ALIEN TECHNOLOGY®

ALR-H450 Developer Guide

January, 2016

ALIEN®

ALR-H450

Legal Notices

Alien Technology®
ALR-H450 Developer Guide

Table of Contents

1 Introduction

The Alien ALR-H450 Software Developer Kit (SDK) provides libraries and sample code for programmatically controlling Alien ALR-H450 handheld readers, which are running on an Android operating system. Alien provides class libraries and sample applications to help you get up-and-running developing your custom applications to run directly on the device.

1.1 Audience

We assume that the readers of this guide:

- are proficient Android developers,
- have minimal previous knowledge of RFID, and other relevant technologies.

1.2 Type Conventions

- Regular text appears in a plain, sans-serif font.
- External files and documents appear in *italic text*.
- Class names appear in a `fixed-width serif font`.
- Things you type in, and sample code appear:

  ```
  indented, in a fixed-width serif font.
  ```

- Longer blocks of sample code appear like below:

```
// Obtain RFIDReader instance
RFIDReader reader = RFID.open();
// Release RFIDReader instance
reader.close();
```

1.3 Overview

This document focuses on controlling the ALR-H450 handheld readers using the Alien Android API. The Android Library provided in the SDK, **alienapi.aar**, supports controlling both the RFID Reader and 1D/2D Barcode Scanner.

Library	Hardware Module	ALR-H450 (Android)
alienapi.aar	RFID Reader	✓
	1D/2D Barcode Scanner	✓ 1D & 2D

1.4 Documentation and Sample Code

This Guide provides complete documentation of all API elements. Additional usage tips can be gleaned by examining and modifying the source code samples provided by Alien. The sample applications, developed for Android Kitkat 4.4.2, demonstrate most of the major features of each hardware module.

1.5 System Requirements

You will need Android Studio 1.4 (or newer) to develop applications for the Alien handhelds.

2 RFID Reader

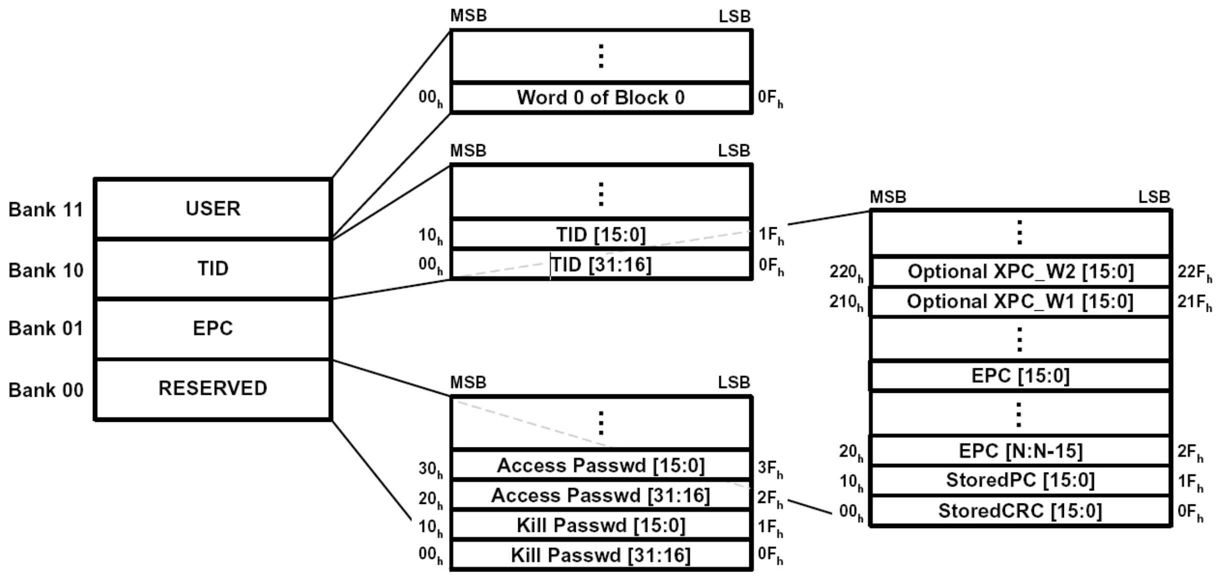
2.1 Introduction

The Alien `alienapi.aar` API library contains classes to provide interface with the RFID module in your Alien handheld reader. After initializing the `RFIDReader` class object, use its methods to communication with the RFID module.

You have control over most aspects of the Gen2 protocol, including access operations like write, lock, and kill, using tag masks, as well as reading tags' EPC IDs and Return Signal Strength Indication (RSSI).

To take full advantage of the RFID module and the Gen2 protocol, you are encouraged to read and understand the relevant portions of the EPCglobal Gen2 specification. One thing you must be aware of when accessing individual portions of memory in the tag is the layout of tag memory. As shown in the diagram below, all Gen 2 tags have four banks of memory (RESERVER, EPC, TID, USER) and each of those banks can be broken down into fields (Kill Password and Access Password within the RESERVED bank).

When reading and writing tag memory, all data operations are performed in one-word (2 bytes, 16 bits) increments, and only on word boundaries within a single bank. Some masking operations allow addressing memory down to the nibble (1/2 byte, 4 bits) or bit level.



2.2 Enumerations

2.2.1 Bank

The Bank enumeration is used to specify a particular memory bank, for locking, reading, and writing. See the Gen2 memory diagram at the start of this chapter for details of each bank.

```
public enum Bank{
  RESERVED,
  EPC,
  TID,
  USER
}
```

2.2.2 Target

Controls which Gen2 protocol A/B target is used when performing tag inventories.

```
public enum Target {
  A,
  B
}
```

2.2.3 Session

Controls which Gen2 protocol session is used when performing tag inventories.

```
public enum Session {
  S0,
  S1,
  S2,
  S3
}
```

2.2.4 LockType

Controls the lock type to use for lock/unlock operations.

```
public enum LockType {
  LOCK,
  UNLOCK,
  PERMALOCK,
  PERMAUNLOCK
}
```

2.2.5 RFIDInfo

Defines the available RFID subsystem information.

```
public enum RFIDInfo {
  FIRMWARE_VER,
  HARDWARE_VER,
  MODULE_ID,
  REGION,
}
```

2.3 Connecting to RFID Module

The Alien API's `RFIDReader` class allows controlling the RFID module. Using the `RFIDReader` instance, an application can adjust radio parameters, and execute tag operations (inventory, read, write, kill, lock) according to the ISO 18000 EPC Class 1 Generation 2 (Gen2) tag protocol.

Obtaining `RFIDReader` instance:

```
// Obtain RFIDReader instance
RFIDReader reader = RFID.open();
```

The `RFID.open()` method returns a **global** instance of `RFIDReader` which will be used by all `RFIDReader` methods to control the RFID module.

When finished using the `RFIDReader`, you should call the `close()` method to release the `RFIDReader` resources.

```
// Release RFIDReader instance and resources
reader.close();
```

2.4 Tag Operations

2.4.1 Mask

The Alien API provides the `Mask` class to give an application the ability to filter which tags participate in RFID operations, such as inventory, read, write, kill, and lock.

<u>**Constructor**</u>:

`public Mask(Bank bank, int bitOffset, int bitLength, String data)`

Parameters:

`bank`	which bank for masking
`bitOffset`	the bit offset where mask `data` start
`bitLength`	the bit length of the mask `data`
`pattern`	the data in hex string to filter

Example:
To mask on a tag which has the EPC bank value of "11AA" (16 bits) at the bit offset of 32, set the mask as follows:

```
Mask mask = new Mask(Bank.EPC, 32, 16, "11AA");
```

<u>**No filtering**</u>:
The `Mask` class has the static `NONE` instance which disables masking for tag operations.

Example:

```
Mask mask = Mask.NONE;
```

Mask on the EPC starting with a pattern:
For more convenience, the `Mask` class has a static `maskEPC()` method to create masks that would match tags that have EPC start with a certain data string.

`public static Mask maskEPC(String data)`

Parameters:
`data` the data the tag's EPC starts with

Returns:
`Mask` object that contains mask information

Example:
To mask on tags that have an EPC starting with "11AA".

```
Mask mask = Mask.maskEPC("11AA");
```

2.4.2 Reading a single tag

The `RFIDReader` class has the `read` method to perform an inventory that would read a single tag.

`public RFIDResult read(Mask mask [optional]) throws ReaderException`

Parameters:

`mask`	the mask to use when reading tags

Returns:

`RFIDResult`	object that contains `Tag` object. Calling `RFIDResult.getData()` will return a `Tag` object that contains EPC and RSSI information

Exceptions:
`ReaderException` is raised if the operation fails.

