



Building ROS 2 enabled Android apps with C++ □

Shane Loretz, ROSCon 2022

Android is a trademark of Google LLC.



- **Why Android and ROS?**
 - Comparison with existing projects
 - How to build Apps with only C++?
 - What are the downsides?

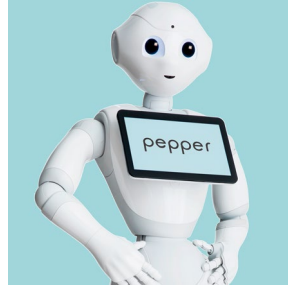
Why Android + ROS?



Because Android is useful to Robots



Spot
Tablet
Boston
Dynamics



Pepper
SoftBank
Robotics



Astrobee
NASA

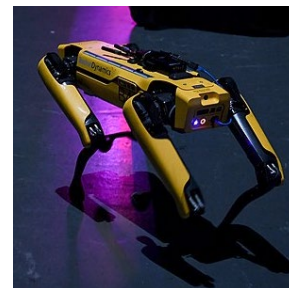
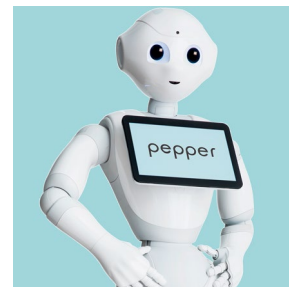
Spot image: Web Summit, CC BY 2.0 <<https://creativecommons.org/licenses/by/2.0/>>, via Wikimedia Commons (edited)

Pepper Image: Softbank Robotics Europe, CC BY-SA 4.0 <<https://creativecommons.org/licenses/by-sa/4.0/>>, via Wikimedia Commons (edited)

Because Android is useful to Robots

Android is used by

- Astrobbee to enable Guest Science apps
- Pepper to show relevant information to users
- Spot Tablet to control Spot



Spot image: Web Summit, CC BY 2.0 <<https://creativecommons.org/licenses/by/2.0/>>, via Wikimedia Commons (edited)

Pepper Image: Softbank Robotics Europe, CC BY-SA 4.0 <<https://creativecommons.org/licenses/by-sa/4.0/>>, via Wikimedia Commons (edited)

Comparison with existing projects



What already exists?

ROS projects usable on Android

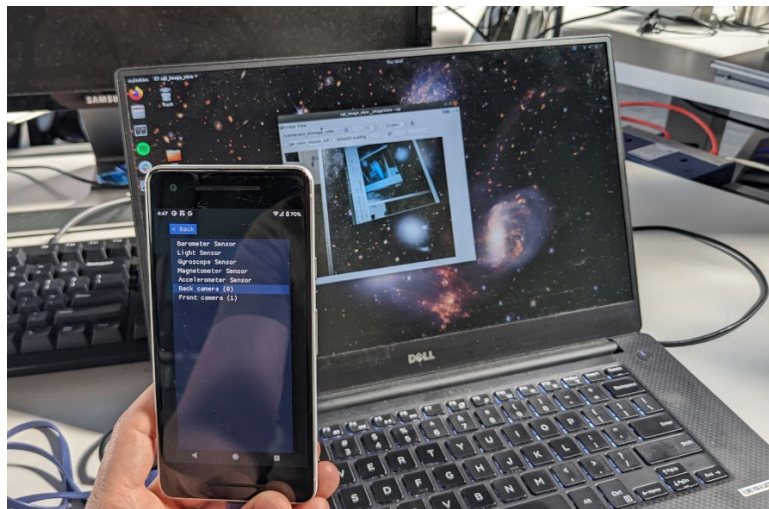
	Purpose	ROS 1 or ROS 2?
rosjava	Client Library	1
ROS-Mobile-Android	App	1
ros2_java (aka rcljava)	Client Library	2

What already exists?

ROS projects usable on Android

	Purpose	ROS 1 or ROS 2?	Cross-compiles ROS packages?
rosjava	Client Library	1	No
ROS-Mobile-Android	App	1	No (uses rosjava)
ros2_java (aka rcljava)	Client Library	2	Yes (up to rcl)
Sensors for ROS	App	2	Everything (No Java or Kotlin)

What is Sensors for ROS?



