CMER
Centre for Mobile Education and Research

Developing Apps for BB 10 with BlackBerry HTML5 WebWorks

Lab 1: A GeoLocation Application using Google Maps and GPS

Introduction

The objective of this tutorial is to introduce you to mobile application development using BlackBerry 10 HTML5 WebWorks. This tutorial will serve as a high-level introduction to the process and procedures involved in mobile application development for the BlackBerry smartphone. The application featured in this hands-on tutorial will give you a flavor of the work involved in developing mobile applications.

A basic knowledge of JavaScript and HTML is highly recommended before attempting this lab.

Setting up the HTML5 Webworks SDK, Ripple, and the BB10 Simulator

Setting up the SDK, Ripple Emulator Beta, and the BB10 Simulator can be a little intimidating. The first step is to download the most recent version of the Java Development Kit, or JDK. It is available here:
http://www.oracle.com/technetwork/java/javase/downloads/jdk-7u4-downloads-1591156.html

Download and install the appropriate release for your operating system. Make sure to choose the appropriate release (32 bit vs 64 bit) for your machine.

Now we will move on to download the Ripple Emulator. This provides a quick, easy to use tool that will allow you to develop your HTML5 application and see the results on screen quickly. Ensuring that you are obtaining the Ripple emulator for BB10, download the emulator here:
https://developer.blackberry.com/html5/download/ripple

Next, we have to download the HTML5 Webworks SDK. The version that we want is the BlackBerry 10 version, NOT the smartphone or tablet version. You can find the SDK for download here:
https://developer.blackberry.com/html5/download/sdk

Included in the Webworks SDK is a VMWare virtual machine image. The simulator is used to test your application thoroughly on a complete virtual recreation of a BlackBerry device. In order to do this, we have download and install a virtualization application. The tool of choice is VMWare player, a free version of the famous VMWare Workstation. You can find the download for it here:
http://www.vmware.com/products/player

After downloading all this software, it is time to start installing it. Start with the JDK, followed by Ripple Emulator, WebWorks SDK, and VMWare Player, in this order.

After installing all the above software, we should test it in order to ensure that everything was installed correctly. Let's start with the Ripple Emulator.

1. Create a folder in your user directory called ***RippleSites***. This is where you will save and develop the sites that you wish to test using the Ripple emulator.
2. Navigate to the default install directory for the Ripple Emulator. Usually this is found in the C***:\Program Files\Research in Motion\Ripple <version number>\.***
3. Open Google Chrome and open a new tab. Drag the file ***ripple_ui.crx*** into the main Chrome window. Click on "Allow" on any dialogs asking for confirmation regarding the install. A new icon should appear by the end of your address bar.
4. Continue to the .\services\bin\ subdirectory. Run the ***ripple-services.bat*** script.
5. Open notepad, and paste the following text into it:
```
<html>
        <head><title>Test</title></head>
        <body><p>This is a test</p></body>
</html>
```
6. Save the file as test.html in the RippleSites folder we created earlier.
7. Open a new chrome window, and navigate to the site http://localhost:9110/test.html. You should see a page saying "Test is a test".
8. Next, go to the icon beside the address bar, which appeared after installing the Ripple plugin, and use it to enable the emulator. When prompted, choose BB10 as your target platform.
9. Hit refresh. You should now see a BlackBerry displaying the same test page we just visited.

Setting up the simulator is easy in comparison. Open VMWare player, and click on ***Open a Virtual Machine.*** Navigate to the directory where you installed the WebWorks SDK (usually C:\Program Files\Research In Motion\BlackBerry 10 WebWorks SDK <version number>\). Open the ***simulator*** subdirectory and select the ***BlackBerry10Simulator.vmx***. Follow the instructions on screen and then run the Virtual Machine. During Boot, you will be presented with two options: a normal BB10 boot, or a "safe" mode. If you do not have a hardware-accelerated graphics card with support for OpenGL, select the safe mode. Otherwise, just let the boot process continue. You will eventually reach a home screen of a BB10 device.

