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.

Implementing custom interfaces

Goal: Learn more ways to implement custom interfaces in ROS 2.

Tutorial level: Beginner

Time: 15 minutes

Contents

Background

In a previous tutorial, you learned how to create custom msg and srv interfaces.

While best practice is to declare interfaces in dedicated interface packages, sometimes it can be convenient to declare, create and use an interface all in one package.

Recall that interfaces can currently only be defined in CMake packages. It is possible, however, to have Python libraries and nodes in CMake packages (using ament_cmake_python), so you could define interfaces and Python nodes together in one package. We'll use a CMake package and C++ nodes here for the sake of simplicity.

This tutorial will focus on the msg interface type, but the steps here are applicable to all interface types.

Prerequisites

We assume you've reviewed the basics in the Creating custom msg and srv files tutorial before working through this one.

You should have ROS 2 installed, a workspace, and an understanding of creating packages.

As always, don't forget to source ROS 2 in every new terminal you open.

Tasks

1 Create a package

In your workspace `src` directory, create a package `more_interfaces` and make a directory within it for msg files:

```
ros2 pkg create --build-type ament_cmake --license Apache-2.0 more_interfaces
mkdir more_interfaces/msg
```

2 Create a msg file

Inside `more_interfaces/msg`, create a new file `AddressBook.msg`, and paste the following code to create a message meant to carry information about an individual:

```
uint8 PHONE_TYPE_HOME=0
uint8 PHONE_TYPE_WORK=1
uint8 PHONE_TYPE_MOBILE=2

string first_name
string last_name
string phone_number
uint8 phone_type
```

This message is composed of these fields:

- first_name: of type string
- last_name: of type string
- phone_number: of type string

- phone_type: of type uint8, with several named constant values defined

Note that it's possible to set default values for fields within a message definition. See Interfaces for more ways you can customize interfaces.

Next, we need to make sure that the msg file is turned into source code for C++, Python, and other languages.

2.1 Build a msg file

Open `package.xml` and add the following lines:

```
<buildtool_depend>rosidl_default_generators</buildtool_depend>

<exec_depend>rosidl_default_runtime</exec_depend>

<member_of_group>rosidl_interface_packages</member_of_group>
```

Note that at build time, we need `rosidl_default_generators`, while at runtime, we only need `rosidl_default_runtime`.

Open `CMakeLists.txt` and add the following lines:

Find the package that generates message code from msg/srv files:

```
find_package(rosidl_default_generators REQUIRED)
```

Declare the list of messages you want to generate:

```
set(msg_files
  "msg/AddressBook.msg"
)
```

By adding the .msg files manually, we make sure that CMake knows when it has to reconfigure the project after you add other .msg files.

Generate the messages:

```
rosidl_generate_interfaces(${PROJECT_NAME}
  ${msg_files}
)
```

Also make sure you export the message runtime dependency:

```
ament_export_dependencies(rosidl_default_runtime)
```

Now you're ready to generate source files from your msg definition. We'll skip the compile step for now as we'll do it all together below in step 4.

2.2 (Extra) Set multiple interfaces

Note

You can use `set` in `CMakeLists.txt` to neatly list all of your interfaces:

```
set(msg_files
  "msg/Message1.msg"
  "msg/Message2.msg"
  # etc
  )

set(srv_files
  "srv/Service1.srv"
  "srv/Service2.srv"
   # etc
  )
```

And generate all lists at once like so:

```
rosidl_generate_interfaces(${PROJECT_NAME}
  ${msg_files}
  ${srv_files}
)
```

3 Use an interface from the same package

Now we can start writing code that uses this message.

In `more_interfaces/src` create a file called `publish_address_book.cpp` and paste the following code:

```
#include <chrono>
#include <memory>

#include "rclcpp/rclcpp.hpp"
#include "more_interfaces/msg/address_book.hpp"

using namespace std::chrono_literals;

class AddressBookPublisher : public rclcpp::Node
{
public:
  AddressBookPublisher()
  : Node("address_book_publisher")
  {
    address_book_publisher_ =
      this->create_publisher<more_interfaces::msg::AddressBook>("address_book", 10);

    auto publish_msg = [this]() -> void {
        auto message = more_interfaces::msg::AddressBook();

        message.first_name = "John";
        message.last_name = "Doe";
        message.phone_number = "1234567890";
        message.phone_type = message.PHONE_TYPE_MOBILE;

        std::cout << "Publishing Contact\nFirst:" << message.first_name <<
          "  Last:" << message.last_name << std::endl;

        this->address_book_publisher_->publish(message);
      };
    timer_ = this->create_wall_timer(1s, publish_msg);
  }

private:
  rclcpp::Publisher<more_interfaces::msg::AddressBook>::SharedPtr address_book_publisher_;
  rclcpp::TimerBase::SharedPtr timer_;
};


int main(int argc, char * argv[])
{
  rclcpp::init(argc, argv);
  rclcpp::spin(std::make_shared<AddressBookPublisher>());
  rclcpp::shutdown();

  return 0;
}
```

3.1 The code explained

Include the header of our newly created `AddressBook.msg`.

```
#include "more_interfaces/msg/address_book.hpp"
```

Create a node and an `AddressBook` publisher.

```
using namespace std::chrono_literals;

class AddressBookPublisher : public rclcpp::Node
{
public:
  AddressBookPublisher()
  : Node("address_book_publisher")
  {
    address_book_publisher_ =
      this->create_publisher<more_interfaces::msg::AddressBook>("address_book");
```

