Introduction and Concepts

Elnaz Balazadeh

balazadeh.elnaz@gmail.com

Installation

Concepts

Example

Installation

Concepts

Example

Installation

Configure your Ubuntu repositories

Installation

Dependencies for building packages

Installing and Configuring Your ROS Environment

Installation

www.wiki.ros.org

Installation

Installing and Configuring Your ROS Environment

www.wiki.ros.org/catkin/Tutorials/create_a_workspace

Installation

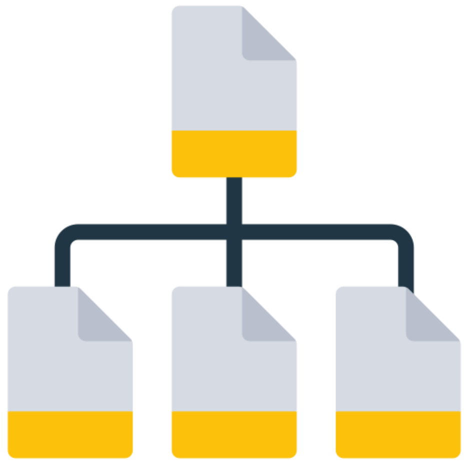
Concepts

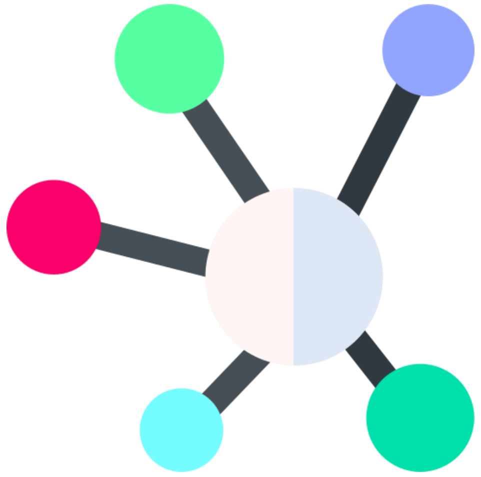
ROS has three levels of concepts

ROS File system Level

ROS Computation Graph Level

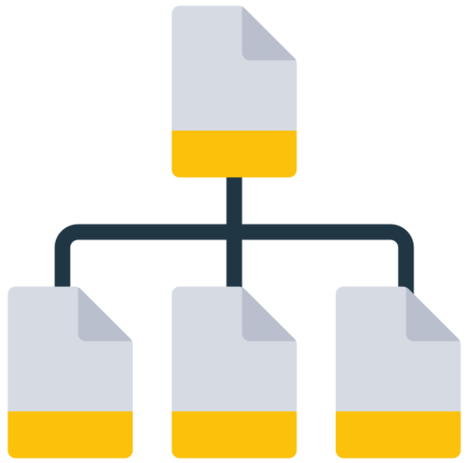
