

ROS Crash Course

Class 6

The ROS logo consists of a 3x3 grid of nine blue dots on the left, followed by the letters "ROS" in a large, bold, blue sans-serif font.

Agenda

- Creating custom msg and srv
- Running rosservice from a node
- HW

Creating custom msg and srv

-what does a msg and a srv look like and what are the differences?

Msg (Pose.msg)

float32 x
float32 y
float32 theta

float32 linear_velocity
float32 angular_velocity

Srv (AddTwoInts.srv)

int64 a
int64 b

int64 sum

Creating custom msg and srv

-what does a msg and a srv look like and what are the differences?

Msg (Pose.msg)

float32 x
float32 y
float32 theta

float32 linear_velocity
float32 angular_velocity

geometry_msgs/Twist twist

-all of the variables in a msg are meant for the subscriber
-it is possible to reference other msgs within a msg

Srv (AddTwoInts.srv)

int64 a
int64 b

int64 sum

-the first two are meant for the service to use then the last one is the response back

Creating custom msg and srv

-how to add them to a package

-first cd into your ros package

-then make a new directory called “msg”

-go into your new directory and run

```
$ gedit test.msg
```

-now add

```
Int64 count
```

-to it and save

Creating custom msg and srv

-Open a new shell and go into your package again and make a new directory “srv”
Don't go the new directory yet

-now run the following command

```
$ roscp rospy_tutorials AddTwoInts.srv srv/AddTwoInts.srv
```

-You just copied the add two ints service from another package to yours the format for the command is as follows

```
$ roscp [name of package you're copying from] [file to copy] [path to paste to]
```

Creating custom msg and srv

-Now open up the package.xml for your package

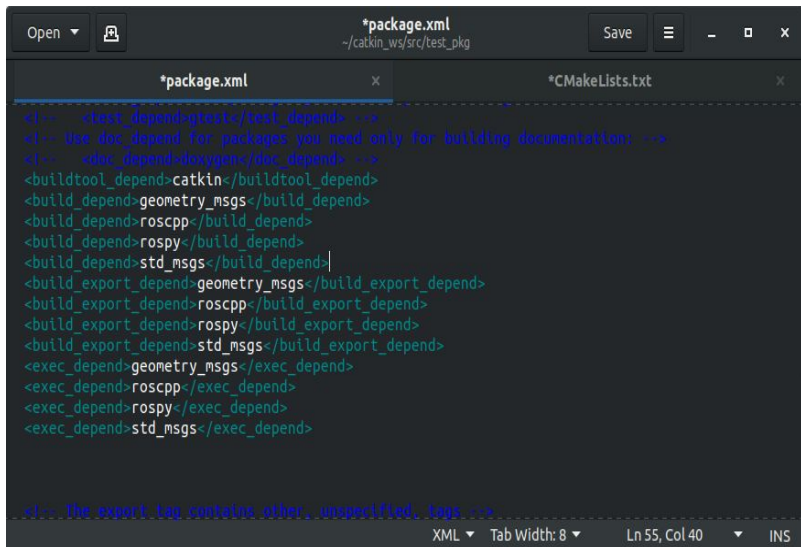
Add these lines where they belong

```
<build_depend>message_generation</build_depend>
```

```
<exec_depend>message_runtime</exec_depend>
```

