Extracted from:

Hello, Android

Introducing Google's Mobile Development Platform, Fourth Edition

The Pragmatic Bookshelf

Dallas, Texas • Raleigh, North Carolina

The Pragmatic Programmers

Updated for Android 5

Hello, Android

Introducing Google's Mobile Development Platform

Fourth Edition

Ed Burnette

Edited by Susannah Davidson Pfalzer

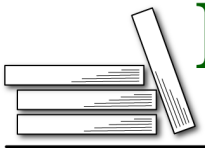
Hello, Android

Introducing Google's Mobile Development Platform, Fourth Edition

Ed Burnette

The Pragmatic Bookshelf

Dallas, Texas • Raleigh, North Carolina



Pragmatic Bookshelf

Our Pragmatic courses, workshops, and other products can help you and your team create better software and have more fun. For more information, as well as the latest Pragmatic titles, please visit us at *https://pragprog.com*.

The team that produced this book includes:

Susannah Davidson Pfalzer (editor)
Potomac Indexing, LLC (indexer)
Liz Welch (copyeditor)
Dave Thomas (typesetter)
Janet Furlow (producer)
Ellie Callahan (support)

For international rights, please contact *rights@pragprog.com*.

Printed in the United States of America.
ISBN-13: 978-1-68050-037-0
Encoded using the finest acid-free high-entropy binary digits.
Book version: P1.0—May 2015

In recent years, more and more functionality has moved into the cloud. Amazon Web Services is a multibillion-dollar business offering cloud-based computing power and storage. Google Apps handle back-office functionality for millions of small and medium-sized businesses. Microsoft is even in the process of turning its ubiquitous Office suite into a cloud service. One thing all these platforms have in common are REST-ful web service interfaces.

REST (REpresentational State Transfer) means many things to different people, but the most pragmatic definition is a technique of building a service on the Internet that you can cause to do something by making simple HTTP (HyperText Transfer Protocol) requests over TCP/IP (Transmission Control Protocol/Internet Protocol) connections. In the most basic terms, there's a server running out there on the web somewhere, which you can connect to over a standard communications port, using a standard protocol, and you can send it requests and commands and get results back.

Web servers such as google.com and microsoft.com that you use every day are a kind of web service. The client (browser) opens a connection on port 80 or 443, sends it a request for a web page or other asset, receives the result, and then closes the connection. On this simple architecture, the entire World Wide Web is built.

In this chapter you'll learn how to make network connections to services on the web from your Android program. This will open up a new world of functionality to your app.

Using Web Services

Android provides a full set of Java-standard networking APIs, such as the `java.net.HttpURLConnection` package, that you can use in your programs. The tricky part is to make the calls asynchronously so that your program's user interface is responsive at all times.

Consider what would happen if you just make a blocking network call in the middle of your user interface code. Suddenly your application would not be able to respond to any events such as touches or button presses. It would appear hung to the user. Obviously, that's something you want to avoid.

The `java.util.concurrent package` is perfect for this kind of work. First created by Doug Lea as a stand-alone library and later incorporated into Java 5, this package supports concurrent programming at a higher level than the regular Java `Thread` class. The `ExecutorService` class manages one or more threads for you, and all you have to do is submit tasks (instances of `Runnable` or `Callable`) to the executor to have them run. An instance of the `Future` class is returned,

which is a reference to some as-yet-unknown future value that will be returned by your task (if any). You can limit the number of threads that are created, and you can interrupt running tasks if necessary.

To illustrate these concepts, let's create a fun little program that calls the Google Suggest API. Have you ever noticed that when you go to a search site like google.com or bing.com and you start typing in a search term, you immediately start seeing suggestions for how to complete the phrase you're typing? For example, if you type in the letters "and" you might see some suggestions having to do with Android. This is implemented with a web service.

