EAN Android Library Guide

Last revised: 2013.03.05

Preface

This document is meant to serve as a step-by-step guide to the EAN Android library. It covers the basics of the library and includes instructions for creating an Android app that uses the library.

The guide is broken down into the following topics:

Learn about the library, its capabilities and purpose

*covered in **Chapter 1: Introduction***

Build the library

*covered in **Chapter 2: Installing the Library***

Run the provided sample Android app

*covered in **Chapter 3: Running the Sample App***

Explore how the library is used to make EAN API requests

*covered in **Chapter 4: Exploring the Sample App***

Create a new Android app that uses the library

*covered in **Chapter 5: Creating a New App***

Chapter 1: Introduction

What is the library?

The EAN (Expedia Affiliate Network) Android library is a Java-based extension of the existing EAN API designed to work with Android devices. The library offers the following functionality:

Function	Description
Destination Lookup	Resolves destinations from a user input string. It returns a list of destinations with categories, names, and EAN destination IDs.
Hotel List	Retrieves a list of available hotels for user defined region. It returns a list of hotels with a single room's pricing details.
Room Availability	Retrieves the list of rooms available for a user defined hotel. It returns a list of rooms with all the details need to book them.
Hotel Information	Retrieves an extended set of hotel information that is not based on availability or pricing. It returns a single object with a large set of data (such as full set of amenities and photos).
Hotel Booking	Makes a booking of a single room for a user. It will require data from the Hotel Room Availability and will return an itinerary ID, confirmation number and pricing details of the booking.
View Itinerary	Looks up an existing itinerary based on the itinerary ID assigned by EAN and the email address that the user provided at time of booking. It will return all the pricing and stay details that were returned at time of booking.
Cancel Itinerary	Cancels an existing booking. It will return only success or failure.

Tip: For more information about the full EAN API, visit http://developer.ean.com/docs

Why use the library?

While it is entirely possible to write an Android app that communicates directly with the EAN API, using the EAN Android library as a bridge between the two offers several benefits. A few highlights:

The library is **fast.**

Speed is a primary focus of the library. Existing EAN API requests and responses contain a lot of elements that tend to be more detrimental than useful in mobile environments. Weeding them out improves communication times significantly.

The library is **simple.**

The library allows Android developers to focus on the app itself instead of wading through complex communication logic and data parsing.

The library is **extensible.**

The library is open source and we encourage developers to explore the source code and make modifications if necessary.

Who can use the library?

Any developer should be able to quickly integrate the library into an application and start making requests. However, this guide does make a couple of assumptions:

- You are familiar with the Java programming language.
- You are familiar with Android-based development.

Familiarity with the full EAN API is preferred, but not necessary for this guide.

What is needed to use the library?

The remainder of this guide will assume that you have the following set up in your development environment:

Name	Version	Download Link
Java Development Kit (JDK)	1.6+	http://www.oracle.com/technetwork/java/javase/downloads/index.html
Apache Ant	1.7+	http://ant.apache.org/bindownload.cgi
Android SDK	14+	http://developer.android.com/sdk/index.html
Git	1.8+	http://git-scm.com

You will also want to have a functioning IDE (Integrated Development Environment) set up for your project. The following examples all assume you are using IntelliJ with the Android Designer plugin installed, but other IDEs with Android support (such as Eclipse) will work as well.

Chapter 2: Installing the Library

Downloading the library

The code for the EAN Android library is located on Github. To download the source code from the git repository, use the following command:

```
git clone git://ExpediaInc/ean-android/ean-android.git
```

Building the library

The library uses Ant as its build tool. To build the library, run the following command from the root of the `ean-android` repository you checked out:

```
ant
```

The build resolves dependencies, checks the project for style issues, runs unit tests, builds the javadoc, and creates the necessary JAR files. It also builds the included sample app that we will use later in this guide.

Tip: For more details on the build process, see the README.md file at https://github.com/ExpediaInc/ean-android

Using the library

The build process creates a JAR file and saves it in `ean-android/api-lib/build/ean-api-lib.jar`. To use the library in an application, the `ean-api-lib.jar` file must be added to the application's classpath.

Chapter 3: Running the Sample App

A sample Android application is included in the ean-android distribution and will be the basis for the upcoming tutorials.

Building the sample app

Assuming you downloaded the library in the previous chapter, the sample application will be located in the following directory:

`~/projects/ean-android/`**`sample-app`**

(your structure may be different depending on where you cloned the `ean-android` repository)

To build the sample app, run the following commands from the `sample-app` directory:

```
android update project -p ./
ant debug
```

Adding the sample app to IntelliJ

To set up the sample app in the IntelliJ IDE, follow these steps after launching the IDE.

Warning: Make sure you have the IntelliJ Android Designer plugin installed before you begin this process!

Close all other open projects.

In the Quick Start window, click **Import Project**.

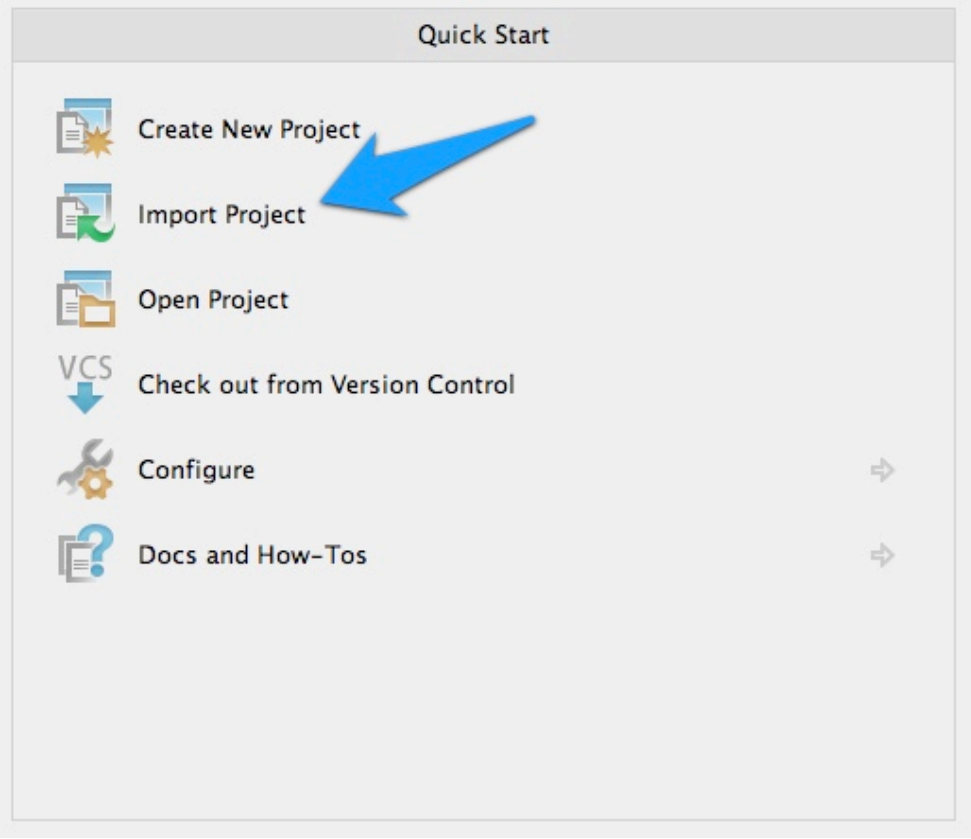

Figure 3.1 - Quick Start Screen

Navigate to the sample-app directory, select it, and click **OK**.

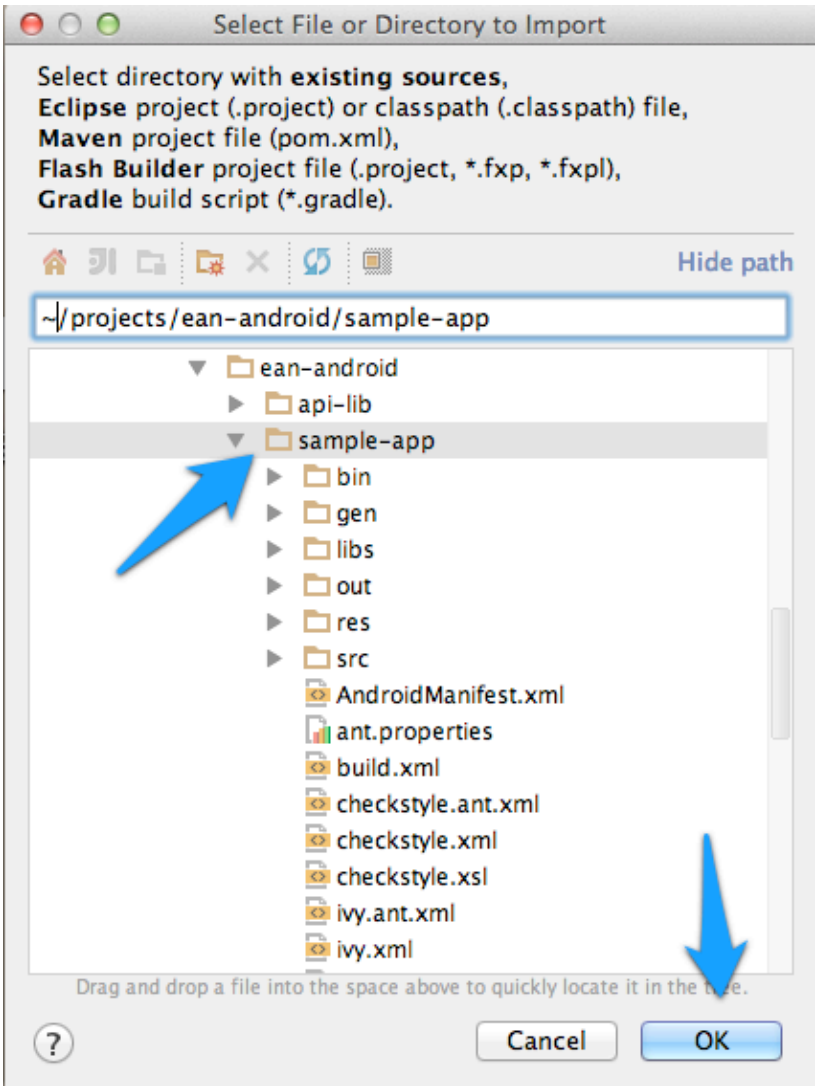

Figure 3.2 - Project location selection

Make sure that **Create project from existing sources** is selected and click **Next**.