-now save the file

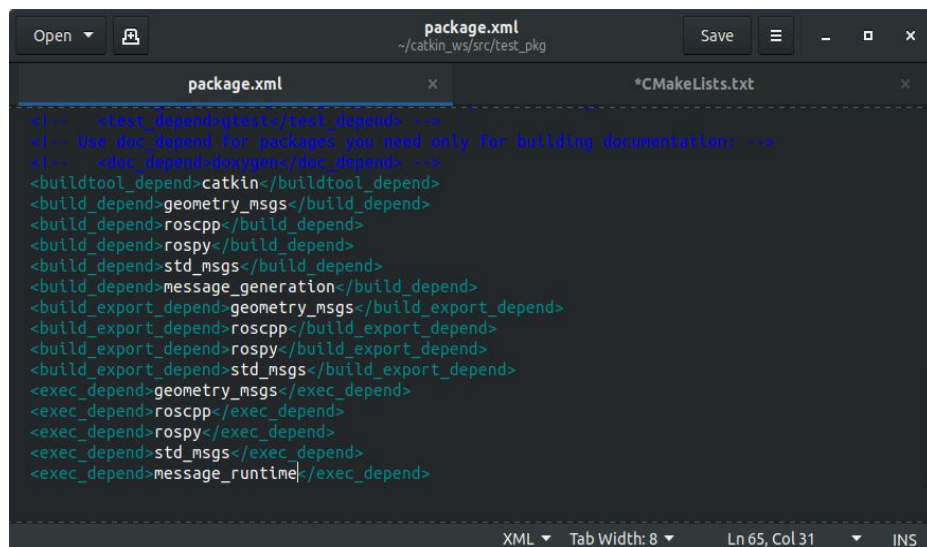
Creating custom msg and srv



```
Open | package.xml | Save | ~ | □ | ×
~/catkin_ws/src/test_pkg

*package.xml | *CMakeLists.txt | ×
<!-- <test_depend>gtest</test_depend> -->
<!-- Use doc_depend for packages you need only for building documentation; -->
<!-- <doc_depend>doxygen</doc_depend> -->
<buildtool_depend>catkin</buildtool_depend>
<build_depend>geometry_msgs</build_depend>
<build_depend>roscpp</build_depend>
<build_depend>rospy</build_depend>
<build_depend>std_msgs</build_depend>
<build_export_depend>geometry_msgs</build_export_depend>
<build_export_depend>roscpp</build_export_depend>
<build_export_depend>rospy</build_export_depend>
<build_export_depend>std_msgs</build_export_depend>
<exec_depend>geometry_msgs</exec_depend>
<exec_depend>roscpp</exec_depend>
<exec_depend>rospy</exec_depend>
<exec_depend>std_msgs</exec_depend>

<!-- The export tag contains other, unspecified, tags -->
XML | Tab Width: 8 | Ln 55, Col 40 | INS
```



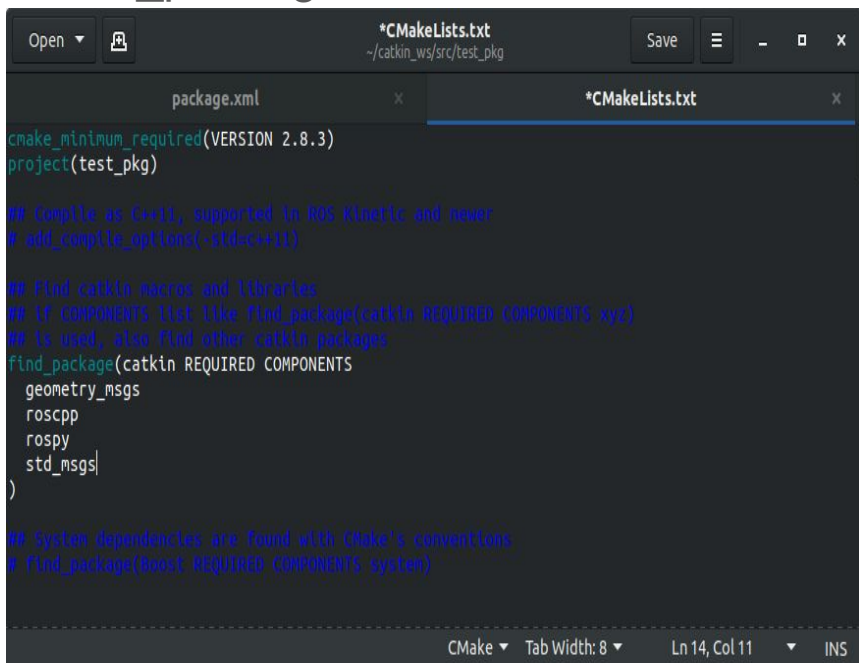
```
Open | package.xml | Save | ~ | □ | ×
~/catkin_ws/src/test_pkg

package.xml | *CMakeLists.txt | ×
<!-- <test_depend>gtest</test_depend> -->
<!-- Use doc_depend for packages you need only for building documentation; -->
<!-- <doc_depend>doxygen</doc_depend> -->
<buildtool_depend>catkin</buildtool_depend>
<build_depend>geometry_msgs</build_depend>
<build_depend>roscpp</build_depend>
<build_depend>rospy</build_depend>
<build_depend>std_msgs</build_depend>
<build_depend>message_generation</build_depend>
<build_export_depend>geometry_msgs</build_export_depend>
<build_export_depend>roscpp</build_export_depend>
<build_export_depend>rospy</build_export_depend>
<build_export_depend>std_msgs</build_export_depend>
<exec_depend>geometry_msgs</exec_depend>
<exec_depend>roscpp</exec_depend>
<exec_depend>rospy</exec_depend>
<exec_depend>std_msgs</exec_depend>
<exec_depend>message_runtime</exec_depend>

XML | Tab Width: 8 | Ln 65, Col 31 | INS
```


