ROS

create workspace

```
$ mkdir -p ~/catkin_ws/src – vytvori slozku catkin_ws a podslozku src
$ cd ~/catkin_ws/ - jsme ve slozce catkin_ws
$ catkin_make


$ source devel/setup.bash


$ echo $ROS_PACKAGE_PATH – mělo by ukazat /home/youruser/catkin_ws/src:/opt
/ros/kinetic/share
```

navigace

```
$ sudo apt-get install ros-<distro>-ros-tutorials – instalace tutorialu
$ rospack find [package_name] – hledani package
$ roscd roscpp/cmake
$ pwd
$ echo $ROS_PACKAGE_PATH
$ roscd log – slozka s ROS log soubory
```

creating package

package – package.xml, CMakeLists.txt, každý package svoje slozka

```
$ cd ~/catkin_ws/src
$ catkin_create_pkg beginner_tutorials std_msgs rospy roscpp –package v src
$ cd ~/catkin_ws
$ catkin_make
$ . ~/catkin_ws/devel/setup.bash – pridani workspace do ROS
$ rospack depends1 beginner_tutorials - dependecies
```

building package

```
$ source /opt/ros/kinetic/setup.bash
$ cd ~/catkin_ws/
$ ls src – vypsani obsahu src
$ catkin_make – vytvoreni package
```

ROS nodes

rospy – nody v pythonu

roscpp – nody v c++

```
$ roscore – otevřít novy terminal
$ rosnode list – vypise bezici nody a informace o nich
$ rosnode info /rosout
$ rosrun [package_name] [node_name] ($ rosrun turtlesim turtlesim_node
__name:=my_turtle – otevřít novy terminal
```

Topics

```
$ roscore – otevřít novy terminal
$ rosrun turtlesim turtlesim_node – otevřít novy terminal
$ rosrun turtlesim turtlesim_teleop_key – otevřít novy terminal
$ sudo apt-get install ros-kinteic-rqt
$ sudo apt-get install ros-kinetic-rqt-common-plugins
$ rosrun rqt_graph rqt_graph – graf zavislosti
$ rostopic –h
$ rostopic echo /turtle1/cmd_vel
$ rostopic list –v
$ rostopic type /turtle1/cmd_vel – typ zprávy posilane topikem
$ rostopic pub -1 /turtle1/cmd_vel geometry_msgs/Twist -- '[2.0, 0.0, 0.0]'
'[0.0, 0.0, 1.8]'
$ rostopic pub /turtle1/cmd_vel geometry_msgs/Twist -r 1 -- '[2.0, 0.0, 0.0
]' '[0.0, 0.0, -1.8]'
$ rostopic hz /turtle1/pose
$ rostopic type /turtle1/cmd_vel | rosmsg show
$ rosrun rqt_plot rqt_plot
```

Services and parameters

```
$ rosservice list
$ rosservice type /clear
$ rosservice call /clear
$ rosservice type /spawn | rossrv show
$ rosservice call /spawn 2 2 0.2 „"
$ rosparam list
rosparam set [param_name]
```

```
rosparam get [param_name]

$ rosparam set /background_r 150

$ rosparam get /

$ rosparam dump params.yaml - zapsani parametru do souboru

$ rosparam load params.yaml copy - nacteni parametru za souboru

$ rosparam get /copy/background_b - vypsani
```

rqt_console a roslaunch

```
$ sudo apt-get install ros-<distro>-rqt ros-<distro>-rqt-common-plugins ros
-<distro>-turtlesim - instalace

$ rosrun rqt_console rqt_console -oboje rozbehneme před spustenim turtlesim

$ rosrun rqt_logger_level rqt_logger_level

$ roslaunch [package] [filename.launch] - pouziti roslaunch

$ cd ~/catkin_ws

$ source devel/setup.bash

$ roscd beginner_tutorials

$ mkdir launch

$ cd launch
```

