


Lesson 10



# WebView Browser

Victor Matos  
Cleveland State University

Notes are based on:

Android Developers  
<http://developer.android.com/index.html>

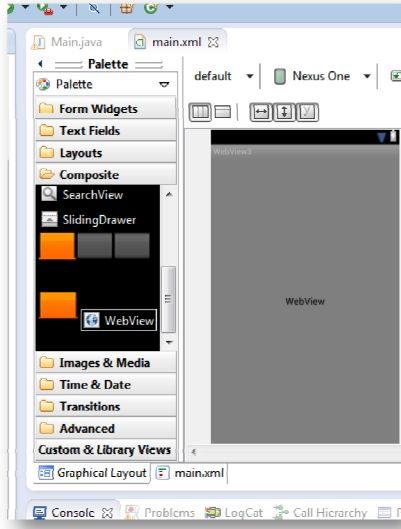
Google Maps Javascript API V3 Basics  
<http://code.google.com/apis/maps/documentation/javascript/basics.html>

The Busy Coder's Guide to Android Development  
by Mark L. Murphy  
Copyright © 2008-2009 CommonsWare, LLC.  
ISBN: 978-0-9816780-0-9

Portions of this page are reproduced from work created and [shared by Google](#) and used according to terms described in the [Creative Commons 3.0 Attribution License](#).

## WebView Browser

- Android provides a built-in *Web browser UI* control called **WebView**.
- WebView is used for displaying local HTML material or browsing Internet pages.
- The Android browser is based on **WebKit**, the same engine that powers *Apple's Safari Web browser*.
- Applications using the **WebView** widget must request **INTERNET permission**.



2

## WebView Browser

The browser will try to access the **Internet** through

- WiFi services,
- cellular network,
- reverse tethering,
- any other technology available.

The **WebKit** engine includes methods to

- navigate forward and backward through a history record,
- zoom in and out,
- perform text searches,
- load data
- stop loading and
- more...

3

## WebView Browser

```
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
  package="com.example.webview_url"
  android:versionCode="1"
  android:versionName="1.0" >

  <uses-sdk
    android:minSdkVersion="8"
    android:targetSdkVersion="15" />
  <uses-permission android:name="android.permission.INTERNET"/>

  <application
    android:icon="@drawable/ic_launcher"
    android:label="@string/app_name"
    android:theme="@style/AppTheme" >

    <activity
      android:name=".MainActivity"
      android:label="@string/title_activity_main" >
      <intent-filter>
        <action android:name="android.intent.action.MAIN" />

        <category android:name="android.intent.category.LAUNCHER" />
      </intent-filter>
    </activity>
  </application>

</manifest>
```

### Warning !!!

In order for your Activity to access the Internet you must add the **INTERNET** permission to your Android Manifest file:

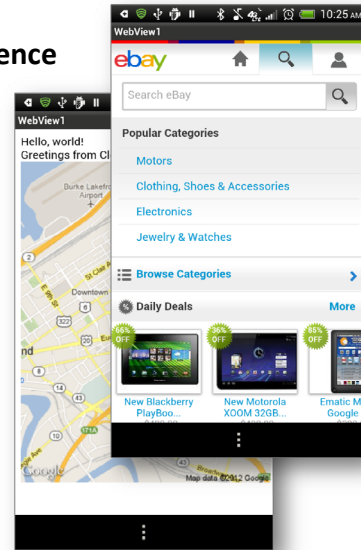
*(see next example)*

## WebView Browser

### Example 1A: A simple browsing experience

This example uses a WebView widget to

- Reach a particular URL (eBay)
- Load a local HTML page in which the user instructs Google Maps to show a static map around given coordinates.



Reference:

<http://developer.android.com/guide/webapps/webview.html>

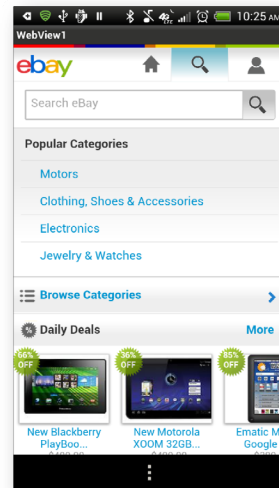
5

## WebView Browser

### Example 1A: A simple browsing experience

Example 2 - Let's go e-shopping

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout
  xmlns:android="http://schemas.android.com/apk/res/android"
  android:orientation="vertical"
  android:layout_width="match_parent"
  android:layout_height="match_parent"
  >
  <WebView
    android:id="@+id/webkit"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
  />
</LinearLayout>
```



6

## WebView Browser

### Example 1A: A simple browsing experience

(1) Let's go e-shopping (2) Visiting Cleveland State U.