Creating custom msg and srv

-Now open your CMakeLists.txt file and add message_generation to your find_package



```
Open [icon] *CMakeLists.txt
~/catkin_ws/src/test_pkg Save [icon] - □ x

package.xml x *CMakeLists.txt x

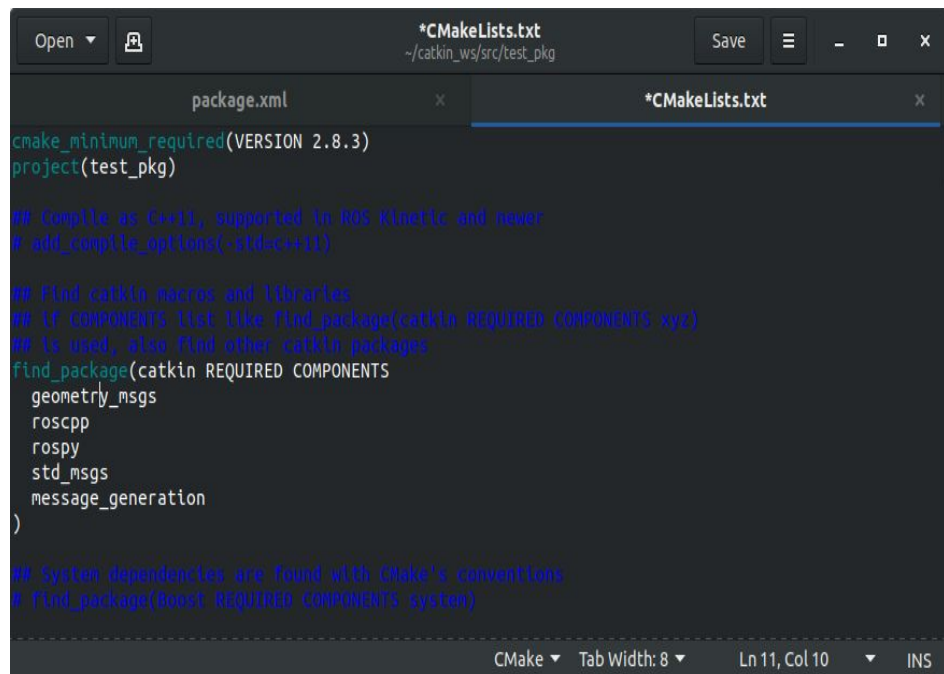
cmake_minimum_required(VERSION 2.8.3)
project(test_pkg)

## Compile as C++11, supported in ROS Kinetic and newer
# add_compile_options(-std=c++11)

## Find catkin macros and libraries
## If COMPONENTS list like find_package(catkin REQUIRED COMPONENTS xyz)
## is used, also find other catkin packages
find_package(catkin REQUIRED COMPONENTS
  geometry_msgs
  roscpp
  rospy
  std_msgs)

## System dependencies are found with CMake's conventions
# find_package(Boost REQUIRED COMPONENTS system)
```

CMake Tab Width: 8 Ln 14, Col 11 INS



```
Open [icon] *CMakeLists.txt
~/catkin_ws/src/test_pkg Save [icon] - □ x

package.xml x *CMakeLists.txt x

cmake_minimum_required(VERSION 2.8.3)
project(test_pkg)

## Compile as C++11, supported in ROS Kinetic and newer
# add_compile_options(-std=c++11)

## Find catkin macros and libraries
## If COMPONENTS list like find_package(catkin REQUIRED COMPONENTS xyz)
## is used, also find other catkin packages
find_package(catkin REQUIRED COMPONENTS
  geometry_msgs
  roscpp
  rospy
  std_msgs
  message_generation)

## System dependencies are found with CMake's conventions
# find_package(Boost REQUIRED COMPONENTS system)
```

CMake Tab Width: 8 Ln 11, Col 10 INS

Creating custom msg and srv

-Find the `catkin_package()` function uncomment it and add `message_runtime` as a `CATKIN_DEPENDS` variable