Deploying BB10 Applications

To deploy your applications to the BB10 Smartphone simulator, please read the documentation available here:
https://developer.blackberry.com/html5/documentation/ww_testing/Using_the_BB10_simulator_2008466_11.html

Notes on BB10 WebWorks Applications

In order to access the BB10 System APIs, a Webworks Application must include the webworks.js script. This should be included in every BB10 app that you create. In addition, your application must "subscribe" to the ***webworksready*** event. An example of the appropriate code can be found here, at step 3: https://developer.blackberry.com/html5/documentation/ww_getting_started/Creating_a_BB10_app_2007539_11.html

Application Description

The application that you will be developing during the course of this tutorial is designed to help you locate your parked car (the idea for this application was inspired after the author lost his car downtown). In order to properly start developing any application, a list of functional requirements should be defined first.
In this case, the application you will develop must be able to:

- Determine the current location of the user.
- Let the user save their current latitude and longitude for future use, in order to determine the location of their car when they have need to find it
- Draw a map showing a user their present position.
- Draw a map showing the location which was previously saved.
- Draw a map displaying a user's current location and the direction to their saved position.
- Display the user's current latitude and longitude numerically so that it can be used for other purposes.
- Let the user choose between the different types of maps that are available for use.

An example usage scenario could be as follows:

1. The user gets out of their car and saves their current location.
2. The user, when returning to their vehicle, gets lost.
3. They determine their current location and see it displayed on a map.
4. They then get a directional indicator as to which way they must proceed in order to find their car.

Here is a sample of what your application should look like by the end of this lab.

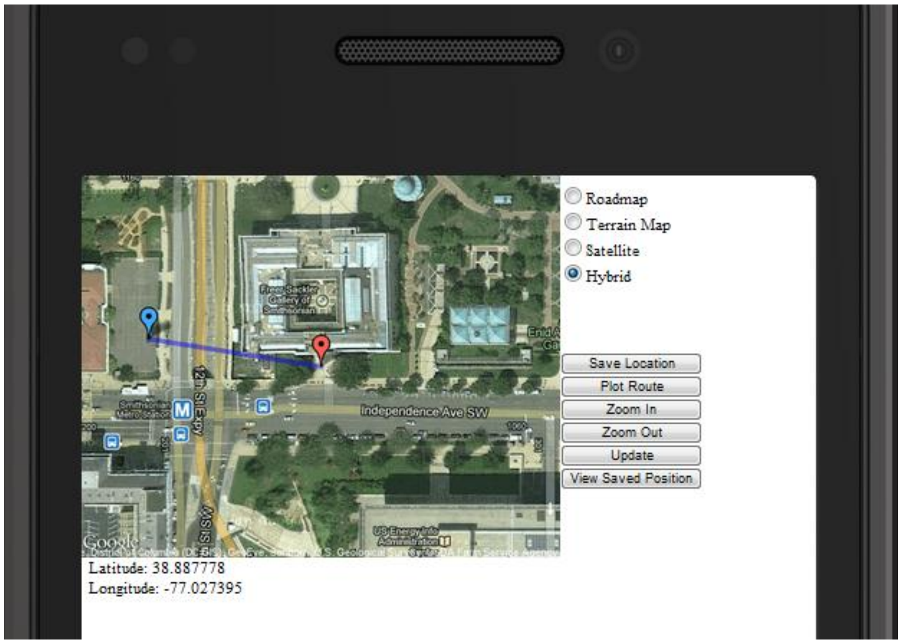


Figure 1: *A screenshot of the application you will have developed by the end of the lab.*

<u>The Interface</u>

The first step in developing our application is to design the user interface. This is the easiest step, and doing this first will allow us to visualize our application requirements which were set out previously.