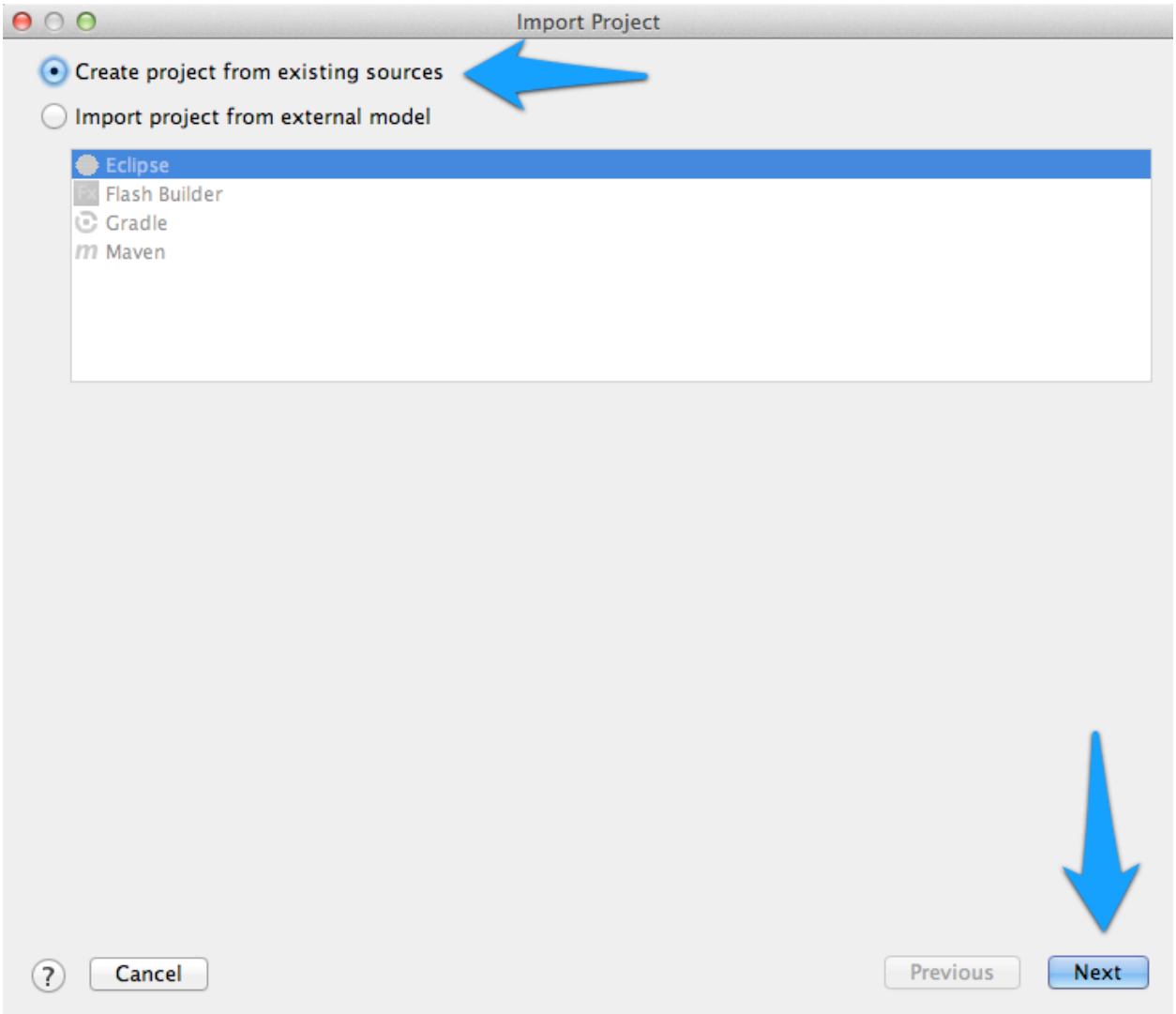

Figure 3.3 - Import type

Accept the default project name and location and click **Next**.

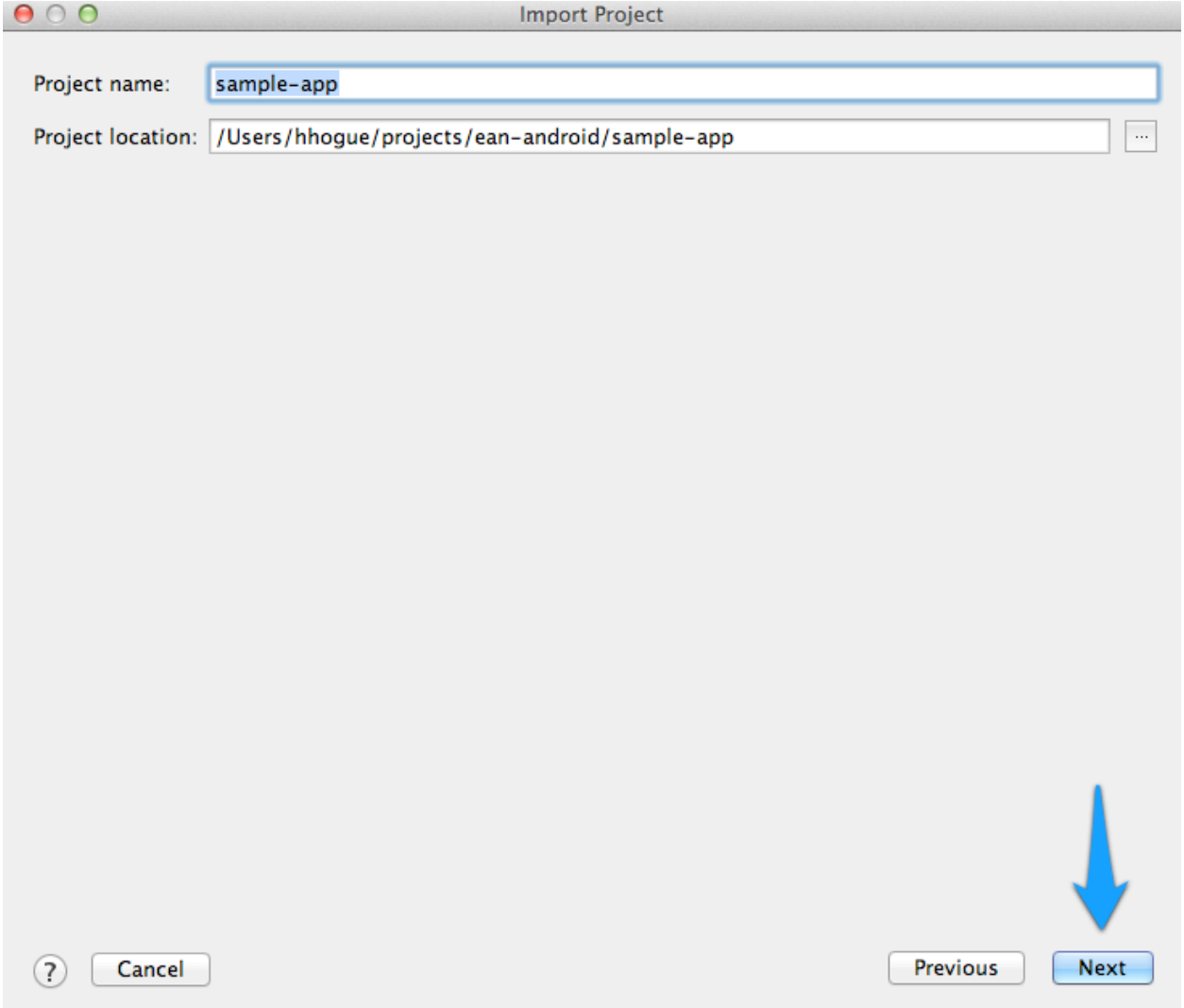

Figure 3.4 - Project name and location

Make sure all directories are selected and click **Next**.