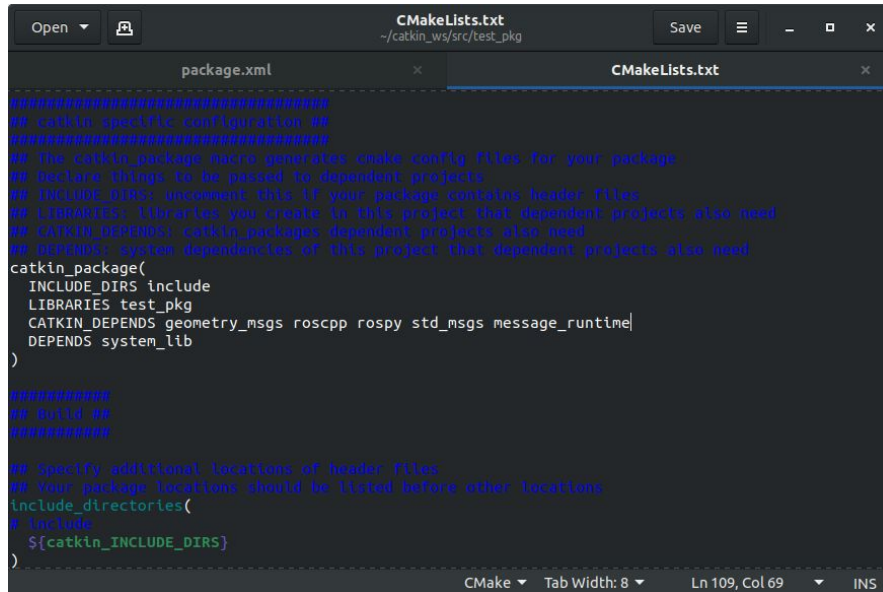


```
#####
## catkin specific configuration ##
#####
## The catkin_package macro generates cmake config files for your package
## Declare things to be passed to dependent projects
## INCLUDE_DIRS: uncomment this if your package contains header files
## LIBRARIES: libraries you create in this project that dependent projects also need
## CATKIN_DEPENDS: catkin_packages dependent projects also need
## DEPENDS: system dependencies of this project that dependent projects also need
catkin_package(
#  INCLUDE_DIRS include
#  LIBRARIES test_pkg
#  CATKIN_DEPENDS geometry_msgs roscpp rospy std_msgs
#  DEPENDS system_lib
)

#####
## Build ##
#####

## Specify additional locations of header files
## Your package locations should be listed before other locations
include_directories(
# include
${catkin_INCLUDE_DIRS}
)
```

CMake Tab Width: 8 Ln 113, Col 12 INS



```
#####
## catkin specific configuration ##
#####
## The catkin_package macro generates cmake config files for your package
## Declare things to be passed to dependent projects
## INCLUDE_DIRS: uncomment this if your package contains header files
## LIBRARIES: libraries you create in this project that dependent projects also need
## CATKIN_DEPENDS: catkin_packages dependent projects also need
## DEPENDS: system dependencies of this project that dependent projects also need
catkin_package(
  INCLUDE_DIRS include
  LIBRARIES test_pkg
  CATKIN_DEPENDS geometry_msgs roscpp rospy std_msgs message_runtime
  DEPENDS system_lib
)

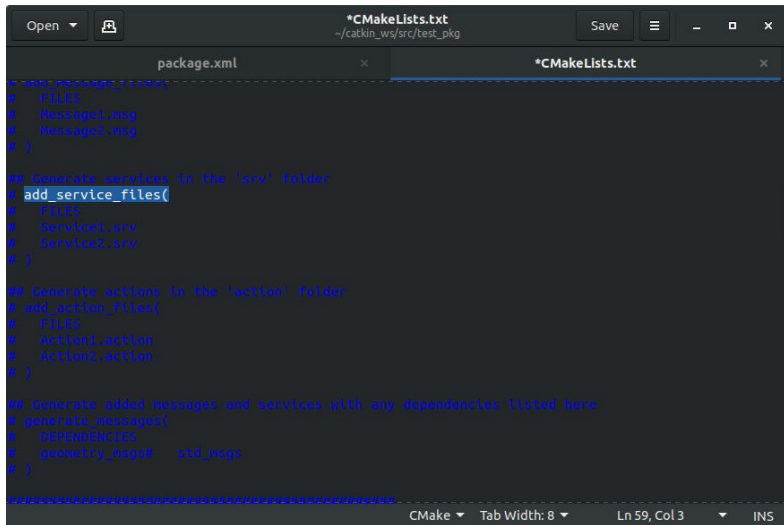
#####
## Build ##
#####

## Specify additional locations of header files
## Your package locations should be listed before other locations
include_directories(
# include
${catkin_INCLUDE_DIRS}
)
```