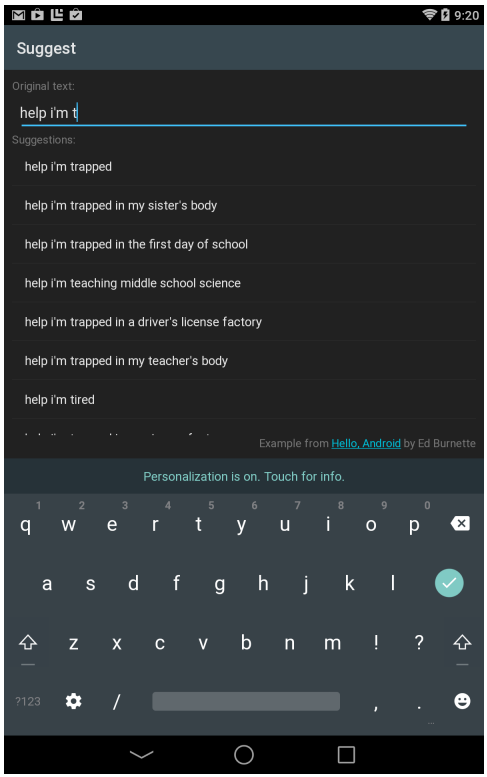
The way it works is that as you type each character into the search box, the browser or web page makes a call to the server to see what could possibly start with the letters you've typed so far. So first you type an "a" and it returns a few likely phrases that begin with a. Then you type an "n" and it gets some phrases that begin with an. And so forth.

Inside the server, it's doing something very clever. Because it knows who you are, where you are, and what you've searched for recently, it customizes the results especially for you. Try this experiment: you and a friend put your computers side by side and go to the same search site. Start typing the same phrase a letter at a time and compare the suggestions. Chances are that they'll be wildly different. The final result from the search after you press Enter will be different too.[1]

The Suggest Example

We're going to create a program that calls the Suggest web service and displays the results just like a search engine or smart address bar would. To use this program, simply start typing a phrase. As you type, the program will use the Suggest web service to fetch suggestions about what comes next.

1. For more information on this phenomenon, see http://en.wikipedia.org/wiki/Filter_bubble.



To create this application, start with a "Hello, Android" application using these parameters:

```
Application name: Suggest
Company Domain: example.org
Form factors: Phone and Tablet
Minimum SDK: API 16: Android 4.1 (Jelly Bean)
Add activity: Blank Activity
Activity Name: MainActivity
Layout Name: activity_main
Title: Suggest
```

Since this example accesses the Internet to make a web service call, we need to tell Android to grant us permission.

Add this line to AndroidManifest.xml before the <application> XML tag:

suggest/src/main/AndroidManifest.xml

```
<uses-permission android:name="android.permission.INTERNET" />
```

The layout for the main activity is pretty simple: a vertical LinearLayout that contains several rows:

suggest/src/main/res/layout/activity_main.xml

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="fill_parent"
    android:layout_height="fill_parent"
    android:orientation="vertical"
    android:padding="10dip" >

    <TextView
        android:layout_width="fill_parent"
        android:layout_height="wrap_content"
        android:text="@string/original_label" />

    <EditText
        android:id="@+id/original_text"
        android:layout_width="fill_parent"
        android:layout_height="wrap_content"
        android:hint="@string/original_hint"
        android:inputType="textNoSuggestions"
        android:padding="10dip"
        android:textSize="18sp" />

    <TextView
        android:layout_width="fill_parent"
        android:layout_height="wrap_content"
        android:text="@string/result_label" />

    <ListView
        android:id="@+id/result_list"
        android:layout_width="fill_parent"
        android:layout_height="0dp"
        android:layout_weight="1" />

    <TextView
        android:id="@+id/eband_text"
        android:layout_width="fill_parent"
        android:layout_height="wrap_content"
        android:gravity="bottom|right"
        android:text="@string/eband" />

</LinearLayout>
```

In this example, we have five rows. The first row is a label telling the user what to type in the second row, which is a text input field. The third row is another label, followed by the list of suggestions. I wanted to publish this example in the Play Store, so the last row is a little advertisement for the book.

Now let's start modifying the MainActivity class. Here's the basic outline:

suggest/src/main/java/org/example/suggest/MainActivity.java