Create a callback to publish the messages periodically.

```
auto publish_msg = [this]() -> void {
```

Create an `AddressBook` message instance that we will later publish.

```
auto message = more_interfaces::msg::AddressBook();
```

Populate `AddressBook` fields.

```
message.first_name = "John";
message.last_name = "Doe";
message.phone_number = "1234567890";
message.phone_type = message.PHONE_TYPE_MOBILE;
```

Finally send the message periodically.

```
std::cout << "Publishing Contact\nFirst:" << message.first_name <<
  "  Last:" << message.last_name << std::endl;

this->address_book_publisher_->publish(message);
```

Create a 1 second timer to call our `publish_msg` function every second.

```
timer_ = this->create_wall_timer(1s, publish_msg);
```

3.2 Build the publisher

We need to create a new target for this node in the `CMakeLists.txt`:

```
find_package(rclcpp REQUIRED)

add_executable(publish_address_book src/publish_address_book.cpp)
ament_target_dependencies(publish_address_book rclcpp)

install(TARGETS
    publish_address_book
  DESTINATION lib/${PROJECT_NAME})
```

3.3 Link against the interface

In order to use the messages generated in the same package we need to use the following CMake code:

```
rosidl_get_typesupport_target(cpp_typesupport_target
  ${PROJECT_NAME} rosidl_typesupport_cpp)

target_link_libraries(publish_address_book "${cpp_typesupport_target}")
```

This finds the relevant generated C++ code from `AddressBook.msg` and allows your target to link against it.

You may have noticed that this step was not necessary when the interfaces being used were from a different package that was built independently. This CMake code is only required when you want to use interfaces in the same package as the one in which they are defined.

4 Try it out

Return to the root of the workspace to build the package:

Linux | macOS | Windows

```
cd ~/ros2_ws
colcon build --packages-up-to more_interfaces
```

Then source the workspace and run the publisher:

Linux | macOS | Windows

```
source install/local_setup.bash
ros2 run more_interfaces publish_address_book
```

You should see the publisher relaying the msg you defined, including the values you set in `publish_address_book.cpp`.

To confirm the message is being published on the `address_book` topic, open another terminal, source the workspace, and call `topic echo`:

Linux | macOS | Windows

```
source install/setup.bash
ros2 topic echo /address_book
```

We won't create a subscriber in this tutorial, but you can try to write one yourself for practice (use Writing a simple publisher and subscriber (C++) to help).

5 (Extra) Use an existing interface definition

Note

You can use an existing interface definition in a new interface definition. For example, let's say there is a message named `Contact.msg` that belongs to an existing ROS 2 package named `rosidl_tutorials_msgs`. Assume that its definition is identical to our custom-made `AddressBook.msg` interface from earlier.

In that case you could have defined `AddressBook.msg` (an interface in the package *with* your nodes) as type `Contact` (an interface in a *separate* package). You could even define `AddressBook.msg` as an *array* of type `Contact`, like so:

```
rosidl_tutorials_msgs/Contact[] address_book
```

To generate this message you would need to declare a dependency on `Contact.msg's` package, `rosidl_tutorials_msgs`, in `package.xml`:

```
<build_depend>rosidl_tutorials_msgs</build_depend>

<exec_depend>rosidl_tutorials_msgs</exec_depend>
```

And in `CMakeLists.txt`:

```
find_package(rosidl_tutorials_msgs REQUIRED)

rosidl_generate_interfaces(${PROJECT_NAME}
  ${msg_files}
  DEPENDENCIES rosidl_tutorials_msgs
)
```

You would also need to include the header of `Contact.msg` in your publisher node in order to be able to add `contacts` to your `address_book`.

```
#include "rosidl_tutorials_msgs/msg/contact.hpp"
```

You could change the callback to something like this:

```
auto publish_msg = [this]() -> void {
   auto msg = std::make_shared<more_interfaces::msg::AddressBook>();
   {
     rosidl_tutorials_msgs::msg::Contact contact;
     contact.first_name = "John";
     contact.last_name = "Doe";
     contact.phone_number = "1234567890";
     contact.phone_type = message.PHONE_TYPE_MOBILE;
     msg->address_book.push_back(contact);
   }
   {
     rosidl_tutorials_msgs::msg::Contact contact;
     contact.first_name = "Jane";
     contact.last_name = "Doe";
     contact.phone_number = "4254242424";
     contact.phone_type = message.PHONE_TYPE_HOME;
     msg->address_book.push_back(contact);
   }

   std::cout << "Publishing address book:" << std::endl;
   for (auto contact : msg->address_book) {
     std::cout << "First:" << contact.first_name << "  Last:" << contact.last_name <<
       std::endl;
   }

   address_book_publisher_->publish(*msg);
 };
```

Building and running these changes would show the msg defined as expected, as well as the array of msgs defined above.

Summary

In this tutorial, you tried out different field types for defining interfaces, then built an interface in the same package where it's being used.

You also learned how to use another interface as a field type, as well as the `package.xml`, `CMakeLists.txt`, and `#include` statements necessary for utilizing that feature.

Next steps

Next you will create a simple ROS 2 package with a custom parameter that you will learn to set from a launch file. Again, you can choose to write it in either C++ or Python.

Related content

There are several design articles on ROS 2 interfaces and the IDL (interface definition language).