CMake Tab Width: 8 Ln 109, Col 69 INS

Creating custom msg and srv

-Now uncomment the `add_service_files()` function, delete the placeholder srvs and add yours to it



```
Open  package.xml  *CMakeLists.txt
~/catkin_ws/src/test_pkg  Save  -  x

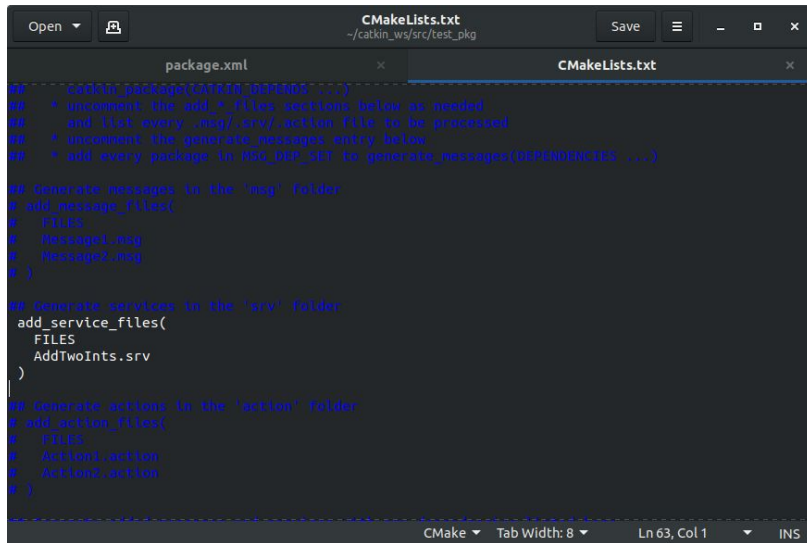
# catkin depends on these
# FILES
# Message1.msg
# Message2.msg
# )

## Generate services in the 'srv' folder
# add_service_files(
#   FILES
#   Service1.srv
#   Service2.srv
# )

## Generate actions in the 'action' folder
# add_action_files(
#   FILES
#   Action1.action
#   Action2.action
# )

## Generate added messages and services with any dependencies listed here
# generate_messages(
#   DEPENDENCIES
#   geometry_msgs@ std_msgs
# )

CMake  Tab Width: 8  Ln 59, Col 3  INS
```



```
Open  package.xml  CMakeLists.txt
~/catkin_ws/src/test_pkg  Save  -  x

## catkin package(CATKIN_DEPENDS ...)
## * uncomment the add_*_files sections below as needed
## and list every .msg/.srv/.action file to be processed
## * uncomment the generate_messages entry below
## * add every package in MSG_DEP_SET to generate_messages(DEPENDENCIES ...)

## Generate messages in the 'msg' folder
# add_message_files(
#   FILES
#   Message1.msg
#   Message2.msg
# )

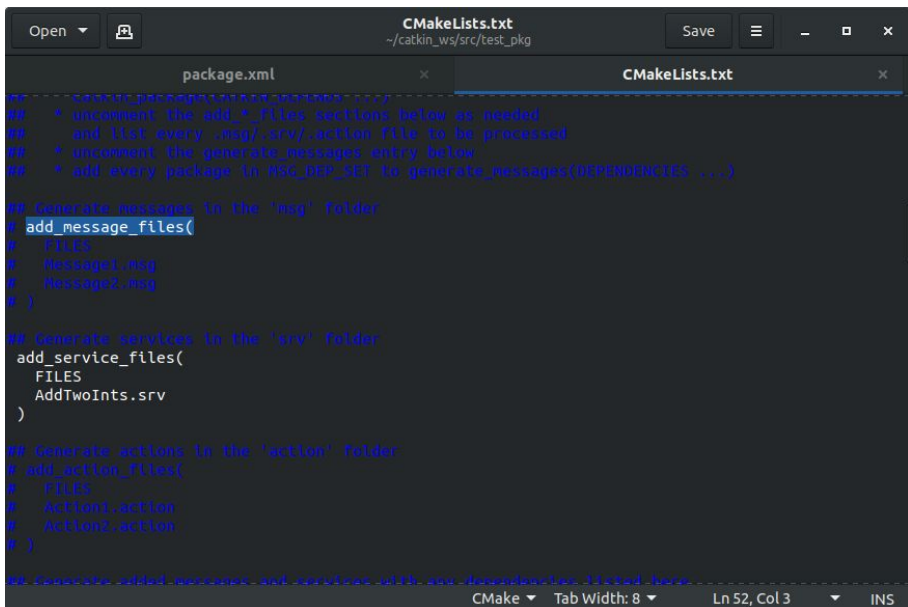
## Generate services in the 'srv' folder
add_service_files(
  FILES
  AddTwoInts.srv
)

## Generate actions in the 'action' folder
# add_action_files(
#   FILES
#   Action1.action
#   Action2.action
# )

CMake  Tab Width: 8  Ln 63, Col 1  INS
```

Creating custom msg and srv

-Now find the `add_message_files()` function uncomment it and replace the placeholder msg files with the one we made



