Lecture 09.01 Introducing ROS actions

actions
action server
asynchronicity
clients
goals
results
feedback

A ROS *action* is effectively a function one node (the *action server*) *asynchronously* provides to other nodes (the *clients*). Note this is just like *service*, but with the asynchronicity of a *topic*. Like a service, an action has a *goal* and a *result*; but unlike a service, an action also provides *feedback* during execution. This makes actions more suitable for goal-oriented tasks that take time, such as:

1. navigating to a location,
2. performing a complex manipulation, or
3. performing a long calculation.

09.01.1 An example action type definition

In this section, we develop a custom action type definition `Timer` in `action/Timer.action` for an action that has as

input a `duration` to wait `time_to_wait`;
output a total actual `duration` waited `time_elapsed` and a total `uint32` number of feedback updates sent `updates_sent`; and
feedback a `duration` waited so far `time_elapsed` and a `duration` left to wait `time_remaining`.

Box 09.1 why a timer though

The `Timer` action is for demonstration purposes only and shouldn't be used to implement timing in a ROS graph. For timing, use `rospy.sleep()`.

We create a new package for this chapter, `my_actions`, which shadows the package included with the book, `rico_actions`. So, in your workspace's `src` directory, use `catkin_create_pkg` to create a package, as follows.

```
catkin_create_pkg my_actions roscpp rospy actionlib_msgs
```

action definition file

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

09.01.1.1 Creating an action definition

From your package root, create it with the following.

```
mkdir action
touch action/Timer.action
```

Now we can edit the contents of `Timer.action` to include the following.

```
# inputs
duration time_to_wait
---
# outputs
duration time_elapsed
uint32 updates_sent
---
# feedback
duration time_elapsed
duration time_remaining
```

Above the first delimiter "`---`" are *input field* types and names; between the delimiters are *output field* types and names; and after the second delimiter are *feedback field* types and names.

input field
output field
feedback field

We are now ready to update the build-system.

09.01.1.2 Updating the build-system configuration

The package we're creating in this chapter, `my_actions`, was created with a bit of forethought: we included as dependencies in our `catkin_create_pkg` call the package `actionlib_msgs` for creating actions. 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 *would have* had to change `package.txt`

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

```
<build_depend>actionlib_msgs</build_depend>
<build_exec_depend>actionlib_msgs</build_exec_depend>
<exec_depend>actionlib_msgs</exec_depend>
```

How we need to change `CMakeLists.txt`

Including `actionlib_msgs` 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.

```
actionlib_msgs
```

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

```
add_action_files(
  DIRECTORY action
  FILES Timer.action
)
```

We have already created the `Timer.action` file.

The `generate_messages(...)` block needs to be uncommented and `actionlib_msgs` added such that it appears as follows.

```
generate_messages(
  DEPENDENCIES
  actionlib_msgs
  std_msgs
)
```

Finally, the `catkin_package` block also needs uncommented and `actionlib_msgs` added such that it appears as follows.

```
catkin_package(
  CATKIN_DEPENDS
  actionlib_msgs
)
```

Now our package is set up to use the action type `Timer`—or, it will be once we `catkin_make` our workspace. (Go ahead and do so now.) As before with services, `catkin_make` will take our action definition and create several message definition `.msg` files. This highlights the fact that an action communicates via services.

We have successfully created an action type! In Lecture 09.02, we'll learn to serve and call this action type.