Example:
To read one tag with no filtering

```
RFIDReader reader = RFID.open();
RFIDResult result = reader.read();
if (result.isSuccess()) {
    Tag tag = (Tag)result.getData();
    String epc = tag.getEPC();
    double rssi = tag.getRSSI();
}
```

2.4.3 Continuous Inventory

The `RFIDReader` class provides the `inventory` method to initiate a continuous inventory and `stop` method to stop it.

The `RFIDCallback` **`interface`** is used to retrieve Tag information from the continuous inventory operation, via the `onTagRead` method callback. The caller needs to implement this method to receive tag information.

The `RFIDReader` class provides the `isRunning` method to check if continuous inventory is running.

```
public void inventory(RFIDCallback callback, Mask mask [optional])
    throws ReaderException
```

Parameters:

`callback`	the callback to receive tag information.
`mask`	the mask object for filtering

Returns:

`Tag`	object that contains EPC and RSSI information. For convenience, the `Tag` object has its own methods to execute operations for that specific tag.

Exceptions:

`ReaderException` is raised if there is an error.

NOTE:

- Continuous inventories run **asynchronously**.
- The reader cannot respond to other commands during continuous inventories. It is necessary to stop the continuous inventory before executing other commands. An `RFIDBusyException` will be raised if you attempt to execute other commands while the continuous inventory is running.

```
public interface RFIDCallback {
  void onTagRead(Tag tag);
}
```

Parameters:

`Tag` – The Tag object that contains tag information.

NOTE: Only EPC and RSSI information can be obtained from the tag object while inside the callback or while continuous inventory is running. To perform other operations on the tag object you must stop the continuous inventory.

```
public void stop() throws ReaderException
```

Exceptions:

`ReaderException` is raised if there is an error.

```
public boolean isRunning()
```

Returns:

`boolean` true if a continuous inventory is running.

Example:

To perform a continuous inventory for tags which EPCs start with "1122"

```
RFIDReader reader = RFID.open();
reader.inventory(new RFIDCallback() {
        @Override
        public void onTagRead(Tag tag) {
            String epc  = tag.getEPC();
            double rssi = tag.getRSSI();
        }
    },
    Mask.maskEPC("1122")
);
```

To stop continuous inventory:

```
reader.stop();
```

To check if continuous inventory is running:

```
boolean isRunning = reader.isRunning();
```

2.4.4 Reading and Writing tag's memory

The `RFIDReader` class provides `read` and `write` methods to read and write data from and to specific locations of tag's memory.

```
public RFIDResult read(Bank bank, int wordPointer, int wordCount,
   Mask mask [optional], String accessPassword [optional]) throws ReaderException
```

Parameters:

`bank`	the memory bank to read data from
`wordPointer`	the offset within the bank where to read the data from (in word units)
`wordCount`	the number of words to read
`mask`	the mask for filtering
`accessPassword`	the access password used when accessing protected memory. If not using password, pass this parameter as an empty string (`""`)

Returns:

`RFIDResult`	the results of the operation. Call `isSuccess()` on the `RFIDResult` object to check if the operation succeed. Call `getData()` on the `RFIDResult` object to get data read from the tag.

Exceptions:

`ReaderException` is raised if there is an error.

```
public RFIDResult write(Bank bank, int wordOffset, String data,
   Mask mask [optional], String accessPassword [optional]) throws ReaderException
```

Parameters:

`bank`	the memory bank to write data to
`wordOffset`	the offset within the bank where to write data to (in word units)
`data`	the data as a hex string to write
`mask`	the mask for filtering
`accessPassword`	the access password used when accessing protected memory. If not using password, passing this parameter as empty string (″″).

Returns:

`RFIDResult`	the results of the operation. Call `isSuccess()` on the `RFIDResult` object to check if the operation succeeded.

Exceptions:

`ReaderException` is raised if there is an error.

Example: To write and read a word from the USER bank

```
RFIDReader reader = RFID.open();
// Write "AABB" to the beginning of the USER memory bank of the tag which
// EPC starts with "1122"
RFIDResult writeResult = reader.write(Bank.USER,0,"AABB",Mask.maskEPC("1122"));
if (writeResult.isSuccess()) {
    // Write operation succeeded.
}

// Read 1 word from the beginning of the USER memory bank of the tag which
// EPC starts with "1122"
RFIDResult readResult = reader.read(Bank.USER,0,1,Mask.maskEPC("1122"));
if (readResult.isSuccess()) {
    // Read operation succeeded.
    String data = readResult.getData(); // data returned as a hex string
}
```

2.4.5 Lock Fields

The Alien API provides the `LockFields` class to be used as a parameter for tag lock/unlock operations. A `LockFields` object specifies which tag's memory fields will be affected when executing lock operations.

The `LockFields` class defines the following fields: `ACCESS_PWD, KILL_PWD, EPC` and `USER`

Descriptions:

`ACCESS_PWD`	Access Password is used to protect tag memory from access. The Access Password is stored within the RESERVED memory bank.
`KILL_PWD`	Kill Password is used when kill a tag. The Kill Password is stored within RESERVED memory bank.
`EPC`	EPC field, same as EPC bank.
`USER`	USER field, same as USER bank

Constructor:

`public LockFields(int fields)`

Parameters:

`fields`	a bitmap of fields to be locked

Example:
Create a `LockFields` bitmap specifying EPC and USER lock fields:

```
LockFields fields = new LockFields(
    LockFields.EPC | LockFields.USER
);
```

2.4.6 Locking and Unlocking Tag Memory

The `RFIDReader` class provides `lock` method to lock, unlock, permlock or permaunlock tag memory fields.

```
public RFIDResult lock(LockFields fieldBitmap, LockType lockType,
   Mask mask [optional], String accessPassword [optional]) throws ReaderException
```

Parameters:

`fieldBitmap`	a bitmap of fields to be locked
`lockType`	lock type (lock, unlock, permalock, permaunlock)
`mask`	the mask object for filtering
`accessPassword`	the access password used when accessing protected memory

Returns:

`RFIDResult`	the results of the operation. Call `isSuccess()` on the `RFIDResult` object to check if the operation succeeded.

Exceptions:

`ReaderException` is raised if there is an error.

NOTE:

- If a field is locked, a correct access password must be supplied in order to write to the memory
- If the EPC or USER fields are locked, they are **read-only** without knowing the access password. To write, you are required to pass the correct access password.
- If the ACCESS_PWD or KILL_PWD fields are locked, they are not accessible at all (**cannot read or write**) without knowing the access password.
- To unlock, it is necessary to know the correct access password.
- Use caution when using `lockType PERMALOCK` or `PERMAUNLOCK` – the results are **permanent**! If a field is permanently locked, it cannot be unlocked. If a field is permanently unlocked, it cannot be locked.

```
RFIDReader reader = RFID.open();
// Lock EPC and USER fields for the tags which EPC start with "1122"
LockFields fields = new LockFields(LockFields.EPC | LockFields.USER);
RFIDResult lockResult = reader.lock(fields, LockType.LOCK, Mask.maskEPC("1122"));
if (lockResult.isSuccess()) {
    // Lock operation succeed.
}

// Unlock EPC and USER fields for the tags which EPC start with "1122"
RFIDResult unlockResult = reader.lock(fields, LockType.UNLOCK,
           Mask.maskEPC("1122"), accessPassword);
if (unlockResult.isSuccess()) {
    // Unlock operation succeed.
}
```

2.4.7 Kill tags

The `RFIDReader` class provides the `kill` method to kill RFID tags. This prevents the tags from communicating further with an RFID reader. Once a tag is killed, that tag can't be recovered.

```
public RFIDResult kill(String killPassword, Mask mask [optional])
                throws ReaderException
```

Parameters:

`killPassword`	non-zero kill password
`mask`	mask object for filtering

Returns:

`RFIDResult`	the result of the operation. Call `isSuccess()` on the `RFIDResult` object to check if the operation succeeded.

Exceptions:

`ReaderException` is raised if there is an error.

2.4.8 Tag class

The Alien API library provides the `Tag` class to provide more convenience for the developer to run operations for a specific tag.

A `Tag` object is returned by reading a single tag, or via a callback from a continuous inventory. The data returned inside the `RFIDResult` is a `Tag` object.

```
RFIDReader reader = RFID.open();

RFIDResult result = reader.read(Mask.NONE);

Tag tag = (Tag)result.getData();

String epc = tag.getEPC();

double rssi = tag.getRSSI();
```

The `Tag` class provides wrapper methods, below, which allow you to run all operations for the tag directly on the tag itself. You don't need to supply a mask for that tag as it's going to automatically mask on the tag's EPC when calling an RFIDReader method.

