1.4.1

ROS Services – code illustration

Mukunda Bharatheesha

Setting up the service definition files

Let's begin by sourcing our setup files, and navigate to our workspace msgs folder:

```
$ source $HOME/hrwros_ws/devel/setup.bash
$ roscd hrwros_msgs/
```

We can see the msg/ folder, but we just learned services are defined in the srv/ folder. So let's create that, and create our service file:

```
$ mkdir srv
$ cd srv
$ touch ConvertMetresToFeet.srv
```

Setting up the service definition files

Let's begin by sourcing our setup files, and navigate to our workspace msgs folder:

ConvertMetresToFeet.srv

```
float64 distance_metres      # Request message: Distance in (m) to
                               be converted to (ft)
---                          # Demarcation
float64 distance_feet        # Response message: Distance in (ft)
                               after conversion
bool success                 # Response message: Success or failure
                               of conversion
```

Setting up the service definition files

Now, just like we did for messages, we add our new service to the CMakeLists.txt file in the hrwros_msgs/ folder. Note that services have their own section, after messages. Add the name of our new service definition file after the FILES entry. Don't forget the .srv file extension.

Finish by building our service by running the catkin b command. We can check if it worked correctly by using the rossrv command:

```
$ rossrv show hrwros_msgs/ConvertMetresToFeet
$ rossrv show hrwros_msgs/ConvertMetresToFeet -r
```

The terminal should show the request message field, and the response message field.

Inspecting our service server code

Now we've created a new service, we can use it in our code. Let's go to the week 1 script folder, where we already have two template implementations. Let's begin by inspecting the server: metres_to_feet_server.py. Note that the used file is also showed below.

We are importing the newly created service definitions from hrwros_msgs.srv.

Note that we not only have to import the service definition, but also explicitly the request and response message definitions (#L7)!

The service request is processed in the service callback function called process_service_request (#L14).

- We first create an object for the response message type (res), and in the example, do a check in the distance input to check if it is a positive real number (req.distance_metres<0).
- If the sanity check fails, we return a default error value in the response.
- Otherwise, we do the conversion and return both the converted value and a flag that indicates the conversion was successful.
- In either case, we return the response.

Inspecting our service server code

Then, we have the next function for the administrative settings of the service server node, called *metres_to_feet_server* (#L34).

- Here, we create a ROS node for our service by using the *rospy.init_node* function (#L36).
- We create the actual service by giving it a name, a type, and a callback function for processing a service request. This is done by using the *rospy.Service* function (#L39)
- Finally, in the main function (#L46) we start the service server node by calling the function *metres_to_feet_server*.