It's an Android app that publishes sensor data to ROS 2 topics.

What is Sensors for ROS?

Supported sensors:

- Accelerometer
- Barometer
- Camera(s)
- Gyroscope
- Illuminance
- Magnetometer

What is Sensors for ROS?

- It uses CMake, C++, and rclcpp
- No Java* or Kotlin!

```
$ mkdir build
$ cd build
$ cmake ../ -DANDROID_HOME=/path/to/android-sdk/
...
$ make -j`nproc`
...
```

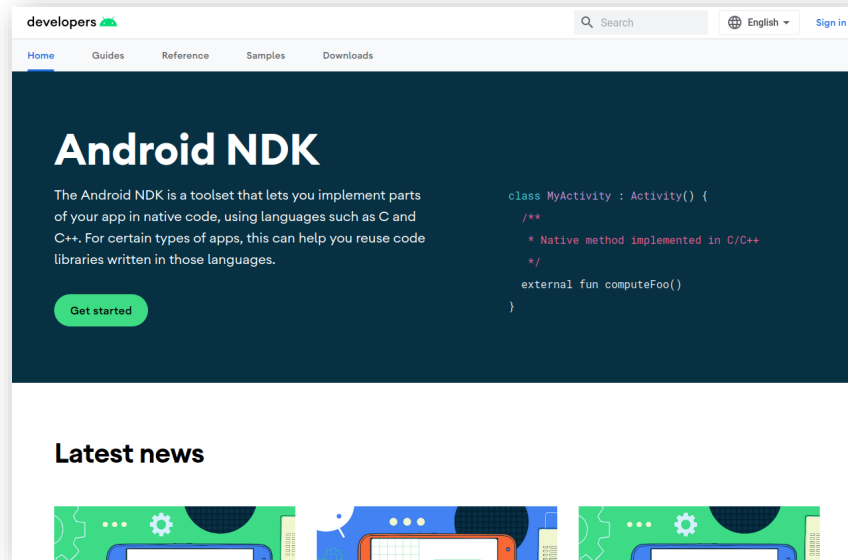
* We won't write any Java, but we still have to interact with the JVM.

How to build Apps with only C++?




```
[ 45%] Built target deps-tracetools
[ 45%] Built target deps-rcutils
[ 47%] Built target deps-rcpputils
[ 47%] Built target deps-rosidl_runtime_c
[ 48%] Built target deps-rcl_logging_interface
[ 49%] Built target deps-rosidl_runtime_cpp
[ 50%] Built target deps-rmw
[ 51%] Built target deps-rosidl_typesupport_c
[ 52%] Built target deps-rcl_logging_noop
[ 53%] Built target deps-rcl_logging_rcutils
[ 55%] Built target deps-rcl_logging_spdlog
[ 55%] Built target deps-rcl_logging_android
[ 56%] Built target deps-rosidl_typesupport_introspection_cpp
[ 57%] Built target deps-rcl_yaml_param_parser
[ 58%] Built target deps-rosidl_typesupport_cpp
[ 59%] Built target deps-builtin_interfaces
[ 60%] Built target deps-lifecycle_msgs
[ 61%] Built target deps-rmw_dds_common
[ 62%] Built target deps-std_srvs
[ 64%] Built target deps-std_msgs
[ 64%] Built target deps-statistics_msgs
[ 65%] Built target deps-rcl_interfaces
[ 66%] Built target deps-rmw_cyclonedds_cpp
[ 67%] Built target deps-rosgraph_msgs
[ 70%] Built target deps-rmw_implementation
[ 70%] Built target deps-geometry_msgs
[ 70%] Built target deps-actionlib_msgs
[ 71%] Built target deps-composition_interfaces
[ 73%] Built target deps-rcl
[ 73%] Built target deps-trajectory_msgs
[ 74%] Built target deps-diagnostic_msgs
[ 77%] Built target deps-nav_msgs
[ 77%] Built target deps-rcl_lifecycle
[ 77%] Built target deps-sensor_msgs
[ 78%] Built target deps-shape_msgs
[ 79%] Built target deps-libstatistics_collector
[ 81%] Built target deps-visualization_msgs
[ 81%] Built target deps-stereo_msgs
[ 82%] Built target deps-rclcpp
[ 83%] Built target deps-rclcpp_lifecycle
[ 83%] Performing build step for 'hello_android'
make[3]: warning: jobserver unavailable: using -j1. Add '+' to parent make rule.
[ 25%] Built target imgui
[100%] Built target android-ros
[ 83%] Performing install step for 'hello_android'
[ 25%] Built target imgui
[100%] Built target android-ros
Install the project...
-- Install configuration: "RELEASE"
-- Up-to-date: /home/sloretz/android_ros/build/hello_android/lib/libandroid-ros.so
[ 84%] Completed 'hello_android'
[ 85%] Built target hello_android
[100%] Built target apk
sloretz@sloretz-XPS-15-9560 ~/a/build> █
```