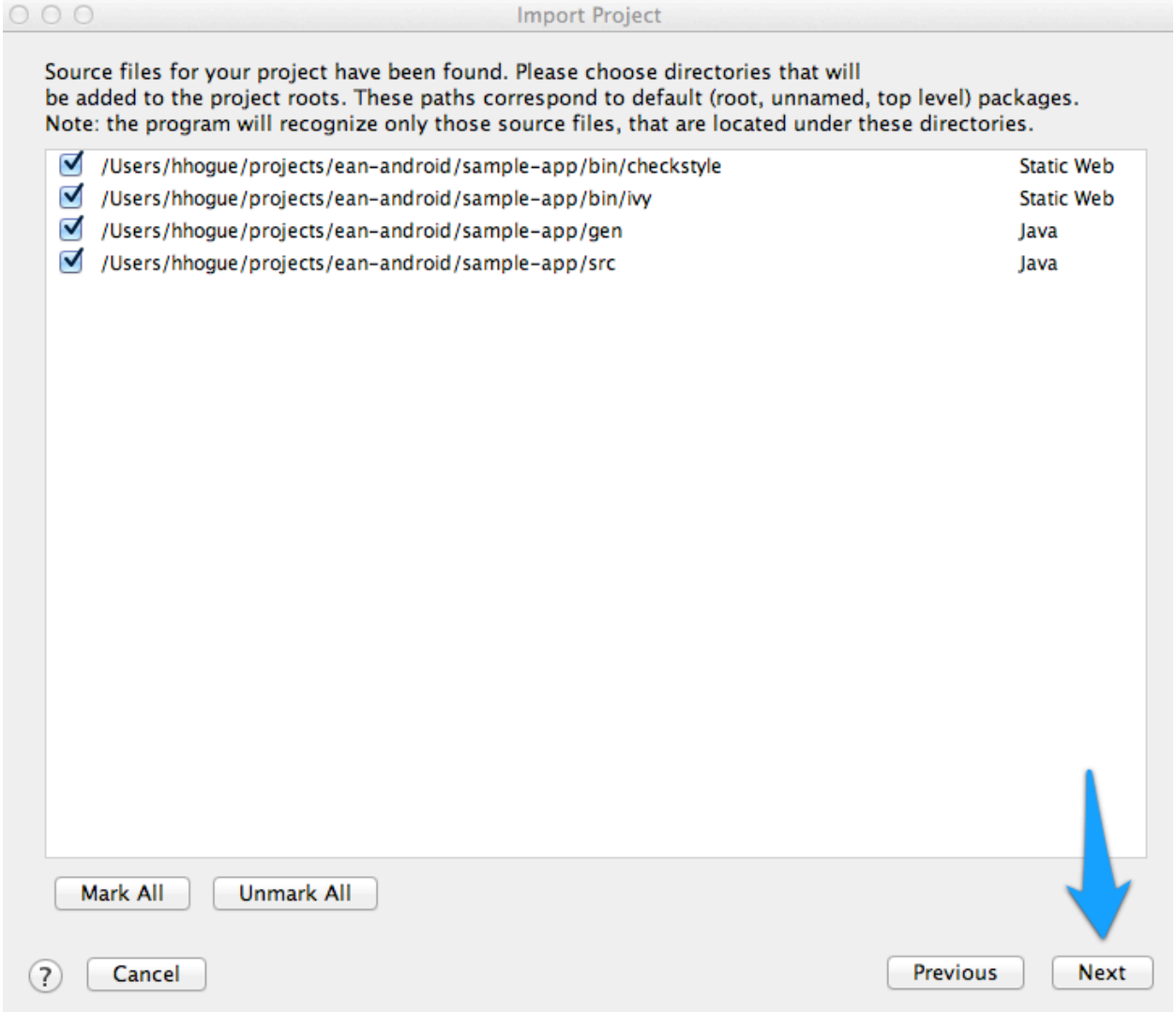

Figure 3.5 - Source file selection

Review the included libraries and click **Next**.

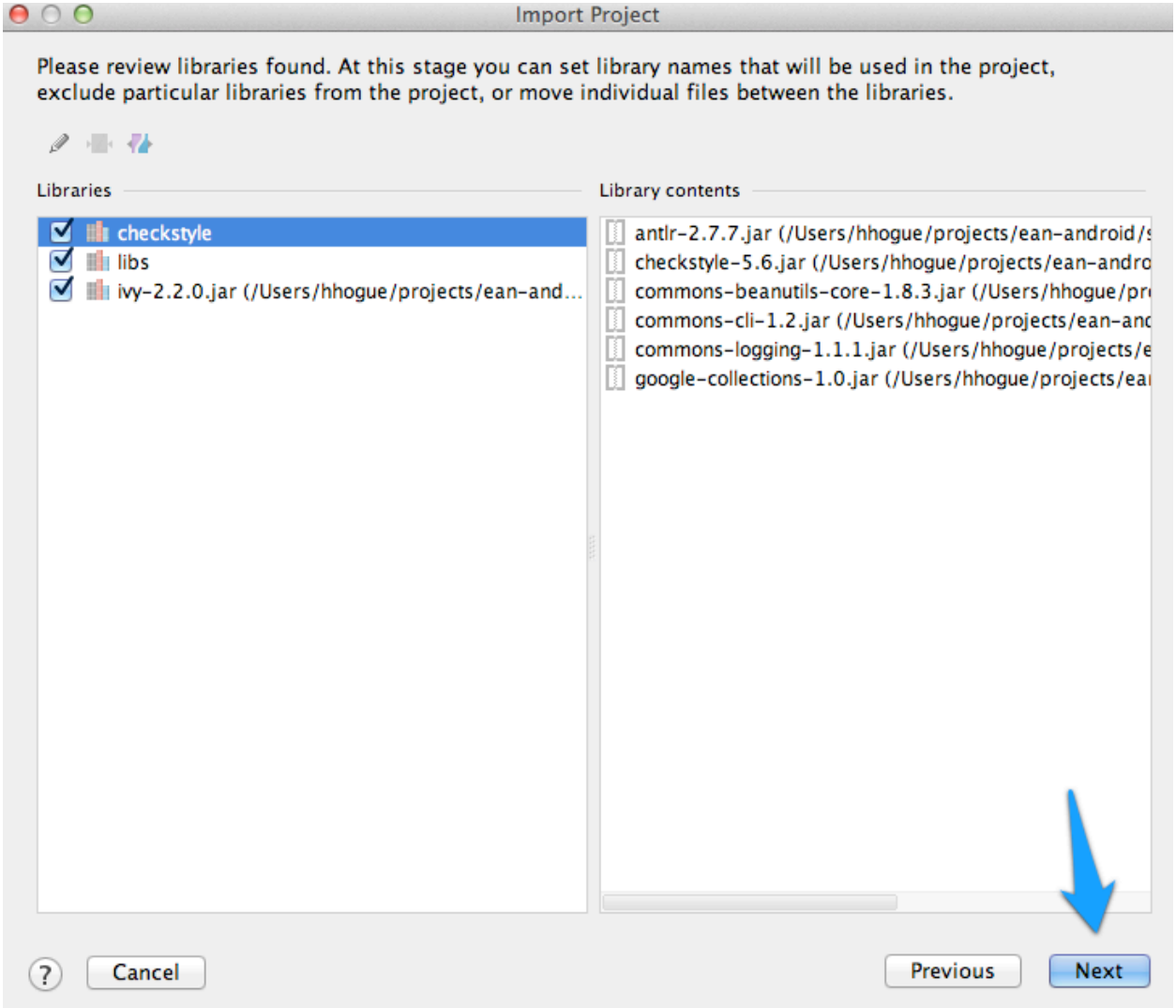

Figure 3.6 - Library selection

Review the module structure and click **Next**.

