int	
uint16	Unsigned 16-bit integer	uint16_t	int	
int32	Signed 32-bit integer	int32_t	int	
uint32	Unsigned 32-bit integer	uint32_t	int	

## Tipos de mensajes

int64	Signed 64-bit integer	int64_t	long
uint64	Unsigned 64-bit integer	uint64_t	long
float32	32-bit IEEE float	float	float
float64	64-bit IEEE float	double	float
string	ASCII string	std::string	ROS does not support Unicode strings; use a UTF-8 encoding
time	secs/nsecs unsigned 32-bit	ros::Time	rospy.Time duration
ints			

## Tipos de mensajes

```
alex@alex-VirtualBox:~$ rosmsg package sensor_msgs
sensor_msgs/BatteryState
sensor_msgs/Chatter
sensor_msgs/CompressedImage
sensor_msgs/FluidPressure
sensor_msgs/Illuminance
sensor_msgs/Image
sensor_msgs/Imu
sensor_msgs/ImuOrientationState
sensor_msgs/Joy
sensor_msgs/JoyFeedback
sensor_msgs/JoyFeedbackArray
sensor_msgs/LaserEcho
sensor_msgs/LaserScan
sensor_msgs/MagneticField
sensor_msgs/MultichannelImage
sensor_msgs/MultichannelLaserScan
sensor_msgs/NavSatFix
sensor_msgs/NavSatStatus
sensor_msgs/PointCloud
sensor_msgs/PointCloud2
sensor_msgs/PointField
sensor_msgs/Range
sensor_msgs/RegionOfInterest
sensor_msgs/RelativeHumidity
sensor_msgs/Temperature
sensor_msgs/TimeReference
alex@alex-VirtualBox:~$
```

## Servicios

- Permiten a un nodo llamar a una función que se ejecuta en otro nodo.
- Se Define las entradas y salidas de esta función
- El servidor (que proporciona el servicio) especifica una devolución de llamada para hacer frente a la solicitud de servicio y anuncia el servicio.
- El cliente (que llama al servicio) accede a este servicio a través de un proxy local.
- Las llamadas de servicio se adaptan bien a las cosas que sólo necesita hacer de vez en cuando y que tomar una cantidad limitada de tiempo para completar.

## Servicios

- Las acciones discretas que el robot podría hacer, como encender un sensor o tomar una imagen de alta resolución con una cámara, son también buenos candidatos para una implementación de llamadas de servicio.
- Existe varios servicios ya definidos por paquetes en ROS.

## Definiendo un servicio

¿Como crear un servicio?

Ejemplo: Servicio que cuenta el número de palabras de una frase

- Primero se debe ubicarse en la raíz de nuestro paquete y crear la carpeta srv
- Después se crea el archivo WordCount.srv

The terminal shows the command to create a new package and then navigate into it to create a 'srv' directory. The code editor shows the 'WordCount.srv' file with the following content:

```
string words
...
uint32 count
```

Annotations in red point to the terminal output (the path), the code editor window, and specific lines of code: 'Variables de entrada' points to 'string words', and 'Variables de Salida' points to 'uint32 count'.

Carpeta que donde se ubican los servicios

## Definiendo un servicio

¿Como crear un servicio?

The terminal shows the creation of a new package. The code editor shows the 'CMakeLists.txt' file with the following content:

```
cmake_minimum_required(VERSION 2.8.3)
project(new_package)

## Add support for C++11, supported in ROS Kinetic and newer
#_add_definitions(-std=c++11)

## Find catkin macros and libraries
## if COMPONENTS list like find_package(catkin REQUIRED COMPONENTS xyz)
## is used, also find other catkin packages
find_package(catkin REQUIRED COMPONENTS
    rospy
    std_msgs
    message_generation
)
```

Annotations in red point to the terminal output (the path), the code editor window, and specific lines of code: 'Agregar el paquete que se usa para los servicios' points to the 'catkin REQUIRED COMPONENTS' section, 'Editar las editar parámetros que debe compilar catkin\_make' points to the '#\_add\_definitions' line.

## Definir un servicio

¿Como crear un servicio?

Agregar el archivo donde se especifican el nombre y tipo de las variables a usar

Agregar dependencias (tipos de variables a usar)

The code editor shows the 'CMakeLists.txt' file with the following content:

```
## Generate services in the 'srv' folder
add_service_files(
  FILES
    WordCount.srv
    # Service1.srv
    # Service2.srv
  )

## Generate actions in the 'action' folder
## add_action_files(
##   FILES
##     Action1.action
##     Action2.action
##   )

## Generate added messages and services with any dependencies listed
## here
generate_messages(
  DEPENDENCIES
    std_msgs
)
```

## Definir un servicio

Verificar que todo va bien

```
alex@alex-VirtualBox:~/catkin_ws/src/new_package$ rosrun roscd new_package/
alex@alex-VirtualBox:~/catkin_ws/src/new_package$ gedit CMakeLists.txt
alex@alex-VirtualBox:~/catkin_ws/src/new_package$ rosrv show WordCount
[new_package/WordCount]:
string words
...
uint32 count
alex@alex-VirtualBox:~/catkin_ws/src/new_package$
```

