ROS 0-60

A comprehensive tutorial of
the Robot Operating System

Installing

```
# update macports to latest version
$ sudo port selfupdate
$ sudo port upgrade outdated

# get enough software to start building ros
$ sudo port install wget cmake py25-yaml python_select
$ sudo python_select python25
$ sudo ln -s /opt/local/Library/Frameworks/Python.framework  /Library/Frameworks

# tell ROS about macports
$ echo “export CPATH=/opt/local/include” >> ~/.profile
$ echo “export LIBRARY_PATH=/opt/local/include” >> ~/.profile
$ source ~/.profile

# get the ROS getter, get ROS, and build just the basics.
$ wget --no-check-certificate http://ros.org/rosconfig -O ~/rosconfig && chmod 755 ~/rosconfig
$ ~/rosconfig bootstrap -s http://ros.org/rosconfigs/all.rosconfig ~/ros roscpp

# configure your bash environment for ros
$ ~/rosconfig setup ~/ros > ~/.bashrc.ros
$ echo “source ~/.bashrc.ros” >> .profile
$ source ~/.profile
```

What is ROS?

- A set of libraries for creating programs that communicate flexible data structures simply and efficiently.
- A set of tools for managing large software systems.
- A set of software created by other Robot developers.

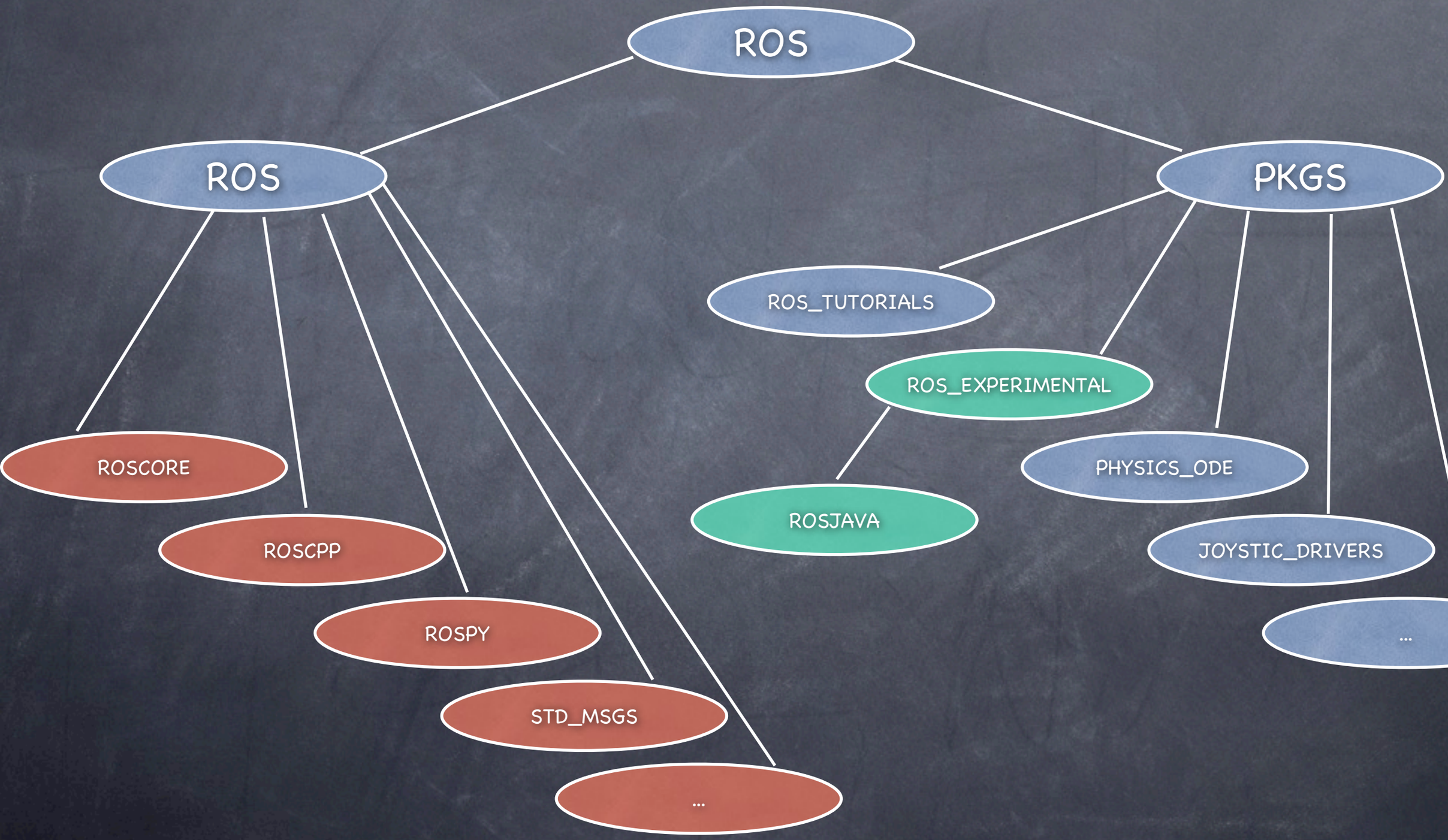
The bad

- Steep learning curve.
- Some features are poorly documented, if they exist.
- API is a moving target -- e.g. the current rosjava is an analogue for a previous API version, including some deprecated things.
  - Hard to port(?), hard to maintain port.
  - Native Java, Windows, Flash
- Free as in kittens

The good

- Navigating around ROS and compiling / running is ridiculously easy.
- Possible to make libraries that "just work" for software that depends on them (don't worry about linking to dependencies, etc.)
- RubiOS is a strict subset of ROS, so existing MPLab software is easily integrated.
- WG is willing to let us help maintain ROS.

What you get


ROS
ROS
PKGS
ROS_TUTORIALS
ROS_EXPERIMENTAL
ROSCORE
ROSCPP
ROSPY
STD_MSGS
...
ROSJAVA
PHYSICS_ODE
JOYSTIC_DRIVERS
...

Verify Installation

1

```
$ roscore
```

2

```
$ rosmake roscpp_tutorials
$ rosrun roscpp_tutorials talker
^C
```

3

```
$ rosrun roscpp_tutorials listener
^C
```

2

```
$ rosrun roscpp_tutorials \
    add_two_ints_server
```

3

```