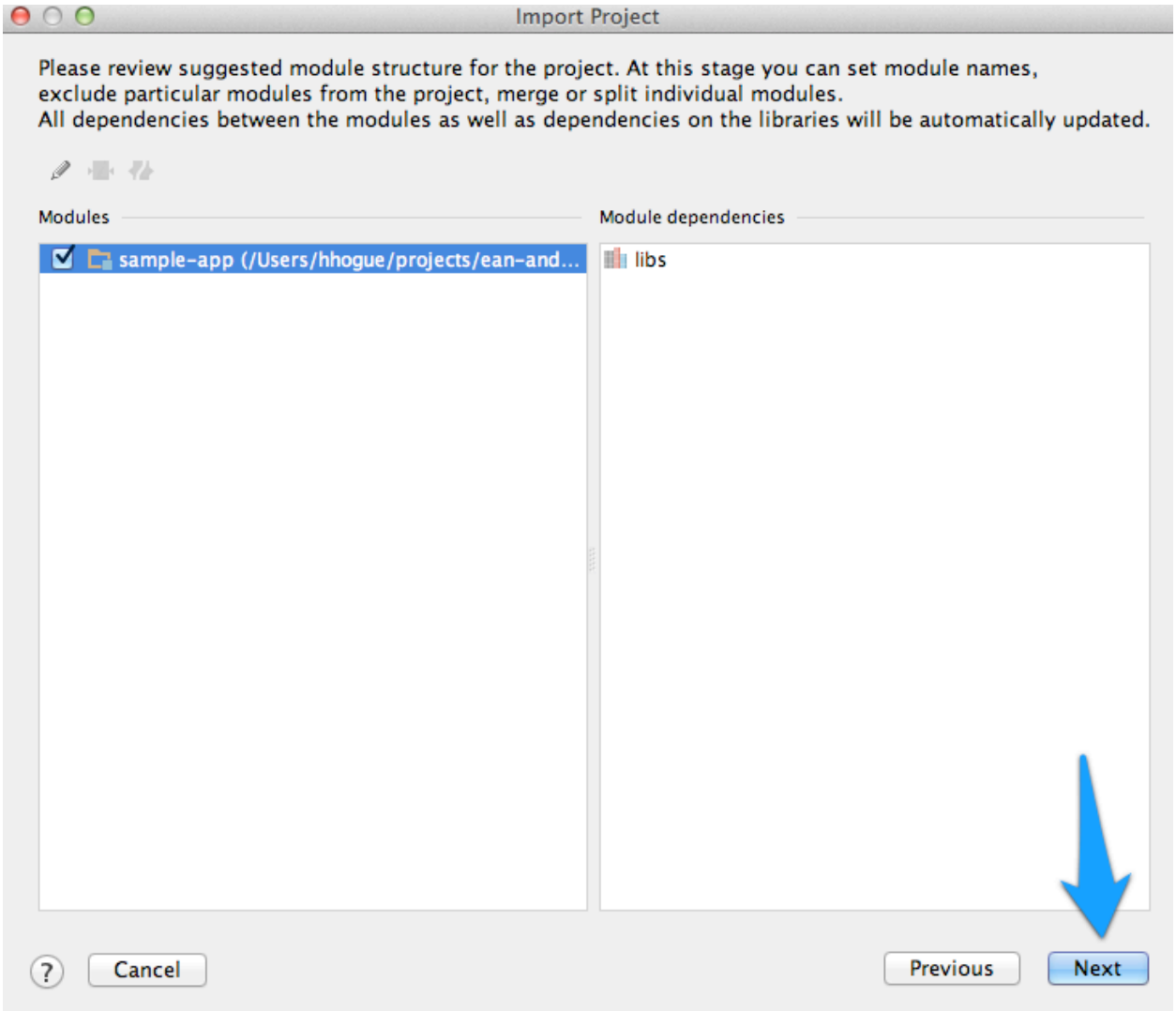

Figure 3.7 - Module review

Select a JDK for the project. The sample application has been confirmed to work with JDK 1.6 and 1.7. This example will use 1.6.

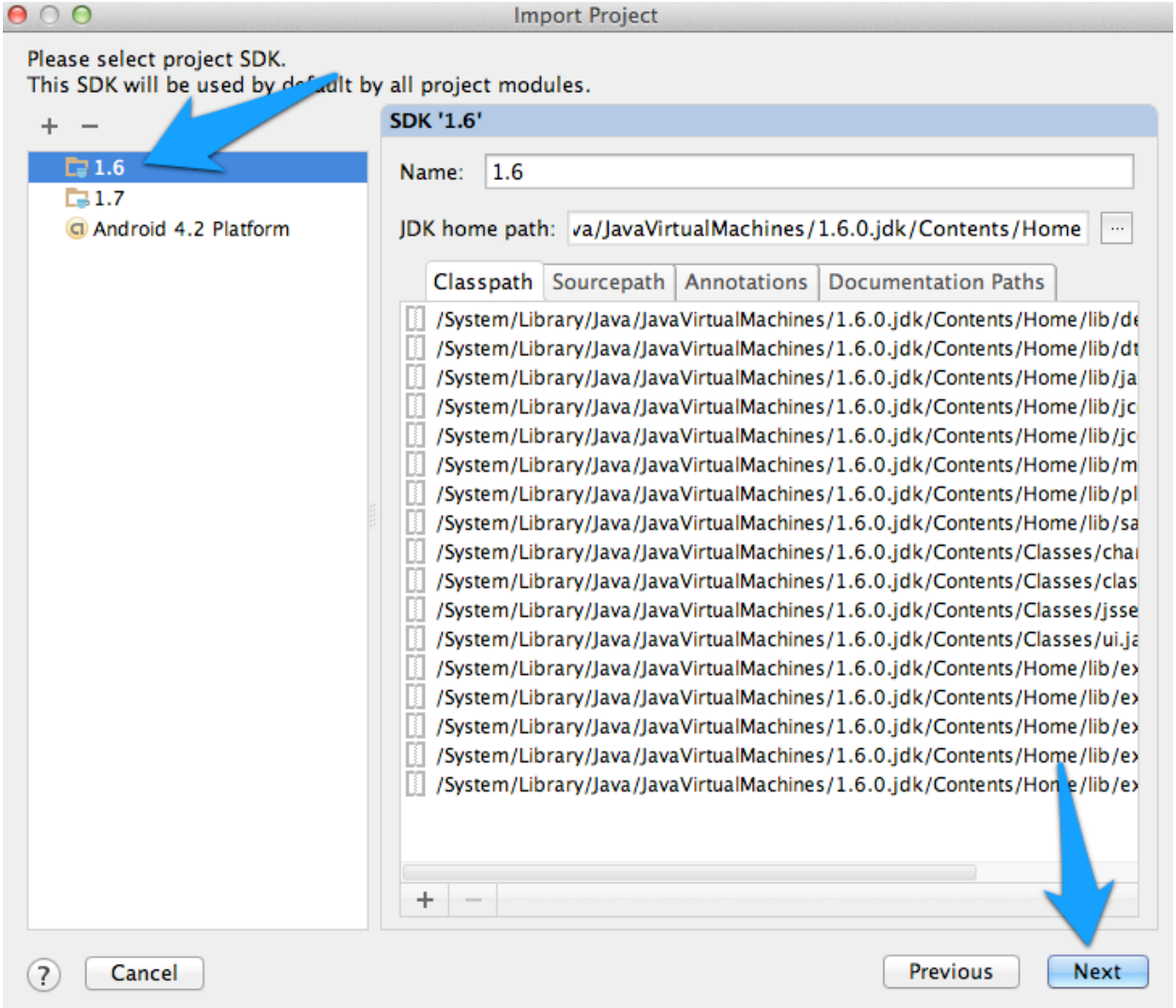

Figure 3.8 - JDK selection

Confirm the Android framework detection by clicking **Finish**.

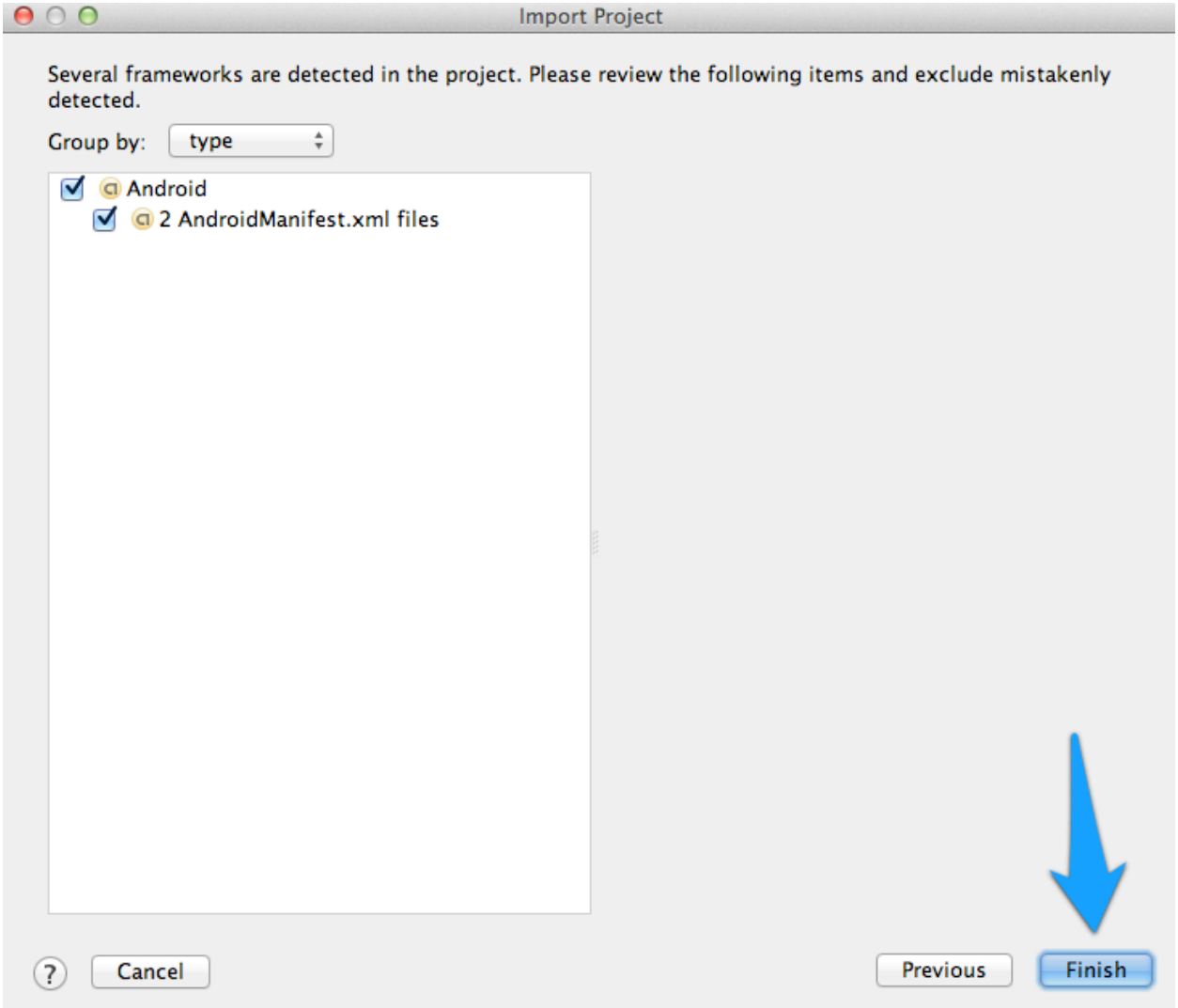

Figure 3.9 - Framework selection

The project will be created and you should see a structure similar to the one in Figure 3.10.