```
public void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.activity_main);

    browser = (WebView) findViewById(R.id.webkit);
    browser.getSettings().setJavaScriptEnabled(true); // allow scripts
    browser.setWebViewClient(new WebViewClient()); // page navigation

    // Try version 1 or 2 (please, one at the time!!!)

    // Version 1.-----
    // provide an URL to some web page outside of the app
    browser.loadUrl("http://www.eBay.com");

    // Version 2. -----
    // code your html in-line page (or store page in "/assets" or SD card)
    // we show a static Google map centered on CSU
    // showMyHomeMadeHtmlPage();
} // onCreate
```

This app is  
hard-wired to  
eBay

7

## WebView Browser

### Example 1A: A simple browsing experience

Part 1 - Let's go Back & For

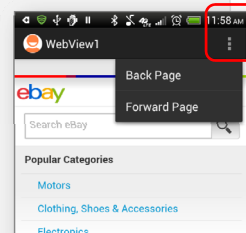
```
// ////////////////////////////////////////
// browser navigation implemented through the menu
@Override
public boolean onCreateOptionsMenu(Menu menu) {
    getMenuInflater().inflate(R.menu.activity_main, menu);
    return true;
} // onCreateOptionsMenu

@Override
public boolean onOptionsItemSelected(MenuItem item) {
    String option = item.getTitle().toString();
    if (option.equals("Forward Page"))
        browser.goForward();

    if (option.equals("Back Page"))
        browser.goBack();

    return true;
} //onOptionsItemSelected

} // class
```



8

## WebView Browser

### Example 1A: A simple browsing experience

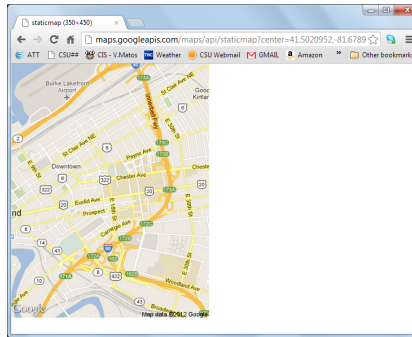
#### Part 2 - Let's visit CSU

Playing with Google Maps: Use your computer to access the following link

<http://maps.googleapis.com/maps/api/staticmap?center=41.5020952,-81.6789717&zoom=14&size=350x450&sensor=false>

It displays a web page showing a static map centered around the the given coordinates (latitude and longitude of Cleveland State University Student's center)

*We want to reproduce this behavior in our Android app.*



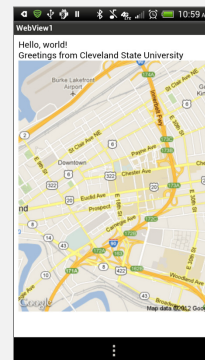
9

## WebView Browser

### Example 1A: A simple browsing experience

#### Part 2 - Let's visit CSU (add this method to the app)

```
private void showMyHomeMadeHtmlPage() {
    String aGoogleMapImage =
        "<img src=\"http://maps.googleapis.com/maps/api/"
        + "staticmap?center=41.5020952,81.6789717&"
        + "zoom=14&size=350x450&sensor=false\"> ";
    String myLocalHtmlPage =
        "<html> "
        + "<body> Hello, world! "
        + "<br> Greetings from Cleveland State University"
        + aGoogleMapImage
        + "</body> "
        + "</html>";
    browser.loadData( myLocalHtmlPage, "text/html", "UTF-8" );
}
```



10

## WebView Browser

### Warning

Your Android application **must** explicitly enable the *JavaScript* code of visited pages to operate .

By default WebViews have *JavaScript* is turned **off**

To activate scripts do this:

```
browser.setSettings().setJavaScriptEnabled(true);
```

11

## WebView Browser

### Warning

To open links clicked by the user (History Record ), provide a **WebViewClient** for your WebView

In our example:

```
WebView browser = (WebView) findViewById(R.id.webview);  
browser.setWebViewClient( new WebViewClient() );
```

12

## WebView Browser

### Browser Commands

There is no navigation toolbar with the WebView widget (*saving space*). You could supply the UI –such as a Menu– to execute the following operations:

- **reload()** to refresh the currently-viewed Web page
- **goBack()** to go back one step in the browser history, and **canGoBack()** to determine if there is any history to trace back
- **goForward()** to go forward one step in the browser history, and **canGoForward()** to determine if there is any history to go forward to
- **goBackOrForward()** to go backwards or forwards in the browser history, where *negative/positive* numbers represent a count of steps to go
- **canGoBackOrForward()** to see if the browser can go backwards or forwards the stated number of steps (following the same positive/negative convention as **goBackOrForward()**)
- **clearCache()** to clear the browser resource cache and **clearHistory()** to clear the browsing history

13

## WebView Browser

Using our running example:

