Android Development - Using the Web Browser

Android, along with several other mobile operating systems uses the open source WebKit web browser. WebKit is the same code base that Apple uses for Safari on the desktop and on the iPhone. One of the great things about WebKit unlike some mobile browsers of the past is that it doesn't attempt to change the page or offer a "mobile view" of the page, instead it simply displays it as it is meant to be displayed on a desktop browser.

WIth the ever increasing capabilities of the browser, WebKit in particular (mobile or otherwise), utilizing a browser component in a native application may give the best of both worlds: device independence in the design (because it is simply HTML, CSS and JavaScript, porting that part to devices running other operating systems is easier) as well as retaining the ability to distribute the application as an application which offers some marketing advantages by using various application stores (such as the Android Market) and the ability to leverage that infrastructure to charge for the app, push out updates and so on.

Launching a web view with an Intent

There are a couple of different ways to utilize a "web view" in an Android application. The easiest way, as usual is by using an Intent. This example uses an Intent to launch the browser application:

```
package com.mobvcasting.webintent;

import android.app.Activity;
import android.content.Intent;
import android.net.Uri;
import android.os.Bundle;
import android.view.View;
import android.view.View.OnClickListener;
import android.widget.Button;

public class WebIntent extends Activity
{
     Button theButton;

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

        theButton = (Button) findViewById(R.id.Button01);
        theButton.setOnClickListener(new OnClickListener()
        {
               public void onClick(View v)
               {
                    Intent intent = new Intent(Intent.ACTION_VIEW);
                    Uri uri = Uri.parse("http://www.mobvcasting.com/mmtest/index.html");
                    intent.setData(uri);
                    startActivity(intent);
               }
        });

     }
}
```

What follows is the HTML source of the index.html page referenced above:

```
<html>
	<head>
		<title>Hello World</title>
		<script type="text/javascript">
			alert("JavaScript Alert");
			document.writeln("Line Generated By JavaScript");
		</script>
	</head>
	<body>
		<h1>Hello There</h1>
		<a href="#" onClick="window.jsinterface.specialClick();">Click Me</a>
	</body>
</html>
```

Running the above code with this HTML, you'll notice a couple of things. First when the page loads an Alert pops up that says "JavaScript Alert". In the body of the document you'll see first a line of text that says "Line Generated By JavaScript" and then a link that says "Click Me".

The two JavaScript references show that JavaScript is running and fully implemented in the browser view. When you click on the "Click Me" link though, nothing will probably happen. This is not a bug, it just points the way to something we'll be doing later on.

Displaying a web page inside an Activity

Using the WebView object we can create our own browser view.

It requires that we added the following permission request to our AndroidManifest.xml file:

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

Here is an example Activity using a WebView object:

```
package com.mobvcasting.webviewexample;

import android.app.Activity;
import android.os.Bundle;
import android.webkit.WebView;

public class WebviewExample extends Activity
{
     WebView webview;

     @Override
    public void onCreate(Bundle savedInstanceState)
     {
        super.onCreate(savedInstanceState);

        webview = new WebView(this);
        setContentView(webview);

        webview.loadUrl("http://www.mobvcasting.com/mmtest/index.html");
    }
}
```

You'll notice that this behaves a bit differently loading the same HTML as the previous example. This one does not show the JavaScript Alert and may or may not show the "Line Generated By JavaScript". In this version the link that says "Click Me" probably still doesn't do anything.

This points to the fact that we have do some additional work in order to get full JavaScript functionality.

To enable JavaScript and create the ability to do JavaScript Alerts we need to use the following code:

```
        // Enable JavaScript overall
        WebSettings settings = webview.getSettings();
        settings.setJavaScriptEnabled(true);

        // Enable JavaScript Alerts
        webview.setWebChromeClient(new WebChromeClient() {
          public boolean onJSAlert(WebView view, String url, String message, JsResult result)
          {
               return true;
          }
        });
```

Here is our example with the extra code added:

```
package com.mobvcasting.webviewexample;

import android.app.Activity;
import android.os.Bundle;
import android.webkit.WebView;

public class WebviewExample extends Activity
{
     WebView webview;

     @Override
    public void onCreate(Bundle savedInstanceState)
     {
        super.onCreate(savedInstanceState);
        webview = new WebView(this);

        // Enable JavaScript overall
        WebSettings settings = webview.getSettings();
        settings.setJavaScriptEnabled(true);

        // Enable JavaScript Alerts
        webview.setWebChromeClient(new WebChromeClient() {
            public boolean onJSAlert(WebView view, String url, String message, JsResult result)
            {
                  return true;
            }
        });

        setContentView(webview);

        webview.loadUrl("http://www.mobvcasting.com/mmtest/index.html");
    }
}
```

Webview to Native View

One nice thing about the webview (beyond the fact that we can author in HTML/JavaScript/CSS) is that we can call other Activities and and so on.