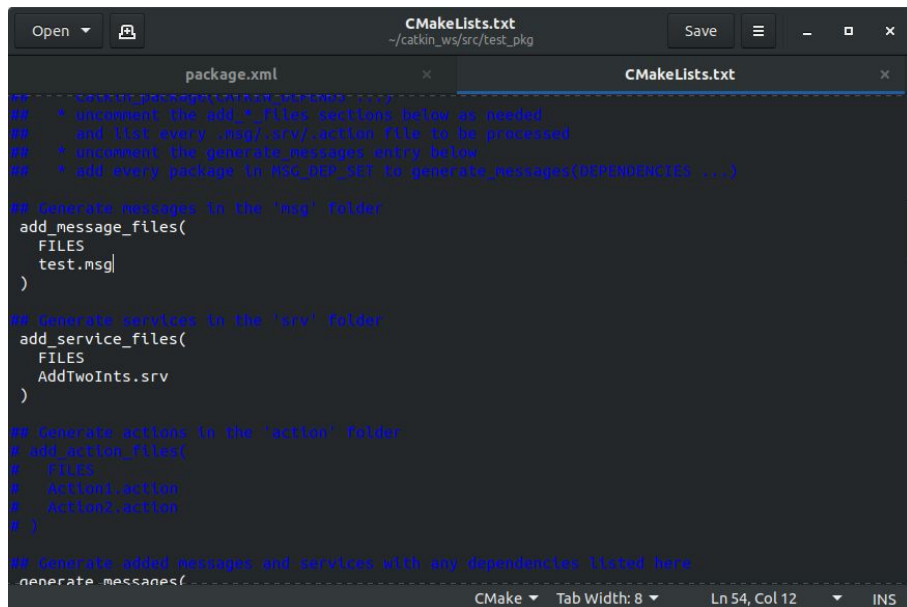
```
----- catkin_package() system dependencies -----
## * uncomment the add_*_files sections below as needed
##   and list every .msg/.srv/.action file to be processed
## * uncomment the generate_messages entry below
## * add every package in MSG_DEP_SET to generate_messages(DEPENDENCIES ...)

## Generate messages in the 'msg' folder
# add_message_files(
#   FILES
#   Message1.msg
#   Message2.msg
# )

## Generate services in the 'srv' folder
add_service_files(
  FILES
  AddTwoInts.srv
)

## Generate actions in the 'action' folder
# add_action_files(
#   FILES
#   Action1.action
#   Action2.action
# )

## Generate added messages and services with any dependencies listed here
generate_messages(
```



```
----- catkin_package() system dependencies -----
## * uncomment the add_*_files sections below as needed
##   and list every .msg/.srv/.action file to be processed
## * uncomment the generate_messages entry below
## * add every package in MSG_DEP_SET to generate_messages(DEPENDENCIES ...)

## Generate messages in the 'msg' folder
add_message_files(
  FILES
  test.msg
)

## Generate services in the 'srv' folder
add_service_files(
  FILES
  AddTwoInts.srv
)

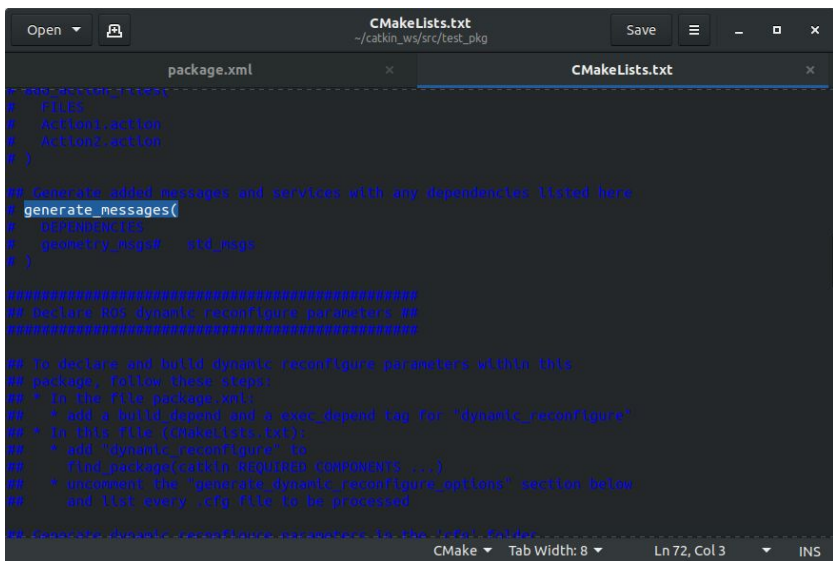
## Generate actions in the 'action' folder
# add_action_files(
#   FILES
#   Action1.action
#   Action2.action
# )

## Generate added messages and services with any dependencies listed here
generate_messages(
```

Creating custom msg and srv

-Finally uncomment the generate_messages()