## Definir un servicio

Compilando los parámetros que se añadieron pasos atrás

```
alex@alex-VirtualBox:~/catkin_ws$ cd build
alex@alex-VirtualBox:~/catkin_ws$ make -j1 -l1" in "/home/alex/catkin_ws/build"
Scanning dependencies of target "new_package_generate_messages_check_deps_WordCount"
[ 0%] Built target "new_package_generate_messages_check_deps_WordCount"
Scanning dependencies of target std_msgs_generate_messages_lisp
[ 0%] Built target std_msgs_generate_messages_lisp
Scanning dependencies of target "new_package_generate_messages_lisp"
[ 25%] Built target "new_package_generate_messages_lisp"
Scanning dependencies of target std_msgs_generate_messages_py
[ 50%] Generating Python code from SRV "new_package/WordCount.srv"
[ 50%] Built target std_msgs_generate_messages_py
Scanning dependencies of target "new_package_generate_messages_py"
[ 75%] Generating Python code from SRV "new_package/WordCount.srv"
[ 75%] Built target "new_package_generate_messages_py"
Scanning dependencies of target std_msgs_generate_messages_cpp
[ 100%] Generating C++ code from SRV "new_package/WordCount.srv"
[ 100%] Built target std_msgs_generate_messages_cpp
Scanning dependencies of target "new_package_generate_messages_cpp"
[ 100%] Built target "new_package_generate_messages_cpp"
Scanning dependencies of target std_msgs_generate_messages_lisp
[ 100%] Built target std_msgs_generate_messages_lisp
Scanning dependencies of target "new_package_generate_messages_lisp"
[ 100%] Built target "new_package_generate_messages_lisp"
alex@alex-VirtualBox:~/catkin_ws$
```

## Implementar un servicio

```
#!/usr/bin/env python
import rospy
from new_package.srv import WordCount, WordCountResponse

def count_words(request):
    return WordCountResponse(len(request.words.split()))

rospy.init_node('service_server')
server = rospy.Service('word_count', WordCount, count_words)
rospy.spin()
```

Importar los servicios definidos

Función callback que define el servicio

Método que separa una frase en un arreglo de palabras que la contiene

## Implementar un servicio

```
#!/usr/bin/env python
import rospy
from new_package.srv import WordCount, WordCountResponse

def count_words(request):
    return WordCountResponse(len(request.words.split()))

rospy.init_node('service_server')
server = rospy.Service('word_count', WordCount, count_words)
rospy.spin()
```

Función callback que define el servicio

Cuenta el número de elementos en el arreglo

## Implementar un servicio

```
#!/usr/bin/env python
import rospy
from new_package.srv import WordCount, WordCountResponse

def count_words(request):
    return WordCountResponse(len(request.words.split()))

rospy.init_node('service_server')           ← Iniciar el nodo
server = rospy.Service('word_count', WordCount, count_words)
rospy.spin()

Especificar: Nombre del servicio, Archivo
donde están las variables y el nombre de
la función "callback".
```

## Implementar un servicio

En el caso de que haya un solo argumento de devolución para el servicio, simplemente puede devolver ese valor:

```
def count_words(request):
    return len(request.words.split())
```

Si hay múltiples argumentos que devolver, puede devolver una tupla o una lista. Esto funciona incluso si sólo hay un valor de retorno:

```
def count_words(request):
    return [len(request.words.split())]
```

Otras formas  
de retornar  
valores

## Implementar un servicio

Primero, poner en marcha el servicio

```
alex@alex-VirtualBox:~/catkin_ws/src/new_package/src$ roscd new_package/
alex@alex-VirtualBox:~/catkin_ws/src/new_package$ cd src
alex@alex-VirtualBox:~/catkin_ws/src/new_package/src$ chmod u+x service_server.py
alex@alex-VirtualBox:~/catkin_ws/src/new_package/src$ rosrun new_package service_server.py
```

Lista de servicios disponibles

```
alex@alex-VirtualBox:~$ rosservice list
/rossout/get_loggers
/rossout/set_logger_level
/service_server/get_loggers
/service_server/set_logger_level
/topic_publisher/get_loggers
/topic_publisher/set_logger_level
/topic_subscriber/get_loggers
/topic_subscriber/set_logger_level
/word_count
alex@alex-VirtualBox:~$
```

Verificar que  
todo va bien

Información del servicio creado

```
alex@alex-VirtualBox:~$ rosservice info word_count
Node: /service_server
URI: rosrpc://alex-VirtualBox:42666
Type: new_package/WordCount
Args: words
alex@alex-VirtualBox:~$
```

## Implementar un servicio

Otras formas  
de retornar  
valores

## Implementar un servicio

También se puede devolver como diccionario, donde las claves son los nombres de los argumentos (dados como Strings):

```
def count_words(request):
    return {'count': len(request.words.split())}
```

En ambos casos, el código de llamada de servicio subyacente en ROS traducirá estos tipos de devolución en un objeto WordCountResponse y lo devolverá al nodo que lo llama, al igual que en el código de ejemplo inicial.

## ¿Cómo usar un servicio?

- La forma más sencilla es llamando usando el comando rosservice

```
alex@alex-VirtualBox:~$ rosservice call word_count 'Estoy aprendiendo ROS'
count: 3
alex@alex-VirtualBox:~$
```

- Pero esto no es de utilidad, la idea es usar un servicio dentro de algún programa.