$ rosrun roscpp_tutorials \
    add_two_ints_client 3 5

$ rosrun roscpp_tutorials \
    add_two_ints_client 12345 98765

$ rosrun roscpp_tutorials \
    add_two_ints_client   \
    8000000000 8000000000
```

Fixing things

1: Choose an editor

```
$ echo “export EDITOR=\”open -a XCode\” >> ~/.profile
$ echo “export SVN_EDITOR=vim” >> ~/.profile
$ source ~/.profile
```

2: Fix add_two_ints_client

```
$ rosed roscpp_tutorials \
    add_two_ints_client.cpp

# change “atoi” to “atoll”

$ rosmake roscpp_tutorials
$ rosrun roscpp_tutorials \
    add_two_ints_client   \
    8000000000 8000000000
```

3: Fix roscreate-pkg

```
$ rosed roscreate roscreatepkg.py
--- roscreatepkg.py  (revision 7388)
+++ roscreatepkg.py  (working copy)
@@ -73,8 +73,9 @@
     if uses_rospy:
         # create package/src/ for python files
         py_path = os.path.join(p, 'src')
-        print "Creating python source directory", py_path
-        os.makedirs(py_path)
+        if not os.path.exists(py_path):
+            print "Creating python source directory", py_path
+            os.makedirs(py_path)

     templates = get_templates()
     for filename, template in templates.iteritems():
```

ROS Packages

- A "package" is a basic unit in ROS.
  - Usually it encapsulates a node (or similar set of nodes) e.g. roscpp_tutorials, or a library e.g. roscpp, rosjava
- A package is a package because it
  - 1. Is descended from a directory in ROS_PACKAGE_PATH (.bashrc.ros)
  - 2. Contains a "manifest.xml" file that says what packages this package depends on (<depend>), and what dependent packages need to know (<export>)
- A package usually has a CMakeLists.txt file, describing how to build the package.
- A package cannot contain other packages.

```
$ roscd ros_tutorials
$ roscreate-pkg beginner_tutorials \
    std_msgs rospy roscpp

$ rosls beginner_tutorials
$ cat `rosls \
    beginner_tutorials/manifest.xml`
$ cat `rosls \
    beginner_tutorials/CMakeLists.txt`

$ rospack depends1 beginner_tutorials
$ rospack depends beginner_tutials
$ rospack depends1 rospy

$ roscd beginner_tutorials
$ touch src/talker.cpp
$ echo "rosbuild_add_executable(talker
 src/talker.cpp)" >> CMakeLists.txt

$ rosed beginner_tutorials talker.cpp
```