- ***Obtaining Data From The Tag Object (without issuing RFID commands):***

```
public String getEPC();  // tag's EPC
public double getRSSI(); // tag's Returned Signal Strength
```

- ***Executing Rfid Operations On The Tag Object*:**

Write EPC:
```
public RFIDResult writeEPC(String epc) throws ReaderException;
```

Read/Write ACCESS_PWD:
```
public RFIDResult readAccessPwd() throws ReaderException
public RFIDResult writeAccessPwd() throws ReaderException
```

Read/Write KILL_PWD:
```
public RFIDResult readKillPwd() throws ReaderException
public RFIDResult writeKillPwd() throws ReaderException
```

Read TID memory bank:
```
public RFIDResult getTID() throws ReaderException
```

Read/Write USER memory bank:
```
public RFIDResult readUser() throws ReaderException
public RFIDResult writeUser() throws ReaderException
```

Read/Write tags's memory:
```
public RFIDResult read(Bank bank, int wordOffset, int wordCount,
        String accessPassword [optional])
public RFIDResult write(Bank bank, int wordOffset, String data,
        String accessPassword [optional]) throws ReaderException
```

Lock/Unlock tag's memory:
```
public RFIDResult lock(LockFields fields, LockType lockType,
        String accessPassword [optional]) throws ReaderException
```

Kill tag:
```
public RFIDResult kill(String killPassword) throws ReaderException
```

2.5 Setting RFID Parameters

The `RFIDReader` class includes methods to configure RFID related parameters, as well as to obtain information about the device.

2.5.1 Transmit Power

The `RFIDReader` class provides `getPower` and `setPower` methods to manage the RFID module's transmit power.

`public int getPower() throws ReaderException`

Parameters:
`None`

Returns:

`int`	the RFID module's power attenuation in dBm

Exceptions:
`ReaderException` is raised if there is an error.

`public void setPower(int power) throws ReaderException`

Parameters:

`power`	the RFID module's transmit power in dBm (1-30)

Exceptions:
`ReaderException` is raised if there is an error.
`InvalidParamException` is raised if the parameter value is not valid

2.5.2 Q Parameter

The `RFIDReader` class provides `getQ` and `setQ` methods to configure the starting Q parameter value that is used in the Gen2 protocol to read tags. It indicates approximately how many tags to expect (2^Q).

`public int getQ() throws ReaderException`

Parameters:
`None`

Returns:

`int`	the starting Q parameter value

Exceptions:
`ReaderException` is raised if there is an error.

`public void setQ(int QValue) throws ReaderException`

Parameters:

`QValue`	starting Q parameter value. Valid Q values are from 0 to 15.

Exceptions:
`ReaderException` is raised if there is an error.
`InvalidParamException` is raised if the parameter value is not valid

2.5.3 Session Parameter

The `RFIDReader` class provides `getSession` and `setSession` methods to configure the session parameter of the Gen2 protocol to keep track of inventoried tags. Please refer to the Gen2 protocol specification for information on the Session value.

`public Session getSession() throws ReaderException`

Parameters:
None

Returns:
`Session` the Gen2 protocol Session parameter

Exceptions:
`ReaderException` is raised if there is an error.

`public void setSession(Session session) throws ReaderException`

Parameters:
`Session` the Gen2 protocol Session parameter

Exceptions:
`ReaderException` is raised if there is an error.

2.5.4 Inventory Target

The `RFIDReader` class provides `getTarget` and `setTarget` methods to configure the Inventory target parameter used in the Gen2 protocol.

`public Target getTarget() throws ReaderException`

Parameters:
None

Returns:
`Target` the Gen2 protocol inventory target parameter

Exceptions:
`ReaderException` is raised if there is an error.

`public void setTarget(Target target) throws ReaderException`

Parameters:
`target` the Gen2 protocol inventory target parameter

Exceptions:
`ReaderException` is raised if there is an error.

2.5.5 RFID Subsystem Information

The `RFIDReader` class provides `getInfo` methods to obtain the device information, such as hardware version, firmware version, and model number.

`public String getInfo(DeviceInfo type)`

Parameters:

`type` the information type to obtain

Return:

`String` device information returned as a string, or null if the information is not available

`public Map<DeviceInfo, String> getInfo()`

Parameters:

None

Return:

`Map` a Map object containing all available device information

Example

```
RFIDReader reader = RFID.open();
String rfidHardwareVersion = reader.getInfo(DeviceInfo.RFID_HARDWARE_VER);
String rfidFirmwareVersion = reader.getInfo(DeviceInfo.RFID_FIRMWARE_VER);
String rfidModuleId = reader.getInfo(DeviceInfo.RFID_ID);
```

2.5.6 Device Information

The `DeviceInfo` class provides several methods to obtain the device information, such as the device model and the device ID.

`public String getDeviceID()`

Return:

`String` device ID returned as a string.

`public String getModel()`

Return:

`String` device model returned as a string.

Example

```
DeviceInfo dev = new DeviceInfo(context);
String id    = dev.getDeviceID();
String model = dev.getModel();
```

3 Barcode Reader

3.1 Introduction

The Alien API library includes classes to control and communicate with the Barcode Scanner module. Use `BarcodeReader` instance's methods to communicate with the Barcode Scanner.

3.2 Controlling Barcode Reader

The `BarcodeReader` allows scanning 1D and 2D barcodes.

`public BarcodeReader(Context context)`

Parameters:
`context` the Android context where the `BarcodeReader` instance is used

Example:

```
// Create BarcodeReader instance
BarcodeReader barcodeReader = new BarcodeReader(androidContext);
```

3.2.1 Scan 1D and 2D barcode

The `start` and `stop` methods control the Barcode Reader module to start and stop barcode scanning.
NOTE: The scanning will automatically stop after a barcode has been successfully scanned.

The `BarcodeCallback` **`interface`** is used to retrieve barcode information from the scanning operation, via the `onBarcodeRead` method callback. The caller needs to implement this method in order to receive barcode information.

The `isRunning` method is used to determine if barcode scanning is currently in progress.

`public void start(BarcodeCallback callback)`

Parameters:
`callback` the callback to receive barcode information

Returns:
None

`public void stop()`

Parameters:
None

Returns:
None

`public boolean isRunning()`

Parameters:
None

Returns:

`true`	if the barcode scanning is currently in progress

Example:

```
// Start barcode scan
barcodeReader.start(new BarcodeCallback() {
        @Override
        public void onBarcodeRead(String barcode) {
            String detectedBarcode = barcode;
            // Scan will automatically stop after successfully scanning a barcode
        }
    }
);
```

Stop barcode scan:

```
// Stop scan barcode
barcodeReader.stop();
```

4 Key Codes

The physical keys on the handheld each generate a unique Key Code, and when you handle key events in your application you look at the reported key code to determine which button was pressed.

The Alien API library provides the `KeyCode` class that defines codes for special physical keys on the handheld.

4.1 ALR-H450 (Android 4.4.2)

The `KeyCode.ALR_H450` class defines the following key codes for the ALR-H450 handheld:
`SCAN, SIDE_LEFT, SIDE_RIGHT.`

Descriptions:

`SCAN`	Key code of the Scan button.
`SIDE_LEFT`	Key code of the button on the Left Side of the unit.
`SIDE_RIGHT`	Key code of the button on the Right Side of the unit, below the Power button.

5 Developing applications with Android Studio

5.1 Install Android Studio

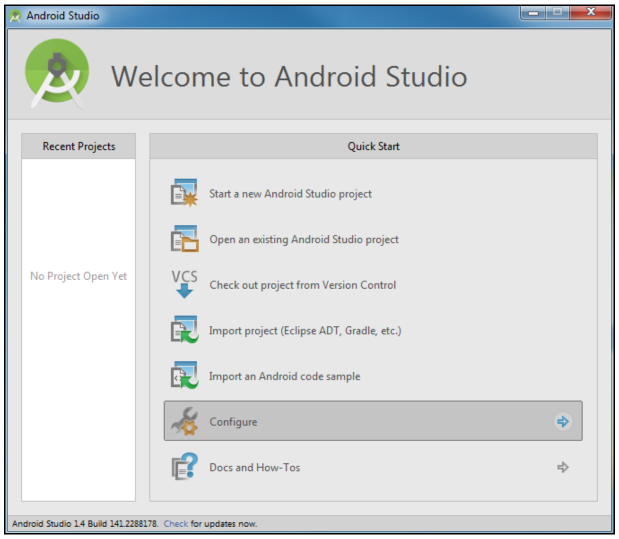
In order to develop RFID applications running on the Alien ALR-H450 handheld, you need to install Android Studio 1.4 (or newer) on your computer.