Ejemplo:

## ¿Cómo usar un servicio?

```
#!/usr/bin/env python
import sys
import rospy
from new_package.srv import WordCount

rospy.init_node('service_client')
rospy.wait_for_service('word_count')

word_counter = rospy.ServiceProxy('word_count', WordCount)
words = ''.join(sys.argv[1:])
word_count = word_counter(words)

print words, '>', word_count.count
```

Se instancia la conexión con el proxy del servicio y se define en una variable

Se especifica el nombre del servicio y el tipo de dato que usa el servicio

Guarda la frase que se escribe al momento de iniciar el programa

## ¿Cómo usar un servicio?

```
#!/usr/bin/env python
import sys
import rospy
from new_package.srv import WordCount

Importar los servicios definidos

crear el nodo

rospy.init_node('service_client') Se espera a que el servicio esté disponible
rospy.wait_for_service('word_count')

word_counter = rospy.ServiceProxy('word_count', WordCount)
words = ''.join(sys.argv[1:])
word_count = word_counter(words)

En caso el servicio no este disponible, el programa falla

print words, '>', word_count.count
```

## ¿Cómo usar un servicio?

## ¿Cómo usar un servicio?

```
#!/usr/bin/env python
import sys
import rospy
from new_package.srv import WordCount

rospy.init_node('service_client')
rospy.wait_for_service('word_count')

word_counter = rospy.ServiceProxy('word_count', WordCount)
words = ''.join(sys.argv[1:])
word_count = word_counter(words)

Se aplica el servicio para la frase ingresada

print words, '>', word_count.count
```

Se imprime la frase y el resultado del servicio

## Usar el Servicio

- Dar los permisos al programa creado:

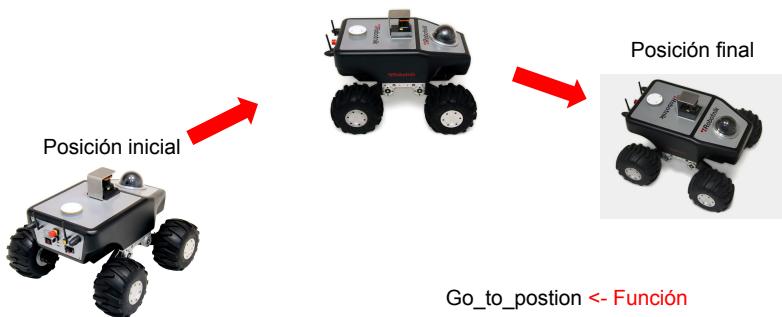
```
alex@alex-VirtualBox:~/catkin_ws/src/new_package/src
alex@alex-VirtualBox:~$ roscd new_package
alex@alex-VirtualBox:~/catkin_ws/src/new_package$ cd src
alex@alex-VirtualBox:~/catkin_ws/src/new_package/src$ chmod u+x service_client.py
```

- Ejecutar el programa de la siguiente forma:

```
alex@alex-VirtualBox:~$ rosrun new_package service_client.py 'Estoy aprendiendo ROS'
Estoy aprendiendo ROS -> 3
alex@alex-VirtualBox:~$
```

## Acciones

¿Qué es?

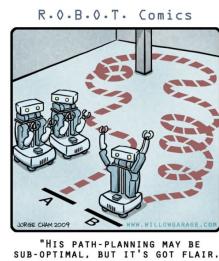


## Acciones

Un servicio:



Aunque los servicios son prácticos para las interacciones simples de hacer/establecer, como consultar el estado y administrar la configuración, no funcionan bien cuando se necesita iniciar una tarea de larga ejecución.



## Acciones

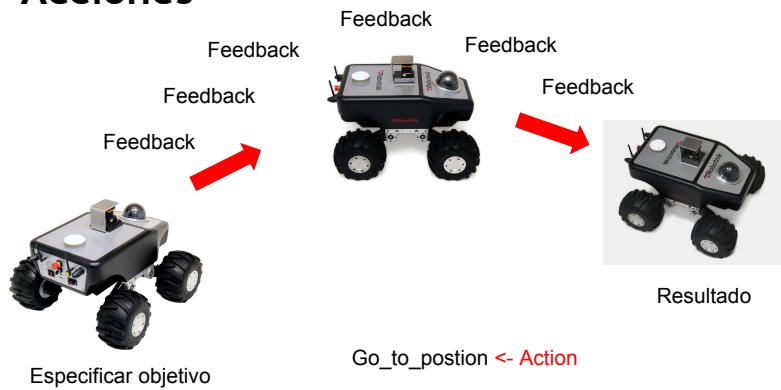
Una acción es esencialmente un protocolo de nivel superior que especifica cómo un conjunto de temas principales:

- Objetivo.
- Resultado.
- Retroalimentación.

Estos deben utilizarse en combinación.

-> Las acciones requieren de mayor trabajo para ser implementadas, pero estas proveen mayor poder y flexibilidad.

## Acciones



## Acciones

Ejemplo:

Temporizador, se establece un tiempo de espera. Se cuenta el tiempo desde que inició el programa hasta terminar el tiempo de espera establecido.



## Definir una Acción

El primero paso es definir un archivo tipo acción como se hizo con el servicio.

Dentro de esta se crea el archivo "Timer.action"

Crear la carpeta "action" donde se va a guardar las variables para este

Objetivo

Resultado

Feedback

```
alex@alex-VirtualBox:~/catkin_ws/src/new_package$ roscd new_package
alex@alex-VirtualBox:~/catkin_ws/src/new_package$ mkdir action
alex@alex-VirtualBox:~/catkin_ws/src/new_package$ cd action/
alex@alex-VirtualBox:~/catkin_ws/src/new_package/action$ gedit Timer.action
```

```
duration time_to_wait
...
duration time_elapsed
uint32 updates_sent
...
duration time_elapsed
duration time_remaining
```

## Definir una Acción

- Definición de cada una de las variables de la acción a implementar.

```
duration time_to_wait
...
duration time_elapsed
uint32 updates_sent
...
duration time_elapsed
duration time_remaining
```

Tiempo que hay para esperar.

