Note: This tutorial assumes that you have completed the previous tutorials: Hello World Publisher (/rosserial_arduino/Tutorials/Hello%20World).

Please ask about problems and questions regarding this tutorial on answers.ros.org (http://answers.ros.org). Don't forget to include in your question the link to this page, the versions of your OS & ROS, and also add appropriate tags.

CMake with rosserial_arduino

Description: This tutorial shows how to use the CMake build system with rosserial_arduino.

Tutorial Level: ADVANCED

Next Tutorial: Servo Controller (/rosserial_arduino/Tutorials/Servo%20Controller)

When you are doing large software projects, the Arduino IDE quickly becomes unwieldy. You often want to be able to compile your project from the command line or use a different IDE like Eclipse where you can use autocompletion. Finally, by using rosserial_client CMake infrastructure, you can build and distribute firmwares using the ROS buildfarm.

For this tutorial, we are going to create a simple hello world firmware.

catkin | rosbuild

Contents

New in Indigo You can now again build rosserial firmwares and other clients inside your regular ROS packages. This functionality is available from Indigo onward. The installation of the rosserial_arduino (/rosserial_arduino) package now also installs arduino-core (http://packages.ubuntu.com/trusty/arduino-core), so there's nothing additional needed to make this work.

1. Making Your Project

Starting your rosserial_arduino project is just like creating any other package. In your catkin workspace's `src` folder:

```
catkin_create_pkg helloworld rosserial_arduino rosserial_client std_msgs
```

As usual, use `catkin_create_pkg` to create a package named `helloworld`. You must depend on rosserial_arduino for the Arduino toolchain, and on rosserial_client for client library generation macros. And finally, since we are going to use std_msgs/String (http://docs.ros.org/api/std_msgs/html/msg/String.html) messages, you must depend on std_msgs (/std_msgs).

2. Source Code

Copy the the source code below and make a file called `firmware/chatter.cpp` in your helloworld package.

Toggle line numbers

```
#include <ros.h>
#include <std_msgs/String.h>

#include <Arduino.h>

ros::NodeHandle nh;

std_msgs::String str_msg;
ros::Publisher chatter("chatter", &str_msg);

char hello[13] = "hello world!";

void setup()
{
  nh.initNode();
  nh.advertise(chatter);
}

void loop()
{
  str_msg.data = hello;
  chatter.publish( &str_msg );
  nh.spinOnce();
  delay(1000);
}
```

This program is almost exactly the same as the hello world covered in the Publisher Tutorial (/rosserial_arduino/Tutorials/Hello%20World).

When you are compiling a cpp file outside of the Arduino IDE, you need to explicitly include a header file which contains all of the Arduino functions (digitalRead, analogRead, delay, etc.).

If you are unsure if you are going to be using a file with the Arduino IDE versus CMake, just add this line at the top of your file. It never hurts and it makes sure your file is always compatible with non-Arduino IDE build systems.

3. CMakeLists.txt

Open the CMakeLists.txt in your package directory and replace the contents with the below:

Toggle line numbers

```
cmake_minimum_required(VERSION 2.8.3)
project(helloworld)

find_package(catkin REQUIRED COMPONENTS
  rosserial_arduino
  rosserial_client
)

catkin_package()

rosserial_generate_ros_lib(
  PACKAGE rosserial_arduino
  SCRIPT make_libraries.py
)

rosserial_configure_client(
  DIRECTORY firmware
  TOOLCHAIN_FILE ${ROSSERIAL_ARDUINO_TOOLCHAIN}
)

rosserial_add_client_target(firmware hello ALL)
rosserial_add_client_target(firmware hello-upload)
```

With rosserial_client (/rosserial_client)'s CMake scripts, we are not actually building the firmware directly, but configuring a separate CMake project, and passing through targets from the catkin package to the sub-project.

The `rosserial_generate_ros_lib` function creates a target called helloworld_ros_lib, which will generate the rosserial client library, including messages headers.

The `rosserial_configure_client` function creates a target which will configure the CMake project in the specified subdirectory, optionally using the supplied toolchain. In this case, we use the Arduino toolchain, helpfully provided by rosserial_arduino (/rosserial_arduino).

Finally, the `rosserial_add_client_target` calls each pass through targets, so that when you run make the `helloworld_firmware_hello` catkin target, it will configure the `firmware` directory and build the `hello` target therein.

Now, we actually need a second `CMakeLists.txt`, and that's the one for the firmware subproject. Create the file `firmware/CMakeLists.txt` in your package, with the following contents:

Toggle line numbers

```
cmake_minimum_required(VERSION 2.8.3)

include_directories(${ROS_LIB_DIR})

# Remove this if using an Arduino without native USB (eg, other than Leonardo)
add_definitions(-DUSB_CON)

generate_arduino_firmware(hello
  SRCS chatter.cpp ${ROS_LIB_DIR}/time.cpp
  BOARD leonardo
  PORT /dev/ttyACM0
)
```

The `generate_arduino_firmware` function is provided by the arduino-cmake toolchain (https://github.com/queezythegreat/arduino-cmake), which this uses. It handles the intricacies of locating Arduino, linking any libraries you use, and so on.

You should be done!

4. Build & Test

The firmware should build by default when you run `catkin_make`, but you can also specify it explicitly:

```
catkin_make helloworld_firmware_hello
```

Now connect an Arduino Leonardo, confirm that it comes up as `/dev/ttyACM0` (or change the firmware's `CMakeLists.txt` accordingly), and program it:

```
catkin_make helloworld_firmware_hello-upload
```

Now you can use the Arduino! Launch the following in separate terminals to see it in action:

```
roscore
```

```
rosrun rosserial_python serial_node.py /dev/ttyACM0
```

```
rostopic echo chatter
```

5. Additional Info

The CMake script which supplies these functions lives here (https://github.com/ros-drivers/rosserial/blob/indigo-devel/rosserial_client/cmake/rosserial_client-extras.cmake), if you want to inspect it to understand better what is going on.

If there's a breakage you don't understand, a good first step is to clean your workspace (delete build and devel trees), and then re-run in verbose mode:

```
catkin_make VERBOSE=1
```