```
browser.goBack();
browser.goForward();
browser.goBackOrForward(-2);
browser.goBackOrForward(+2);

browser.canGoBack();
browser.canGoForward();
browser.canGoBackOrForward(-2);
browser.canGoBackOrForward(+2);

browser.clearCache(true);
browser.clearHistory();
browser.stopLoading();
```

14

## WebView Browser

### Why it is important to allow JavaScript in WebView?

Using JavaScript in a WebView has several important consequences:

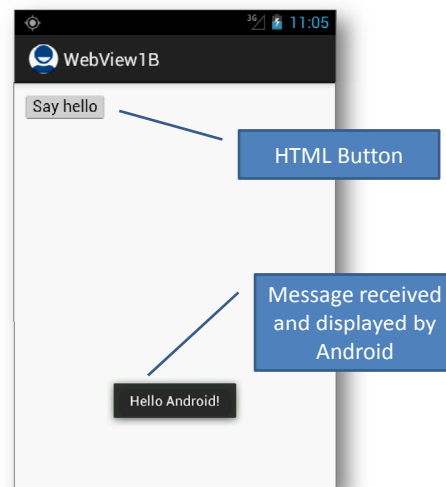
1. Visited pages powered by JavaScript become functional and operate as their designers expected them to be.
2. You could create Android applications whose User-Interface is designed in HTML.
3. Custom-made **Interfaces** could be created to mediate data exchange between the hosting Android application and its locally stored set of HTML pages (powered by JavaScript).

15

## WebView Browser

### Example 1B: JavaScripting

1. This app has two pieces; one is a 'local' HTML page, the other a typical Android app.
2. Android loads the HTML page into a WebView and waits for the user to operate on the page (It contains a button)
3. When the user clicks the (html) button, a JavaScript method is called. The method sends to Android a piece of html data which is shown in an Android Toast view.



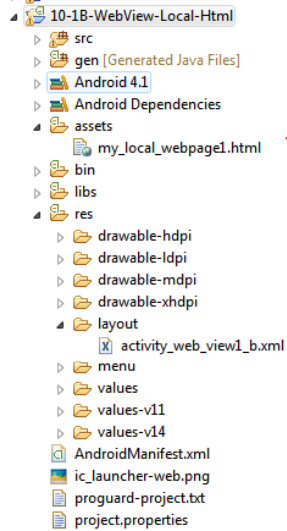
Reference: <http://developer.android.com/guide/webapps/webview.html>

16



## WebView Browser

### Example 1B: JavaScripting



```
<html>
<input type="button"
  value="Say hello"
  onClick="showAndroidToast(
    'Hello Android!')" />

<script type="text/javascript">
  function showAndroidToast(toast) {
    Android.showToast(toast);
  }
</script>
</html>
```

assets/my\_local\_webpage1.html

This local page is stored in the /assets folder. It declares an HTML <input> button and its *onClick* handler.

17

## WebView Browser

### Example 1B: JavaScripting

The Android app defines a WebView control to host HTML pages

```
<RelativeLayout xmlns:android="http://schemas.android.com/apk/res/android"
  xmlns:tools="http://schemas.android.com/tools"
  android:layout_width="match_parent"
  android:layout_height="match_parent" >

  <WebView
    android:id="@+id/webView1"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:layout_alignParentLeft="true" />

</RelativeLayout>
```

18

## WebView Browser

### Example 1B: JavaScripting

An interface object is attached to the WebView. It will allow the Android app and the HTML page to 'talk' to each other.

```
public class WebView1B extends Activity {
    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_web_view1_b);

        WebView browser = (WebView) findViewById(R.id.webView1);
        browser.getSettings().setJavaScriptEnabled(true);
        → browser.addJavascriptInterface(new JavaScriptInterface(this), "AndroidInterface");

        // if the html file is in the app's memory space use:
        → browser.loadUrl("file:///android_asset/my_local_webpage1.html");

        // if the file is in the app's SD card use:
        → // browser.loadUrl("file:///sdcard/my_local_webpage1.html");

        // CAUTION: Manifest must include
        // <uses-permission android:name="android.permission.INTERNET"/>
        // <uses-permission android:name="android.permission.READ_EXTERNAL_STORAGE"/>

    } //onCreate
} //class
```

## WebView Browser

### Example 1B: JavaScripting

The method `showToast()` will be invoked by a javascript `onClick` handler defined in the HTML page, in addition the event will supply data for the `toastMsg` string

```
public class JavaScriptInterface {
    Context mContext;

    /** Instantiate the interface and set the context */
    JavaScriptInterface(Context c) {
        mContext = c;
    }

    /** Show a toast from the web page */
    → public void showToast(String toastMsg) {
        Toast.makeText(mContext, toast, Toast.LENGTH_SHORT).show();
    }
}
```

HTML supplied data

## WebView Browser

### Example 1B: JavaScripting

You need to request Internet permission for accessing pages outside of your app space. If you local HTML pages are stored in the SD card you also need permission to read the device.