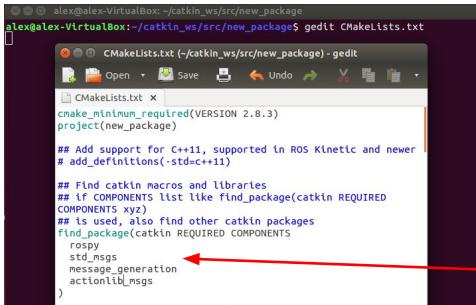
...

Tiempo que se espera  
Cuantas actualizaciones se dio durante la tarea

...

Tiempo transcurrido desde el inicio de la acción  
Tiempo que queda para terminar la tarea

## Definir una Acción

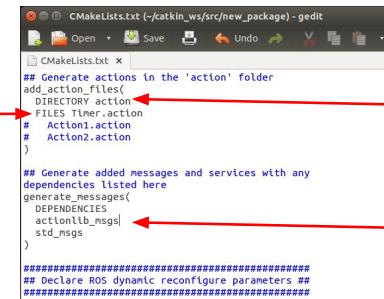


```
alex@alex-VirtualBox: ~/catkin_ws/src/new_package$ gedit CMakeLists.txt
...
# Add support for C++11, supported in ROS Kinetic and newer
# add_definitions(-std=c++11)

## Find catkin macros and libraries
## If COMPONENTS list like find_package(catkin REQUIRED
COMPONENTS xyz)
## is used, also find other catkin packages
find_package(catkin REQUIRED COMPONENTS
  rospy
  std_msgs
  message_generation
  actionlib_msgs
)
```

Agregar el paquete que se usa para las acciones

## Definir una Acción



```
...
## Generate actions in the 'action' folder
add_action_files(
  DIRECTORY action
  FILES Timer.action
  # Action1.action
  # Action2.action
)

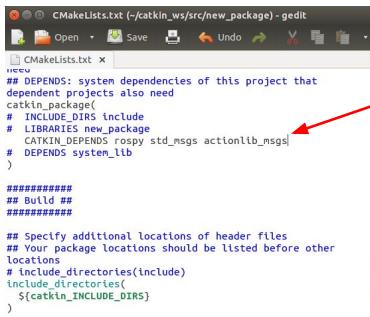
## Generate added messages and services with any
## dependencies listed here
generate_messages(
  DEPENDENCIES
    actionlib_msgs
  std_msgs
)

#####
## Declare ROS dynamic reconfigure parameters ##
#####
```

Indicar la carpeta donde se ubica el archivo con extensión ".action"

Se debe generar mensajes especiales para las acciones

## Definir una Acción



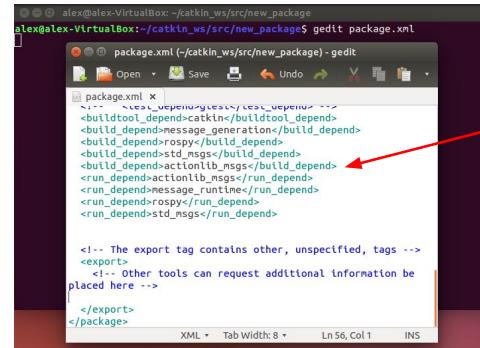
```
...
## DEPENDS: system dependencies of this project that
## dependent objects also need
catkin_package(
  INCLUDE_DIRS include
  # LIBRARIES new_package
  CATKIN_DEPENDS rospy std_msgs actionlib_msgs
  # DEPENDS system lib
  ##

  #####
  ## Build ##
  #####
  ##

  ## Specify additional locations of header files
  ## Your package locations should be listed before other
  ## locations
  # include_directories(include)
  include_directories(
    ${catkin_INCLUDE_DIRS}
  )
)
```

Para terminar, se agrega la dependencias para catkin

## Definir una Acción



```
...
<build_depend>catkin</build_depend>
<build_depend>message_generation</build_depend>
<build_depend>rospy</build_depend>
<build_depend>std_msgs</build_depend>
<build_depend>actionlib_msgs</build_depend>
<run_depend>catkin</run_depend>
<run_depend>message_runtime</run_depend>
<run_depend>rospy</run_depend>
<run_depend>std_msgs</run_depend>

<!-- The export tag contains other, unspecified, tags -->
<export>
  <!-- Other tools can request additional information be
  placed here -->
</export>
</package>
```

