

## G4 - Android Porting and Internals

### Porting the Android System and Framework

#### Objectives

- Understanding the architecture of the Android system.
- Learn to use GIT version manager to download and manage Android sources.
- Learn how to install Linux on your hardware and create a BSP
- Discover the Android version of the Linux kernel.
  - New drivers and IPCs
  - Power Management
- Explore the Android system architecture
  - The Android init process
  - System services
  - The Android Binder
  - The Android Application Framework
  - The Android Hardware Abstraction Layer
  - The Android Multimedia Framework and OpenMAX
- Learn how to install Android on a platform that already supports Linux

*Labs are conducted on i.MX6 or i.MX8 boards*

*We use the last open source version of Android, as available on the board.*

*For on-site trainings, if suitable Linux workstations are not available, we provide virtual machine images for VirtualBox; the only requisite is then a recent 64bit PC with at least 8Gb of RAM and 100Gb of free disk space.*

#### Who should attend this course?

- Engineers that must install an Android platform on a new board
  - Porting the Linux kernel from a supported SoC
  - Adapting the Android Frameworks to the board hardware
  - Tailoring the Android System
  - Connecting hardware graphics accelerators to the Android framework

#### Prerequisite

- Embedded Linux experience (see our [D1 - Embedded Linux with Buildroot and Yocto](#) course)
- Good Linux kernel and driver programming experience (see our [D3 - Linux Drivers](#) course)
- Good C++ and Java programming skills (see our [L3 - Embedded C++](#) course and [L4G - Java for Android](#) courses)

#### Course Environment

- Theoretical course
- PDF course material (in English) supplemented by a printed version.
- The trainer answers trainees' questions during the training and provide technical and pedagogical assistance.
- Practical activities
- Practical activities represent from 40% to 50% of course duration.
- Code examples, exercises and solutions
- One PC (Linux ou Windows) for the practical activities with, if appropriate, a target board.
- One PC for two trainees when there are more than 6 trainees.
- For onsite trainings:

- An installation and test manual is provided to allow preinstallation of the needed software.
- The trainer come with target boards if needed during the practical activities (and bring them back at the end of the course).
- Downloadable preconfigured virtual machine for post-course practical activities
- At the start of each session the trainer will interact with the trainees to ensure the course fits their expectations and correct if needed

### **Target Audience**

- Any embedded systems engineer or technician with the above prerequisites.

### **Evaluation modalities**

- The prerequisites indicated above are assessed before the training by the technical supervision of the trainee in his company, or by the trainee himself in the exceptional case of an individual trainee.
- Trainee progress is assessed in two different ways, depending on the course:
- For courses lending themselves to practical exercises, the results of the exercises are checked by the trainer while, if necessary, helping trainees to carry them out by providing additional details.
- Quizzes are offered at the end of sections that do not include practical exercises to verify that the trainees have assimilated the points presented
- At the end of the training, each trainee receives a certificate attesting that they have successfully completed the course.
- In the event of a problem, discovered during the course, due to a lack of prerequisites by the trainee a different or additional training is offered to them, generally to reinforce their prerequisites, in agreement with their company manager if applicable.

## **Plan**

### **First Day**

#### **Android Overview**

- Linux
- Android
- Android licensing

#### **The GIT and REPO distributed source management system (reminders)**

- Installation and general usage.
- Creating and using a local repository
- Cloning a remote repository
- Working with branches
- Creation of a new branch
- Merging branches
- Team functions
- Creating configuring and managing a public repository
- Working with patches
- The repo tool
- The manifest file
- Downloading the Android source tree
- Adding projects to the manifest
- The Android development process
- Android code lines
- Submitting changes through Gerrit

#### **The Android Linux kernel enhancements**

- Downloading source code
- Adapting the Android-specific kernel drivers
- Alarm
- Ashmem
- Logger
- Low\_memory\_killer
- Timer\_gpio
- Timed\_output
- Buttons and Keypad management
- The Android Binder architecture
- Why a new IPC mechanism

- The Binder in action
  - The Binder kernel driver
  - Android Power Management
  - The Linux Power Management architecture
  - Android Wake Locks
  - The Power Management driver
  - The Android Kernel debugger
  - Configuring an Android Linux kernel
  - Building the kernel
- Exercise: Configuration and build of the Android kernel for the target board*  
*Exercise: Checking the first phases of kernel boot*