-and run catkin_make



```
Open package.xml CMakeLists.txt
~/catkin_ws/src/test_pkg

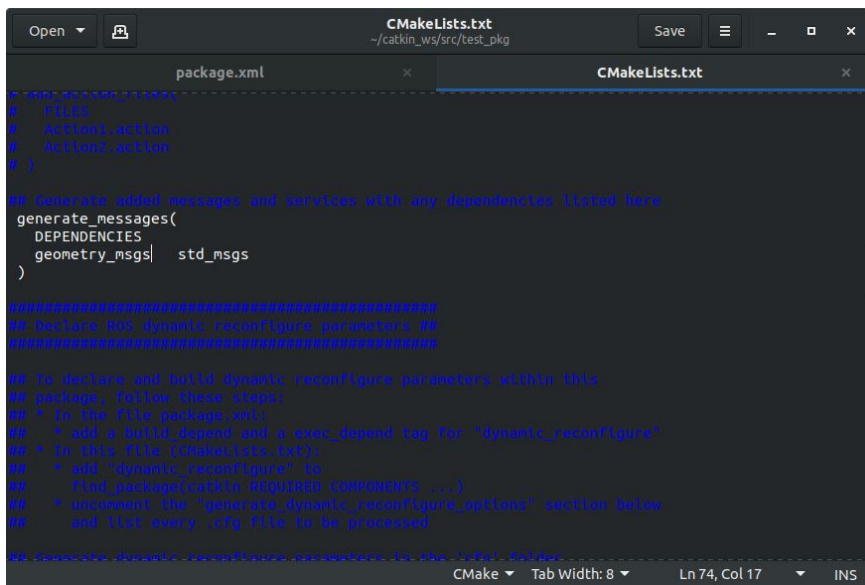
# Find action111.msg
# FILES
# Action1.action
# Action2.action
# }

## Generate added messages and services with any dependencies listed here
# generate_messages(
#   DEPENDENCIES
#   geometry_msgs| std_msgs
# )

#####
## Declare ROS dynamic reconfigure parameters ##
#####

## To declare and build dynamic reconfigure parameters within this
## package, follow these steps:
## * In the file package.xml:
##   * add a build_depend and a exec_depend tag for "dynamic_reconfigure"
## * In this file (CMakeLists.txt):
##   * add "dynamic_reconfigure" to
##     find_package(catkin REQUIRED COMPONENTS ...)
##   * uncomment the "generate_dynamic_reconfigure_options" section below
##   and list every .cfg file to be processed

## Generate dynamic reconfigure parameters in the -cfg- folder
CMake Tab Width: 8 Ln 72, Col 3 INS
```



```
Open package.xml CMakeLists.txt
~/catkin_ws/src/test_pkg

# Find action111.msg
# FILES
# Action1.action
# Action2.action
# }

## Generate added messages and services with any dependencies listed here
# generate_messages(
#   DEPENDENCIES
#   geometry_msgs| std_msgs
# )

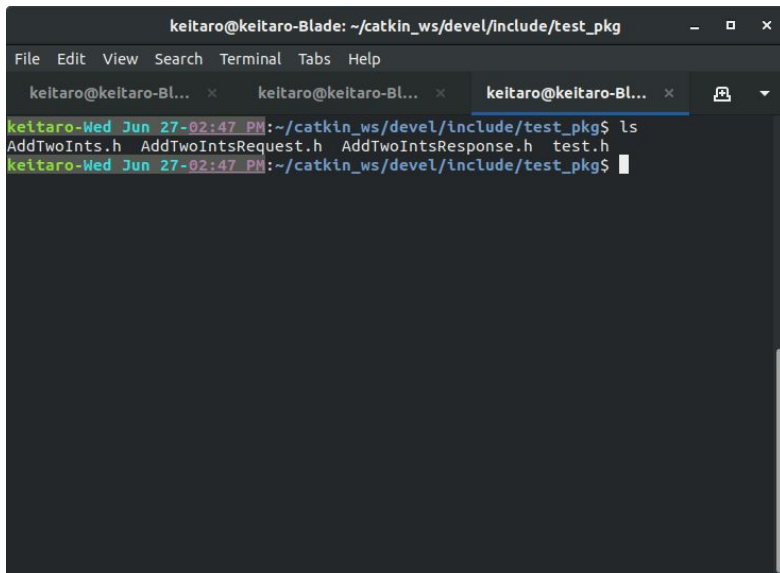
#####
## Declare ROS dynamic reconfigure parameters ##
#####

## To declare and build dynamic reconfigure parameters within this
## package, follow these steps:
## * In the file package.xml:
##   * add a build_depend and a exec_depend tag for "dynamic_reconfigure"
## * In this file (CMakeLists.txt):
##   * add "dynamic_reconfigure" to
##     find_package(catkin REQUIRED COMPONENTS ...)
##   * uncomment the "generate_dynamic_reconfigure_options" section below
##   and list every .cfg file to be processed

## Generate dynamic reconfigure parameters in the -cfg- folder
CMake Tab Width: 8 Ln 74, Col 17 INS
```