```
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
  package="com.example.webview_local_html"
  android:versionCode="1" android:versionName="1.0" >

  <uses-sdk
    android:minSdkVersion="8"
    android:targetSdkVersion="15" />
  <uses-permission android:name="android.permission.INTERNET"/>
  <uses-permission android:name="android.permission.READ_EXTERNAL_STORAGE"/>

  <application
    android:icon="@drawable/ic_launcher"
    android:label="@string/app_name"
    android:theme="@style/AppTheme" >
    <activity
      android:name=".WebView1B"
      android:label="@string/title_activity_web_view1_b" >
      <intent-filter>
        <action android:name="android.intent.action.MAIN" />
        <category android:name="android.intent.category.LAUNCHER" />
      </intent-filter>
    </activity>
  </application>
</manifest>
```

## WebView Browser



### Combining HTML + JAVASCRIPT + ANDROID

#### Advantages offered by Android Development

1. Access to native services on the device, including location services
2. Placement in the Android Market
3. Rapid development using the Android SDK and Eclipse.

#### Advantages offered by Google Maps API

1. Application exists in a server not inside a device.
2. Rapid versioning, removing the requirement for your users to download and install constant updates.
3. More frequent feature additions and bug fixes from Google.
4. Cross-platform compatibility: Using the Maps API allows you to create a single map that runs on multiple platforms.
5. Designed to load *fast* on Android and iPhone devices.

## WebView Browser



### Combining HTML + JAVASCRIPT + ANDROID

#### Learning Strategy

- **WebView2:** Passing complex objects between Android and JS (goal: create interconnectivity)
- **WebView3:** Mapping a fixed location using Google Maps V3 (Pure HTML + JS, just update the server -no need to upgrade ALL devices carrying the application, portability, homogeneous design)
- **WebView4:** Passing a real location object to JS – draw a map centered at given location (mapping current location, combines two above).

23

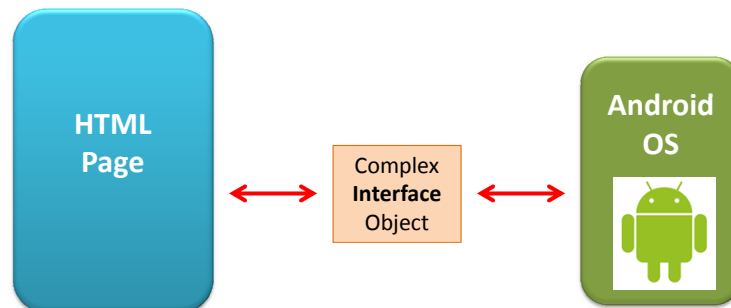
## WebView Browser



### HTML + JAVASCRIPT + ANDROID

#### Example 2: Exchanging objects between Android & JS

1. The app consists of an Android Activity and an HTML page holding JS scripts.
2. The user will interact with the app through the commanding HTML page.
3. A complex interface object will be created by the Android side of the app.
4. The object will include methods to pass data/actions from Android-to-HTML, and HTML-to-Android



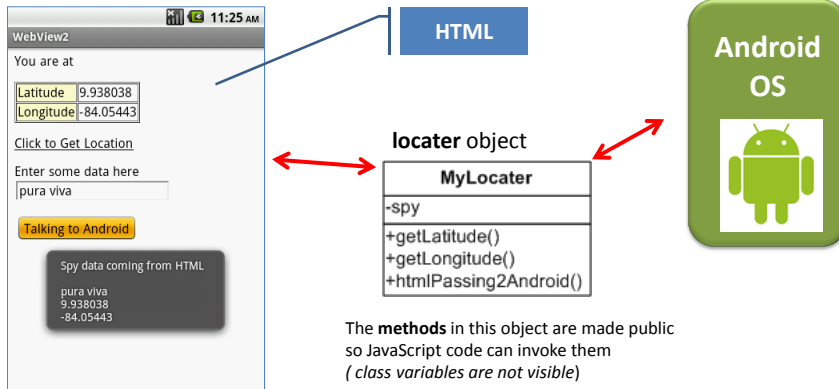
24

## WebView Browser



### HTML + JAVASCRIPT + ANDROID

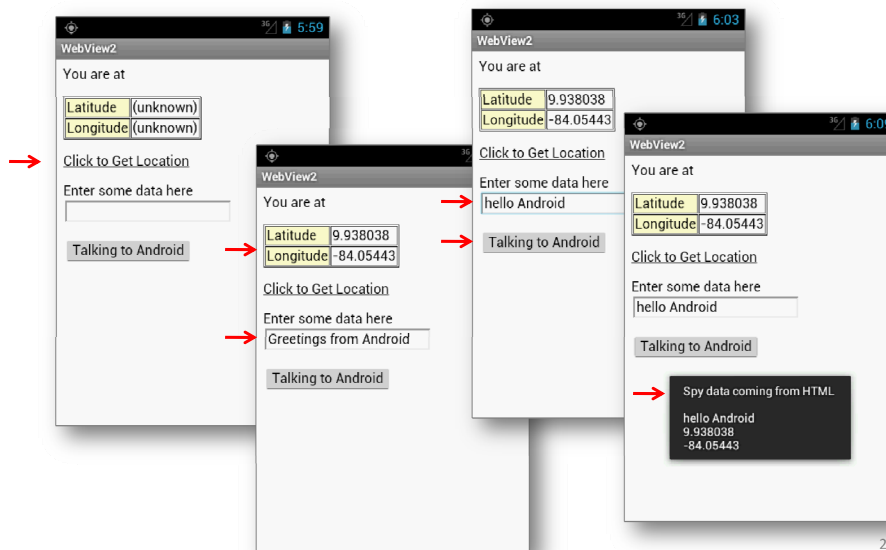
#### Example 2: Exchanging objects between Android & JS



25

## WebView Browser

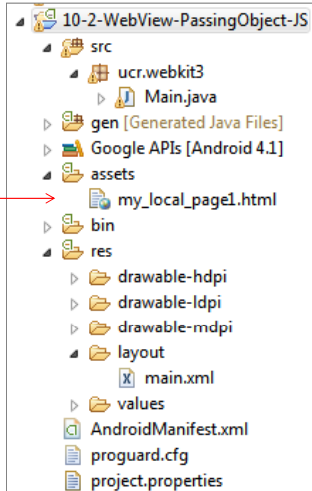
#### Example 2. Passing Objects between Android and JS



26

## WebView Browser

### Example 2. Passing Objects between Android and JS



#### Putting the pieces together:

1. Place a **WebView** in the main.xml file
2. Place html page in the **assets** folder
3. Create the Java **object** to share with JS

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

    <WebView
        android:id="@+id/webview"
        android:layout_width="fill_parent"
        android:layout_height="fill_parent"/>

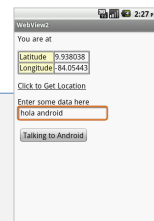
</LinearLayout>
```

