You're reading the documentation for an older, but still supported, version of ROS 2. For information on the latest version, please have a look at Iron.

Creating an action

Goal: Define an action in a ROS 2 package.

Tutorial level: Intermediate

Time: 5 minutes

Contents

Background

You learned about actions previously in the Understanding actions tutorial. Like the other communication types and their respective interfaces (topics/msg and services/srv), you can also custom-define actions in your packages. This tutorial shows you how to define and build an action that you can use with the action server and action client you will write in the next tutorial.

Prerequisites

You should have ROS 2 and colcon installed.

Set up a workspace and create a package named `action_tutorials_interfaces`:

(Remember to source your ROS 2 installation first.)

Linux | macOS | Windows

```
mkdir -p ros2_ws/src #you can reuse existing workspace with this naming convention
cd ros2_ws/src
ros2 pkg create action_tutorials_interfaces
```

Tasks

1 Defining an action

Actions are defined in `.action` files of the form:

```
# Request
---
# Result
---
# Feedback
```

An action definition is made up of three message definitions separated by `---`.

- A *request* message is sent from an action client to an action server initiating a new goal.
- A *result* message is sent from an action server to an action client when a goal is done.
- *Feedback* messages are periodically sent from an action server to an action client with updates about a goal.

An instance of an action is typically referred to as a *goal*.

Say we want to define a new action "Fibonacci" for computing the Fibonacci sequence.

Create an `action` directory in our ROS 2 package `action_tutorials_interfaces`:

Linux | macOS | Windows

```
cd action_tutorials_interfaces
mkdir action
```

Within the `action` directory, create a file called `Fibonacci.action` with the following contents:

```
int32 order
---
int32[] sequence
---
int32[] partial_sequence
```

The goal request is the `order` of the Fibonacci sequence we want to compute, the result is the final `sequence`, and the feedback is the `partial_sequence` computed so far.

2 Building an action

Before we can use the new Fibonacci action type in our code, we must pass the definition to the rosidl code generation pipeline.

This is accomplished by adding the following lines to our `CMakeLists.txt` before the `ament_package()` line, in the `action_tutorials_interfaces`:

```
find_package(rosidl_default_generators REQUIRED)

rosidl_generate_interfaces(${PROJECT_NAME}
  "action/Fibonacci.action"
)
```

We should also add the required dependencies to our `package.xml`:

```
<buildtool_depend>rosidl_default_generators</buildtool_depend>

<depend>action_msgs</depend>

<member_of_group>rosidl_interface_packages</member_of_group>
```

Note, we need to depend on `action_msgs` since action definitions include additional metadata (e.g. goal IDs).

We should now be able to build the package containing the `Fibonacci` action definition:

```
# Change to the root of the workspace
cd ~/ros2_ws
# Build
colcon build
```

We're done!

By convention, action types will be prefixed by their package name and the word `action`. So when we want to refer to our new action, it will have the full name `action_tutorials_interfaces/action/Fibonacci`.

We can check that our action built successfully with the command line tool:

```
# Source our workspace
# On Windows: call install/setup.bat
. install/setup.bash
# Check that our action definition exists
ros2 interface show action_tutorials_interfaces/action/Fibonacci
```

You should see the Fibonacci action definition printed to the screen.

Summary

In this tutorial, you learned the structure of an action definition. You also learned how to correctly build a new action interface using `CMakeLists.txt` and `package.xml`, and how to verify a successful build.

Next steps

Next, let's utilize your newly defined action interface by creating an action service and client (in Python or C++).

Related content

For more detailed information about ROS actions, please refer to the design article.