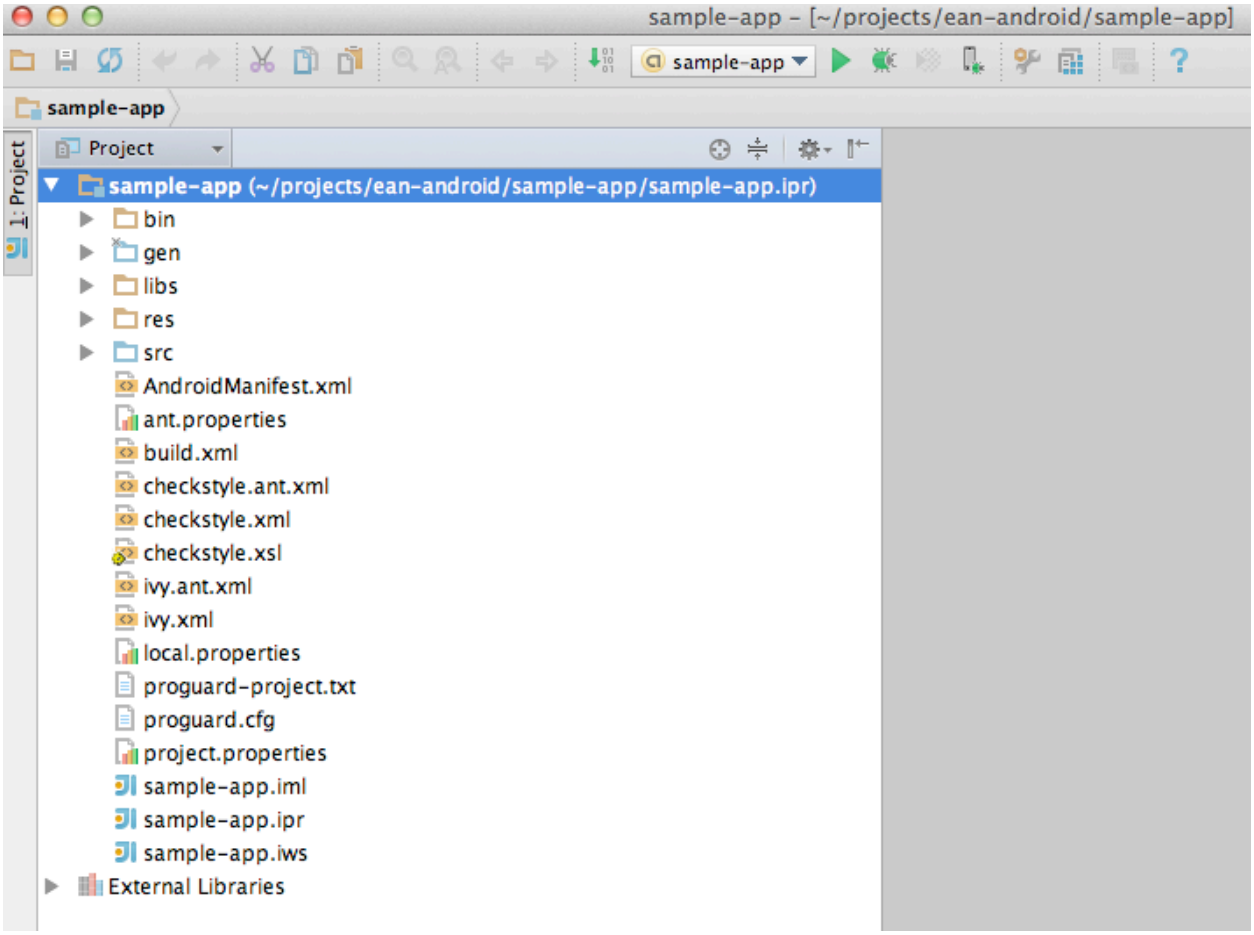

Figure 3.10 - Project structure

Running the sample app

Once the project is set up in IntelliJ, running it is a simple matter of selecting the ***Run 'sample app'*** option from the Run menu.

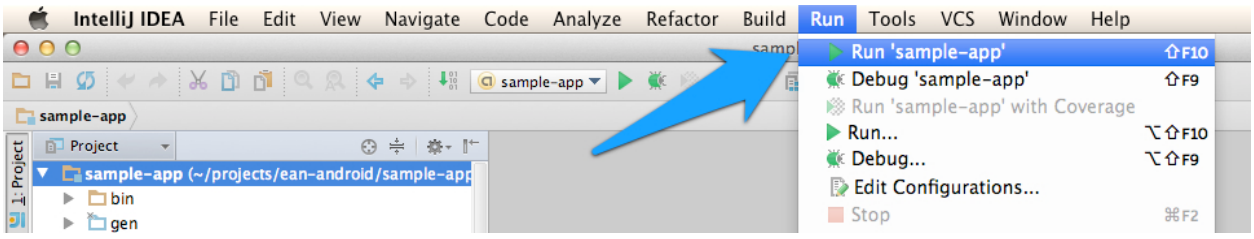

Figure 3.11 - Running the app

You should eventually see a screen similar to the one shown in Figure 3.12.

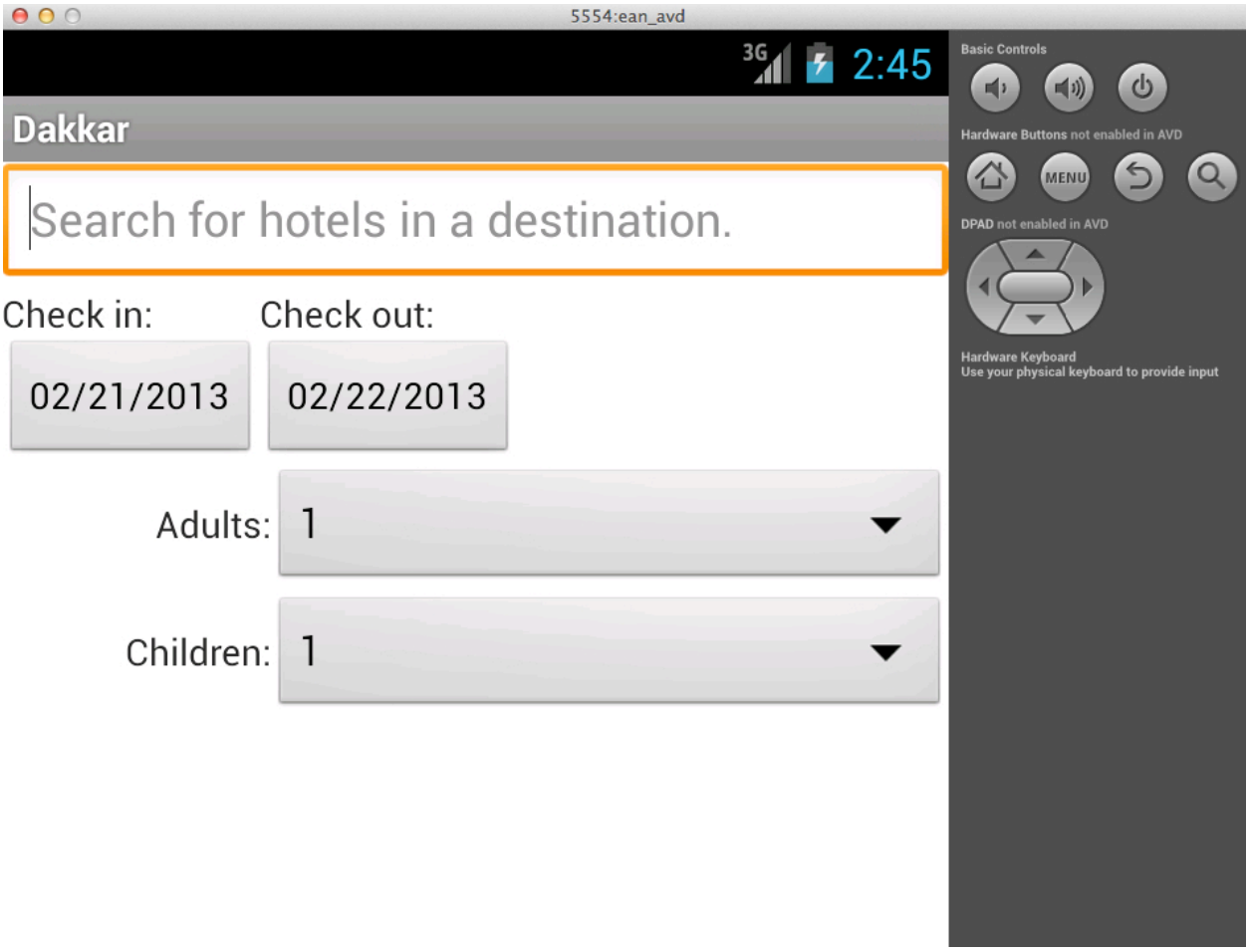

Figure 3.11 - Sample app

Tip: Android emulators tend to be rather sluggish. It would be preferable to use a physical device for testing if one is available.