Download Android Studio from http://developer.android.com/sdk/index.html

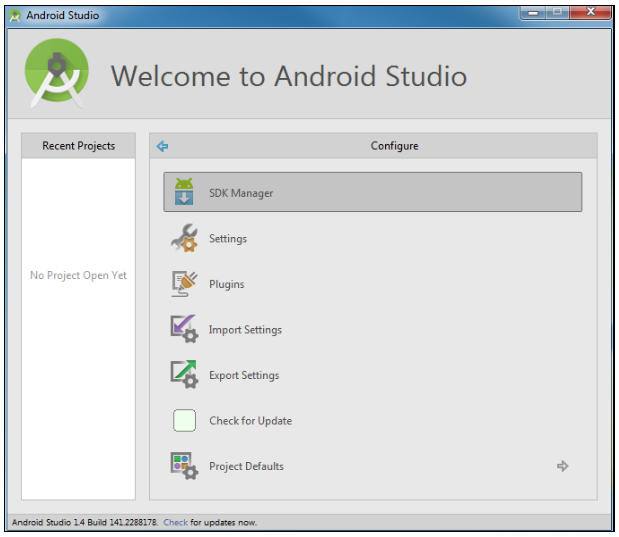
Run the installer and use all the default settings. The installer will automatically download and install the required components, including Android Support Repository and Android SDK Tools.

5.2 Install Google USB Driver

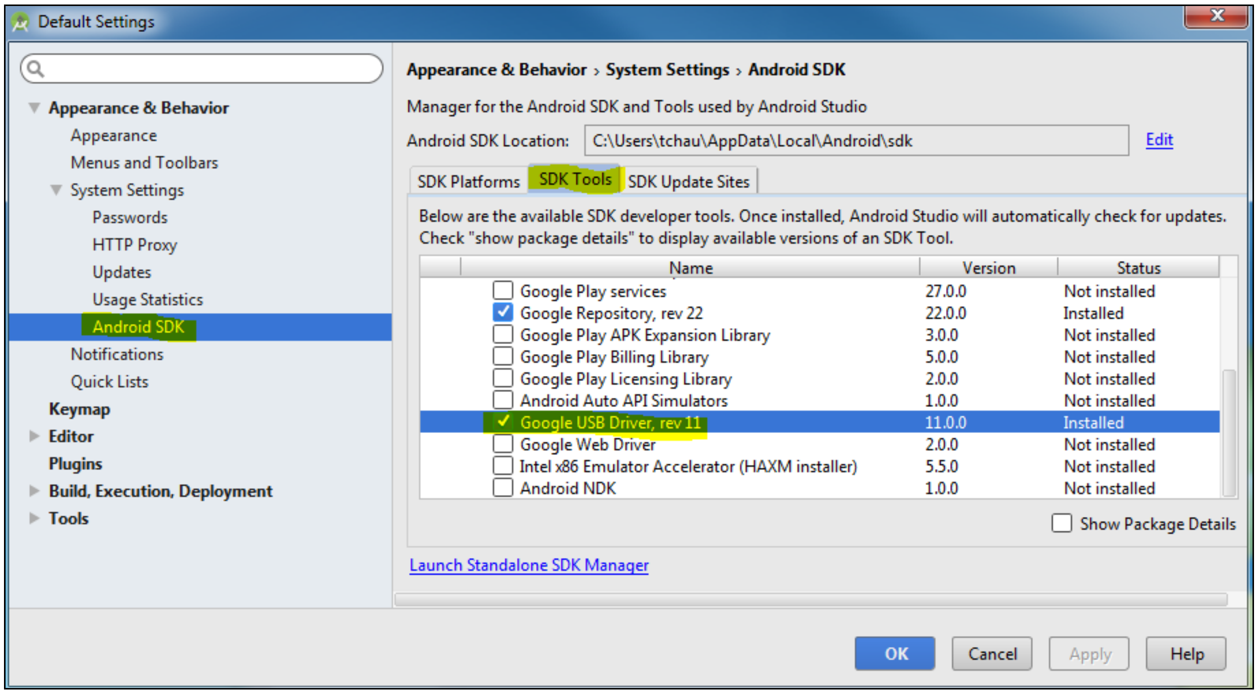
Open Android Studio. On the “Welcome to Android Studio” screen select Configure:


Android Studio
Welcome to Android Studio
Recent Projects
No Project Open Yet
Quick Start
Start a new Android Studio project
Open an existing Android Studio project
Check out project from Version Control
Import project (Eclipse ADT, Gradle, etc.)
Import an Android code sample
Configure
Docs and How-Tos
Android Studio 1.4 Build 141.2288178. Check for updates now.

Select “SDK Manager”:



Select “SDK Manager”, then “Android SDK”, then “SDK Tools” tab, and check the “Google USB Driver”. Click OK:


Default Settings

Appearance & Behavior › System Settings › Android SDK

Manager for the Android SDK and Tools used by Android Studio

Android SDK Location: C:\Users\tchau\AppData\Local\Android\sdk

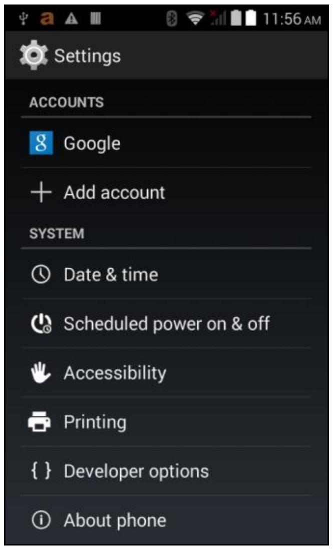
Below are the available SDK developer tools. Once installed, Android Studio will automatically check for updates. Check "show package details" to display available versions of an SDK Tool.

	Name	Version	Status
☐	Google Play services	27.0.0	Not installed
✓	Google Repository, rev 22	22.0.0	Installed
☐	Google Play APK Expansion Library	3.0.0	Not installed
☐	Google Play Billing Library	5.0.0	Not installed
☐	Google Play Licensing Library	2.0.0	Not installed
☐	Android Auto API Simulators	1.0.0	Not installed
✓	Google USB Driver, rev 11	11.0.0	Installed
☐	Google Web Driver	2.0.0	Not installed
☐	Intel x86 Emulator Accelerator (HAXM installer)	5.5.0	Not installed
☐	Android NDK	1.0.0	Not installed

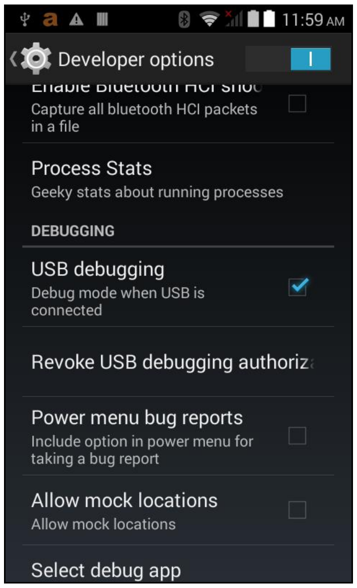
Make sure your PC is connected to the Internet.
Configure Windows to automatically download device drivers:
- Go to Control Panel > Devices and Printers
- Right click on the Computer icon and select “Device installation Settings”
- Select “Yes” to automatically download device drivers

5.3 Enable Developer Mode for your handheld

Go to the “Settings” app on the handheld, tap on “About Phone”. Then tap the “Build Number” 7 times. This will enable Developer Mode for your handheld.
Go back to the “Settings” and you will see a new “Developer options” menu item right above the “About Phone”.:



Tap on “Developer options”, check “USB Debugging” and click OK to enable USB debugging:


11:59 AM

Developer options

Enable Bluetooth HCI snoop
Capture all bluetooth HCI packets in a file

Process Stats
Geeky stats about running processes

DEBUGGING

USB debugging ✓
Debug mode when USB is connected

Revoke USB debugging authoriz

Power menu bug reports
Include option in power menu for taking a bug report

Allow mock locations
Allow mock locations

Select debug app

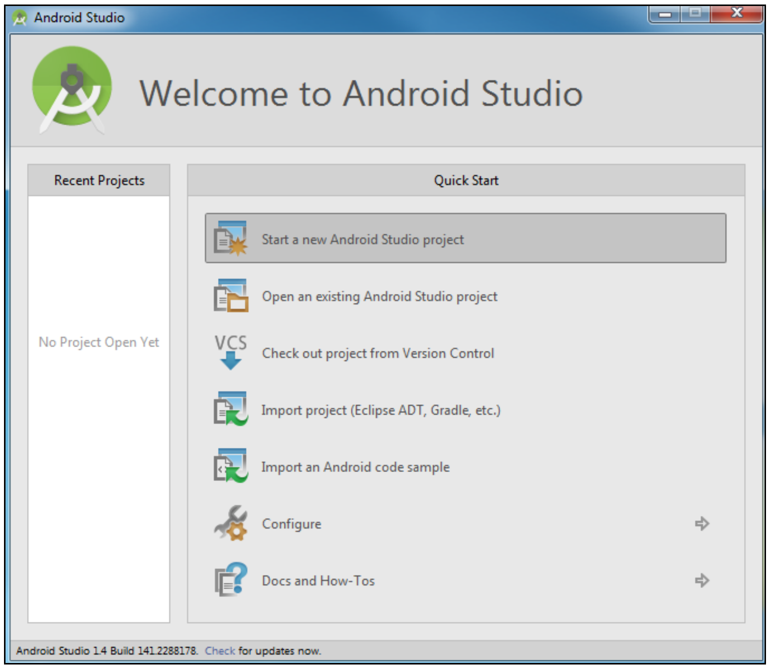
Verify your device is properly detected by your PC:
- Connect the cradle to the PC with a USB cable and insert the handheld into the cradle.
- Open the Device Manager to verify that the device has been identified under “Android USB Devices”

Now you are ready to develop applications for the handheld.