soubor turtlemimic.launch

```
<launch>

  <group ns="turtlesim1">
    <node pkg="turtlesim" name="sim" type="turtlesim_node"/>
  </group>

  <group ns="turtlesim2">
    <node pkg="turtlesim" name="sim" type="turtlesim_node"/>
  </group>

  <node pkg="turtlesim" name="mimic" type="mimic">
    <remap from="input" to="turtlesim1/turtle1"/>
    <remap from="output" to="turtlesim2/turtle1"/>
  </node>

</launch>
```

```
$ roslaunch beginner_tutorials turtlemimic.launch

$ rostopic pub /turtlesim1/turtle1/cmd_vel geometry_msgs/Twist -r 1 -- '[2.
0, 0.0, 0.0]' '[0.0, 0.0, -1.8]'

$ rqt_graph
```

msg a srv

```
$ roscd beginner_tutorials
```

```
$ mkdir msg
$ echo "int64 num" > msg/Num.msg
```

v package.xml odkomentovat

```
<build_depend>message_generation</build_depend>
<exec_depend>message_runtime</exec_depend>
```

zmeny v CMakeLists.txt

```
find_package(catkin REQUIRED COMPONENTS
   roscpp
   rospy
   std_msgs
   message_generation
)
add_message_files(
  FILES
  Num.msg
)
generate_messages(
  DEPENDENCIES
  std_msgs
)
```

```
$ rosmsg show [message type]
$ rosmsg show beginner_tutorials/Num | $ rosmsg show Num
$ roscd beginner_tutorials
$ mkdir srv
$ roscp [package_name] [file_to_copy_path] [copy_path] - kopirovani srv def
inice z jiného packege
$ roscp rospy_tutorials AddTwoInts.srv srv/AddTwoInts.srv
```

kouknout jestli v package.xml je

```
<build_depend>message_generation</build_depend>
<exec_depend>message_runtime</exec_depend>
```

změna v CMakeLists.txt

```
find_package(catkin REQUIRED COMPONENTS
  roscpp
  rospy
  std_msgs
  message_generation
)
add_service_files(
  FILES
  AddTwoInts.srv
)
```

```
$ rossrv show <service type>
$ rossrv show beginner_tutorials/AddTwoInts | $ rossrv show AddTwoInts
# In your catkin workspace
$ roscd beginner_tutorials
$ cd ../..
$ catkin_make install
$ cd -
$ rosmsg -h
$ rosmsg show -h
```

rospack – informace o package

roscd – **c**hange **d**irectory

rosls – **l**ist**s** files to/from package

roscp – **co**pies files from/to package

Publisher and subscriber C++

```
roscd beginner_tutorials
mkdir -p src
```

src/talker.cpp

```
#include "ros/ros.h"
#include "std_msgs/String.h"

#include <sstream>

/**
```

```
 * This tutorial demonstrates simple sending of messages over the ROS syste
m.
 */
int main(int argc, char **argv)
{
  /**
   * The ros::init() function needs to see argc and argv so that it can per
form
   * any ROS arguments and name remapping that were provided at the command
line.
   * For programmatic remappings you can use a different version of init()
which takes
   * remappings directly, but for most command-line programs, passing argc
and argv is
   * the easiest way to do it.  The third argument to init() is the name of
the node.
   *
   * You must call one of the versions of ros::init() before using any othe
r
   * part of the ROS system.
   */
  ros::init(argc, argv, "talker");

  /**
   * NodeHandle is the main access point to communications with the ROS sys
tem.
   * The first NodeHandle constructed will fully initialize this node, and
the last
   * NodeHandle destructed will close down the node.
   */
  ros::NodeHandle n;

  /**
   * The advertise() function is how you tell ROS that you want to
   * publish on a given topic name. This invokes a call to the ROS
   * master node, which keeps a registry of who is publishing and who
   * is subscribing. After this advertise() call is made, the master
   * node will notify anyone who is trying to subscribe to this topic name,
   * and they will in turn negotiate a peer-to-peer connection with this
   * node.  advertise() returns a Publisher object which allows you to
   * publish messages on that topic through a call to publish().  Once
   * all copies of the returned Publisher object are destroyed, the topic
   * will be automatically unadvertised.
   *
   * The second parameter to advertise() is the size of the message queue
   * used for publishing messages.  If messages are published more quickly
   * than we can send them, the number here specifies how many messages to
   * buffer up before throwing some away.
   */
  ros::Publisher chatter_pub = n.advertise<std_msgs::String>("chatter", 100
0);

  ros::Rate loop_rate(10);

  /**
   * A count of how many messages we have sent. This is used to create
   * a unique string for each message.
   */
  int count = 0;
  while (ros::ok())
  {
```

```
    /**
     * This is a message object. You stuff it with data, and then publish i
t.
     */
    std_msgs::String msg;

    std::stringstream ss;
    ss << "hello world " << count;
    msg.data = ss.str();

    ROS_INFO("%s", msg.data.c_str());

    /**
     * The publish() function is how you send messages. The parameter
     * is the message object. The type of this object must agree with the t
ype
     * given as a template parameter to the advertise<>() call, as was done
     * in the constructor above.
     */
    chatter_pub.publish(msg);

    ros::spinOnce();

    loop_rate.sleep();
    ++count;
  }


  return 0;
}
```