Hello World (1)

src/talker.cpp

```
#include <ros/ros.h>
#include <std_msgs/String.h>
#include <sstream>
int main(int argc, char** argv)
{
  ros::init(argc, argv, "talker");
  ros::NodeHandle n;
  ros::Publisher chatter_pub =
      n.advertise<std_msgs::String>("chatter", 100);
  ros::Rate loop_rate(5);
  int count = 0;
  while (ros::ok())
  {
    std::stringstream ss;
    ss << "Hello there! This is message ["
       << count << "]";
    std_msgs::String msg;
    msg.data = ss.str();
    chatter_pub.publish(msg);
    ROS_INFO("I published [%s]", ss.str().c_str());
    ros::spinOnce();
    loop_rate.sleep();
    ++count;
  }
}
```

src/listener.cpp

```
#include <ros/ros.h>
#include <std_msgs/String.h>
void chatterCallback(const
                     std_msgs::StringConstPtr& msg)
{
  ROS_INFO("Received [%s]", msg->data.c_str());
}
int main(int argc, char** argv)
{
  ros::init(argc, argv, "listener");
  ros::NodeHandle n;
  ros::Subscriber chatter_sub =
    n.subscribe("chatter", 100, chatterCallback);
  ros::spin();
}
```

*If your callback doesn't take an argument (const [T]ConstPtr&), you will get unhelpful compiler errors for n.subscribe()!

Example from http://www.ros.org/wiki/ROS/Tutorials/WritingPublisherSubscriber%28c%2B%2B%29
Python Version at http://www.ros.org/wiki/ROS/Tutorials/WritingPublisherSubscriber%28python%29

msg & srv

- msg are custom datatypes for publish/subscribe
  - We cheated and used std_msgs::String
- Put in msg/ directory
  - .msg suffix
  - uncomment #rosbuild_genmsg() in CMakeLists.txt

srv/num.msg

```
int64 num
```

- srv are RPC function prototypes
  - e.g. "Add"
  - input: int A, int B
  - output: int Sum
- Put in srv/ directory
  - .srv suffix
  - uncomment #rosbuild_gensrv() in CMakeLists.txt

srv/AddTwoInts.srv

```
int64 A
int64 B
---
int64 Sum
```

```
$ roscd beginner_tutorials && mkdir srv && makdir msg
# edit srv/AddTwoInts.srv and msg/num.msg and CMakeLists.txt
$ make
$ rosed beginner_tutorials AddTwoInts.h
$ rosed beginner_tutorials _AddTwoInts.py
# Also AddTwoInts.java, AddTwoInts.lisp, beginner_tutorials_AddTwoInts.m
```

Hello World (2)