5.4 Developing your first RFID application

5.4.1 Create Android Project

To create a new project, open Android Studio and click “Start new Android Studio project”:


Android Studio

Welcome to Android Studio

Recent Projects

No Project Open Yet

Quick Start

Start a new Android Studio project

Open an existing Android Studio project

Check out project from Version Control

Import project (Eclipse ADT, Gradle, etc.)

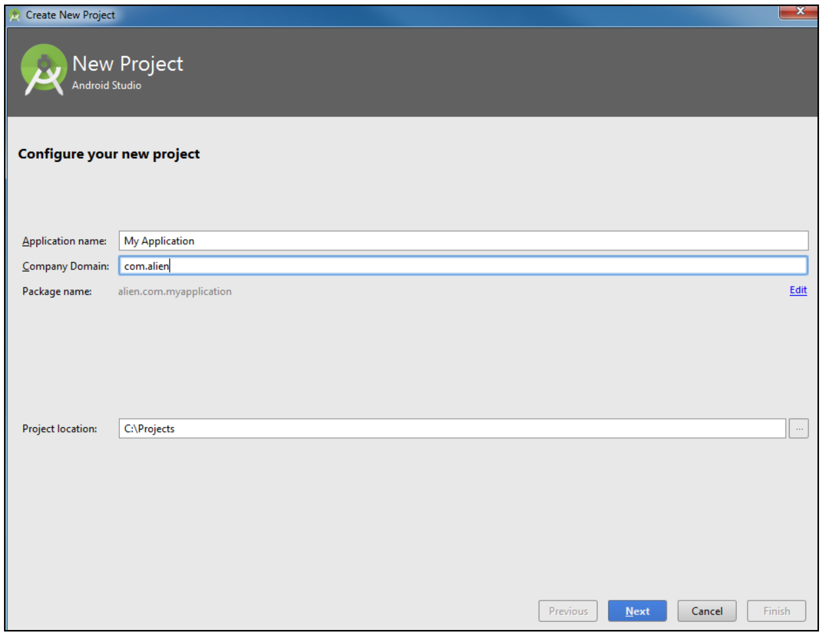
Import an Android code sample

Configure

Docs and How-Tos

Android Studio 1.4 Build 141.2288178. Check for updates now.

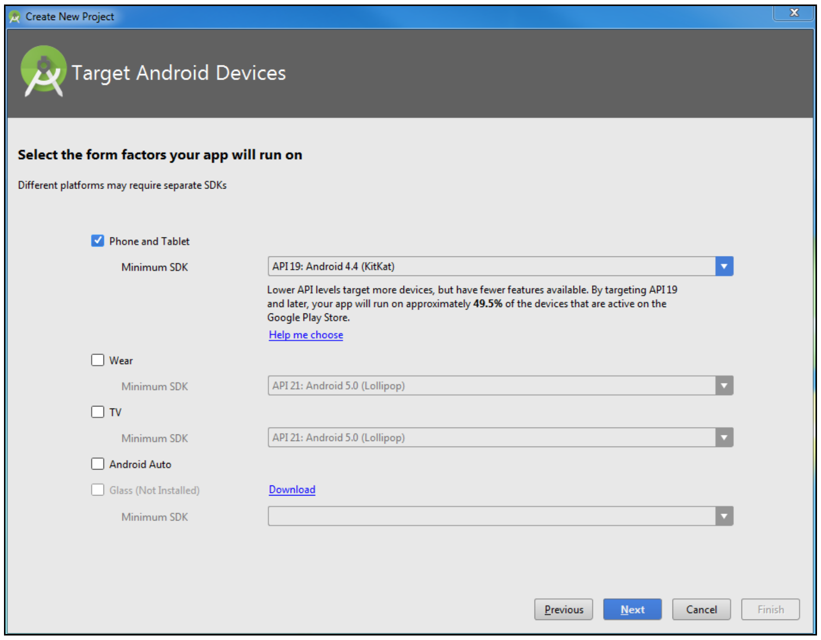
In the "New Project" dialog, use the default settings or enter new names for Application Name, Domain and Project location for your project. Click "Next":



Configure your new project

Application name: My Application
Company Domain: com.alien
Package name: alien.com.myapplication Edit
Project location: C:\Projects

In the "Target Android Devices" dialog, check the "Phone and Tablet" checkbox and select "API19: Android 4.4 (KitKat)" as the Minimum SDK. Click "Next":

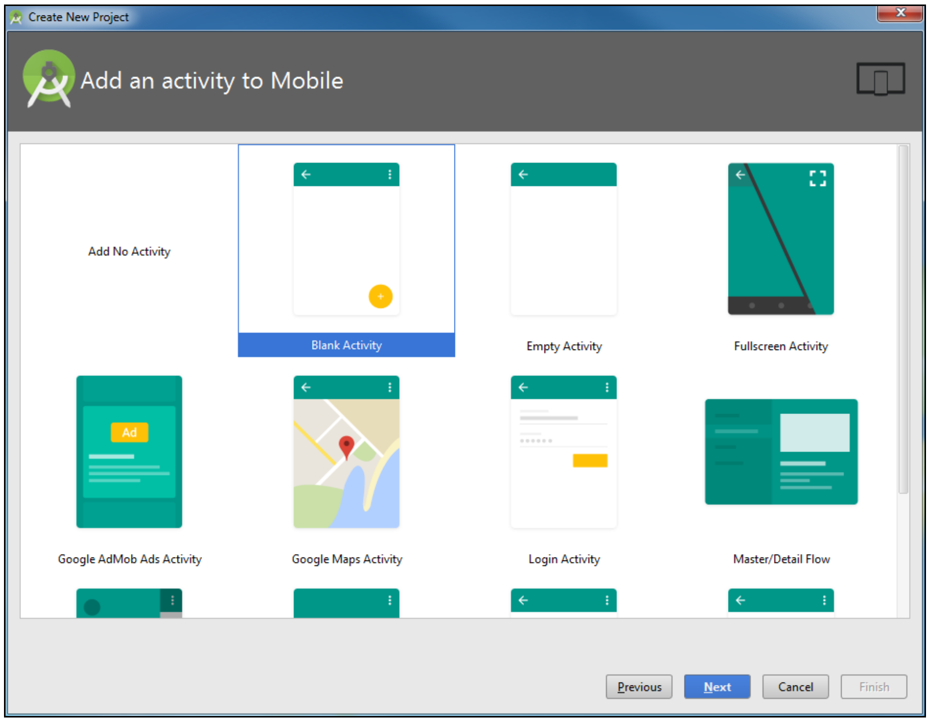


Select the form factors your app will run on

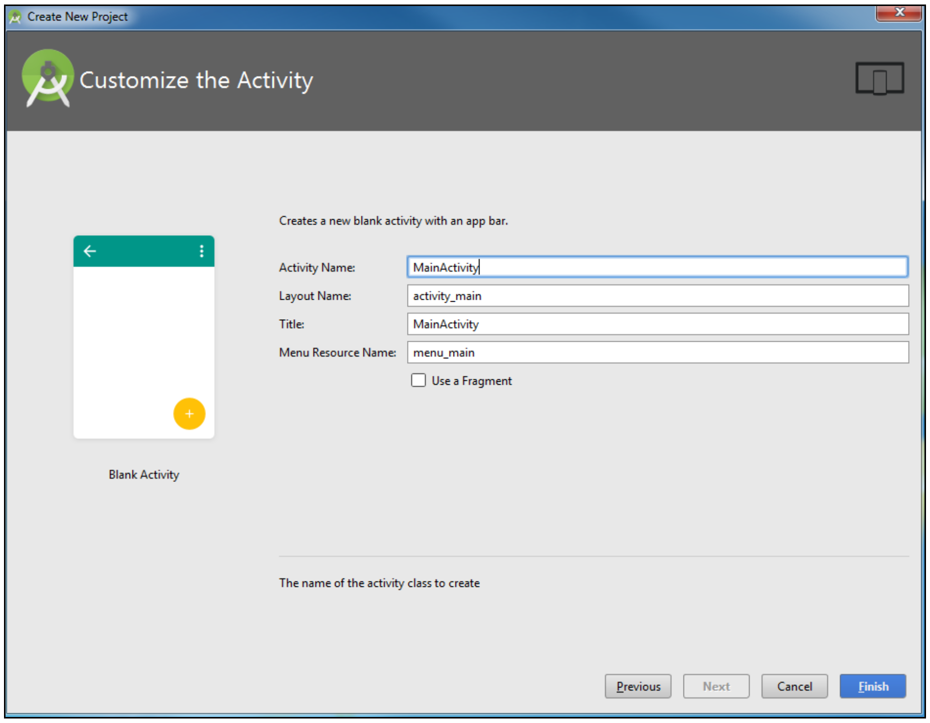
Different platforms may require separate SDKs