src/listener.cpp

```
#include "ros/ros.h"
#include "std_msgs/String.h"

/**
 * This tutorial demonstrates simple receipt of messages over the ROS syste
m.
 */
void chatterCallback(const std_msgs::String::ConstPtr& msg)
{
  ROS_INFO("I heard: [%s]", msg->data.c_str());
}

int main(int argc, char **argv)
{
  /**
   * The ros::init() function needs to see argc and argv so that it can per
form
   * any ROS arguments and name remapping that were provided at the command
line.
   * For programmatic remappings you can use a different version of init()
which takes
   * remappings directly, but for most command-line programs, passing argc
and argv is
   * the easiest way to do it.  The third argument to init() is the name of
the node.
   *
   * You must call one of the versions of ros::init() before using any othe
r
   * part of the ROS system.
```

```
   */
  ros::init(argc, argv, "listener");

  /**
   * NodeHandle is the main access point to communications with the ROS sys
tem.
   * The first NodeHandle constructed will fully initialize this node, and
the last
   * NodeHandle destructed will close down the node.
   */
  ros::NodeHandle n;

  /**
   * The subscribe() call is how you tell ROS that you want to receive mess
ages
   * on a given topic.  This invokes a call to the ROS
   * master node, which keeps a registry of who is publishing and who
   * is subscribing.  Messages are passed to a callback function, here
   * called chatterCallback.  subscribe() returns a Subscriber object that
you
   * must hold on to until you want to unsubscribe.  When all copies of the
Subscriber
   * object go out of scope, this callback will automatically be unsubscrib
ed from
   * this topic.
   *
   * The second parameter to the subscribe() function is the size of the me
ssage
   * queue.  If messages are arriving faster than they are being processed,
this
   * is the number of messages that will be buffered up before beginning to
throw
   * away the oldest ones.
   */
  ros::Subscriber sub = n.subscribe("chatter", 1000, chatterCallback);

  /**
   * ros::spin() will enter a loop, pumping callbacks.  With this version,
all
   * callbacks will be called from within this thread (the main one).  ros:
:spin()
   * will exit when Ctrl-C is pressed, or the node is shutdown by the maste
r.
   */
  ros::spin();

  return 0;
}
```

na konec CMakeLists.txt

```
add_executable(talker src/talker.cpp)

target_link_libraries(talker ${catkin_LIBRARIES})

add_dependencies(talker beginner_tutorials_generate_messages_cpp)


add_executable(listener src/listener.cpp)

target_link_libraries(listener ${catkin_LIBRARIES})
```

```
add_dependencies(listener beginner_tutorials_generate_messages_cpp)
```

```
# In your catkin workspace
$ cd ~/catkin_ws
$ catkin_make
```