## **Second Day**

### **The Android Build system**

- The Android code base
  - Building Android
  - The Android build environment
  - The Android build system
  - The Android.mk files
- Exercise: Compiling a single component*
- Creating a new Android platform
  - Declaring a new vendor
  - Creation of platform-specific parameter files
  - Choosing platform-dependent compilation options
- Exercise: Creating and building a new Android platform*

### **Android Application Structure**

- Structure of an Android Application
  - Android application components
  - Activity
  - Service
  - Broadcast receiver
  - Content provider
  - Manifest file
  - Application components declaration
  - Permissions
- Exercise: Hello world application*

### **Android System Initialization**

- Android properties
  - Automatic properties
  - Default properties
  - Persistent properties
  - The Android initialization
  - Structure of the init process
  - The Android initialization language
  - The Dalvik zygote process
- Exercise: Modify the init process to handle (simulated) firmware signature check*

### **The Android BSP components**

- The Dalvik Java virtual machine
  - The Dalvik machine structure
  - The Dalvik bytecodes
  - The Dalvik JIT compiler
  - Porting the Dalvik interpreter
  - Porting the JIT compiler
  - Adding native components
  - Adding native executables
  - The Android NDK
  - Defining Java methods in C or C++
  - JNI for Android
  - Using SWIG
- Exercise: Add a native Linux system component to the Android BSP*

## Third Day

### **The Android Native Framework**

- The bionic C library
- Why a new C library
- The bionic Android-specific features
- What is missing in bionic
- Compiling against bionic
- Creating and using shared libraries
- *Exercise: Create a simple program and shared library using the bionic library*
- WebKit
- The Media Framework
- The OpenCORE PacketVideo platform
- Supported audio, video and still formats
- Hardware and software codec plug-ins
- SQLite
- FreeType
- SSL
- OpenGL/ES

### **The Android Application Framework**

- The Core platform services
- Activity Manager
- Package Manager
- Window Manager
- Resource Manager
- Content Providers
- View System
- The system services
- What is a system service
- Telephony Service
- Location Service
- Bluetooth Service
- WiFi Service
- USB Service
- Sensor Service
- Power Service
- The Android binder
- Writing services in C++
- The binder's C++ interface
- Adding a new system service
- *Exercise: Creating a system service in C++*

## Fourth Day

### **The Android Native Servers**

- Android Renderscript layer
- The Android framework renderscript API
- The Reflected layer mapping renderscript code to Java classes
- Renderscript code
- The renderscript graphics and compute engine
- Interfacing the Renderscript engine with hardware accelerators
- The Surface Flinger
- The Binder interface
- OpenGL/ES interface
- Using hardware accelerators and composers
- Double buffering using page-flip from the frame buffer
- The Audio Flinger
- Handling output devices
- Handling audio routing

### **The Hardware Abstraction Layer**

- Why a HAL?
- The Acme Component-oriented Architecture Definition Language

- Defining HAL components in Acme
  - Loading and using HAL component
  - The standard HAL components
  - Graphics
  - Audio
  - Camera
  - Bluetooth
  - GPS
  - Radio (RIL the Radio Interface Layer)
  - WiFi
- Exercise: Create a simple HAL component*

## **Fifth Day**

### **OpenMAX for Android Multimedia**

- Multimedia in an Android device
- Data formats and File formats
- Codec and Demux
- The Android Multimedia Framework
- OpenCORE the initial Android Media Framework
- OpenCORE architecture
- Stagefright
- OpenMAX Overview
- The Khronos Group
- OpenMAX/DL: the Development Layer
- OpenMAX/IL: the Integration Layer
- OpenMAX/AL: the Application Layer
- OpenMAX and OpenSL/ES
- OpenMAX in the Android Media Framework
- Interface between Android and OpenMAX
- The OpenMAX/IL Architecture

### **The Linux BSP**

- Linux BSP architecture
  - Overall structure
  - The ARM BSP
  - The Linux build system
  - Linux kernel memory usage
  - Defining and initializing the board
  - Linux kernel debugging
  - Debugging with a JTAG probe
  - Debugging with traces
  - Debugging with kgdb and kdb
- Exercise: Debugging the first steps of kernel startup*
- Exercise: Debugging kernel modules and drivers*
- The Linux driver model
  - Kernel objects
  - Devices, Classes and Drivers
  - Hotplug events
  - Power Management in drivers

## **Renseignements pratiques**

**Inquiry : 5 days**