- [x] Phone and Tablet
  Minimum SDK: API 19: Android 4.4 (KitKat)
  Lower API levels target more devices, but have fewer features available. By targeting API 19 and later, your app will run on approximately **49.5%** of the devices that are active on the Google Play Store.
  Help me choose
- [ ] Wear
  Minimum SDK: API 21: Android 5.0 (Lollipop)
- [ ] TV
  Minimum SDK: API 21: Android 5.0 (Lollipop)
- [ ] Android Auto
- [ ] Glass (Not Installed) Download
  Minimum SDK:

In the “Add an activity to Mobile” dialog, select “Blank Activity” and click “Next”:



In the “Customize the Activity” dialog, use the default names or enter new names for Activity Name, Layout Name, Title and Menu Resource Name. Click “Finish”:



Creates a new blank activity with an app bar.

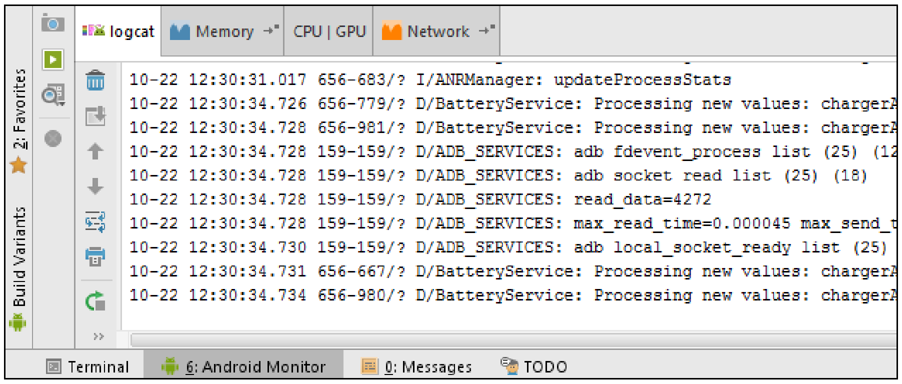
Activity Name:	MainActivity
Layout Name:	activity_main
Title:	MainActivity
Menu Resource Name:	menu_main

☐ Use a Fragment

The name of the activity class to create

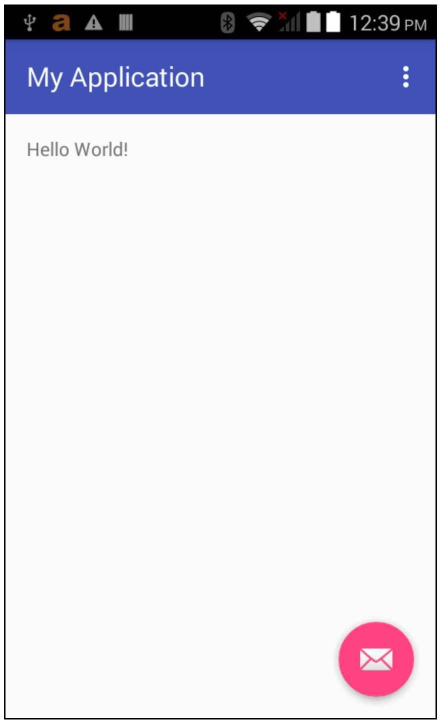
Connect the cradle to your PC, and insert the handheld into the cradle. You should see “Connected as USB storage” and “USB Debugging Enabled” notifications on the handheld screen.

In Android Studio, click “Android Monitor” at the bottom left. Then the “Allow USB debugging” will show up on your handheld’s screen. Check “Always allow from this computer” and tap OK. This will allow Android Studio to deploy the application into your handheld as well as to get log info from the handheld. Logs from the handheld will display in the “logcat” tab:



```
10-22 12:30:31.017 656-683/? I/ANRManager: updateProcessStats
10-22 12:30:34.726 656-779/? D/BatteryService: Processing new values: chargerA
10-22 12:30:34.728 656-981/? D/BatteryService: Processing new values: chargerA
10-22 12:30:34.728 159-159/? D/ADB_SERVICES: adb fdevent_process list (25) (12
10-22 12:30:34.728 159-159/? D/ADB_SERVICES: adb socket read list (25) (18)
10-22 12:30:34.728 159-159/? D/ADB_SERVICES: read_data=4272
10-22 12:30:34.728 159-159/? D/ADB_SERVICES: max_read_time=0.000045 max_send_t
10-22 12:30:34.730 159-159/? D/ADB_SERVICES: adb local_socket_ready list (25)
10-22 12:30:34.731 656-667/? D/BatteryService: Processing new values: chargerA
10-22 12:30:34.734 656-980/? D/BatteryService: Processing new values: chargerA
```

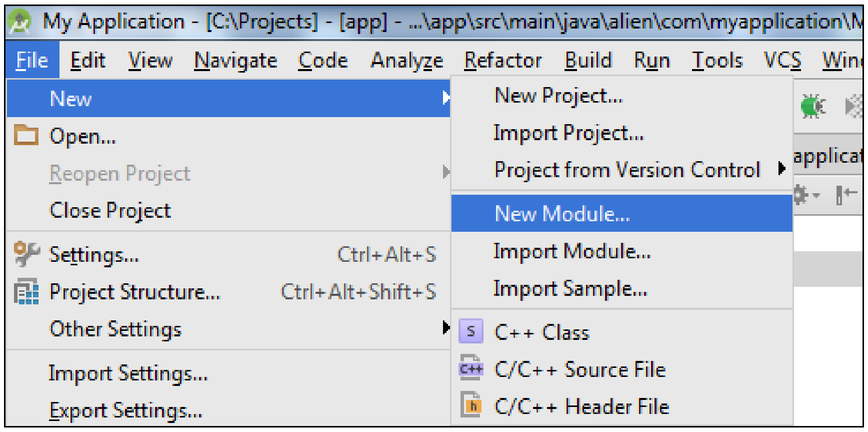
In Android Studio, click “Run app” toolbar button to run your application. The “Device Chooser” dialog will pop up. Select your device and click OK to run your “Hello World” application on the handheld:



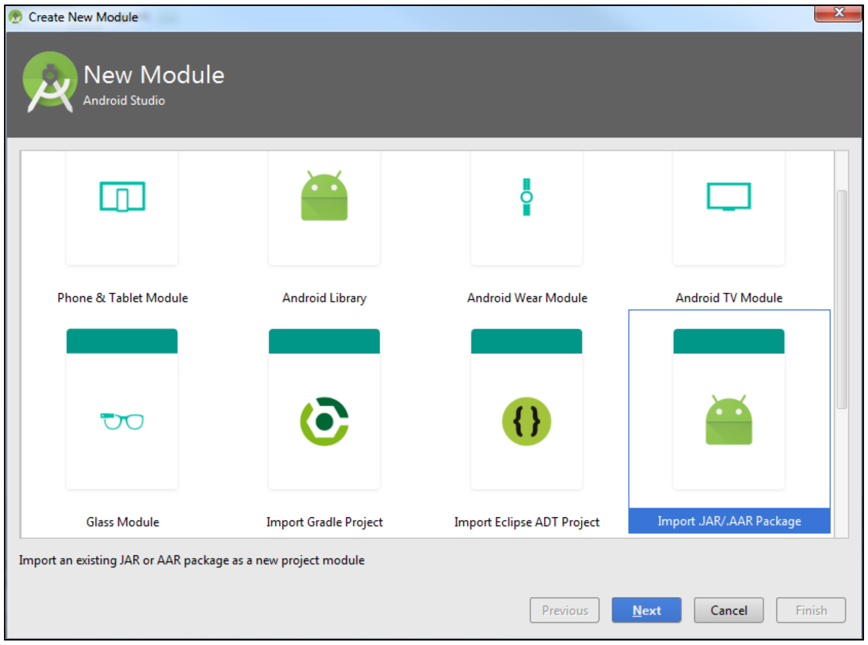
5.4.2 Use RFID functionality in your project

In order to use RFID functionality, you have to add the Alien `alienapi.aar` library to your project.