Here is a very simple Activity called "WebviewOtherActivity".

```
package com.mobvcasting.webviewexample;

import android.app.Activity;
import android.os.Bundle;
import android.view.View;
import android.view.View.OnClickListener;
import android.widget.Button;

public class WebviewOtherActivity extends Activity {

     Button aButton;

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

        aButton = (Button) findViewById(R.id.Button01);
        aButton.setOnClickListener(new OnClickListener() {
               public void onClick(View v) {
                    WebviewOtherActivity.this.finish();
               }
          });
     }
}
```

Don't forget to add the following line the the AndroidManifest.xml so Android knows about our new Activity:

```
        <activity android:name=".WebviewOtherActivity"></activity>
```

Now in our WebviewExample we can create a custom JavaScript interface that allows us to run some native code from a JavaScript call. In the below example, the native code will be to load the WebviewOtherActivity with an Intent:

```
                    Intent nintent = new Intent(WebviewExample.this,WebviewOtherActivity.class);
                    startActivityForResult(nintent,0); // if we did startActivityForResult we could get the data back??
```

Here is the full code:

```
package com.mobvcasting.webviewexample;

import android.app.Activity;
import android.content.Intent;
import android.os.Bundle;
import android.os.Handler;
import android.util.Log;
import android.webkit.WebChromeClient;
import android.webkit.WebSettings;
import android.webkit.WebView;
import android.webkit.WebViewClient;
import android.webkit.JsResult;
import android.widget.Toast;

public class WebviewExample extends Activity
{
     WebView webview;

    private Handler handler = new Handler();

     @Override
    public void onCreate(Bundle savedInstanceState)
     {
        super.onCreate(savedInstanceState);

        webview = new WebView(this);
        setContentView(webview);

        // Enable JavaScript overall
        WebSettings settings = webview.getSettings();
        settings.setJavaScriptEnabled(true);

        // Enable JavaScript Alerts
        webview.setWebChromeClient(new WebChromeClient() {
          public boolean onJSAlert(WebView view, String url, String message, JsResult result)
          {
                return true;
          }
        });

        //webview.addJavascriptInterface(obj, interfaceName);
        webview.addJavascriptInterface(new JSInterface(), "jsinterface");

        webview.loadUrl("http://www.mobvcasting.com/mmtest/index.html");
      }

      public class JSInterface {

          //window.jsinterface.specialClick()
        /**
         * This is not called on the UI thread. Post a runnable to invoke
         * loadUrl on the UI thread.
         */
        public void specialClick() {
            handler.post(new Runnable() {
                public void run() {
                    //webview.loadUrl("javascript:alert('Special Click Was Called');");

                    // Could do other things here, like load a different activity
                    Intent nintent = new Intent(WebviewExample.this,WebviewOtherActivity.class);
                    startActivityForResult(nintent,0); // if we did startActivityForResult we could get the data back??
                }
            });
        }
      }

     @Override
     protected void onActivityResult(int requestCode, int resultCode, Intent intent)
     {
          super.onActivityResult(requestCode, resultCode, intent);

          //Bundle extras = intent.getExtras();

          switch(requestCode)
          {
                case 0:
                     if (resultCode == RESULT_OK)
                     {
                          Log.v("WEBVIEW","Got Results");
                     }
                     break;
          }
     }
}
```

You'll notice in the above code that we are adding this line to the webview setup:

```
        webview.addJavascriptInterface(new JSInterface(), "jsinterface");
```

Later in the code, you see a class called JSInterface defined which does the native work.

The second argument in the "addJavascriptInterface" is the name of the JavaScript object that we are making up (jsinterface) to reference the Java Object (JSInterface).

Now any time we call in JavaScript:

window.jsinterface

we will be referencing the JSInterface Java Class.

The JSInterface Java Class has a method called "specialClick" which is what is being called by the onClick method of the "Click Me" link from our HTML:

<a href="#" onClick="window.jsinterface.specialClick();">Click Me</a>

The specialClick method has the Intent to launch the WebviewOtherActivity Activity.

Loading HTML and images from package instead from internet

HTML/CSS/JavaScript and other assets that will be used in a Webview don't have to be on the Internet. They can just as easily be in a file saved as part of the application.

For example, if you save the following HTML as "local.html" and put it in your project's "assets" folder:

```
<html>
	<head>
		<title>Hello World</title>
		<script type="text/javascript">
			alert("JavaScript Alert");
			document.writeln("Line Generated By JavaScript");
		</script>
	</head>
	<body>
		<h1>Hello There</h1>
		<a href="#" onClick="window.jsinterface.specialClick();">Click Me</a>
	</body>
</html>
```

It can be referred to with this as a URL: `file:///android_asset/local.html`

So we only need to change the webview loadUrl method to:

```
	webview.loadUrl("file:///android_asset/local.html");
```

Additional resources can be added to the "assets" folder as well. For instance, I added a JPEG image called "happydog.jpg" to a folder called "images" within the "assets" folder.

To display that image as part of the web page, I can use it's relative path:

```
	<img src="images/happydog.jpg" width="320" />
```

or it's absolute path:

```
	<img src="file:///android_asset/images/happydog.jpg" width="320" />
```

Similarly other assets can be loaded in the same manner.