Chapter 4: Exploring the Sample App

The sample app is a simple Android application that contains the following basic functionality:

- Retrieve a list of destinations
- Retrieve a list of hotels
- Retrieve detailed information for a specific hotel
- Retrieve room availability for a specific hotel
- Book a hotel

Exploring the application is a good place to start when learning how the EAN Android library works. It is also a good template to use for your own Android app that uses the EAN Android library.

Project structure

The sample app contains the following directories:

- */* - project configuration and build settings
- */bin* - all compiled classes and build reports (checkstyle, ivy)
- */gen* - code generated by the Android SDK
- */libs* - all libraries used by the sample app (including the EAN Android library)
- */res* - Android UI components and assets
- */src* - all source files for the sample app (where this chapter spends the majority of its time)

The next several sections will detail the sample app's usage of the EAN Android library and serve as a foundation for building your own Android app.

Common parameter initialization

In order to successfully make requests to the EAN API using the Android library, some initial parameters need to be set at runtime. These parameters are common to most requests so setting them once (before any requests are run) is preferable to setting them in every single request.

The common parameters are:

- CID
- API key
- Shared secret

- Customer user agent
- Locale
- Currency code

Tip: For more information on setting up these values, visit the Getting Started section of the API doc: http://dev.ean.com/docs/read/Getting_Started

Instead of having to pass the same common parameters in every request, they can be set one time as static thread-safe values stored in the `CommonParameters` class. The sample app sets the common parameters in the overridden `onCreate` method in the `com.ean.mobile.SampleApp` class. This ensures that the values are set before any requests are made.

```
@Override
public void onCreate() {
    super.onCreate();
    setupHttpConnectionStuff();
    CommonParameters.cid = "55505";
    CommonParameters.apiKey = "cbrzfta369qwyrm9t5b8y8kf";
    CommonParameters.customerUserAgent = "Android";
    CommonParameters.locale = Locale.US.toString();
    CommonParameters.currencyCode = Currency.getInstance(Locale.US).getCurrencyCode();
}
```

Figure 4.1 - Common parameter initialization

Requests

All requests in the library follow the same basic flow:

Create the appropriate Request object:

```
Request request = new InformationRequest(12345L);
```

Use the RequestProcessor to execute the request and return the appropriate response data:

```
HotelInformation hotelInformation =
RequestProcessor.run(request);
```

The `RequestProcessor` will send the request to the EAN API and parse the raw JSON that is returned.

Since the `run` method contains network operations, it should never be called from the main application thread. It should instead be called via a subclass of the Android SDK's `AsyncTask`.

```
private class ExtendedInformationLoaderTask
        extends AsyncTask<Void, Integer, com.ean.mobile.hotel.HotelInformation> {

    private final long hotelId;

    public ExtendedInformationLoaderTask(final long hotelId) {
        this.hotelId = hotelId;
    }

    @Override
    protected com.ean.mobile.hotel.HotelInformation doInBackground(final Void... voids) {
        try {
            return RequestProcessor.run(new InformationRequest(hotelId));
        } catch (EanWsError ewe) {
            Log.d(SampleConstants.LOG_TAG, "Unexpected error occurred within the api", ewe);
        } catch (UrlRedirectionException ure) {
            SampleApp.sendRedirectionToast(getApplicationContext());
        }
        return null;
    }

    @Override
    protected void onPostExecute(final com.ean.mobile.hotel.HotelInformation hotelInformation) {
        SampleApp.EXTENDED_INFOS.put(hotelId, hotelInformation);
        setExtendedInfoFields();
    }
}
```

Figure 4.2 - Using AsyncTask to execute requests

Now you simply use the following line to start the asynchronous task:

```
new ExtendedInformationLoaderTask(hotel.hotelId).execute((Void) null);
```

Figure 4.3 - Starting the custom asynchronous task

Exception handling

The RequestProcessor.run method throws two exceptions.

`EanWsError` - This is virtually the same exception thrown by the full EAN API when an error message is contained in the response.

`UrlRedirectionException` - This exception is thrown when a request is unexpectedly redirected and is usually caused by a network sign on or connectivity issue.

```
try {
    return RequestProcessor.run(new InformationRequest(hotelId));
} catch (EanWsError ewe) {
    Log.d(SampleConstants.LOG_TAG, "Unexpected error occurred within the api", ewe);
} catch (UrlRedirectionException ure) {
    SampleApp.sendRedirectionToast(getApplicationContext());
}
```

Figure 4.4 - Exception handling

Chapter 5: Creating a New App

The previous chapters served as an introduction and guided tour of the library and the provided sample app. This chapter will serve as a walkthrough for creating your own application that uses the EAN Android library.

We will create an app with two screens. The first screen will contain a text field to enter a hotel ID and a search button to submit the hotel ID. The second screen will contain details for the corresponding hotel. The logic in between will utilize the EAN Android library.

Creating the application

As with the sample app, these examples will be using IntelliJ. It will be easier to follow the instructions if you use IntelliJ as well, but any IDE with Android support will work.

Start by going into the New Project selection screen. You will be presented with several different types of projects. Select "Application Module" under Android and choose a name. We will name our project hotel-details. Click Next.

Warning: You need to have the IntelliJ Android plugin installed for these options to appear!

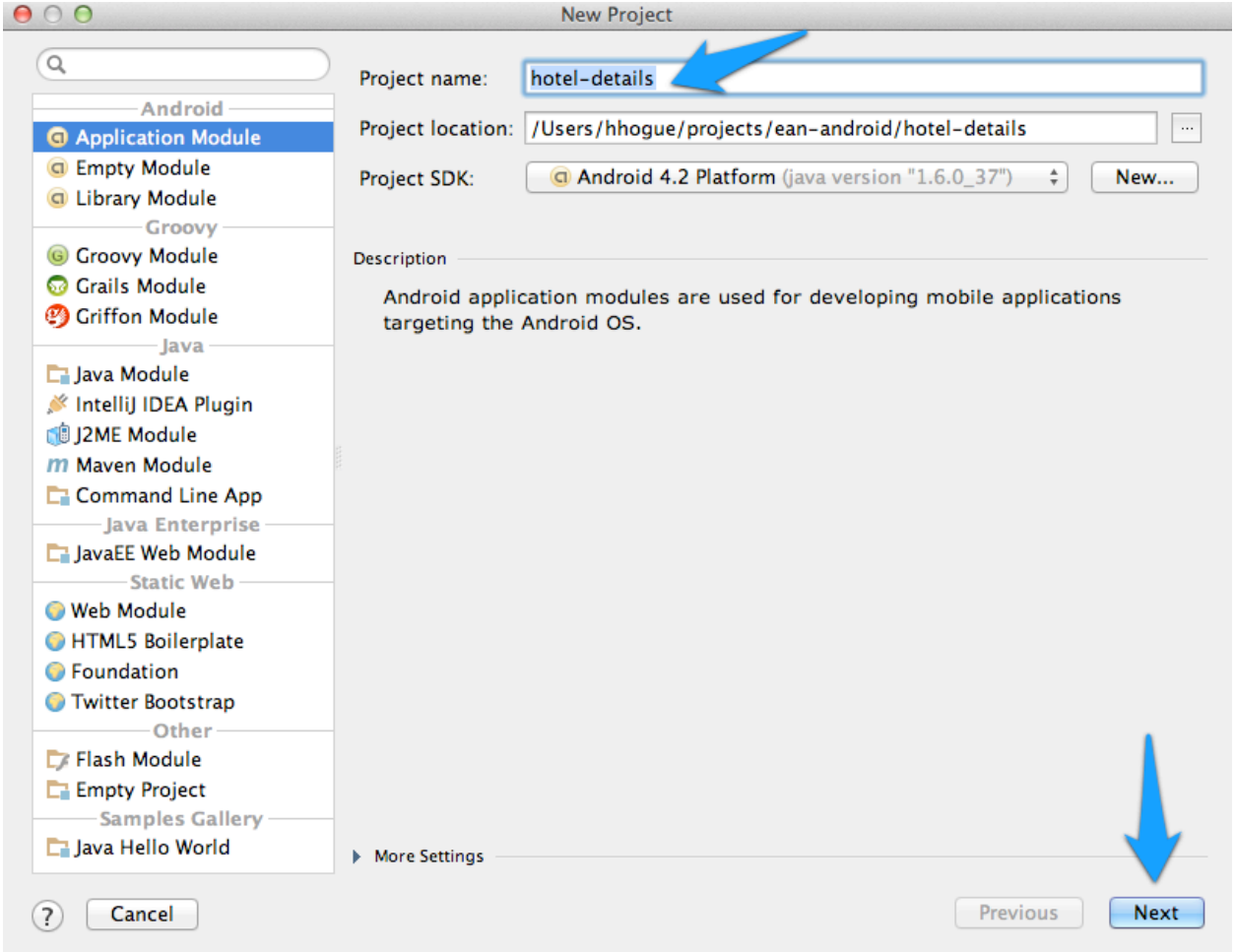

Figure 5.1 - Project type and selection

The next screen allows you to choose a base package name and a name for the default activity. Choose whatever package you'd like and enter HotelSearchActivity for the activity (we'll change the "Hello, World!" part later).