```
Line 1  package org.example.suggest;
-
-   import java.util.ArrayList;
-   import java.util.List;
5   import java.util.concurrent.ExecutorService;
-   import java.util.concurrent.Executors;
-   import java.util.concurrent.Future;
-   import java.util.concurrent.RejectedExecutionException;
-
10  import android.app.Activity;
-   import android.app.SearchManager;
-   import android.content.Intent;
-   import android.os.Bundle;
-   import android.os.Handler;
15  import android.text.Editable;
-   import android.text.TextWatcher;
-   import android.text.method.LinkMovementMethod;
-   import android.view.View;
-   import android.widget.AdapterView;
20  import android.widget.AdapterView.OnItemClickListener;
-   import android.widget.ArrayAdapter;
-   import android.widget.EditText;
-   import android.widget.ListView;
-   import android.widget.TextView;
25
-   public class MainActivity extends Activity {
-      private EditText origText;
-      private ListView suggList;
-      private TextView ebandText;
30
-      private Handler guiThread;
-      private ExecutorService suggThread;
-      private Runnable updateTask;
-      private Future<?> suggPending;
35    private List<String> items;
-     private ArrayAdapter<String> adapter;
-
-     @Override
-     public void onCreate(Bundle savedInstanceState) {
40       super.onCreate(savedInstanceState);
-
-        setContentView(R.layout.activity_main);
-        initThreading();
-        findViews();
45      setListeners();
-       setAdapters();
-
-     }
-  }
```

After declaring a few variables, we define the onCreate() method starting at line 39 to initialize the threading and user interface. Don't worry; we'll fill out all those other methods it calls as we go.

The findViews() method, called from line 44, just gets a handle to all the user interface elements defined in the layout file:

suggest/src/main/java/org/example/suggest/MainActivity.java

```
private void findViews() {
   origText = (EditText) findViewById(R.id.original_text);
   suggList = (ListView) findViewById(R.id.result_list);
   ebandText = (TextView) findViewById(R.id.eband_text);
}
```

The setAdapters() method, called from onCreate() on line 46, defines a data source for the suggestion list:

suggest/src/main/java/org/example/suggest/MainActivity.java

```
/** Set up adapter for list view. */
private void setAdapters() {
   items = new ArrayList<String>();
   adapter = new ArrayAdapter<String>(this,
         android.R.layout.simple_list_item_1, items);
   suggList.setAdapter(adapter);
}
```

In Android, an Adapter is a class that binds a data source (in this case, the list of suggestions) to a user interface control (in this case, a ListView). We use the standard layouts provided by Android for individual items in the list.

Next we set up the user interface handlers in the setListeners() routine (called from line 45 of onCreate()):

suggest/src/main/java/org/example/suggest/MainActivity.java

```
private void setListeners() {
   // Define listener for text change
   TextWatcher textWatcher = new TextWatcher() {
      public void beforeTextChanged(CharSequence s, int start,
            int count, int after) {
         /* Do nothing */
      }
      public void onTextChanged(CharSequence s, int start,
            int before, int count) {
         queueUpdate(1000 /* milliseconds */);
      }
      public void afterTextChanged(Editable s) {
         /* Do nothing */
      }
   };
```

```
    // Set listener on the original text field
    origText.addTextChangedListener(textWatcher);

    // Define listener for clicking on an item
    OnItemClickListener clickListener = new OnItemClickListener() {
        @Override
        public void onItemClick(AdapterView<?> parent, View view,
                int position, long id) {
            String query = (String) parent.getItemAtPosition(position);
            doSearch(query);
        }
    };

    // Set listener on the suggestion list
    suggList.setOnItemClickListener(clickListener);

    // Make website link clickable
    ebandText.setMovementMethod(LinkMovementMethod.getInstance());
}

private void doSearch(String query) {
    Intent intent = new Intent(Intent.ACTION_WEB_SEARCH);
    intent.putExtra(SearchManager.QUERY, query);
    startActivity(intent);
}
```

We define two listeners: one that is called when the input text is changed and one that is called when one of the suggestions is clicked. queueUpdate() puts a delayed update request on the main thread's to-do list using a Handler. We arbitrarily use a 1,000-millisecond delay for text changes.

At the end of the function we call the setMovementMethod() method on the advertisement view. This makes the hyperlink in the ad text live. If the user taps on the link, a browser window will open on that address.