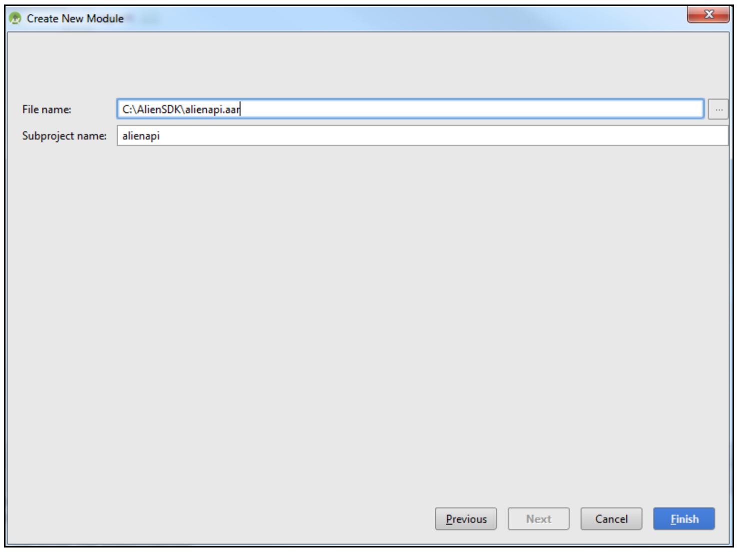
Select menu: File > New > New Module:



In the “New Module” dialog, select “Import .JAR/.AAR Package” and click “Next”:

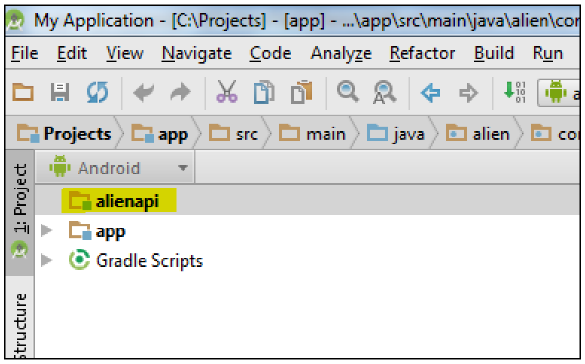


Browse to select the `alienapi.aar` library file and click “Finish”:

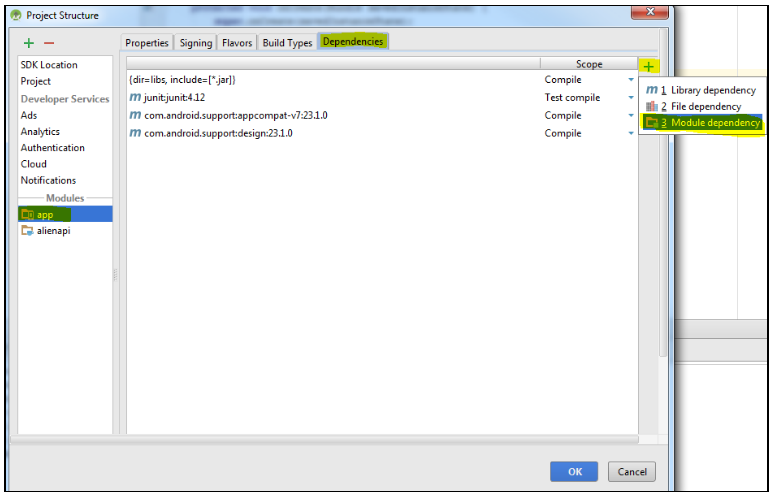


File name: C:\AlienSDK\alienapi.aar
Subproject name: alienapi

Now, you should see the “**alienapi**” module added to your project:

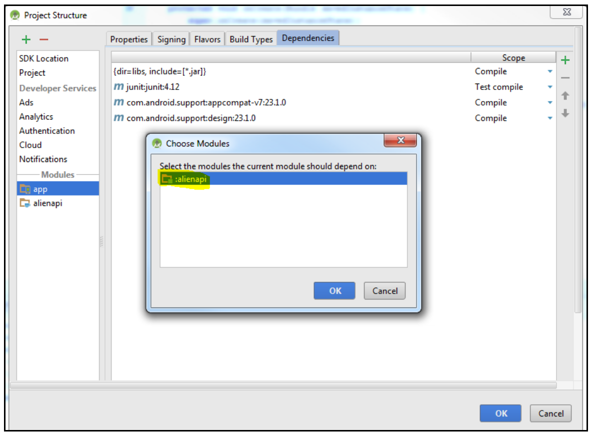


Next, you need to configure the “app” module to use “alienapi” module. To do this:
- Right click on the “app” module, and select “Open Module Settings”.
- Select the “app” module, select the “Dependencies” tab, click “+” button and select “Module dependencies”.



	Scope
{dir=libs, include=[*.jar]}	Compile
junit:junit:4.12	Test compile
com.android.support:appcompat-v7:23.1.0	Compile
com.android.support:design:23.1.0	Compile

Select ":alienapi" and click OK:



Project Structure

SDK Location, Project, Developer Services, Ads, Analytics, Authentication, Cloud, Notifications — Modules: app, alienapi

Properties | Signing | Flavors | Build Types | Dependencies

	Scope
{dir=libs, include=[*.jar]}	Compile
junit:junit:4.12	Test compile
com.android.support:appcompat-v7:23.1.0	Compile
com.android.support:design:23.1.0	Compile

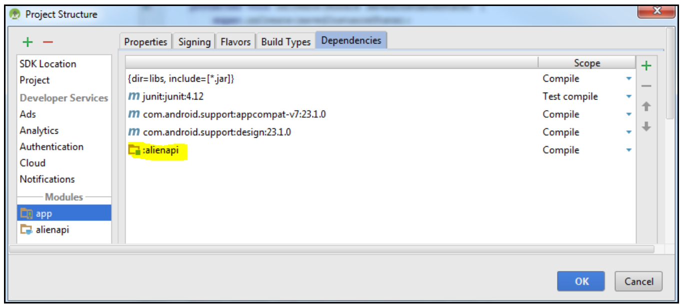
Choose Modules

Select the modules the current module should depend on:

:alienapi

OK Cancel

Now you should see the ":alienapi" in the Dependencies list. Click "OK":



Project Structure

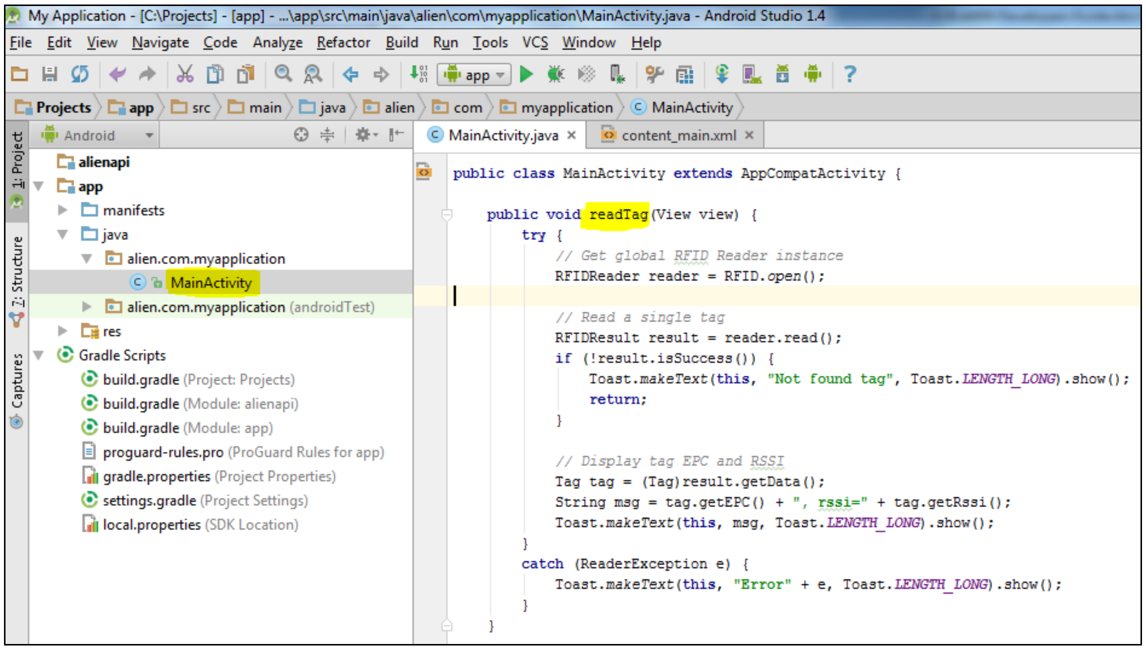
SDK Location, Project, Developer Services, Ads, Analytics, Authentication, Cloud, Notifications — Modules: app, alienapi

Properties | Signing | Flavors | Build Types | Dependencies

	Scope
{dir=libs, include=[*.jar]}	Compile
junit:junit:4.12	Test compile
com.android.support:appcompat-v7:23.1.0	Compile
com.android.support:design:23.1.0	Compile
:alienapi	Compile

OK Cancel

Add the `readTag()` method to your `MainActivity` class as below:



