ProTech *"Charting the Course ...*

protechtraining.com

... to Your Success!"

Android Internals

Course Summary

Description

The Android Internals training course is designed for those who are already familiar with basics of the Android SDK and are looking to customize and/or extend the functionality of the platform.

Android Internals focuses on the Android NDK, and Android IDL APIs, to give you a clean access to underlying hardware and services, with future compatibility in mind. You will learn how to build custom images and hack the platform.

Objectives

By the end of this course, students will be able to:

- Explain the anatomy of the Android platform and get is physiology (layer interactions)
- Build native applications in Android using JNI and NDK
- Take advantage of Android AIDL to build IPC-enabled bound services
- Build the entire Android platform from source and get what's what
- Customize and extend the Android platform to build custom ROMs
- Modify and extend Android frameworks and services
- Take advantage of custom hardware with Android
- Understand where Android departs from standard Linux

Topics

- Android Overview Module
- Android Stack Module
- Android Native Layer
- Android Application Framework Layer
- Android Applications Layer
- Java Native Interface (JNI) and the Android Native Development Kit (NDK)
- Android Inter-Process Communication (IPC) with Binder and AIDL
- Android Security Essentials
- Building Android from Source
- Android Startup
- Android Subsystems
- Creating a Customized Android System Image
- Android Tools and Debugging

Audience

This Android Internals course is for developers who want to dig deeper than the standard Android SDK. It is for those who want to hack the system a bit in order to add system services and hardware support for non-standard components or port Android to completely new boards.

Android Internals

Course Summary (cont'd)

Prerequisites

To take this course, you should have a firm understanding of the following:

Java
You should be able to answer most of the following questions:

- What is a difference between a class and an object?
- What is the difference between static and non-static field?
- What is the difference between extends and implements keywords?
- What is an anonymous inner class?
- What is the purpose of @Override?

C/C++/Make
To get the most benefit out of this class you must have a basic understanding of C and C++. For example, you should be able to answer the following:

- What is a header file?
- What is gcc and how to use it?
- Basic usage of sprintf()
- What is make and how does it work?
- Be able to read and understand basic Makefiles
- Be able to read and understand shell scripts

Linux
You should be familiar with basic Linux operating system. For example, you should be able to answer most of the following:

- How do you use use the following commands: ls, ps, cp, mv, pwd, cat, chmod, chown, mount, and similar.
- What is the init process?
- What are users and groups in Linux and how do r/w/x permissions work?

Duration

Five days

Android Internals

Course Outline

I. Android Overview Module

This module introduces you to the Android operating system to ensure a basic familiarity with the Android architecture, background, and terminology. You also learn additional sources of information for modifying the Android OS, porting it to new hardware, and complying with Google's Android compatibility requirements.

A. Overview of the Android platform architecture
B. Introduction to the structure and lifecycle of Android applications
C. Internet resources for Android platform developers
D. Android compatibility requirements: the Compatibility Definition Document (CDD) and Compatibility Test Suite (CTS)
E. Open source licensing issues with Android and the Linux Kernel

II. Android Stack Module

The Android OS is categorized into four layers. From the bottom up they are the kernel layer, the user-space native layer, the application framework layer, and the applications layer. In this module you explore each of the layers in depth to understand their role and the components they contain.

III. Android Kernel Layer

In this module you learn how Google has extended the standard Linux kernel for Android and how the upper layers of the Android OS interact with these extensions.

A. Where to find the source for Android's custom kernel
B. Binder: Android's primary inter-process communication (IPC) mechanism
C. Anonymous shared memory (ashmem): Android's replacement for POSIX SHM
D. Android's physical memory allocator, ION
E. Android power management extensions: wakelocks, early suspend, and alarms
F. The Android logging system
G. Android's user and group management and kernel security enforcement (the "paranoid network security" kernel option)

IV. Android Native Layer

In this module you explore the portion of Android user-space that is implemented in native C/C++ code. You learn: in what ways Android's libc implementation, Bionic, is not POSIX-compliant (and why); how Android's user-space Hardware Abstraction Layer (HAL) defines a standard API for exposing hardware to the rest of the Android OS; what significant libraries, frameworks, and daemons are incorporated in Android; and how Android's Dalvik Virtual Machine (VM) enables both application and platform developers to write Android code in Java.

A. Bionic: Android's implementation of libc and how it differs from the BSD libc
B. User-space Hardware Abstraction Layer (HAL): standard APIs for accessing hardware
C. Overview of standard Android native daemons (e.g., adbd, rild, ueventd, etc.) and their purpose
D. Android's low-memory process killer (lmkd)
E. Overview of other Android function libraries and frameworks
F. The Android Runtime (ART) and how it differs from a Java VM

V. Android Application Framework Layer

Android's application framework layer not only exposes a Java API for application developers, but also implements much of the Android operating system in Java. In this module, you discover how Java-based system services manage the operating system, applications, and device hardware. You also learn the general model by which client-side manager classes interact with these system services through Binder IPC to access system features.

A. System service architecture
B. Significant system services and their roles (e.g., ActivityManagerService, PackageManagerService, ConnectivityManagerService, etc.)

VI. Android Applications Layer

This module describes the basic structure of an Android application and how it is distributed as an Android application package (APK). It also identifies the standard set of Android system applications and where they are implemented in the source distribution.