Agregar las dependencias que no fueron añadidas cuando se creó el paquete

## Definir una Acción

```
alex@alex-VirtualBox:~/catkin_ws$ catkin_make
Base path: /home/alex/catkin_ws
Source space: /home/alex/catkin_ws/src
Build space: /home/alex/catkin_ws/build
Devel space: /home/alex/catkin_ws/devel
Install space: /home/alex/catkin_ws/install
Using existing command: "make cmake_check_build_system" in "/home/alex/catkin_ws/builtd"
...
Using CATKIN_DEVEL_PREFIX: /home/alex/catkin_ws/devel
Using CMAKE_PREFIX_PATH: /home/alex/catkin_ws/devel:/opt/ros/indigo
This workspace contains:
  - No package dependencies
  - Using PYTHON_EXECUTABLE: /usr/bin/python
  - Using Debain Python package layout
  - Using envpy: /usr/bin/envpy
  - Using build_type: "catkin"
  - Call enable_testing(): ON
  - Using CATKIN_TEST_RESULTS_DIR: /home/alex/catkin_ws/build/test_results
  - Found gtest sources under "/usr/src/gtest": gtests will be built
  - Using nose tests: /usr/bin/nosetests-2.7
  - catkin 0.6.18
  - BUILD_SHARED_LIBS is on
...
alex@alex-VirtualBox:~/catkin_ws$
```

```
alex@alex-VirtualBox:~/catkin_ws$ 
...
traversing 2 packages in topological order:
- new_package
- sensactu
...
*** processing catkin package: "new_package"
...
Using these message generators: genpy;genlisp;genpy
- Generating .msg files for action new_package/Timer /home/alex/catkin_ws/src/n
ew_package/TimerAction.msg
- New package: 7 messages, 1 services
  *** processing catkin package: "sensactu"
  ...
  Using these message generators: genpy;genlisp;genpy
  sensactu: 0 messages, 1 services
- Configuring done
- Generating done
- Creating deb
- Build files have been written to: /home/alex/catkin_ws/build
...
alex@alex-VirtualBox:~/catkin_ws$ make -j1 -l1 << "/home/alex/catkin_ws/build"
...
alex@alex-VirtualBox:~/catkin_ws$ 
Scanning dependencies of target _new_package_generate_messages_check_deps_TimerResult
result
[ 0%] Built target _new_package_generate_messages_check_deps_TimerResult
alex@alex-VirtualBox:~/catkin_ws$
```

## Definir una Acción

```
alex@alex-VirtualBox:~/catkin_ws$ 
[ 60%] Generating C++ code from new_package/WordCount.srv
[ 60%] Built target new_package_generate_messages_cpp
[ 60%] Built target std_msgs_generate_messages_lisp
Scanning dependencies of target actionlib_msgs_generate_messages_lisp
[ 60%] Built target actionlib_msgs_generate_messages_lisp
[ 63%] Generating Lisp code from new_package/TimerActionGoal.msg
[ 66%] Generating Lisp code from new_package/TimerActionResult.msg
[ 73%] Generating Lisp code from new_package/TimerActionResult.msg
[ 76%] Generating Lisp code from new_package/TimerFeedback.msg
[ 80%] Generating Lisp code from new_package/TimerActionFeedback.msg
[ 83%] Generating Lisp code from new_package/WordCount.srv
[ 86%] Generating Lisp code from new_package/WordCount.srv
[ 86%] Built target new_package_generate_messages_lisp
[ 86%] Built target new_package_generate_messages_lisp
[ 86%] Built target geometry_msgs_generate_messages_lisp
[ 86%] Built target sensactu_generate_messages_check_deps_SensorFake
[ 90%] Built target sensactu_generate_messages_lisp
[ 90%] Built target geometry_msgs_generate_messages_lisp
[ 93%] Built target sensactu_generate_messages_cpp
[ 93%] Built target geometry_msgs_generate_messages_py
[100%] Built target sensactu_generate_messages_py
[100%] Built target sensactu_generate_messages
alex@alex-VirtualBox:~/catkin_ws$
```

Catkin genera algunas mensajes-definiciones. Estas son implementadas en el código de la acción.

## Implementar un servidor de la acción

```
#!/usr/bin/env python
import rospy
import time
import actionlib
from new_package.msg import TimerAction, TimerGoal, TimerResult

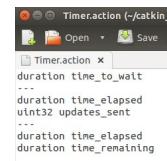
def do_timer(goal):
    start_time = time.time()
    time.sleep(goal.time_to_wait.to_sec())
    result = TimerResult()
    result.time_elapsed = rospy.Duration.from_sec(time.time() - start_time)
    result.updates_sent = 0
    server.set_succeeded(result)

rospy.init_node('timer_action_server')
server = actionlib.SimpleActionServer('timer', TimerAction, do_timer, False)
server.start()
rospy.spin()
```

Agregamos la librería que trabaja con períodos de tiempo como segundos, días, años, etc

Importar el paquete actionlib que provee la clase SimpleActionServer

Se importa de los mensajes generados por catkin los mensajes tipo action, goal, result respectivamente



## Implementar un servidor de la acción

```
#!/usr/bin/env python
import rospy
import time
import actionlib
from new_package.msg import TimerAction, TimerGoal, TimerResult

def do_timer(goal):
    start_time = time.time()
    time.sleep(goal.time_to_wait.to_sec())
    result = TimerResult()
    result.time_elapsed = rospy.Duration.from_sec(time.time() - start_time)
    result.updates_sent = 0
    server.set_succeeded(result)

rospy.init_node('timer_action_server')
server = actionlib.SimpleActionServer('timer', TimerAction, do_timer, False)
server.start()
rospy.spin()
```

Función Goal, esta función es llamada cuando se recibe un nuevo objetivo

Definimos el inicio

Se espera el tiempo establecido para el "timer"

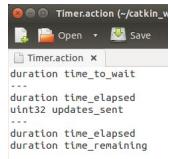
Se define cuánto duró la acción

## Implementar un servidor de la acción

```
#!/usr/bin/env python
import rospy
import time
import actionlib
from new_package.msg import TimerAction, TimerGoal, TimerResult

def do_timer(goal):
    start_time = time.time()
    time.sleep(goal.time_to_wait.to_sec())
    result = TimerResult()
    result.time_elapsed = rospy.Duration.from_sec(time.time() - start_time)
    result.updates_sent = 0
    server.set_succeeded(result)

rospy.init_node('timer_action_server')
server = actionlib.SimpleActionServer('timer', TimerAction, do_timer, False)
server.start()
rospy.spin()
```