Creating custom msg and srv

-After running `catkin_make` you can find the header files generated for the msg and srv files in the `~/catkin_ws/devel/include/[your package name]` directory



```
keitaro@keitaro-Blade: ~/catkin_ws/devel/include/test_pkg
File Edit View Search Terminal Tabs Help
keitaro@keitaro-Bl... x keitaro@keitaro-Bl... x keitaro@keitaro-Bl... x
keitaro-Wed Jun 27-02:47 PM:~/catkin_ws/devel/include/test_pkg$ ls
AddTwoInts.h AddTwoIntsRequest.h AddTwoIntsResponse.h test.h
keitaro-Wed Jun 27-02:47 PM:~/catkin_ws/devel/include/test_pkg$
```

Running rosservice from a node

- There are two sets of code when running a rosservice from a node:

- Server

- The code that is waiting for a request and sending a response

- Client

- The code sending the request and waiting for the response

Running rosservice from a node

-Please refer to the “server” code

```
#include “test_pkg/AddTwoInts.h”
```

This is to add the srv to this code so we can use it. All msg and srv files made have a corresponding .h file in the build folder

Running rosservice from a node

```
bool add(test_pkg::AddTwoInts::Request  &request,
         test_pkg::AddTwoInts::Response &response)
{
    response.sum = request.a + request.b;
    ROS_INFO("request: x=%ld, y=%ld", (long int)request.a, (long int)request.b);
    ROS_INFO("sending back response: %ld", (long int)response.sum);
    return true;
}
```

The function used to when a request is sent to the server from the client. Similar to how a subscriber runs when a msg is received over a topic these functions need to be bool so as to know if the service is successful or not

Running rosservice from a node

```
ros::ServiceServer service = n.advertiseService("add_two_ints", add);
```

Similar to how a msg is sent over a topic there is a name for each service that the request and response must be sent over.

Running rosservice from a node

-Please refer to the “client” code

```
#include “test_pkg/AddTwoInts.h”
```

Don't forget to add the header for the srv so the code know the format

Running rosservice from a node

```
if(argc != 3)
{
    ROS_INFO("usage: add_client X Y");
    return 1;
}
```

This will exit the code if the user doesn't give the two numbers to add at the time of running

Running rosservice from a node

```
ros::ServiceClient client = n.serviceClient<test_pkg::AddTwoInts>("add_two_ints");
```

Similar to the subscriber code to tell the code where to send the service request
Note how there isn't a buffer variable since there is no need for one

Running rosservice from a node

```
test_pkg::AddTwoInts service;  
service.request.a = atoll(argv[1]);  
service.request.b = atoll(argv[2]);
```

This is defining and populating the service request to be sent to the server

Running rosservice from a node

```
if(client.call(service))
{
    ROS_INFO("Sum: %ld", (long int)service.response.sum);
}
else
{
    ROS_ERROR("Failed to call service add_two_ints");
    Return 1;
}
```

This is the code that is sending the request and waiting for the response from the server. Though in this code it is only run once. You could run it multiple times by having it inside of a loop. There is no need for the `ros::spin()` code for a service.

Running rosservice from a node

-If you haven't already add both codes into your src directory of your ros package as "server.cpp" and "client.cpp"

-Then open your CMakeLists.txt file and add the codes as executables by adding the following code to the bottom of your file.

```
add_executable(add_server src/server.cpp)
target_link_libraries(add_server ${catkin_LIBRARIES})
add_dependencies(add_server test_pkg_gencpp)
```

```
add_executable(add_client src/client.cpp)
target_link_libraries(add_client ${catkin_LIBRARIES})
add_dependencies(add_client test_pkg_gencpp)
```


Running rosservice from a node

-Finally run `catkin_make` from your workspace directory

-To run the service run

```
$ roscore
```

-Then in another shell run

```
$ rosrun [your package name] add_server
```

-Finally in another shell run

```
$ rosrun [your package name] add_client 5 2
```

HW

-Create a node that allows you to run the clear, reset, kill, and spawn services all from one node that is able to freely do it for up to 3 different turtles. You should still be able to freely move all three individually from each other.

-Edit your previous node so that you can send a custom msg to another node you will make that can call the clear, reset, kill, spawn, and one of the teleport services at will for up to three turtles. After which it will tell the user to “stop spawning turtles”

-the location for the teleports are the center, top left, and top right of the simulator.

-hint: use the `teleport_absolute` or `teleport_relative` services to do this.

-extra credit if you can choose which turtle goes where when teleporting