First, open the ***index.html*** file, provided in the start-up package. Examples of how to use each of the tags for the HTML elements, which will be needed to develop your application have been included. Let's start with a high level overview of the structure of our application.

The first observation you should make is that all of the editing you will be required to do will take place in <div> named (i.e. whose id is) "content". No changes outside of this area are necessary. Inside this are four main sections:

- The area for the map to be displayed: ***<div id="imagediv">***
- An area for map-type selection controls: ***<div id="mapType">***
- An area for displaying the user's current location***: <div id="locationInfo">***
- An area for buttons which control the application: ***<div id="controls">***

Let's start by creating controls which will allow the user to select between the different map types. In our application, there will be four map-types available: roads, topographic (height), satellite, and a hybrid of the roads and satellite maps. The radio button for the road map has already been created. Create three radio buttons in the same <form>. They should have the values/labels pairs:

- Topographic:
    - Value: “terrain”
    - Label: “Topographic Map”
- Satellite:
    - Value: “satellite”
    - Label: “Satellite”
- Hybrid:
    - Value: “hybrid”
    - Label: “Hybrid”

All the other attributes should be the same as the example radio button element, with one exception: the example radio button is the only one which has the “checked” attribute specified. This means it will be the default selected radio button when the application is started.
Please take notice of the ***onclick*** attribute in each of the radio buttons; it is an event attribute, which will perform an action when the radio button is pressed. setMap() is a javascript function we will call when one of the radio buttons is pressed by the user. You can see the skeletal outline of this function in the ***skeleton.js*** file.
The next task is to implement the buttons which will control program functionality, in ***<div id=”controls”>***. The **<*button*>** element is a type of control that can be used outside of a form. Take a look at the example button in the skeleton file that is provided. Create five more buttons with the properties defined in the table below

Id	**Onclick**	**Regular text**
plotButton	getRoute()	Plot Route
upZoomButton	increaseZoom()	Zoom In
downZoomButton	decreaseZoom()	Zoom Out
updateButton	getMap()	Update
oldPosition	getOldLocation()	View Saved Position

Each button that you have created will execute the one of the requirements that we defined earlier. The names of the buttons are self-explanatory.
The last two areas present in the document are:

- The area for presenting the users location in numerical form: ***<div id=”locationInfo”>*** and its two child ***<div>*** elements for latitude and longitude.
- The map area: ***<div id=”imagediv”>***

Both of these areas will remain blank for now; they will be modified when the user runs our application.
Congratulations, you have finished designing the interface for your application! Take a moment to test your application in the ripple emulator. Your screen should look something like this:

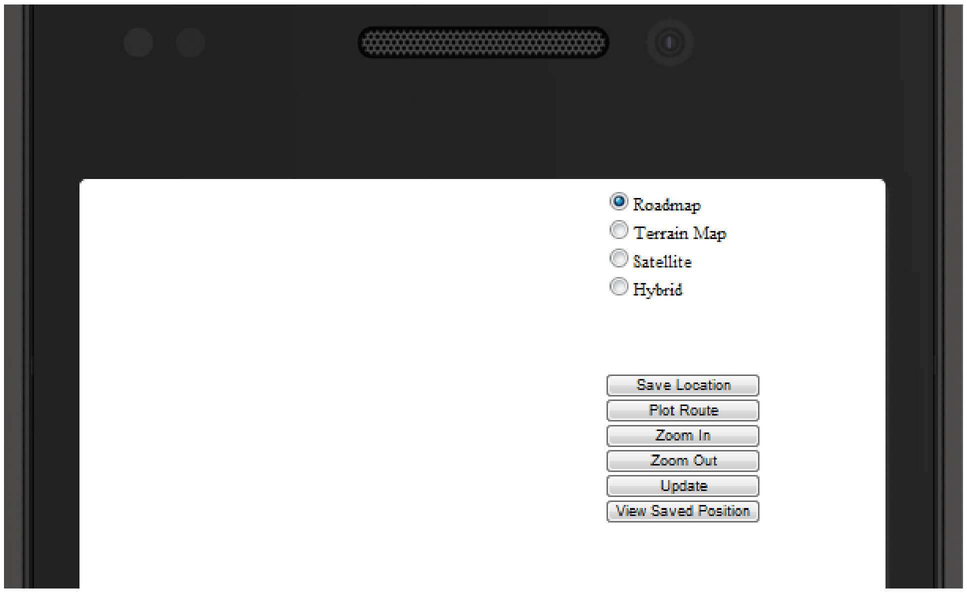


Figure 2: A screenshot of the current state of the application.

Next, we are going to start implementing the actual functionality of our program!

<u>JavaScript Functions</u>

Important: Open the skeleton.js file for the source code required to do this portion of the lab. Please open it while reading this document in order to do the required sections.

It's time to make your application do something! Let's start by examining the ***getMap()*** function. This is the function that we will use to display the map to our user. Before we can do this, we must be able to determine the user's real-world location. This can be accomplished by interacting with a special HTML5 JavaScript object: the ***navigator.geolocation***. This object allows any HTML5 application or web page to interact with a mobile device in order to determine their precise location, using GPS. This can be accomplished by using the ***getCurrentLocation***() method. It takes three parameters: a success function, a failure function, and an array consisting of any extra options. Passing an undefined array will make it use default settings.

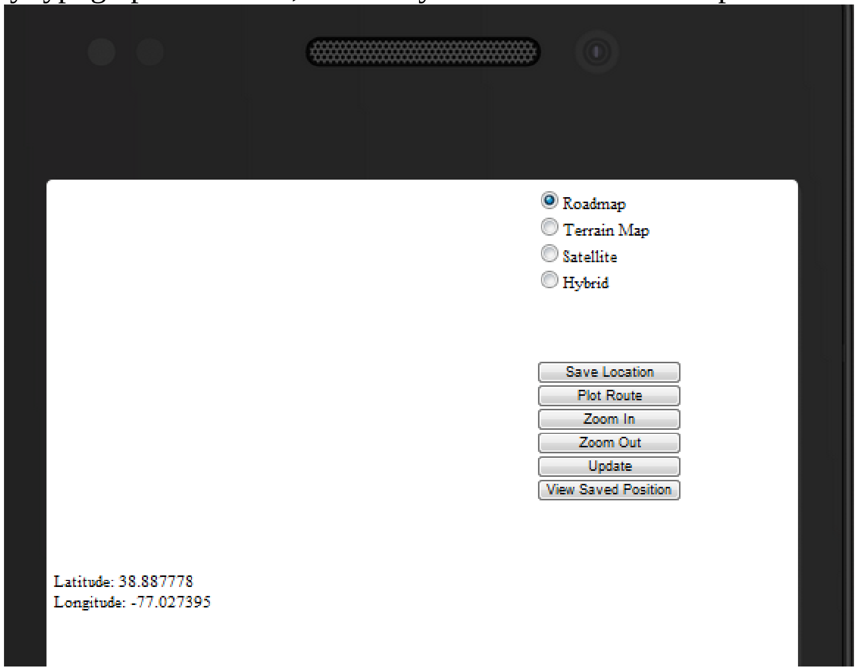
The ***getCurrentLocation()*** method will attempt to ascertain the user's position, and if successful, call the success function. In our applications, this is the ***geolocationSucess()*** method.

Our first task, once the success function is called, is to use the coordinates which were found and display them numerically on the screen. This is accomplished using the ***position.coordinates*** object. This object holds the latitude and longitude which were returned by the mobile device . To successfully display this information, you should now:

1. Create a variable called ***longitude*** and set it equal to the longitude returned from the device.
    a. The function already stores the latitude which was returned. Use this example to save the longitude in the same way.
2. The pre-written function ***showPosInfo(lat,lon)*** will display the coordinates on the screen to the user. Pass it the appropriate parameters; make sure that you don't accidently switch the lat-lon axes.