ROS Community Level





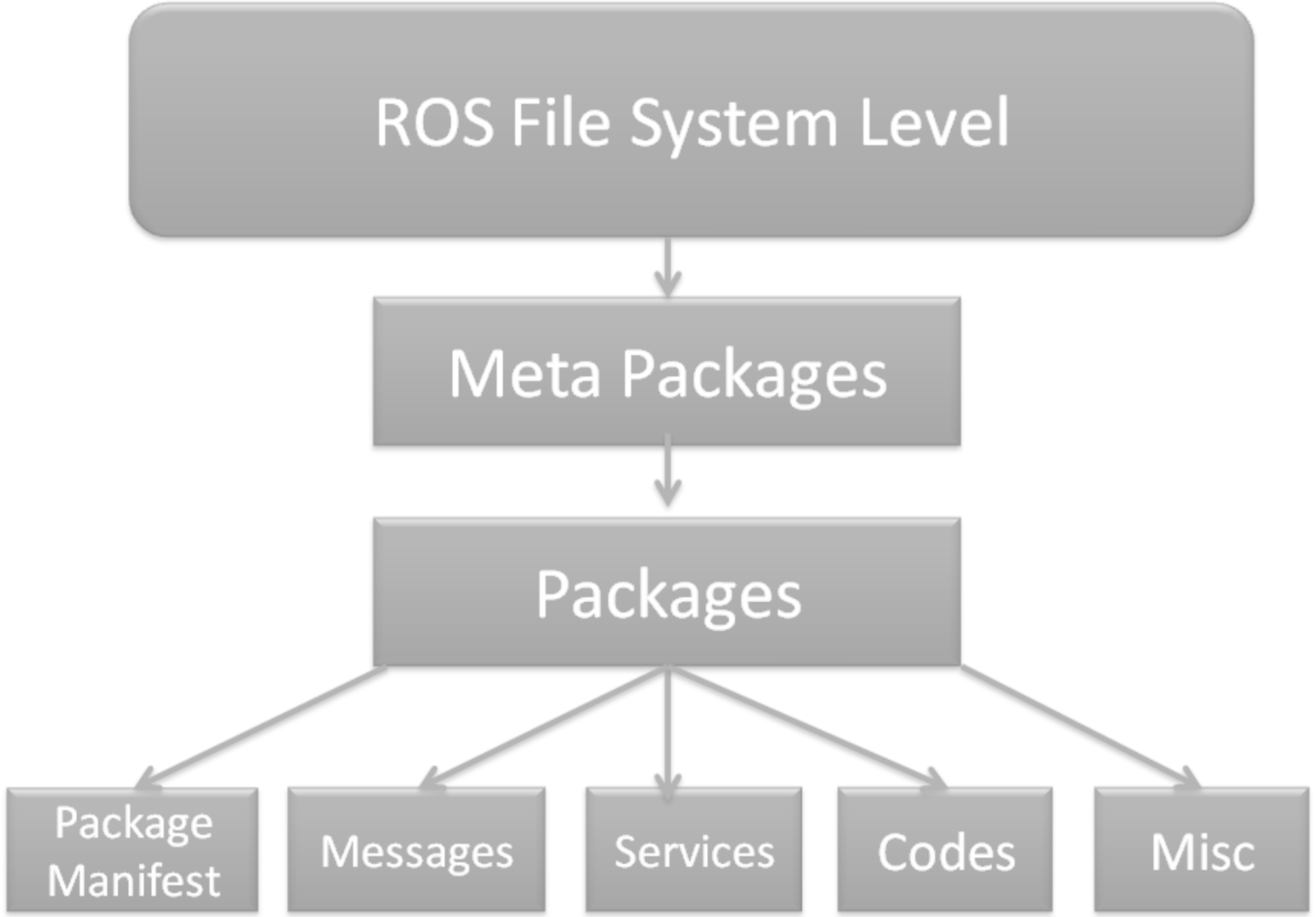
Concepts

ROS File system Level



Concepts

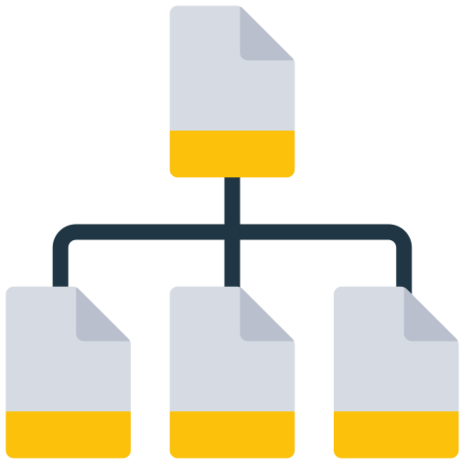
ROS File system Level



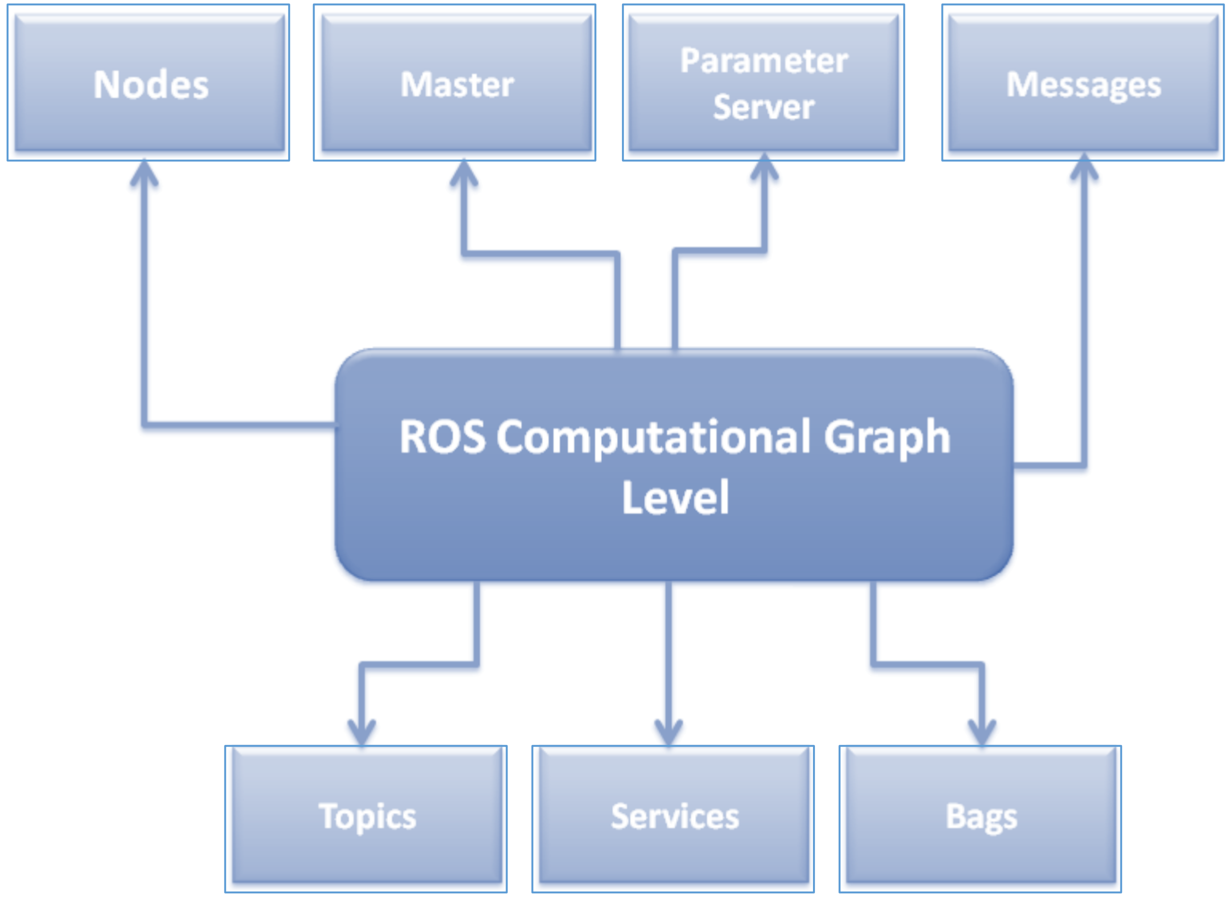
Concepts

ROS File system Level

ROS Computation Graph Level



Concepts



Joseph, L. and Cacace, J., 2018. *Mastering ROS for Robotics Programming: Design, build, and simulate complex robots using the Robot Operating System*. Packt Publishing Ltd.

Installation

Concepts

Example

Installation

Concepts

Example

Example

1. Create your package

```
$ cd ~/catkin_ws/src
# catkin_create_pkg <package_name> [depend1] [depend2] [depend3]
$ catkin_create_pkg my_package std_msgs rospy roscpp
```

Example

2. Create your first node

Create a python file **pub_node.py**

```
$ cd ~/catkin_ws/src/my_package/src/
$ > pub_node.py
```

Enable execute permission

```
$ chmod +x pub_node.py
```

Example

2. Create your first node

Write python script in **pub_node.py**

```
#!/usr/bin/env python3
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 " + str(rospy.get_time())
        rospy.loginfo(hello_str)
        pub.publish(hello_str)
        rate.sleep()

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

Example

2. Create your first node

Run python script **pub_node.py**