27

## WebView Browser

### Example 2. HTML Page (1 of 2) Passing Objects between Android and JS

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN">
<html>
<head>
<title>Android GeoWebOne Demo</title>
<script language="javascript">
function whereami() {
    // html asks android to provide data using object's GET methods
    document.getElementById("lat").innerHTML=locater.getLatitude();
    document.getElementById("lon").innerHTML=locater.getLongitude();
    document.getElementById("myText").value = locater.getCommonData();
}
function talkBack2Android() {
    // bridge object used to send local (html) data to android app
    locater.setCommonData("Greetings from html");
    var spyHtml = "Spy data coming from HTML\n"
        + "\n" + document.getElementById("myText").value
        + "\n" + document.getElementById("lat").innerHTML
        + "\n" + document.getElementById("lon").innerHTML;
    locater.htmlPassing2Android(spyHtml);
}
</script>
</head>
```



## WebView Browser

### Example 2.

#### HTML Page (2 of 2) Passing Objects between Android and JS

```

<body>
  <p>You are at</p>
  <table border="1" cellspacing="1" cellpadding="1">
    <tr>
      <td bgcolor="#FFFFCC"> Latitude </td>
      <td <span id="Lat"> (unknown) </span></td>
    </tr>
    <tr>
      <td bgcolor="#FFFFCC"> Longitude </td>
      <td <span id="Lon"> (unknown) </span></td>
    </tr>
  </table>

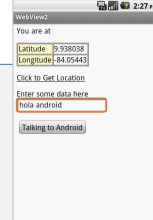
  <p><a onClick="whereami()" ><u>Click to Get Location</u></a></p>

  <p> Enter some data here <input type="text" id="myText" />

  <p> <input type="button"
    onClick= "talkBack2Android()"
    onClick= "talkBack2Android()"
    value="Talking to Android">

</body>
</html>

```



## WebView Browser

### Example 2. Passing Objects between Android and JS

```

public class Main extends Activity {

  WebView browser;
  MyLocater locater = new MyLocater();
  Location mostRecentLocation;

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

    // get a location fix (lat, lon)
    mostRecentLocation = fakeGetLocation();