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;
}
```

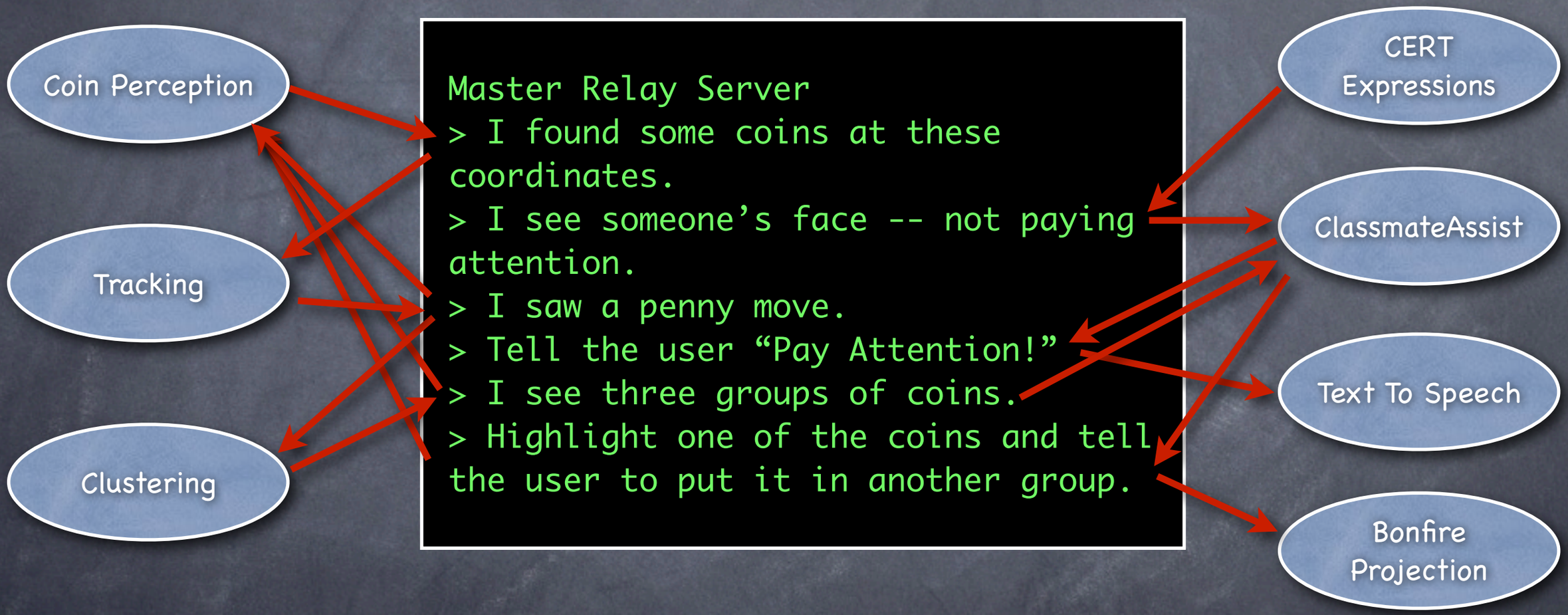
src/add_two_ints_client.cpp

```
#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::AddTwoInts>
                                      ("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;
}
```

Translation

- Translations of the HelloWorlds are available in the ROSTutorial.pdf document for Python & Java
- rosjava is based on an old API, and is not quite 1-to-1
  - Needs JNI update (pretty straightforward)
    - Change node name from commandline (duplicate nodes)
    - Check for user ctrl+c (hacked in by Nick)
  - Improvements will help all rosjava users.

Overview of RUBIOS



- Chat-Room architecture, simple socket message.
- Different processes connect - multi-language, multi-architecture, multi-machine.
- Each sends whatever messages it wants
- All messages heard by AllNodes, they decide whether a message is relevant

RUBI-ROS / RubiRosRelay

- Each node publishes and subscribes to messages of type std_msgs::String.
- All messages are sent to the topic AllNodes
  - Additional topic "AllNodesIncoming" for potential bridge messages.

```
public RUBIOSNode(String[] args, double _dt, int nStateDimensions) {
    dt = _dt;
    q1 = new LinkedList<String>();
    q2 = new LinkedList<String>();
    ros = Ros.getInstance();
    ros.init(new CurrentClassGetter().getClassName());
    n  = ros.createNodeHandle();
    callback=
    new Subscriber.Callback<ros.pkg.std_msgs.msg.String>() {
          public void call(ros.pkg.std_msgs.msg.String msg) {
               q1.offer(msg.data);
          }
    };
    try {
          pub = n.advertise("AllNodes",
                            new ros.pkg.std_msgs.msg.String(),
                            100);
          sub = n.subscribe("AllNodes",
                            new ros.pkg.std_msgs.msg.String(),
                            callback,
                            100);
          sub = n.subscribe("AllNodesIncoming",
                            new ros.pkg.std_msgs.msg.String(),
                            callback, 100);
    } catch (RosException e){
          ros.logError("Failed to connect to ROS. Is roscore started?");
          return ;
    }
}

public final void sendMessage(String msg) {
    ros.pkg.std_msgs.msg.String rosmsg = new ros.pkg.std_msgs.msg.String();

    rosmsg.data = msg;

    pub.publish(rosmsg);
}
```

CMake

- CMake is an advanced scripting language specifically designed to build large projects.
- A few built-in commands, plus user-defined macros.
- ros defines custom cmake functionality via macros in .cmake files.
- Packages can export .cmake files to help dependent packages build smoothly.

CMakeLists.txt

- CMake commands:
  - cmake_minimum_required, include, set, string, add_custom_target
- ros macros:
  - rosbuild_init, rosbuild_gensrv
- rosjava macros:
  - add_java_source_dir, rospack_add_java_executable, add_deps_classpath, add_java_source_dir_internal

rosjava_tutorials/CMakeList.txt