```
# rosrun [package_name] [node_name]
$ rosrun my_package pub_node.py
```

Example

3. Create your subscriber node

Create a python file **subs_node.py**

```
$ cd ~/catkin_ws/src/my_package/src/
$ > subs_node.py
```

Enable execute permission

```
$ chmod +x subs_node.py
```

Example

3. Create your subscriber node

Write python script in **subs_node.py**

```
#!/usr/bin/env python3
import rospy
from std_msgs.msg import String

def callback(data):
    rospy.loginfo("I heard ‶ + data.data)

def listener():

    # In ROS, nodes are uniquely named. If two nodes with the same
    # name 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()
```

3. Create your subscriber node

Run python script **subs_node.py**

```
# rosrun [package_name] [node_name]
$ rosrun my_package subs_node.py
```

Example

4. Create your custom message

Create msg/ directory in package folder

```
$ mkdir msg
```

Example

4. Create your custom message

Create **demo_msg.msg** file

```
$ > demo_msg.msg
```

Specify message type and name in demo_msg.msg file

```
# msg type/msg name
string greeting
int32 number
```

Example

4. Create your custom message

Modify CMakeLists.txt and package.xml

package.xml

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

CMakeLists.txt

```
find_package(catkin REQUIRED COMPONENTS
  roscpp
  rospy
  std_msgs
  message_generation
)

add_message_files(
   FILES
   demo_msg.msg
#   Message2.msg
)

generate_messages(
   DEPENDENCIES
   std_msgs
 )
```

Example

4. Create your custom message

Build your package

```
$ cd ~/catkin_ws/
$ catkin_make
```

Example

5. Writing a Simple Service and Client with custom service

Create srv/ directory in package folder

```
$ mkdir srv
```

Example

5. Writing a Simple Service and Client with custom service

Create **demo_srv.srv** file

```
$ > demo_srv.srv
```

Specify Request and Response message type and name in demo_srv.srv file

```
# Request msg type
string in_name
---
# Response msg type
string out_greeting
```

Example

5. Writing a Simple Service and Client with custom service

Modify CMakeLists.txt and package.xml

package.xml

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

CMakeLists.txt

```
find_package(catkin REQUIRED COMPONENTS
  roscpp
  rospy
  std_msgs
  message_generation
)

 add_service_files(
   FILES
   demo_srv.srv
#   Service2.srv
)
```

Example

5. Writing a Simple Service and Client with custom service

Build your package

```
$ cd ~/catkin_ws/
$ catkin_make
```

Example

5. Writing a Simple Service and Client with custom service

Create executable **demo_service_server.py** file

```
$ > demo_service_server.py
```

Example

5. Writing a Simple Service and Client with custom service

Write python script in **demo_service_server.py**

```
#!/usr/bin/env python3
import rospy
from my_package.srv import demo_srv

def say_hello_server():
    rospy.init_node("demo_service_server")
    server = rospy.Service('greeting_service', demo_srv, handler=say_hello)
    print("Server is ready to say greeting :)")
    rospy.spin())

def say_hello(req):

    resp = 'Hello dear ' + '"' + req.in_name + '"'
    print(resp)
    return (resp)


if __name__ == ‘__main__’:
    say_hello_server()
```

Example

5. Writing a Simple Service and Client with custom service

Create executable **demo_service_client.py** file

```
$ > demo_service_client.py
```

Example

5. Writing a Simple Service and Client with custom service

Write python script in **demo_service_client.py**

```
#!/usr/bin/env python3
import rospy
from my_package.srv import demo_srv
import sys

def send_name(in_name):
    rospy.wait_for_service('greeting_service')
    client_greeting = rospy.ServiceProxy('greeting_service', demo_srv)
    response = client_greeting(in_name)
    print(f"Server responded ---> {response.out_greeting}")

if __name__ == "__main__":
     in_name = str(sys.argv[1])
     send_name(in_name)
```