Android Native Development Kit



The screenshot shows the Android NDK page on the developers.android.com website. The page has a dark blue header with the title "Android NDK" and a sub-header "The Android NDK is a toolset that lets you implement parts of your app in native code, using languages such as C and C++. For certain types of apps, this can help you reuse code libraries written in those languages." Below the text is a green "Get started" button. To the right of the text is a code snippet in C++ showing a native method implementation. Below the main content is a "Latest news" section with a decorative banner featuring icons of a smartphone, a gear, and a document.

developers 

Search English Sign in

Home Guides Reference Samples Downloads


Android NDK

The Android NDK is a toolset that lets you implement parts of your app in native code, using languages such as C and C++. For certain types of apps, this can help you reuse code libraries written in those languages.

[Get started](#)

```
class MyActivity : Activity() {  
    /**  
     * Native method implemented in C/C++  
     */  
    external fun computeFoo()  
}
```

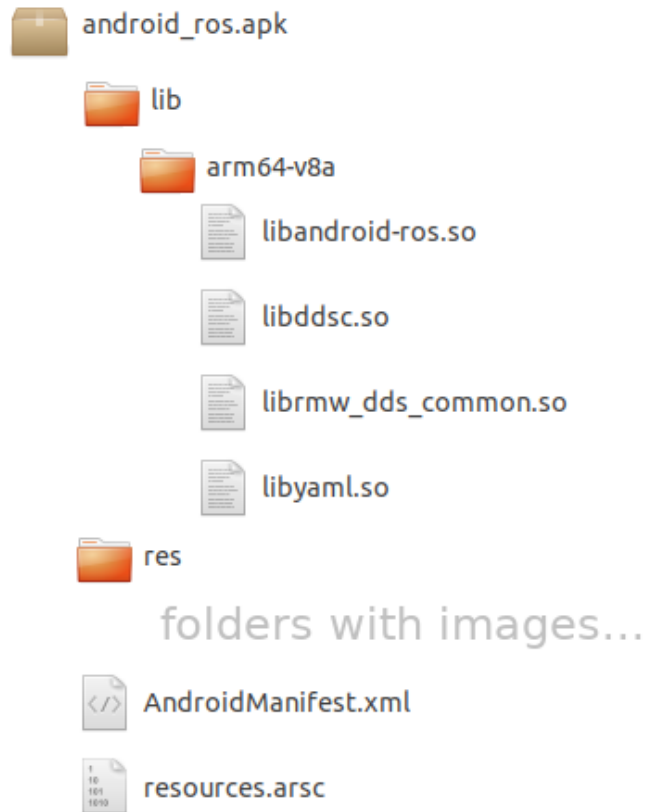
Latest news



“The Android NDK is a toolset that lets you implement parts of your app in native code, using languages such as C and C++. For certain types of apps, this can help you reuse code libraries written in those languages. “