```
cmake_minimum_required(VERSION 2.4.6)
include($ENV{ROS_ROOT}/core/rosbuild/rosbuild.cmake)
rosbuild_init()

#set the default path for built executables to the "bin" directory
set(EXECUTABLE_OUTPUT_PATH ${PROJECT_SOURCE_DIR}/bin)

#uncomment if you have defined messages
#rosbuild_genmsg()
#uncomment if you have defined services
rosbuild_gensrv()

add_java_source_dir(${PROJECT_SOURCE_DIR}/src)
rospack_add_java_executable(Talker Talker)
rospack_add_java_executable(Listener Listener)
rospack_add_java_executable(AddTwoIntsServer AddTwoIntsServer)
rospack_add_java_executable(AddTwoIntsClient AddTwoIntsClient)
```

RubiosNodeAPIJava/CMakeList.txt

```
cmake_minimum_required(VERSION 2.4.6)
include($ENV{ROS_ROOT}/core/rosbuild/rosbuild.cmake)
rosbuild_init()

rosbuild_find_ros_package(rosjava)

#include(${rosjava_PACKAGE_PATH}/cmake/rosjava.cmake)

add_deps_classpath()
set(_targetname _java_compile_${JAVA_OUTPUT_DIR})
string(REPLACE "/" "_" _targetname ${_targetname})
add_custom_target(${_targetname} ALL)
add_java_source_dir_internal(${_targetname} ${PROJECT_SOURCE_DIR}/src)
```

Exercise

- We want an "externalLibs" directory of jar files that all our nodes can link to.
- rosjava has macros for adding paths to the java compile and runtime classpaths.
- Create and "externalLibs" package that uses <export> in the manifest to export a cmake file that uses rosjava's macros to add .jar files to the java classpath.

Solution

externalLibs/cmake/externalLibs.cmake

```
# Add all the jar files under a given directory to the classpath
macro(add_jar_dir _jardir)
    file(GLOB_RECURSE _jar_files ${_jardir}/*.jar)
    message("\n\n!!!!!!!! Glob found files ${_jar_files}\n\n")
    foreach(_jar ${_jar_files})
           message("\n\n!!!!!!!! Adding ${_jar} to classpath\n\n")
        add_classpath(${_jar})
        add_runtime_classpath(${_jar})
    endforeach(_jar)
endmacro(add_jar_dir)

rosbuild_find_ros_package(externalLibs)
message("\n\n!!!!!!!! Adding jars in ${externalLibs_PACKAGE_PATH}\n\n")
add_jar_dir(${externalLibs_PACKAGE_PATH}/jar)
```

externalLibs/manifest.xml

```
 ...
 <depend package="roslang"/>
 <export>
   <roslang cmake="${prefix}/cmake/externalLibs.cmake"/>
 </export>
...
```

Making an API

NMPT/CMakeLists.txt

```
cmake_minimum_required(VERSION 2.4.6)
include($ENV{ROS_ROOT}/core/rosbuild/rosbuild.cmake)
rosbuild_init()

set(OPENCV /opt/local)
set(EXECUTABLE_OUTPUT_PATH ${PROJECT_SOURCE_DIR}/bin)
set(LIBRARY_OUTPUT_PATH ${PROJECT_SOURCE_DIR}/lib)
find_library(QTKIT QTKit)
find_library(FOUNDATION Foundation)
find_library(QUARTZCORE QuartzCore)
find_library(APPKIT AppKit)

message("Searching ${OPENCV} for cxcore")

find_library(CXCORE NAMES cxcore PATHS ${OPENCV}/lib NO_DEFAULT_PATH)
message("Found ${CXCORE}")
find_library(CV cv PATHS ${OPENCV}/lib NO_DEFAULT_PATH)
find_library(HIGHGUI highgui PATHS ${OPENCV}/lib NO_DEFAULT_PATH)

include_directories(${OPENCV}/include/opencv include/NMPT)