3. Lastly, you need to store the latitude and longitude for future reference. Set the global variables, ***currentLatitude*** and ***currentLongitude*** equal to the latitude/longitude values we have already used in the current function scope.

Now, open up the Ripple emulator, and set some GPS coordinates using the Location pane in the bottom right-hand side of the browser window. Go to the webpage localhost:9910/skeleton/***index.html***. Change the device location in the emulator, and try hitting the "Update" button. If the geographical positions that you set show up on the bottom-left of the device screen, then congratulations! Figure 3 shows you what your application should look like in the emulator. You have made the first functional component of your application. If it doesn't work, please go back and ensure that you did not make any typographical errors, and that you followed all the steps outlined above.



Latitude: 38.887778
Longitude: -77.027395

Figure 3: A screen shot of your application demonstrating its current functionality.

<u>Making the Map</u>

Important: Open the skeleton.js file for the source code required to do this lab. Please open it while reading this document in order to do the required sections.

So far, we can determine where the user presently is and display their geographic coordinates to them on the screen. Most people, however, cannot use decimal degrees to figure out where their car is! This is why we are going to create a map which will allow the user to see where they are.

Thankfully, we don't have to create a map from scratch. Google generously provides the static Maps API for this purpose. Given a set of coordinates, it will provide a map in the form of an image, which can be downloaded and displayed normally on any web page or web-based application.

Sending locational data to Google's map service is accomplished using *parameter passing* during an HTTP GET Request. An HTTP request is what happens when your computer requests a web page from a server on the internet, so that it can receive it and display the page on your screen. Parameters are passed by appending a '?' character onto the end of a URL, followed by the parameter name/value pairs, separated by ampersands. The format can be better understood by looking at an example: *http://www.example.com?parameter_name1=value1&parameter_name2=value2*.
Take a look at the global variable called ***mapLink*** at the top of the skeleton.js file. It contains a link to a Google map server which will generate an image we can use and insert into our web page. In order for us get a map and center it on a visible point we need to provide the following information, in the format specified as follows:

1. The latitude and longitude of a location where the map will be centered: e.g. **center=-40,80**
2. The zoom level we want, ranging from 1-14: e.g. **zoom=2**.
3. The size/dimensions of the image being returned, in length by width format: e.g. **size=400x200**
4. The map type: e.g. **maptype=roadmap**.

Now go to the function ***centerMapURL(lat,lon,zoom,length,width)***. This function will construct a URL which can be used to retrieve a map image from one of Google's servers. It has already been partially constructed. So far, it creates a string of parameters which will create a point that will be centered on the map. It uses the ***encodeURI(string)*** to make ensure certain characters are transmitted correctly as part of the http request (this isn't something that we will cover in this lab). Next, append an ampersand and then the rest of the required parameters in the proper format. Refer to the list above for examples of to format them. Notice how the parameters are appended to the variable ***mapLink*** at the end of this function.
Next, we have to create a function which can make a string which encodes information for a *marker*. A marker is a point on the map which can indicates a location on the map. We markers are created by specifying a color and location, and are sent as part of the HTTP request specified above.
The next task that our app has to accomplish is to specify a marker on the screen, by generating another parameter to pass to the Map API. This parameter is created by creating another formatted parameter string containing the marker color, along with its position on the map. Note that coordinates are given with the latitude specified first, and longitude specified second. An example would be: **marker=color:red|80,40.**
Go to the function ***createMarker(latitude,longitude,markerColor).*** Finish initializing the ***markerString*** variable using the parameters that are passed in. The color of the marker has already been added. See the skeleton.js file for an example. The construction of this string should take place within the ***encodeURI(string)*** function which already exists in the file.
After implementing these functions, go back to the ***geolocationSuccess*** function which we worked on earlier. Uncomment the last three lines of code, as indicated by the comments on line 32 of the skeleton.js file.