En este ejemplo no se implementara el "feedback"

Definir el resultado de la acción

## Verificar la acción

```
alex@alex-VirtualBox:~$ rosrun new_package simple_action_server.py
[...]
alex@alex-VirtualBox:~$ rostopic list
/rossout
/rossout_agg
/timer/cancel
/timer/feedback
/timer/goal
/timer/result
/timer/status
alex@alex-VirtualBox:~$
```

```
alex@alex-VirtualBox:~$ rostopic info /timer/goal
Type: new_package/TimerActionGoal
Publishers: None
Subscribers:
* /timer_action_server (http://alex-VirtualBox:36121/)

alex@alex-VirtualBox:~$
```

## Implementar un servidor de la acción

```
#!/usr/bin/env python
import rospy
import time
import actionlib
from new_package.msg import TimerAction, TimerGoal, TimerResult
```

```
def do_timer(goal):
    start_time = time.time()
    time.sleep(goal.time_to_wait.to_sec())
    result = TimerResult()
    result.time_elapsed = rospy.Duration.from_sec(time.time() - start_time)
    result.updates_sent = 0
    server.set_succeeded(result)
```

```
rospy.init_node('timer_action_server')
server = actionlib.SimpleActionServer('timer', TimerAction, do_timer, False)
server.start()
rospy.spin()
```

Nombre de la acción

Tipo de acción

Función callback cuando se ejecuta la acción

Se define un servidor de tipo de acción simple.

Define si debe iniciar el servidor automáticamente

## Verificar la acción

```
alex@alex-VirtualBox:~$ rosmsg show TimerActionGoal
[new_package/TimerActionGoal]:
std_msgs/Header header
duration time_to_wait
actionlib_msgs/GoalID goal_id
time stamp
string frame_id
actionlib_msgs/GoalID goal_id
time stamp
string id
new_package/TimerGoal goal
duration time_to_wait
alex@alex-VirtualBox:~$
```

```
alex@alex-VirtualBox:~$ rosmsg show TimerGoal
[new_package/TimerGoal]:
duration time_to_wait
alex@alex-VirtualBox:~$
```

## Implementado el cliente de la acción

```
#!/usr/bin/env python
import rospy
import actionlib
from new_package.msg import TimerAction, TimerGoal, TimerResult

rospy.init_node('timer_action_client')
client = actionlib.SimpleActionClient('timer', TimerAction)
client.wait_for_server()           ← Importar el paquete actionlib que provee la clase SimpleActionServer
goal = TimeGoal()                ← Se importa de los mensajes generados por catkin los mensajes tipo action, goal, result respectivamente
goal.time_to_wait = rospy.Duration.from_sec(5.0)
client.send_goal(goal)
client.wait_for_result()
print('Timer elapsed: %f'%(client.get_result().time_elapsed.to_sec()))
```

Nombre de la acción Variables a usar

Se espera a que inicie la conexión con el servidor

## Implementado el cliente de la acción

```
#!/usr/bin/env python
import rospy
import actionlib
from new_package.msg import TimerAction, TimerGoal, TimerResult

rospy.init_node('timer_action_client')
client = actionlib.SimpleActionClient('timer', TimerAction)
client.wait_for_server()           ← Aquí se establece el tiempo que debe esperar el "timer"
goal = TimeGoal()
goal.time_to_wait = rospy.Duration.from_sec(5.0)          ← Se envía el resultado esperado la servidor y se espera que este lo reciba
client.send_goal(goal)
client.wait_for_result()
print('Timer elapsed: %f'%(client.get_result().time_elapsed.to_sec()))
```

Diferencia entre servidor y acción, síncrona y asíncrona respectivamente

Se imprime el resultado final

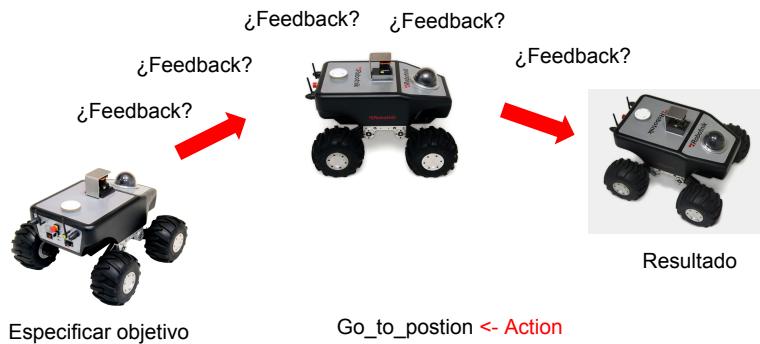
## Ejecutar la acción

```
alex@alex-VirtualBox:~/catkin_ws/src/new_package/src$ chmod u+x simple_action_client.py
alex@alex-VirtualBox:~/catkin_ws/src/new_package/src$ ./simple_action_client.py
```

```
alex@alex-VirtualBox:~$ rosrun new_package simple_action_server.py
alex@alex-VirtualBox:~$ rosrun new_package simple_action_client.py
Time elapsed: 5.000556
alex@alex-VirtualBox:~$
```