A. Android application structure
B. Standard Android system applications, wallpapers, and input method editors

Android Internals

Course Summary (cont'd)

VII. Java Native Interface (JNI) and the Android Native Development Kit (NDK)

Much of the Android platform is implemented in Java, and yet that code needs to access functionality written in C/C++ and compiled to native machine code. Additionally, an application developer may want to implement portions of an app in C/C++ for performance or to incorporate existing native libraries and frameworks. Java Native Interface (JNI) is a standard Java technology for integrating Java and native code.

In this module, you learn the fundamentals of JNI and how to use it as a bridge between the native and Java-based runtimes. You also learn how to use Android's Native Development Kit (NDK) to implement portions of an application in native code and distribute it for use on multiple machine architectures.

A. JNI development process overview
B. Implementing Java methods in native code
C. Mapping Java types to native types
D. Managing object references in native code
E. Managing Strings, arrays, and other Java object types in native code
F. Throwing and catching exceptions in native code
G. Using NDK to incorporate native code in an app
H. Supporting multiple machine architectures with NDK
I. NDK "stable" APIs
J. Lab: NDK

VIII. Android Inter-Process Communication (IPC) with Binder and AIDL

Binder is Android's primary inter-process communication mechanism. (In fact, most standard POSIX IPC mechanisms are not available for use in Android.) Even higher-level Android IPC techniques, such as broadcast Intents and interactions between system services and client-side manager classes, use Binder as the underlying transport mechanism.

In this module, you learn: the capabilities of Binder; how to generated Binder-based interfaces in Java using Android Interface Definition Language (AIDL); how Binder is used by client-side manager classes to communication with system services; and how applications can expose their own Binder-based interfaces to other applications.

A. Overview of Binder and its capabilities
B. Higher-level Android IPC mechanisms based on Binder
C. Binder communication and service discovery
D. Generating Binder-based service interfaces in Java using Android Interface Definition Language (AIDL)
E. Creating custom Parcelable Java types for use with Binder
F. Exposing a Binder-based interface from an application
G. Asynchronous Binder interactions
H. Binder limitations
I. Binder security
J. Detecting and handling Binder "death notifications"
K. Lab: Binder-based application service

IX. Android Security Essentials

Android extends standard Linux security to control access to device features like network interfaces, cameras, and stored personal information. In this module you learn how Android's permission model interacts with standard Linux security and how to define and enforce custom permissions to restrict access to system extensions.

A. User and group ID management
B. SELinux Policies in Android
C. Android permission enforcement
D. Declaring custom Android permissions
E. Lab: Custom permissions
F. Securing application components using permissions

X. Building Android from Source

In this module you learn how to set up an Android build system, download the Android source, build Android system images, and run them on emulators and real hardware.

A. Setting up an Android build system
B. Obtaining the Android source tree
C. Selecting the target product and build variant
D. Building Android system images from source
E. Running custom Android images on emulators and real hardware

Android Internals

Course Summary (cont'd)

XI. Android Startup

This module describes Android system startup including bootloading the kernel, launching standard Linux daemons, and initializing a variety of Binder-based system services. It also explains the importance of the Zygote daemon for reducing Android application startup time and memory consumption. Additionally, you learn how to customize the system boot process through custom init scripts.

A. Bootloading the kernel
B. The init process and Android's init scripting language
C. The standard boot process and how to customize it
D. The purpose of the Zygote daemon
E. Features added for 64-bit user space support
F. Startup of system services

XII. Android Subsystems

This module presents an architectural overview of several of the most significant Binder-based system services in Android, explores interactions between services and client processes and between the various Android platform layers, identifies key source files in the implementation of the services, and provides references to additional resources for subject matter experts to extend and modify many services.

A. Vibrator
B. Power Service
C. Alarm Service
D. Package Service
E. WiFi Service
F. Location Service
G. Android Media Framework
H. Telephony
I. Device Policy Service
J. SurfaceFlinger
K. Camera Service
L. NFC Service

XIII. Creating a Customized Android System Image

This module integrates concepts from the entire course in the hands-on creation of a custom Android system image. You learn how to implement customizations at all Android stack levels, including custom kernels, HAL user-space libraries, executables, daemons, Java libraries, system applications, and Binder-based system services. Additionally, you'll learn how to support third-party developers for your Android devices by creating an SDK add-on that exposes your custom Java APIs and provides developers with a custom image they can use to create Android Virtual Devices (AVDs) on which they can run and test their code.

A. Setting up a custom device directory structure
B. Registering a custom device with Android's build system
C. Adding the Makefile plumbing for a device
D. Generating platform signing keys (Optional)
E. Adding a custom kernel
F. Adding a custom native library and executable
G. Adding a custom daemon
H. Creating a custom Java library to expose a native library (i.e., JNI in the platform)
I. Consuming a custom Java/JNI library via a custom application (Optional)
J. Implementing a custom Binder-based system service
K. Building a custom application using a custom client-side manager class
L. Creating and distributing a custom SDK add-on to support third-party developers (Optional)

XIV. Android Tools and Debugging

In addition to standard utilities for monitoring and troubleshooting Linux-based systems, Android includes several custom tools of its own. This module shows you how to use these tools to monitor and troubleshoot the Android kernel, processes, system services, and applications.

A. Debugging native code on Android
B. Debugging Java code on Android
C. Debugging applications and system services