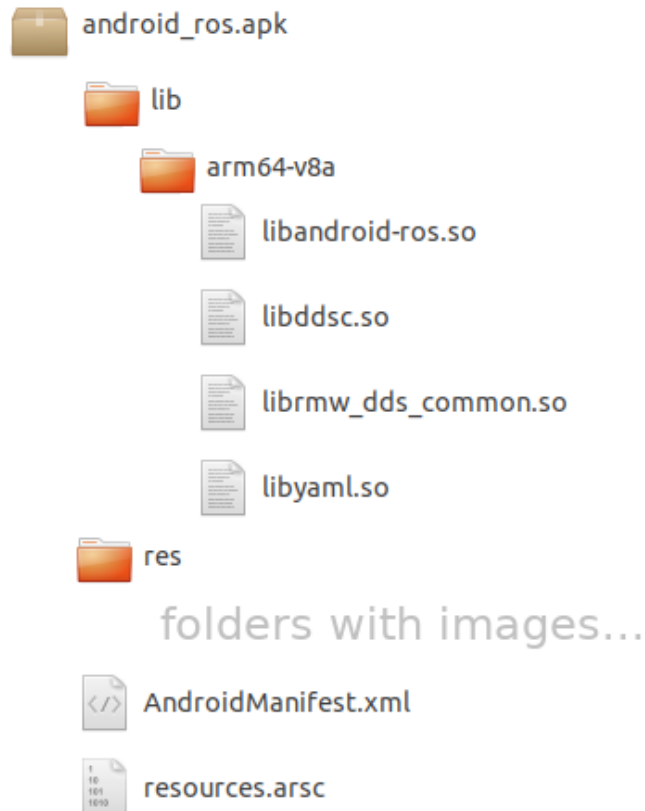
What is an Android App?

- It's an .apk
- Zip archive containing:
 - Compiled code
 - Resources
 - App Icon
 - Strings
 - A manifest



Steps to make an App

1. Create a manifest
2. Cross-compile code
3. Convert resources
4. Package it into an .apk





1. Creating the manifest

Create the manifest

- Hand written
- Describes what's in the app
 - Required permissions
 - **Activity** (ies) in the app

2. Cross-compiling everything

Cross compile everything from source

- Anything that runs on the device
 - ROS 2 Humble
 - `rcl_logging_android`
 - App specific code
- Not cross-compiling build tool dependencies

2. Cross-compiling everything

1. Get ROS source code with `roscpp` and `vcstool`
2. Use CMake toolchain provided by NDK

2a. Making a native Activity

An App needs an Activity

- Entry point for an Android App
- Apps have a “main” activity
- An **Activity** is a Java class

2a. Making a native Activity

ANativeActivity

- C API provided by Android NDK
- Using it requires:
 - Creating shared library
 - Implementing an “on create” function
 - Implementing **Activity** callbacks

3. Converting resources

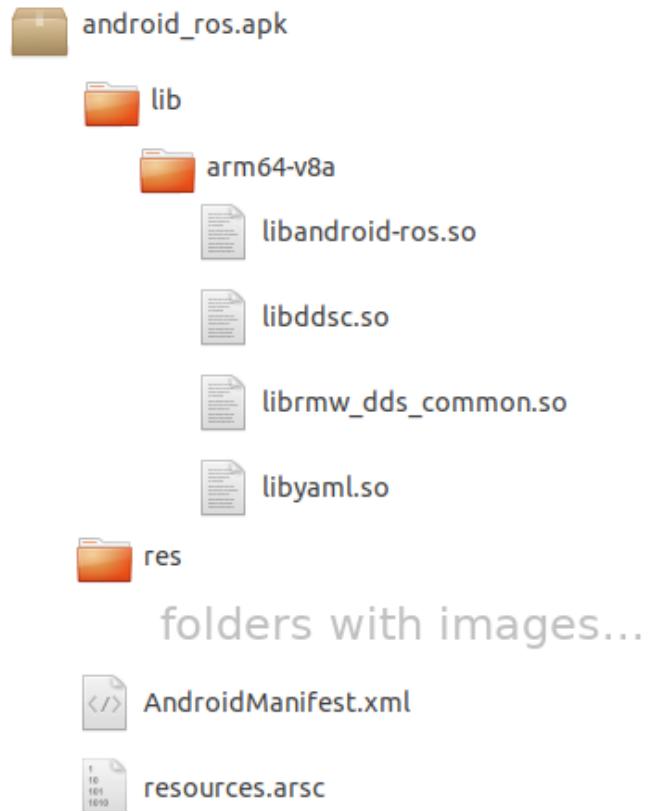
Resources

- The App icon
- Localized strings
- Use **aapt2** to convert to Android format

4. Packaging an App

Packaging the .apk

- Create folder structure
- Create a zip archive
- use **zipalign** on it
- sign it with **apksigner**