rosbuild_add_library(${PROJECT_NAME} src/BlockTimer.cpp src/BoxFeature.cpp
    src/ConvolutionalLogisticPolicy.cpp src/DetectionEvaluator.cpp
    src/FastPatchList.cpp src/FastSaliency.cpp src/Feature.cpp
    src/FeatureRegressor.cpp src/GentleBoostCascadedClassifier.cpp
    src/GentleBoostClassifier.cpp src/HaarFeature.cpp src/ImageDataSet.cpp
    src/ImagePatch.cpp src/ImagePatchPyramid.cpp src/MIPOMDP.cpp
    src/MultinomialObservationModel.cpp src/MultiObjectTrackingState.cpp
    src/NCvCapture.mm src/ObjectDetector.cpp src/OpenCVBoxFilter.cpp
    src/OpenCVHaarDetector.cpp src/OpenLoopPolicies.cpp
    src/PatchDataset.cpp src/PatchList.cpp)

target_link_libraries(${PROJECT_NAME} ${CXCORE} ${CV} ${HIGHGUI} ${QTKIT}
      ${FOUNDATION} ${QUARTZCORE} ${APPKIT})

rosbuild_add_executable(TestQTKitCapture src/TestQTKitCapture.cpp)
```

NMPT/manifest.xml

```
...

  <depend package="roscpp"/>
  <export>
    <cpp os="osx"
         cflags="-I${prefix}/include -I/opt/local/include/opencv
-I/opt/local/include"
         lflags="-L${prefix}/lib -L/opt/local/lib -lNMPT -
lcxcore -lcv -lhighgui"/>
  </export>
 ...
```

Using an API

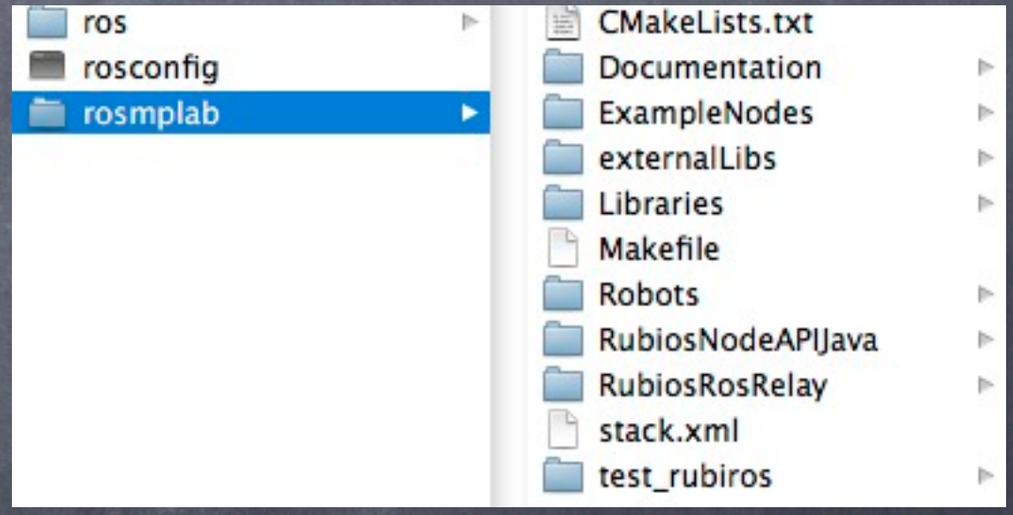
SaliencyNode/manifest.xml

```
...
 <depend package="NMPT"/>
...
```

rosmplab

```
$ svn co svn+ssh://mplab.ucsd.edu/rosmplab
```

- rosmplab is a ros "Stack", i.e. something that ros knows about that holds several packages (identified by stack.xml)
- Make sure to add rosmplab to the relevant environment variables in ~/.bashrc.ros



OpenCV

- For consistency and ease of linking, OpenCV should be installed via MacPorts
- On 10.6, OpenCV should be built with the flags:
  - --without-quicktime --without-carbon --with-gtk
- Instructions here: http://blog.mgrundmann.com/?p=5
- This disables portions of OpenCV that depend on Quicktime, but they are reenabled via the NMPT package in rosmplab.
  - CvCapture -> NCvCapture

Unsolved Mysteries

- Prospect of ROS under Windows
  - Tingfan is stuck on log4cxx
- Spec for implementing ROS-compatible code (e.g. Java, Flash)
- How to export resources, e.g. xml files for classifiers.
- Why does <export cmake> export to non-dependent packages? Is there a way to prevent this?
- How to apply NCvCapture patch to the full OpenCV