    // set up the webview to show location results
    browser = (WebView) findViewById(R.id.webview);
    browser.getSettings().setJavaScriptEnabled(true);

    browser.addJavascriptInterface(locater, "locater");

    browser.loadUrl("file:///android_asset/my_local_page1.html");
  } //onCreate

```

## WebView Browser

### Example 2. Passing Objects between Android and JS

```
private Location fakeGetLocation() {
    // faking the obtaining of a location object (discussed later!)
    Location fake = new Location("fake");
    fake.setLatitude(9.938038);
    fake.setLongitude(-84.054430);
    return fake;
}

// "MyLocater" is used to pass data back and forth between
// Android and JS code-behind

public class MyLocater {
    private String commonData = "XYZ";

    public double getLatitude() {
        if (mostRecentLocation == null) return (0);
        else return mostRecentLocation.getLatitude();
    }

    public double getLongitude() {
        if (mostRecentLocation == null) return (0);
        else return mostRecentLocation.getLongitude();
    }
}
```

## WebView Browser

### Example 2. Passing Objects between Android and JS

```
public void htmlPassing2Android(String dataFromHtml) {
    Toast.makeText(getApplicationContext(),
        "1\n" + commonData, 1).show();

    commonData = dataFromHtml;

    Toast.makeText(getApplicationContext(),
        "2\n" + commonData, 1).show();
}

public String getCommonData(){
    return commonData;
}

public void setCommonData(String actualData){
    commonData = actualData;
}

} // MyLocater

} // class
```



## WebView Browser

### Example 3. Using Google JavaScript Maps API V3

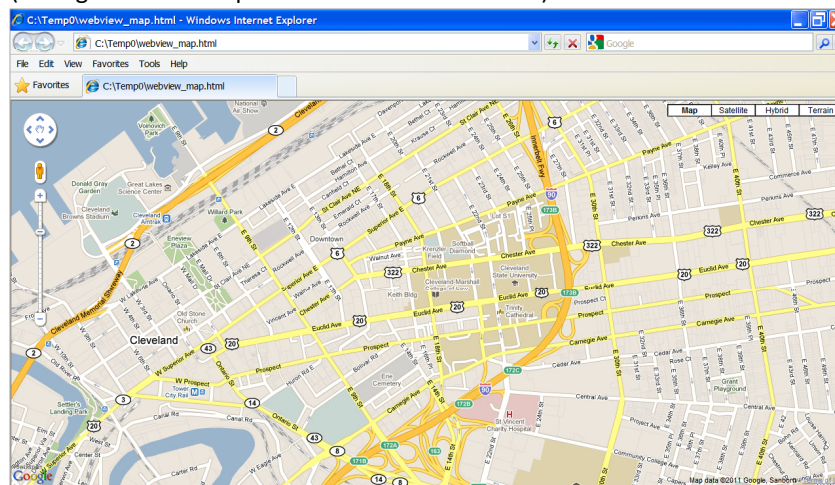
- The **Google Maps Javascript API** is a free service that lets you embed Google Maps in your own web pages.
- It is especially designed to be faster and more applicable to mobile devices (as well as traditional desktop browser applications)
- The API provides a number of utilities for manipulating maps (just like on the <http://maps.google.com> web page) and adding content to the map through a variety of services, allowing you to create robust maps applications on your website/app.

Link: <https://developers.google.com/maps/documentation/javascript/>

33

## WebView Browser

### Example 3. Using Google JavaScript Maps API V3 A Google map centered around “Cleveland State University, Ohio” (seeing here with IExplorer on Windows machine)

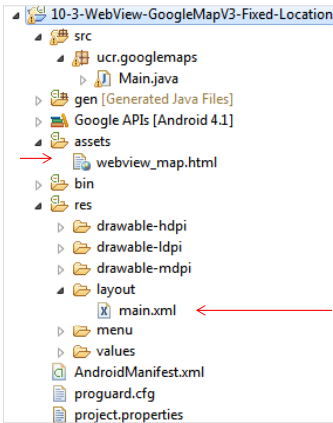


Link: <http://code.google.com/apis/maps/documentation/javascript/basics.html>

34

## WebView Browser

### Example 3. Using Google JavaScript Maps API V3



#### Putting the pieces together:

1. Place a **WebView** in the main.xml file
2. Place html page in the **assets** folder
3. Add **permission** requests to manifest
4. Connect to Java code

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

  <WebView
    android:id="@+id/webview"
    android:layout_width="fill_parent"
    android:layout_height="fill_parent"/>

</LinearLayout>
```