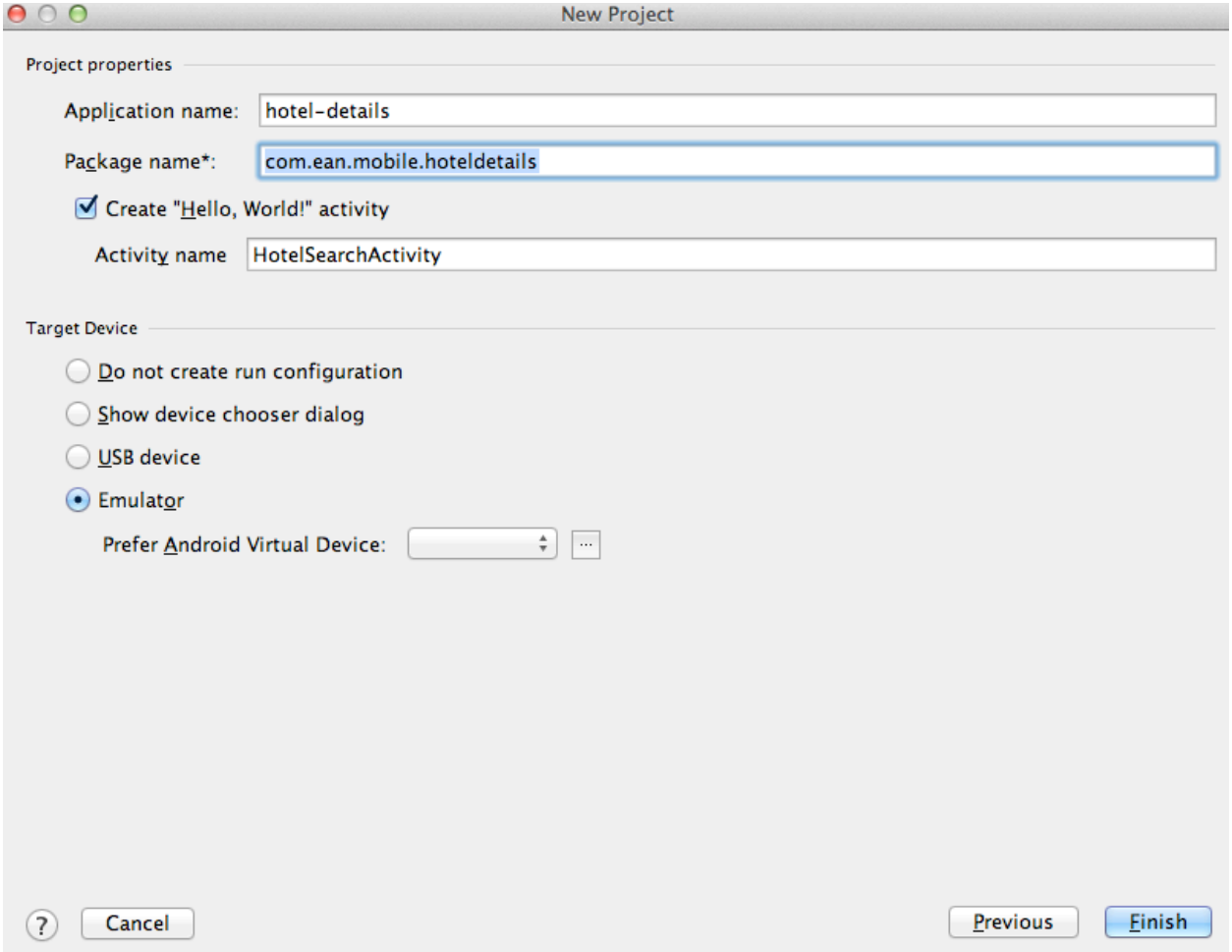

Figure 5.2 - Package name selection

Tip: You can name these files and packages whatever you want, but it will be easier to follow along if you use the same names.

Adding the hotel information search

The project should now be created. The first thing we need to change is the initial view. To do this, open the `main.xml` file in the `res` directory. You will be shown a simple app view with a single line of text. This is our initial search page, so it will need an editable text field and a button. The Palette menu on the right side of the screen contains both -

`EditText` and `Button`. To add them, drag them from the Palette to where you would like to place them on the view.

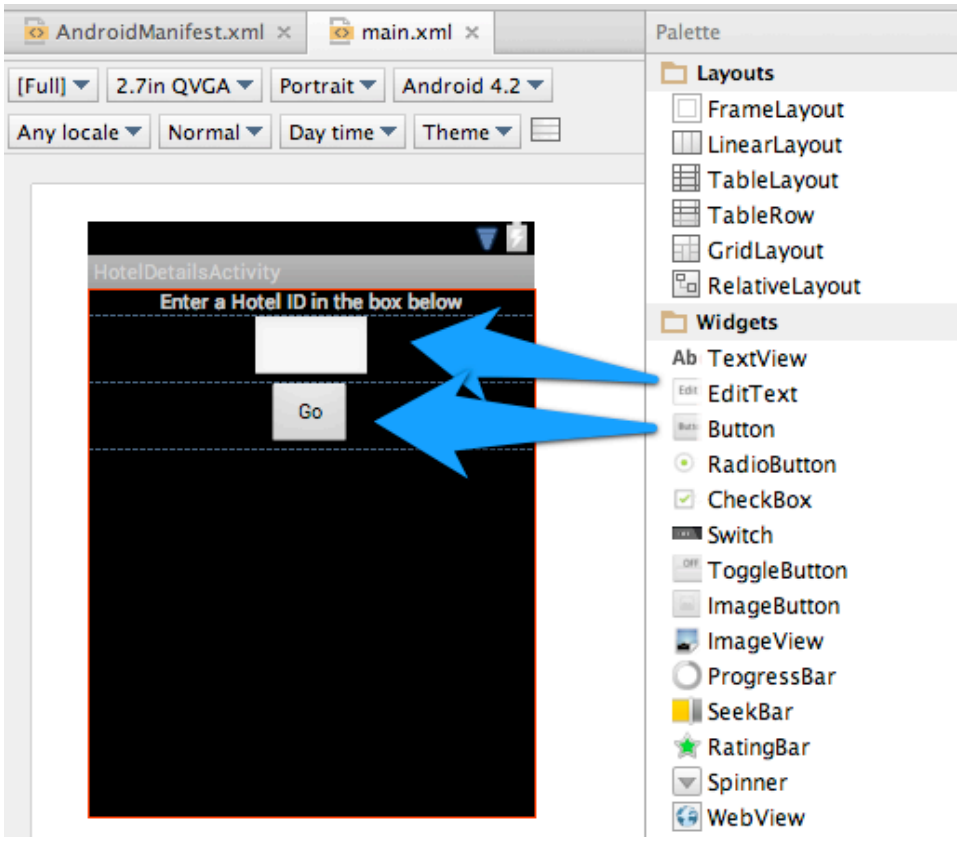

Figure 5.3 - Adding widgets to the main view

Tip: Don't worry about how the UI looks - this walkthrough focuses on the code, not the design.

After they have been added, they will need to be given unique IDs so they can be referenced from the activity code we will be writing. To do this, double click on each widget and change the ID to something unique that you will remember. This walkthrough will be using `@+id/hotelId` and `@+id/goButton`, respectively.
Now that we have the design of the search view ready, we need to add functionality to the text field and button. Since we chose to have a default activity created, it already exists in `src/com.ean.mobile.hoteldetails.HotelSearchActivity`. Open the file to see that it isn't doing much at the moment. It's simply setting the current view to the main view we just modified.

It needs two things:

- A reference to the value of the `hotelId` text box.

- A listener that triggers when the `goButton` is pressed.

We'll deal with the listener first. The class will first need to be modified to implement the `View.OnClickListener` interface and its `onClick` method. The `onCreate` method will also need to include code that creates a reference to the button we added earlier and code that ties it to the listener.

The end product looks like this:

```
package com.ean.mobile.hoteldetails;

import ...

public class HotelDetailsActivity extends Activity implements View.OnClickListener
{
    /** Called when the activity is first created. */
    @Override
    public void onCreate(Bundle savedInstanceState)
    {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.main);

        Button button = (Button)findViewById(R.id.goButton);
        button.setOnClickListener(this);
    }

    public void onClick(View v) {
        // handle button click
    }
}
```

Figure 5.4 - Adding a button listener

The `onClick` method needs to include logic to extract the `hotelId` and send it off to the details view (which hasn't been created yet - we'll get to that later). The `hotelId` can be extracted the same way the `Button` object was created - by using the `findViewById` method. The only difference is that this time we are dealing with an `EditText` object instead of a `Button`. The resulting code is shown in Figure 5.5.

```
public void onClick(View v) {
    EditText hotelIdField = (EditText)findViewById(R.id.hotelId);
    String hotelId = hotelIdField.getText().toString();
}
```

Figure 5.5 - Retrieving the value of the hotelId field

Just a hotel ID won't do a lot of good - we need full hotel details before we can show any new views. This is where the EAN Android library comes in. We will make a call to the library that takes a hotel ID and returns an object full of information specific to the corresponding hotel.

Before going any further, we will need to add the EAN Android library to the project's classpath. Copy the `ean-api-lib.jar` file you built earlier into the project's `libs`

directory. Now go into the Project Structure window (**File --> Project Structure...**) and select the **Libraries** option. Click the plus (**+**) button to add a new Java library and choose the libs directory as its location. Your resulting screen should look like Figure 5.6.