Publisher and subscriber Python

```
$ roscd beginner_tutorials
$ mkdir scripts
$ cd scripts
```

vytvořit talker.py

```
#!/usr/bin/env python
# license removed for brevity
import rospy
from std_msgs.msg import String

def talker():
    pub = rospy.Publisher('chatter', String, queue_size=10)
    rospy.init_node('talker', anonymous=True)
    rate = rospy.Rate(10) # 10hz
    while not rospy.is_shutdown():
        hello_str = "hello world %s" % rospy.get_time()
        rospy.loginfo(hello_str)
        pub.publish(hello_str)
        rate.sleep()

if __name__ == '__main__':
    try:
        talker()
    except rospy.ROSInterruptException:
        pass
```

```
$ wget https://raw.github.com/ros/ros_tutorials/kinetic-devel/rospy_tutorials/001_talker_listener/talker.py
$ chmod +x talker.py  - spustitelnost
```

listener.py

```
#!/usr/bin/env python
import rospy
from std_msgs.msg import String

def callback(data):
    rospy.loginfo(rospy.get_caller_id() + "I heard %s", data.data)

def listener():

    # In ROS, nodes are uniquely named. If two nodes with the same
    # node are launched, the previous one is kicked off. The
    # anonymous=True flag means that rospy will choose a unique
```

```
    # name for our 'listener' node so that multiple listeners can
    # run simultaneously.
    rospy.init_node('listener', anonymous=True)

    rospy.Subscriber("chatter", String, callback)

    # spin() simply keeps python from exiting until this node is stopped
    rospy.spin()

if __name__ == '__main__':
    listener()
```

```
$ roscd beginner_tutorials/scripts/

$ wget https://raw.github.com/ros/ros_tutorials/kinetic-devel/rospy_tutorials/001_talker_listener/listener.py

$ chmod +x listener.py - spustitelnost

$ cd ~/catkin_ws

$ catkin_make
```

Running Publisher and subscriber

```
$ roscore

# In your catkin workspace

$ cd ~/catkin_ws

$ source ./devel/setup.bash

$ rosrun beginner_tutorials talker       (C++)

$ rosrun beginner_tutorials talker.py    (Python)

$ rosrun beginner_tutorials listener      (C++)

$ rosrun beginner_tutorials listener.py  (Python)
```

Service and client C++

```
 roscd beginner_tutorials
```

soubor src/add_two_ints_server.cpp

```
#include "ros/ros.h"
#include "beginner_tutorials/AddTwoInts.h"

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

```
int main(int argc, char **argv)
{
  ros::init(argc, argv, "add_two_ints_server");
  ros::NodeHandle n;

  ros::ServiceServer service = n.advertiseService("add_two_ints", add);
  ROS_INFO("Ready to add two ints.");
  ros::spin();

  return 0;
}
```

soubor src/add_two_ints_client.cpp

```
#include "ros/ros.h"
#include "beginner_tutorials/AddTwoInts.h"
#include <cstdlib>

int main(int argc, char **argv)
{
  ros::init(argc, argv, "add_two_ints_client");
  if (argc != 3)
  {
    ROS_INFO("usage: add_two_ints_client X Y");
    return 1;
  }

  ros::NodeHandle n;
  ros::ServiceClient client = n.serviceClient<beginner_tutorials::AddTwoInt
s>("add_two_ints");
  beginner_tutorials::AddTwoInts srv;
  srv.request.a = atoll(argv[1]);
  srv.request.b = atoll(argv[2]);
  if (client.call(srv))
  {
    ROS_INFO("Sum: %ld", (long int)srv.response.sum);
  }
  else
  {
    ROS_ERROR("Failed to call service add_two_ints");
    return 1;
  }

  return 0;
}
```

na konec CMakeLists.txt

```
add_executable(add_two_ints_server src/add_two_ints_server.cpp)
target_link_libraries(add_two_ints_server ${catkin_LIBRARIES})
add_dependencies(add_two_ints_server beginner_tutorials_gencpp)

add_executable(add_two_ints_client src/add_two_ints_client.cpp)
target_link_libraries(add_two_ints_client ${catkin_LIBRARIES})
add_dependencies(add_two_ints_client beginner_tutorials_gencpp)
```

```
# In your catkin workspace

cd ~/catkin_ws

catkin_make

$ roscore
```

```
$ rosrun beginner_tutorials add_two_ints_server - mělo by se objevit Ready to add two ints.