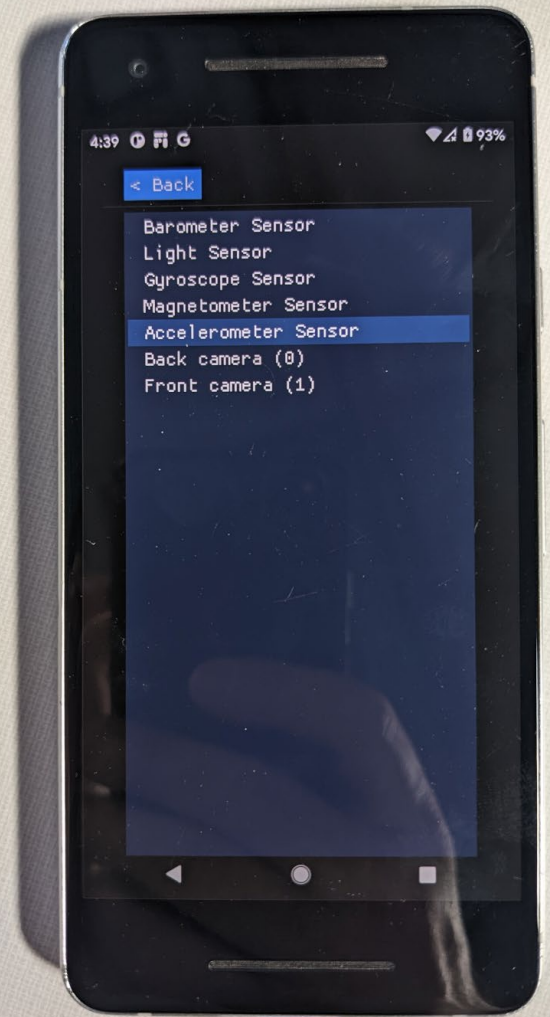


4. Packaging an App

Tools needed to make APKs

- aapt2
- zipalign
- apksigner

What are the downsides?



What are the downsides?

ANativeActivity requires using OpenGL

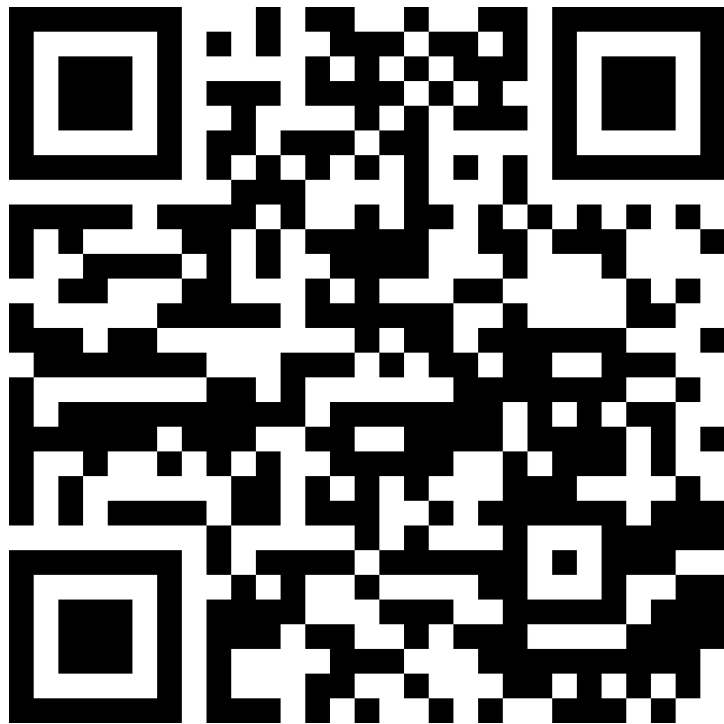
- Can't use native GUI widgets
- Can't use Android XML UI layouts
- **Sensors for ROS** uses **Dear ImGui**

What are the downsides?

Some APIs aren't in the NDK

- Requesting CAMERA permissions
 - Workaround: Call Java APIs using Java Native interface (JNI)

Building ROS 2 enabled Android apps with C++



https://github.com/sloretz/sensors_for_ros

https://github.com/sloretz/rcl_logging_android