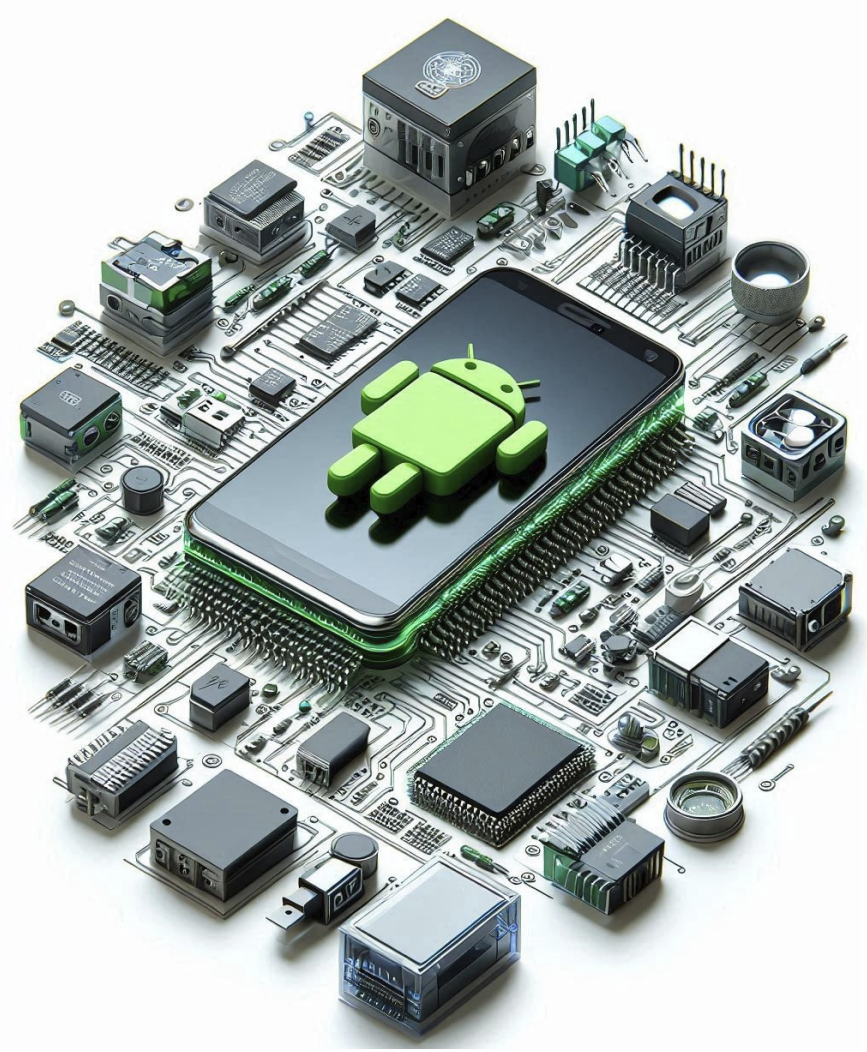
Android Automotive

AOSP Customization Training Track – ITI 2024

Islam Samak



Timeline

- Week 1: Introduction to AOSP Automotive
  - 1.1 Introduction to AOSP and Android Automotive
  - 1.2 Setting Up the Environment
  - 1.3 Basic Android Build System and Commands
- Week 2: Android Automotive Architecture and Framework
  - 2.1 Android Automotive Architecture
  - 2.2 Car Service, Car API, and Car Framework
  - 2.3 Exploring HAL for Automotive

Soong Build System

Lab3

Android App

1) Create ***Android.bp*** in the app's directory
2) Add ***android_app*** object with attributes

1)name	*M*odule name => typical to app's name
2)srcs	Path to Java/Kotlin source files
3)resources_dirs	Path to resources files
4)sdk_version	"current" sdk, or specify SDK version
5)package_name	App's package identifier
6)certificate	Signing certificate "platform, presigned ..."
7)privileged	Grant additional permissions for system apps
8)product_specific	Indicates App should be available only on this product

Android App

```
android_app {
      name: "MyCustomApp",
      srcs: ["src/**/*.java"],
      resource_dirs: ["res"],
      sdk_version: "current",
      package_name: "com.example.mycustomapp",
      certificate: "platform",
      privileged: true,
      product_specific: true,
}
```

Custom Service (Binary Service)

1) Create ***Android.bp*** in the service's directory
2) Add ***cc_binary*** object with attributes

1)name	Module name
2)srcs	Path to C/C++ source files
3)cflags	Additional compiler flags
4)static_libs	Static libraries required for the service
5)shared_libs	Shared libraries required for the service
6)init_rc	Specifies an init.rc file to start the service during boot (used to define service configurations like permissions)
7)vendor	Indicates the service is specific to vendor implementations (especially useful for automotive or customized hardware-related services).

