Lecture 08.01 Introducing ROS services

services
server
clients

A ROS *service* is effectively a function one node (the *server*) provides to other nodes (the *clients*).

Box 08.1 *i can haz service?* a script

Pretty much, if we have Node A [server] and Node B [client]:
Node A: "Yo I can do X service [registers a service],"
Node B: "Node A, do X for me plz? [requests service]" and waits
Node A: does X [service occurs]
Node A: sends Node B the result of doing X [server returns values]
Node B: "thnks fam" [I just assume this happens].

synchronous

A key aspect to services is that they are *synchronous*: a client *waits*, doing nothing else, while the server "services" it. So obviously this only works well for tasks that take a limited amount of time, such as:

1. getting a sensor value,
2. setting a parameter, or
3. performing a computation.

08.01.1 An example service type definition

In this section, we develop a custom service type definition `WordCount` in `srv/WordCount.srv` for a service that has as input a string and as output the number of words in that string. We create a new package for this chapter, `my_services`, which shadows the package included with the book, `rico_services`. So use, in your workspace's `src` directory, use `catkin_create_pkg` to create a package, as follows.

```
catkin_create_pkg my_services \
  roscpp rospy message_generation message_runtime
```

The first thing when creating a custom service definition is to create the service definition file.

08.01.1.1 Creating a service definition

From your package root, create it with the following.

```
mkdir srv # traditional directory for service definitions
touch srv/WordCount.srv
```

Now we can edit the contents of `WordCount.srv` to include the following.

```
string words
---
uint32 count
```

Above the delimiter "`---`" are *input field* types and names and below the delimiter are *output field* types and names.

input field
output field

We are now ready to update the build-system.

08.01.1.2 Updating the build-system configuration

The package we're creating in this chapter, `my_services`, was created with a bit of forethought: we included as dependencies in our `catkin_create_pkg` call the packages `message_runtime` and `message_generation`. If we hadn't had such foresight, we would have to make several changes in our package's `package.xml` and `CMakeLists.txt` files before proceeding to create our own message description. As it stands, we still need to make a few changes to them.

How we need to change `package.xml`

Including `message_runtime` and `message_generation` in our `catkin_create_pkg` call yielded the following lines in our `package.xml`, which would otherwise need to be added manually.

```
<build_depend>message_generation</build_depend>
<exec_depend>message_runtime</exec_depend>
```

However, we still need to add `message_runtime` as a `<build_depend>`.

```
<build_depend>message_runtime</build_depend>
```

How we need to change `CMakeLists.txt`

Including `message_runtime` and `message_generation` in our `catkin_create_pkg` call yielded the following lines in our `CMakeLists.txt`, which would otherwise need to be added manually.

As an additional line in the `find_package(...)` block, we would need the following.

```
message_generation
```

The rest of the changes we do need to make manually. The `add_service_files(...)` block needs uncommented and edited to appear as follows.

```
add_service_files(
  FILES
  WordCount.srv
)
```

We have already created the `WordCount.srv` file.

Finally, the `generate_messages(...)` block needs to be uncommented such that it appears as follows.

```
generate_messages(
  DEPENDENCIES
  std_msgs
)
```

Now our package is set up to use the service type `WordCount`—or, it will be once we `catkin_make` our workspace. (Go ahead and do so now.)

08.01.1.3 See it with `rossrv`

The package `rosmsg` (already installed) includes the command `rossrv`, which gives information about services.

```
rossrv
```

```
rossrv is a command-line tool for displaying information about ROS
↪  Service types.
Commands:
  rossrv show     Show service description
  rossrv info     Alias for rossrv show
  rossrv list     List all services
  rossrv md5      Display service md5sum
  rossrv package  List services in a package
  rossrv packages List packages that contain services
```

We could try it on our new service type `WordCount` as follows.

```
rossrv show WordCount
```

```
[my_services/WordCount]:
string words
---
uint32 count

[rico_services/WordCount]:
string words
---
uint32 count
```

So the `WordCount` service type is available in both packages `rico_services` and `my_services`. We have successfully created a service type! In Lecture 08.02, we'll learn to serve and call this service type.