35

## WebView Browser

### Example 3. Using Google JavaScript Maps API V3 1 of 2

```
<!DOCTYPE html>
<html>
<head>
  <meta name="viewport" content="initial-scale=1.0, user-scalable=no" />
  <style type="text/css">
    html { height: 100% }
    body { height: 100%; margin: 0px; padding: 0px }
    #map_canvas { height: 100% }
  </style>
  <script type="text/javascript"
    src="http://maps.google.com/maps/api/js?sensor=false">
  </script>
  <script type="text/javascript">
    function initialize() {
      var latlng = new google.maps.LatLng(41.5020952, -81.6789717);
      var myOptions = { zoom: 15,
        center: latlng,
        mapTypeId: google.maps.MapTypeId.ROADMAP };

      var map = new google.maps.Map( document.getElementById("map_canvas"),
        myOptions );
    }
  </script>
</head>
```

This is the HTML page:  
webview\_map.html

36

## WebView Browser

### Example 3. Using Google JavaScript Maps API V3 2 of 2

```

<body onload="initialize()">
  <div id="map_canvas" style="width:100%; height:100%"></div>
</body>

</html>

```

This is the HTML page: webview\_map.html

37

## WebView Browser

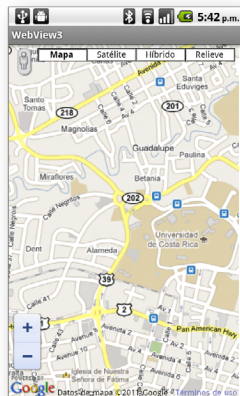
### Example 3. Using Google JavaScript Maps API V3

Add the following permission requests to the AndroidManifest.xml file

```

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

```



Map image shown on an Android device

38

## WebView Browser

### Example 3. Using Google JavaScript Maps API V3

```
public class WebView4 extends Activity {
    WebView browser;

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

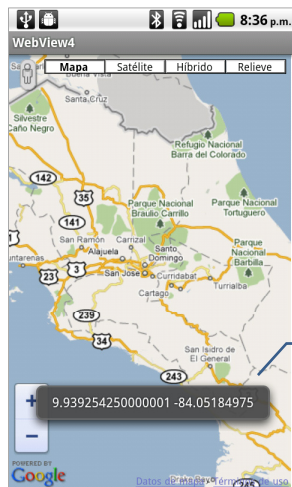
        // connect browser to local html file showing map
        browser = (WebView) findViewById(R.id.webview);
        browser.getSettings().setJavaScriptEnabled(true);
        browser.loadUrl("file:///android_asset/webview_map.html");
    }
}
```

39

## WebView Browser

### Example 4. Using Google JavaScript Maps API V3 (real locations)

Delay until  
Service  
Lesson



This experience combines the two previous examples:

- The goal is to use an Android object to pass 'real location' data to an html webpage.
- The page contains a JavaScript fragment to draw a map centered on the given coordinates.

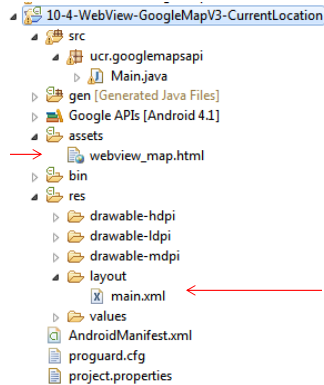
Latitude and longitude detected by the device.  
Image taken from an Android phone.

**Warning:** Make sure your target is: **Google APIs (API Level 8) or higher.**

40

## WebView Browser

### Example 4. Using Google JavaScript Maps API V3 (real locations)



#### Putting the pieces together:

1. Place a **WebView** in the main.xml file
2. Place html page in the **assets** folder
3. Add **permission** requests to manifest
4. Connect to Java code

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

    <WebView
        android:id="@+id/webview"
        android:layout_width="fill_parent"
        android:layout_height="fill_parent"/>

</LinearLayout>
```

41

## WebView Browser

### Example 4. assets/webview\_map.html

This HTML page creates a map using 'real' coordinates passed inside the 'locater' object

```
<!DOCTYPE html>
<html>
<head>
    <meta name="viewport" content="initial-scale=1.0, user-scalable=no" />
    <meta http-equiv="content-type" content="text/html; charset=UTF-8"/>
    <title>Google Maps JavaScript API v3 Example4: Marker Simple</title>

<style type="text/css">
    html { height: 100% }
    body { height: 100%; margin: 0px; padding: 0px }
    #map_canvas { height: 100% }
</style>

<script type="text/javascript" src="http://maps.google.com/maps/api/js?sensor=false"></script>
<script type="text/javascript">
    function initialize() {
        //var myLatLng = new google.maps.LatLng(41.5020952, -81.6789717); //CSU coordinates
        var myLatLng = new google.maps.LatLng(locater.getLatitude(), locater.getLongitude());
        var myOptions = { zoom: 17,
                        center: myLatLng,
                        mapTypeId: google.maps.MapTypeId.ROADMAP
                      }
        var map = new google.maps.Map(document.getElementById("map_canvas"), myOptions);

        var marker = new google.maps.Marker( { position: myLatLng, map: map });
    }
</script>
</head>
<body onload="initialize()">
    <div id="map_canvas"></div>
</body>
</html>
```

## WebView Browser

### Example4. Main Activity Android & Google Map V3 App (real locations)