3) Add ***cc_library*** -optionally- object as native library

Binary Service

```
cc_binary {
      name: "my_custom_service",
      srcs: ["service/my_custom_service.cpp"],
      cflags: ["-Wall"],
      static_libs: ["libutils", "libbinder"],
      shared_libs: ["libcutils"],
      init_rc: ["init.my_custom_service.rc"],
      vendor: true,
}
cc_library {
      name: "my_custom_lib",
      srcs: ["src/**/*.cpp"],
      shared_libs: ["libc", "libm"],
      stl: "libc++",
      include_dirs: ["include"],
      vendor_available: true,
      sdk_version: "current",
}
```

Custom Service (Java Library/Service)

1) Create ***Android.bp*** in the service's directory
2) Add ***`cc_binary`*** object with attributes

`1)name`	*M*odule name
`2)srcs`	Path to Java source files
`3)sdk_version`	Set the SDK version
`4)installable`	Indicates the service can be installed in the system
`5)static_libs`	Java libraries required by the service

Java Library/Service

```
java_library {
     name: "MyCustomJavaService",
     srcs: ["src/**/*.java"],
     sdk_version: "current",
     installable: true,
     static_libs: ["android.frameworks.some_library"],
}
```

Custom System Configuration

System configurations could be:

- Custom properties
- Permissions

Defined as:

- ***prebuilt_etc*** to install configuration files ***<partition>/etc/<sub_dir> directory***
- ***sysprop_library*** to define system properties

Example: prebuilt_etc

```
prebuilt_etc {
     name: "my_custom_config",
     src: "config/my_custom_config.xml",
     sub_dir: "system/etc", // Target directory for
installation
}
```

Example: sysprop_library

```
sysprop_library {
     name: "my_custom_sysprop",
     src: [
          "sysprop/my_custom_sysprop.sysprop",
     ],
}
```

src = Path to the .sysprop file, which defines custom properties accessible by other components on the system

Setting Dependencies Between Modules

- Inside the ***android_app*** object add attributes
  - *1)static_libs* Includes the Prebuilt and/or Custom libraries

```
android_app {
     name: "MyCustomApp",
     srcs: ["src/**/*.java"],
     resource_dirs: ["res"],
     static_libs: ["my_custom_library"],
     sdk_version: "current",
}
```

Advanced Properties in (*android_app*)

- ***overlays***:
  - Define the paths to resources overlays for customization UI based on different build variant
- ***dex_preopt:***
  - Optimizes Dex files for faster app startup
- ***jni_libs***
- ***aaptflags:***
  - Custom AAPT (Android Asset Packaging Tool) flags for additional control over resource processing, useful for enforcing optimizations.
- ***required:***
  - Lists `system features required by the app`

Advanced Properties in (*cc_binary*)

- ***`init_rc`***:
  - Define the paths to resources overlays for customization UI based on different build variant
- ***`relative_install_path:`***
  - Defines a custom path for the binary within the target directory. This helps organize binaries for different vendors.
- ***`group & user:`***
  - Sets the user and group permissions, which are critical for security, especially in automotive environments.
- ***`vendor_available:`***
  - Makes the binary available to the vendor partition, often used for vendor-specific implementations in automotive and other embedded systems.

Advanced Properties in (*prebuilt_etc*)

```
prebuilt_etc {
    name: "my_config_file",
    src: "config/my_config.xml",
    sub_dir: "system/etc/custom_config",
    owner: "system",          // Owner of the config file
    group: "system",          // Group of the config file
    mode: "0644",             // File permissions
}
```

Advanced Properties in (*sysprob_library*)

```
sysprop_library {
    name: "my_custom_properties",
    src: ["sysprop/my_custom_properties.sysprop",],
    scope: "internal | public"
    api_packages: ["com.example.sysprops"], // Packages
 that can access these properties
 }
```

`scope`: Sets access level of the properties.
"*internal*" limits it to system components
"*public*" is available to apps.

`api_packages`: Specifies packages allowed to access these properties, providing fine-grained control for sensitive data.