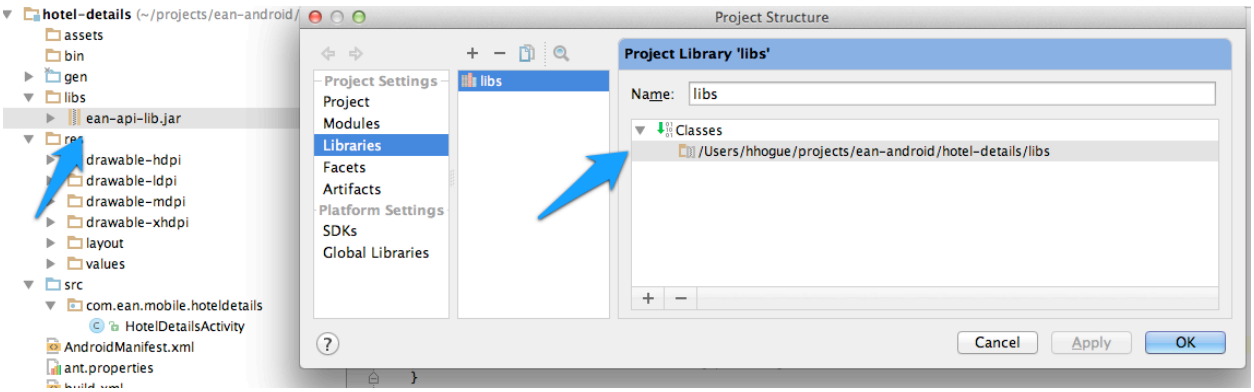

Figure 5.6 - Adding the EAN Android library to the project

You will also need to add the library's lone dependency - Joda Time. You can find the Joda Time jar in the sample app's `libs` directory. Just copy it from there to this project's `libs` directory.

Now we're able to use the library in this project. First thing's first - initialize the common parameters before making any actual requests. We can add the initialization to the `onCreate` method of `HotelDetailsActivity`. We'll use the same test credentials that the sample app uses, shown in Figure 5.7.

```
@Override
public void onCreate(Bundle savedInstanceState)
{
    super.onCreate(savedInstanceState);
    setContentView(R.layout.main);

    CommonParameters.cid = "55505";
    CommonParameters.apiKey = "cbrzfta369qwyrm9t5b8y8kf";
    CommonParameters.customerUserAgent = "Android";
    CommonParameters.locale = Locale.US.toString();
    CommonParameters.currencyCode = Currency.getInstance(Locale.US).getCurrencyCode();

    Button button = (Button)findViewById(R.id.goButton);
    button.setOnClickListener(this);
}
```

Figure 5.7 - Setting common parameters

Warning: Remember that these credentials are only meant to be used for testing purposes!

You may recall from a previous chapter that whenever a call is made from the library, it needs to be done asynchronously via the Android SDK's `AsyncTask`. We will do that here by creating an inner class that extends `AsyncTask`. The necessary code is located in Figure 5.8.

```
public void onClick(View v) {
    EditText hotelIdField = (EditText)findViewById(R.id.hotelId);
    String hotelId = hotelIdField.getText().toString();

    new HotelInformationLoaderTask(Long.parseLong(hotelId)).execute((Void) null);
}

private class HotelInformationLoaderTask extends AsyncTask<Void, Integer, HotelInformation> {

    private final long hotelId;

    public HotelInformationLoaderTask(final long hotelId) {
        this.hotelId = hotelId;
    }

    @Override
    protected HotelInformation doInBackground(final Void... voids) {
        try {
            return RequestProcessor.run(new InformationRequest(hotelId));
        } catch (EanWsError ewe) {
            Log.d("HotelDetails", "Unexpected error occurred within the api", ewe);
        } catch (UrlRedirectionException ure) {
            Log.d("HotelDetails", "The request was unexpectedly redirected", ure);
        }
        return null;
    }

    @Override
    protected void onPostExecute(final HotelInformation hotelInformation) {

    }
}
```

Figure 5.8 - Implementing the API call asynchronously

Tip: You'll want to add better exception handling in a real production app. We're just keeping it simple here for readability.

As you can see, the `onClick` method simply creates a new `HotelInformationLoaderTask`. The task is then responsible for leveraging the EAN Android library to request hotel information using the specified hotel ID. Once the information is returned from the library, the `onPostExecute` method will trigger. This is where we will need to send the data to a new view.

This is a good time to create the second view. We'll keep it simple and just display the description of the hotel in its raw form (which isn't exactly pretty, but that can be an exercise for the reader). To create a new view, right click on the `res/layout` directory

and choose New --> Layout resource file. We'll call this one `details.xml`. All we need to add to it is a single `TextView` widget with an ID of `@+id/hotelDescription`.

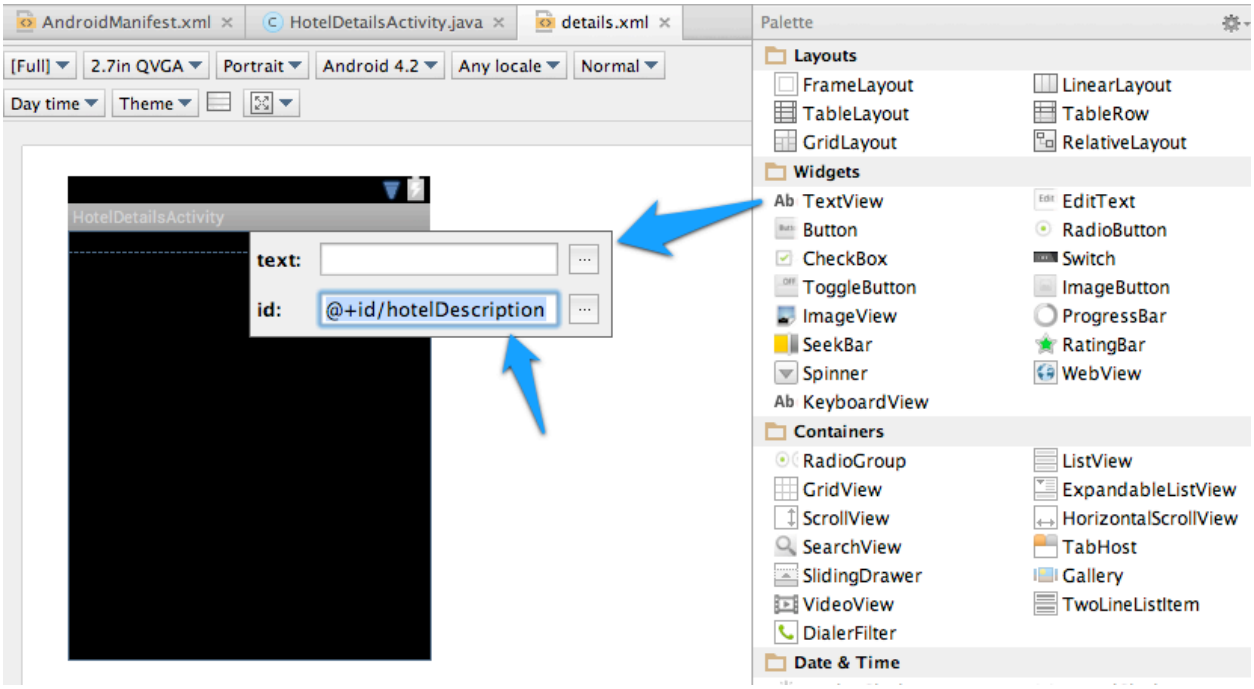

Figure 5.9 - Adding the description label widget

Back to the code, all we have to do now is set the `TextView` widget text to the hotel description and set the current view to the details view we just created. Figure 5.10 shows how this is done.

```
@Override
protected void onPostExecute(final HotelInformation hotelInformation) {
    setContentView(R.layout.details);
    ((TextView)findViewById(R.id.hotelDescription)).setText(hotelInformation.longDescription);
}
```

Figure 5.10 - Setting the description

Warning: Order is important here! You must set the new content view before you set the text view!

Almost there! One more thing to do - give the app permission to access the internet for our API calls. This can be done by adding the following line to the `AndroidManifest.xml`:

Figure 5.11 - Giving the app permission to access the internet

Time to run the application! Go to **Run --> Run 'hotel-details'** and type in 122212 as the hotel ID once the app comes up. Clicking Go should take you to a view containing a (very raw) detailed description of the hotel.

Congratulations, you've just created your first Android app using the EAN Android library!

Where to go from here

There is a lot of opportunity to enhance the sample app. Since it was developed to specifically highlight the key features of the library, it does not cover error handling (it'll crash if anything out of the ordinary happens), data validation (a user can enter anything they want), or detailed UI design (it just spits out a raw string right now), to name a few. However, it is a functional app that uses the EAN Android library and can serve as a stepping stone to a bigger, better app.

Appendix

Resources

https://github.com/ExpediaInc/ean-android/tree/master/sample-app - source code for the sample app
https://bitbucket.org/halhogue/hotel-details - source code for the hotel-details app

Further reading

http://developer.ean.com/docs - EAN API documentation
https://github.com/ExpediaInc/ean-android/wiki - EAN Android library documentation
http://developer.android.com/index.html - Android developer hub