$ rosrun beginner_tutorials add_two_ints_client 1 3 - napise Sum: 4, v serveru by se mělo ukazat request: x=1, y=3, sending back response: [4]
```

Service a client Python

```
$ roscd beginner_tutorials
```

scripts/add_two_ints_server.py

```
#!/usr/bin/env python

from beginner_tutorials.srv import *
import rospy

def handle_add_two_ints(req):
    print "Returning [%s + %s = %s]"%(req.a, req.b, (req.a + req.b))
    return AddTwoIntsResponse(req.a + req.b)

def add_two_ints_server():
    rospy.init_node('add_two_ints_server')
    s = rospy.Service('add_two_ints', AddTwoInts, handle_add_two_ints)
    print "Ready to add two ints."
    rospy.spin()

if __name__ == "__main__":
    add_two_ints_server()
```

```
chmod +x scripts/add_two_ints_server.py - spustitelnost
```

scripts/add_two_ints_client.py

```
#!/usr/bin/env python

import sys
import rospy
from beginner_tutorials.srv import *

def add_two_ints_client(x, y):
    rospy.wait_for_service('add_two_ints')
    try:
        add_two_ints = rospy.ServiceProxy('add_two_ints', AddTwoInts)
        resp1 = add_two_ints(x, y)
        return resp1.sum
    except rospy.ServiceException, e:
        print "Service call failed: %s"%e

def usage():
    return "%s [x y]"%sys.argv[0]

if __name__ == "__main__":
    if len(sys.argv) == 3:
        x = int(sys.argv[1])
        y = int(sys.argv[2])
    else:
        print usage()
        sys.exit(1)
```

```
    print "Requesting %s+%s"%(x, y)
    print "%s + %s = %s"%(x, y, add_two_ints_client(x, y))
```

```
$ chmod +x scripts/add_two_ints_client.py - spustitelnost

# In your catkin workspace

$ cd ~/catkin_ws

$ catkin_make
```

Running service a client

```
$ rosrun beginner_tutorials add_two_ints_server     (C++)

$ rosrun beginner_tutorials add_two_ints_server.py  (Python) - Ready to add
two ints.

$ rosrun beginner_tutorials add_two_ints_client 1 3     (C++)

$ rosrun beginner_tutorials add_two_ints_client.py 1 3  (Python) - Requesti
ng 1+3, 1 + 3 = 4
```

Recording and playing back

```
roscore

rosrun turtlesim turtlesim_node

rosrun turtlesim turtle_teleop_key

rostopic list -v - vypise public topics - jedine co lze nahravat v data log
souboru

mkdir ~/bagfiles

cd ~/bagfiles

rosbag record -a
```

```
-  pohyb zelvou, ukoncit rosbag record
```

```
rosbag info <your bagfile> - info o souboru
```

```
-  kill turtle_teleop_key
```

```
rosbag play <your bagfile>

rosrun turtlesim turtlesim_node

rosrun turtlesim turtle_teleop_key

rosbag record -O subset /turtle1/cmd_vel /turtle1/pose - nahrani jen nejaky
ch topicu
```

roswtf

```
$ roscd
```

```
$ roswtf - hleda problémy

$ ROS_PACKAGE_PATH=bad:$ROS_PACKAGE_PATH roswtf
```

Spuštění mojí mapy + ovládání, mapování

```
roscore

rosrun stage_ros stageros /home/lukas/catkin_ws2/srcobotcraft2017_maze/worl
d/myworld.world

rosrun teleop_twist_keyboard teleop_twist_keyboard.py

rosrun gmappimg slam_gmapping scan:=base_scan

rosrun rviz rviz -d `rospack find stage_ros`/rviz/stage.rviz
```