Storing Data

So far, we have an application which can display a user's location on a map, using an image which is generated by passing parameters to Google's Maps API. The next step in our application is adding the ability to save a location for future reference. This can be accomplished using the localstorage object, part of the HTML5 specification.
Go to the function ***saveLocation***(), and examine the code which is already provided. The first thing it does is error checking; it ensures that the current latitude and longitude have already been determined and notifies the user if either of the parameters are not present. The ***var data ={};*** creates an object to hold information. The application will store the current latitude and longitude in the variables ***data.lat*** and ***data.lon***. Examine the global variables at the very beginning of the skeleton.js file,and set ***data.lat*** and data.long to the currentLatitude and currentLongitude. Now create a variable called ***storageString***, and set it equal to the following: **JSON.stringify(data).** This command changes the data object that you created into a formatted string that is appropriate for long-term storage using the ***localStorage API***; Next, set ***localStorage.location*** equal to storageString. This will save the data in long term storage on the user's device.
In order to retrieve this information, we need to read the data and return it in a format which is usable by the rest of your application. Go to the ***loadLocationData()*** function and set the variable ***data*** equal to ***JSON.parse(localtionStorage.locationData)***. This loads and converts the data which we saved into a usable object, which is then returned to the application.
Lastly, we need to create a function which can retrieve this data and present it on the screen. This is accomplished using the ***getOldLocation()*** function, which loads the old position information using your ***loadLocationData()*** function. It then uses the ***centerMapURL(latitude,longitude)*** and ***createMarker(latitude, longitude, markerColor)*** ,which we worked on earlier, to create a new map based on the data we saved previously, and calls ***insertImage(url)*** to place the image in the application.

Showing Direction

The last piece of functionality that our program must implement is to create a path from the user's current location to the point that they have provided. Go to the ***getRoute()*** function in the skeleton.js file. The first part of the function, as indicated by its comments, creates a string which consists of a map centered on the user's location, along with two markers to indicate the user's current position and the point that was previously saved. The last parameter, that we must create is used to make a path between two points. We must provide two different coordinate positions, starting with the current latitude and longitude, and then the latitude and longitude of the saved data. An example of how we do this is: ***path=80,20|40,60.*** Using the skeleton code provided, finish setting the variable ***string***, by creating the two latitude and longitude for the current position and saved position, inside the ***encodeURI(string)*** at the end of the ***getRoute()*** function. Lastly, call ***insertImage(string)***, passing the variable ***string*** as a parameter. This will update your map with a route shown between the two points.

Testing the App

Now open the application in the ripple emulator. Set a location using the Ripple GUI, and hit the "Update" button. You should see a salmon-colored marker indicating your position on the map. Next, try using the zoom out and zoom in buttons. Hit the "Update" button after changing the zoom level in order to update the map on the screen. Now try saving your location and then retrieving it using the appropriate buttons. Refer to Figure 1 for an example of what your application should now look like.

Finalizing your application

In order to finalize your application, you must package it and sign it using the ripple emulator. This also requires valid keys. Once it is packaged and signed, it can be deployed for testing on a Blackberry device. Below are links for instructions on the setup process:

1. Obtaining signing keys: https://developer.blackberry.com/html5/documentation/ww_publishing/Signing_setup_smartphone_apps_1920010_11.html
2. Packaging your application with Ripple: https://developer.blackberry.com/html5/documentation/ww_developing/Packaging_your_app_in_Ripple_1904611_11.html
3. Signing your app in ripple: https://developer.blackberry.com/html5/documentation/ww_publishing/Signing_setup_smartphone_apps_1920010_11.html
4. Loading your application in a smartphone: https://developer.blackberry.com/html5/documentation/ww_testing/Loading_your_app_on_a_smartphone_1876977_11.html