## Usando una acción más elaborada



## Servidor de acción más elaborado

```
#!/usr/bin/env python
import rospy
import time
import actionlib
from new_package.msg import TimerAction, TimerGoal, TimerResult,
TimerFeedback

def do_timer(goal):
    start_time = time.time()
    update_count = 0

    if goal.time_to_wait.to_sec() > 60.0:
        result = TimeGoal()
        result.time_elapsed = rospy.Duration.from_sec(time.time() - start_time)
        result.updates_sent = update_count
        server.set_aborted(result, "Timer aborted due to too-long wait")
        return

    Se importa de los mensajes generados por catkin los mensajes tipo action, goal, result respectivamente

    Función callback para la acción definida.

    Inicio del contador y las actualización de los estados
    Restricción que el contador no puede ser mayor a 60 segundos

    If goal.time_to_wait.to_sec() > 60.0:
        result = TimeGoal()
        result.time_elapsed = rospy.Duration.from_sec(time.time() - start_time)
        result.updates_sent = update_count
        server.set_aborted(result, "Timer aborted due to too-long wait")
        return
```

Se importa de los mensajes generados por catkin los mensajes tipo action, goal, result respectivamente

Función callback para la acción definida.

Inicio del contador y las actualización de los estados

Restricción que el contador no puede ser mayor a 60 segundos

If goal.time\_to\_wait.to\_sec() > 60.0:  
 result = TimeGoal()  
 result.time\_elapsed = rospy.Duration.from\_sec(time.time() - start\_time)  
 result.updates\_sent = update\_count  
 server.set\_aborted(result, "Timer aborted due to too-long wait")  
 return

## Servidor de acción más elaborado

```
#!/usr/bin/env python
import rospy
import time
import actionlib
from new_package.msg import TimerAction, TimerGoal, TimerResult,
TimerFeedback
```

```
def do_timer(goal):
    start_time = time.time()
    update_count = 0
```

```
If goal.time_to_wait.to_sec() > 60.0:
    result = TimeGoal()
    result.time_elapsed = rospy.Duration.from_sec(time.time() - start_time)
    result.updates_sent = update_count
    server.set_aborted(result, "Timer aborted due to too-long wait")
    return
```

```
Timer.action x
duration time_to_wait
duration time_elapsed
uint32 updates_sent
...
duration time_elapsed
duration time_remaining
```

Informa del error y el estado de la acción

## Servidor de acción más elaborado

```
while (time.time() - start_time) < goal.time_to_wait.to_sec():
    If server.is_preempt_request():
        result = TimeResult()
        result.time_elapsed = rospy.Duration.from_sec(time.time() - start_time)
        result.updates_sent = update_count
        server.set_preempted(result, "Timer preempted")
        return

    feedback = TimeFeedback()
    feedback.time_elapsed = rospy.Duration.from_sec(time.time() - start_time)
    feedback.time_remaining = goal.time_to_wait - feedback.time_elapsed
    server.publish_feedback(feedback)
    update_count += 1

    Se ejecuta antes de que se haya cumplido el tiempo establecido como objetivo

    Función en caso se cancele la acción

    Definir las variables al cancelar la acción

    Definir la variable feedback

    time.sleep(1.0)
```

Se ejecuta antes de que se haya cumplido el tiempo establecido como objetivo

Función en caso se cancele la acción

Definir las variables al cancelar la acción

Definir la variable feedback

time.sleep(1.0)

## Servidor de acción más elaborado

```
while (time.time() - start_time) < goal.time_to_wait.to_sec():
```

```
If server.is_preempt_request():
    result = TimeResult()
    result.time_elapsed = rospy.Duration.from_sec(time.time() - start_time)
    result.updates_sent = update_count
    server.set_preempted(result, "Timer preempted")
    return
```

```
feedback = TimeFeedback()
feedback.time_elapsed = rospy.Duration.from_sec(time.time() - start_time)
feedback.time_remaining = goal.time_to_wait - feedback.time_elapsed
server.publish_feedback(feedback)
update_count += 1
```

```
Timer.action x
duration time_to_wait
...
duration time_elapsed
uint32 updates_sent
...
duration time_elapsed
duration time_remaining
```

Aumentar la cuenta

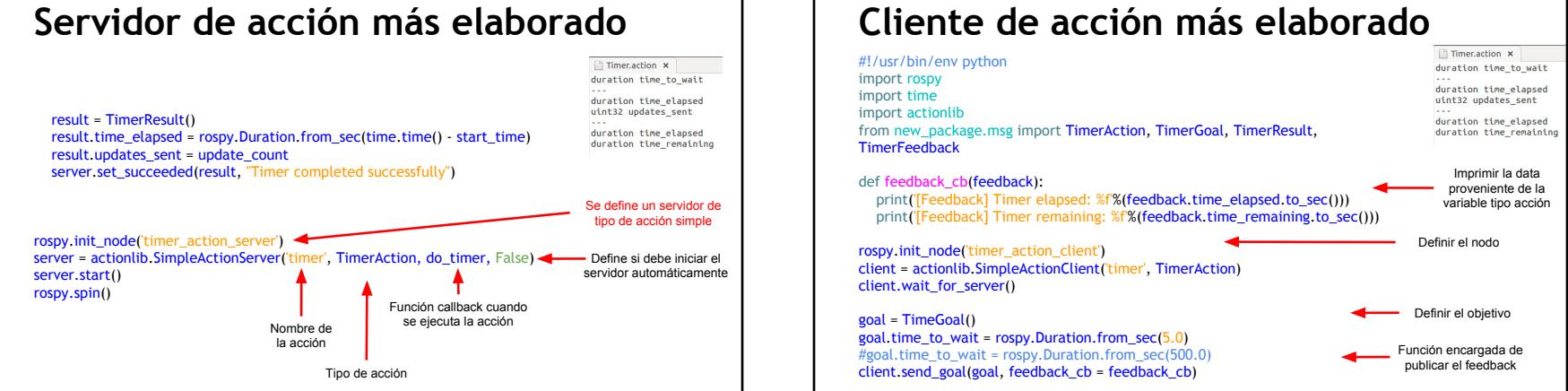
## Servidor de acción más elaborado

```
result = TimerResult()
result.time_elapsed = rospy.Duration.from_sec(time.time() - start_time)
result.updates_sent = update_count
server.set_succeeded(result, "Timer completed successfully")
```

```
rospy.init_node('timer_action_server')
server = actionlib.SimpleActionServer('timer', TimerAction, do_timer, False)
server.start()
rospy.spin()
```