My Application - [C:\Projects] - [app] - ...\app\src\main\java\alien\com\myapplication\MainActivity.java - Android Studio 1.4

File Edit View Navigate Code Analyze Refactor Build Run Tools VCS Window Help
Projects app src main java alien com myapplication MainActivity
Android
MainActivity.java
content_main.xml

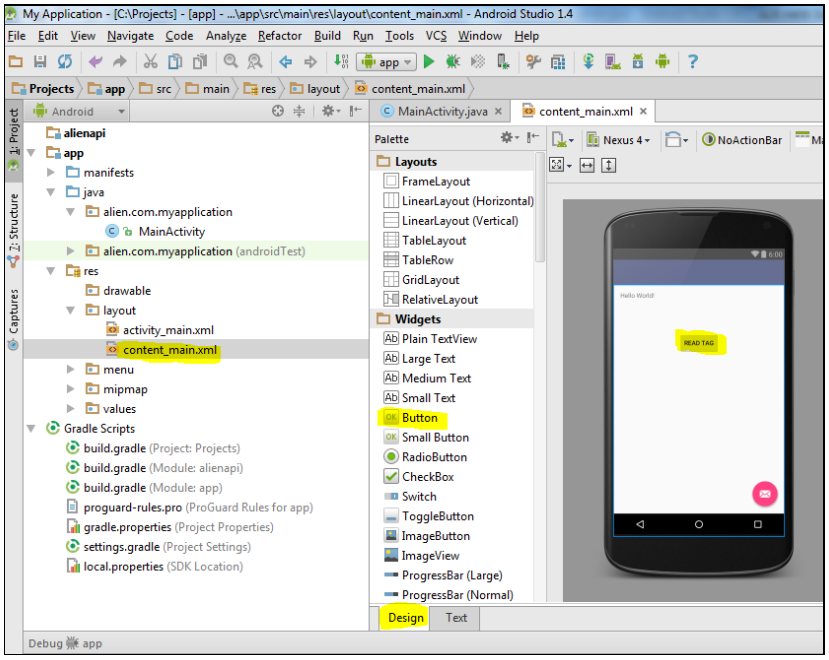
```
alienapi
app
    manifests
    java
        alien.com.myapplication
            MainActivity
        alien.com.myapplication (androidTest)
    res
Gradle Scripts
    build.gradle (Project: Projects)
    build.gradle (Module: alienapi)
    build.gradle (Module: app)
    proguard-rules.pro (ProGuard Rules for app)
    gradle.properties (Project Properties)
    settings.gradle (Project Settings)
    local.properties (SDK Location)
```

```
public class MainActivity extends AppCompatActivity {

    public void readTag(View view) {
        try {
            // Get global RFID Reader instance
            RFIDReader reader = RFID.open();

            // Read a single tag
            RFIDResult result = reader.read();
            if (!result.isSuccess()) {
                Toast.makeText(this, "Not found tag", Toast.LENGTH_LONG).show();
                return;
            }

            // Display tag EPC and RSSI
            Tag tag = (Tag)result.getData();
            String msg = tag.getEPC() + ", rssi=" + tag.getRssi();
            Toast.makeText(this, msg, Toast.LENGTH_LONG).show();
        }
        catch (ReaderException e) {
            Toast.makeText(this, "Error" + e, Toast.LENGTH_LONG).show();
        }
    }
```

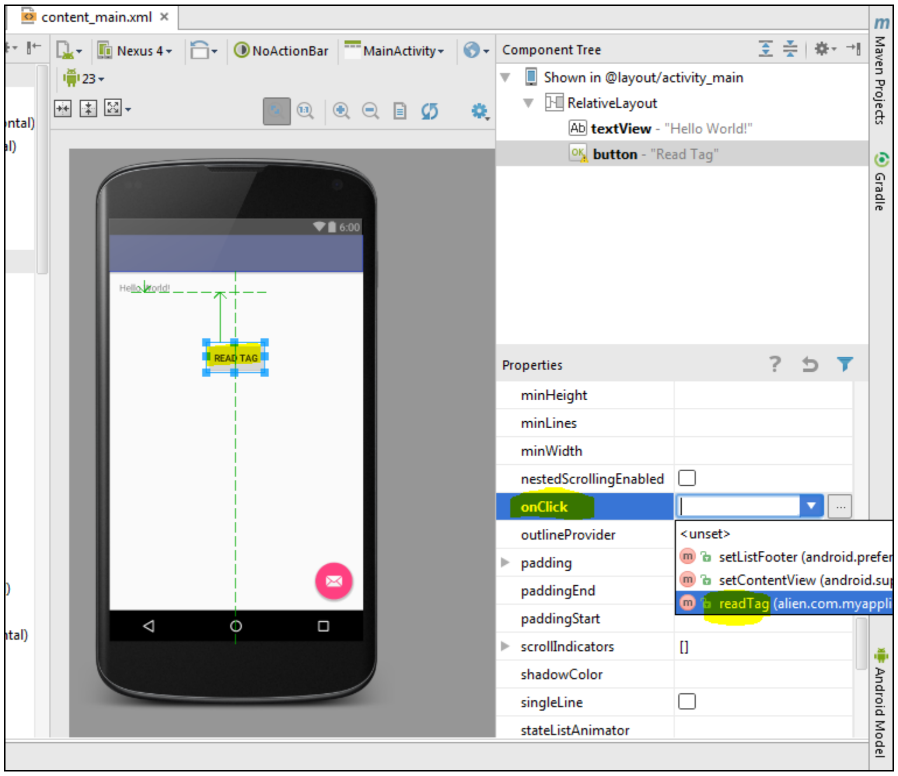
```
public void readTag(View view) {
    try {
        // Get global RFID Reader instance
        RFIDReader reader = RFID.open();
        // Read a single tag
        RFIDResult result = reader.read();
        if (!result.isSuccess()) {
            Toast.makeText(this, "No tags found ", Toast.LENGTH_LONG).show();
            return;
        }
        // Display tag EPC and RSSI
        Tag tag = (Tag)result.getData();
        String msg = tag.getEPC() + ", rssi=" + tag.getRSSI();
        Toast.makeText(this, msg, Toast.LENGTH_LONG).show();
    }
    catch (ReaderException e) {
        Toast.makeText(this, "Error: " + e, Toast.LENGTH_LONG).show();
    }
}
```

Next add a Button to call the `readTag` method that you just created:

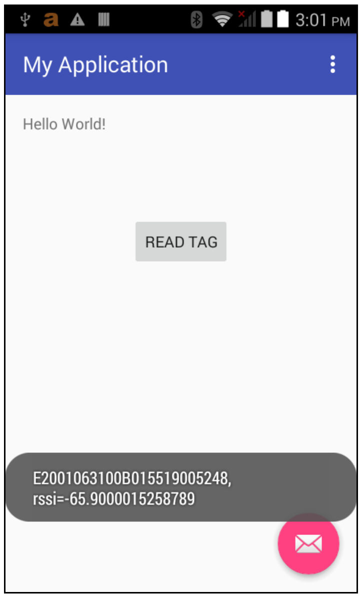
- Double click on “content_main.xml”.
- Select “Design” tab.
- Drag and Drop a “Button” widget into the view.
- Double click the button and change the name to “READ TAG”.


My Application - [C:\Projects] - [app] - ...\app\src\main\res\layout\content_main.xml - Android Studio 1.4
File Edit View Navigate Code Analyze Refactor Build Run Tools VCS Window Help
Projects app src main res layout content_main.xml
Android
MainActivity.java content_main.xml
alienapi
app
manifests
java
alien.com.myapplication
MainActivity
alien.com.myapplication (androidTest)
res
drawable
layout
activity_main.xml
content_main.xml
menu
mipmap
values
Gradle Scripts
build.gradle (Project: Projects)
build.gradle (Module: alienapi)
build.gradle (Module: app)
proguard-rules.pro (ProGuard Rules for app)
gradle.properties (Project Properties)
settings.gradle (Project Settings)
local.properties (SDK Location)
Palette
Layouts
FrameLayout
LinearLayout (Horizontal)
LinearLayout (Vertical)
TableLayout
TableRow
GridLayout
RelativeLayout
Widgets
Plain TextView
Large Text
Medium Text
Small Text
Button
Small Button
RadioButton
CheckBox
Switch
ToggleButton
ImageButton
ImageView
ProgressBar (Large)
ProgressBar (Normal)
Nexus 4
NoActionBar
Hello World!
READ TAG
Design
Text
Debug app

- Select the button, then select “onClick” in the “Properties” panel, and choose “readTag”:



Now test the application:
- Click “Run App” toolbar button to start the application on your handheld.
- Place a RFID tag in front of your handheld.
- Tap the “READ TAG” button in the application window to read and display a tag as shown below



```
E2001063100B015519005248,
rssi=-65.9000015258789
```