```
public class Main extends Activity implements LocationListener {

    private WebView browser;
    LocationManager locationManager;
    MyLocater locater = new MyLocater();

    @Override
    protected void onDestroy() {
        super.onDestroy();
        // cut location service requests
        locationManager.removeUpdates(this);
    }

    private void getLocation() {
        locationManager = (LocationManager)
            getSystemService(Context.LOCATION_SERVICE);
        Criteria criteria = new Criteria();
        criteria.setAccuracy(Criteria.ACCURACY_FINE); // use GPS device
        //criteria.setAccuracy(Criteria.ACCURACY_COARSE); // towers, wifi
        String provider = locationManager.getBestProvider(criteria, true);

        // In order to make sure the device is getting the location, request
        // updates [wake up after changes of: 5 sec. or 10 meter]
        locationManager.requestLocationUpdates(provider, 5, 10, this);
        locater.setNewLocation(locationManager.getLastKnownLocation(provider));
    }
}
```

43

## WebView Browser

### Example4. Main Activity Android & Google Map V3 App (real locations)

```
@Override
public void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.main);
    getLocation();
    setupBrowser();
    this.setRequestedOrientation(ActivityInfo.SCREEN_ORIENTATION_PORTRAIT);
} //onCreate

/** Set up the browser object and load the page's URL */
@SuppressLint("SetJavaScriptEnabled")
private void setupBrowser() {
    final String centerMapURL = "javascript:centerAt("
        + locater.getLatitude() + ","
        + locater.getLongitude() + ")";

    // set up the browser to show location results
    browser = (WebView) findViewById(R.id.webview);
    browser.getSettings().setJavaScriptEnabled(true);
    browser.addJavaScriptInterface(locater, "locater");
    browser.loadUrl("file:///android_asset/webview_map.html");

    // Wait for the page to load then send the location information
    browser.setWebViewClient(new WebViewClient() {
        @Override
        public void onPageFinished(WebView view, String url) {
            browser.loadUrl(centerMapURL);
        }
    });
} //setupBrowser
```

## WebView Browser

### Example4. Main Activity Android & Google Map V3 App (real locations)

```

@Override
public void onLocationChanged(Location location) {
    String lat = String.valueOf(location.getLatitude());
    String lon = String.valueOf(location.getLongitude());
    Toast.makeText(getApplicationContext(), lat + "\n" + lon, 1).show();
    locater.setNewLocation(location);
}

@Override
public void onProviderDisabled(String provider) {
    // needed by Interface. Not used
}

@Override
public void onProviderEnabled(String provider) {
    // needed by Interface. Not used
}

@Override
public void onStatusChanged(String provider, int status, Bundle extras) {
    // needed by Interface. Not used
}

```

45

## WebView Browser

### Example4. Main Activity Android & Google Map V3 App (real locations)

```

//////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
// An object of type "MyLocater" will be used to pass data back and
// forth between the Android app and the JS code behind the html page.
//////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
public class MyLocater {
    private Location mostRecentLocation;

    public void setNewLocation(Location newCoordinates){
        mostRecentLocation = newCoordinates;
    }

    public double getLatitude() {
        if (mostRecentLocation == null) return (0);
        else return mostRecentLocation.getLatitude();
    }

    public double getLongitude() {
        if (mostRecentLocation == null) return (0);
        else return mostRecentLocation.getLongitude();
    }
} // MyLocater

} // class

```

46

## WebView Browser

**Example4. Main Activity** Android & Google Map V3 App (real locations)

Testing the app using the Emulator Control Panel

The screenshot shows the Eclipse IDE interface. On the left, the 'Devices' tab lists various system components for the 'IceCream4x-16 [emulator-5554]' device. The 'Emulator Control' tab is active, showing settings for telephony and location. The 'Location Controls' section has 'Manual' selected, and the 'Send' button is highlighted with a red box and an arrow. To the right, the mobile emulator displays a Google Map with a red location pin and labels for 'Amphitheatre Pkwy' and 'Charleston Rd'. The page number '47' is visible in the bottom right corner.

## WebView Browser

**Where to go next?**

**Google Maps Developers Documentation**

<https://developers.google.com/maps/documentation/>

**Google Maps API – Webservices**

<http://code.google.com/apis/maps/documentation/webservices/index.html>

**Google Maps JavaScript API V3**

<http://code.google.com/apis/maps/documentation/javascript/tutorial.html>

**Questions ?**