Nombre de la acción  
Tipo de acción  
Función callback cuando se ejecuta la acción  
Define si debe iniciar el servidor automáticamente

```
Timer.action x
duration time_to_wait
...
duration time_elapsed
uint32 updates_sent
...
duration time_elapsed
duration time_remaining
```



## Cliente de acción más elaborado

```
#!/usr/bin/env python
import rospy
import time
import actionlib
from new_package.msg import TimerAction, TimerGoal, TimerResult,
TimerFeedback
```

```
def feedback_cb(feedback):
    print("[Feedback] Timer elapsed: %f%(feedback.time_elapsed.to_sec())
    print("[Feedback] Timer remaining: %f%(feedback.time_remaining.to_sec())
rospy.init_node('timer_action_client')
client = actionlib.SimpleActionClient('timer', TimerAction)
client.wait_for_server()
```

Imprimir la data proveniente de la variable tipo acción  
Definir el nodo  
Definir el objetivo  
Función encargada de publicar el feedback

## Cliente de acción más elaborado

```
#!/usr/bin/env python
import rospy
import time
import actionlib
from new_package.msg import TimerAction, TimerGoal, TimerResult,
TimerFeedback

def feedback_cb(feedback):
    print("[Feedback] Timer elapsed: %f%(feedback.time_elapsed.to_sec())
    print("[Feedback] Timer remaining: %f%(feedback.time_remaining.to_sec())
rospy.init_node('timer_action_client')
client = actionlib.SimpleActionClient('timer', TimerAction)
client.wait_for_server()

goal = TimeGoal()
goal.time_to_wait = rospy.Duration.from_sec(5.0)
#goal.time_to_wait = rospy.Duration.from_sec(500.0)
client.send_goal(goal, feedback_cb = feedback_cb)
```

La acción fue definida para no contar con tiempos largos

```
Timer.action x
duration time_to_wait
...
duration time_elapsed
uint32 updates_sent
...
duration time_elapsed
duration time_remaining
```

## Cliente de acción más elaborado

```
#Uncomment these lines to test goal preemption, cancel after 3 seconds.
#time.sleep(3.0)
#client.cancel_goal()
```

```
client.wait_for_result()
print("[Result] State: %d %(client.get_state())
print("[Result] Status: %s %(client.get_goal_status_text())
print("[Result] Time elapsed: %f %(client.get_result().time_elapsed.to_sec())
print("[Result] Updates sent: %d %(client.get_result().updates_sent))
```

Descomentar para cancelar la acción 3 segundos después  
Espera por resultado de la acción  
Imprimir resultados al final de la acción

```
Timer.action x
duration time_to_wait
...
duration time_elapsed
uint32 updates_sent
...
duration time_elapsed
duration time_remaining
```

## Ejecutar la acción más elaborada

```
alex@alex-VirtualBox:~$ rosrn new_package fancy_action_server.py
alex@alex-VirtualBox:~$ rosrn new_package fancy_action_client.py
[Feedback] Time elapsed: 0.00028
[Feedback] Time remaining: 4.999972
[Feedback] Time elapsed: 1.001376
[Feedback] Time remaining: 3.998624
[Feedback] Time elapsed: 2.002634
[Feedback] Time remaining: 1.997766
[Feedback] Time elapsed: 3.003202
[Feedback] Time remaining: 1.995798
[Feedback] Time elapsed: 4.005585
[Feedback] Time remaining: 0.994415
[Result] State: 3
[Result] Status: Timer completed successfully
[Result] Time elapsed: 5.006922
[Result] Updates sent: 5
alex@alex-VirtualBox:~$
```

## Ejecutar la acción más elaborada

```
alex@alex-VirtualBox:~$ rosrn new_package fancy_action_server.py
alex@alex-VirtualBox:~$ rosrn new_package fancy_action_client.py
[Feedback] Time elapsed: 0.000013
[Feedback] Time remaining: 2.999987
[Feedback] Time elapsed: 1.001297
[Feedback] Time remaining: 1.998703
[Feedback] Time elapsed: 2.003129
[Feedback] Time remaining: 0.996871
[Result] State: 1
[Result] Status: Timer completed successfully
[Result] Time elapsed: 3.004596
[Result] Updates sent: 3
alex@alex-VirtualBox:~$
```

```
alex@alex-VirtualBox:~$ rosrn new_package fancy_action_server.py
alex@alex-VirtualBox:~$ rosrn new_package fancy_action_client.py
[Result] State: 4
[Result] Status: Timer aborted due to too-long wait
[Result] Time elapsed: 0.000012
[Result] Updates sent: 0
alex@alex-VirtualBox:~$
```

**¡Gracias!**

¡La única pregunta tonta es la que no se hace!