Developing Apps for the BlackBerry PlayBook

Lab # 2: *Getting Started with JavaScript*

The objective of this lab is to review some of the concepts in JavaScript for creating WebWorks application for the BlackBerry PlayBook. In this lab, we'll be dealing with event handling and DOM in the application.

Before attempting this lab, please be sure to download and install the BlackBerry WebWorks SDK for Tablet OS available at https://bdsc.webapps.blackberry.com/html5/download/sdk. Follow the steps under 'How to get up and running' to ensure you have everything you need. After completing the steps, you should have installed the Adobe AIR SDK, the BlackBerry WebWorks SDK for Tablet OS and VMware Player. You do not need to worry about signing your application as we'll be testing it on the simulator. Instructions on how to setup the simulator in VMware Player can be found here:
http://docs.blackberry.com/en/developers/deliverables/27278/Configure_VM_BB_tablet_simulator_1594361_11.jsp. This lab also assumes that you have knowledge of basic HTML, CSS and JavaScript.

Exercise 1: Event handling in the HTML application

- Use your completed code from Lab 1. Open index.html in your favorite text editor.
- Open script.js in your favorite text editor as well.
- If you open the index.html file, and click on any of the buttons, you'll realize that nothing happens. That's because there is no JavaScript event tied to any of the buttons yet. In this lab, we'll take care of that.
- Let's start with the 'Visit CMER website' button. In the index.html page, find the following line:

```
<input type='button' value='Visit the CMER website'/>
```

- Next, add the attribute 'onclick'. This attribute allows us to call a JavaScript function when the user clicks the button. Assign the JavaScript function for that button to the function called 'openCMERWebsite()'. The line should now look like:

```
<input type='button' onclick='openCMERWebsite();' value='Visit
CMER'/>
```

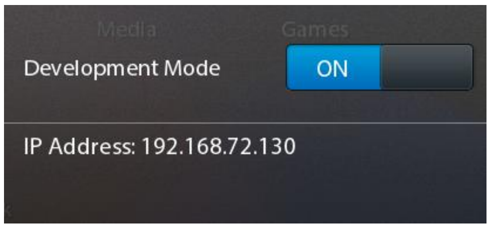
- Now, you'll create a function in the script.js file to open a new page that loads the CMER website when the user clicks the button. Remember, the name of the function has to be 'openCMERWebsite()'. It is best practice in JavaScript to place all global variables at the top of the file. Create a variable called 'cmerlink' which is a String that has the URL of the CMER website. The URL is 'http://cmer.socs.uoguelph.ca/'.
- To have the PlayBook open a URL in the browser, the following code is needed:

```
var args = new blackberry.invoke.BrowserArguments(cmerlink);
blackberry.invoke.invoke(blackberry.invoke.APP_BROWSER, args);
```

- Place this code in the 'openCMERWebsite()' function. Afterwards, in the index.html, add the following line in the <head> section:

```
<script type="text/javascript" src="scripts/script.js"></script>
```

- This allows the application to include the code from the JavaScript file you're working on now.
- Now, test the application. The code you've placed in this function will not work in a regular web browser; you'll need to test it in the PlayBook simulator.
- Open VMware Player. You should have already setup the PlayBook simulator in VMware Player using the instruction found at http://docs.blackberry.com/en/developers/deliverables/27278/Configure_VM_BB_tablet_simulator_1594361_11.jsp.
- If you have not setup the PlayBook simulator, follow the steps below:
    - On the VMware Player Home screen, click Open a Virtual Machine
    - On the Open a Virtual Machine screen, navigate to the folder where you installed the BlackBerry® WebWorks™ SDK. The default folder is C:\Program Files\Research In Motion\BlackBerry WebWorks SDK for TabletOS <version>
    - In the bbwp/blackberry-tablet-sdk/BlackBerryPlayBookSimulator-<version> subfolder, click the BlackBerryPlayBookSimulator.vmx file.
    - Click Open.
- Double click on the 'BlackBerry PlayBook Simulator option under 'Home'. Wait for the VM to initialize and bring you to the PlayBook home screen.
- While testing an application, you may notice that you cannot switch back to a Windows application. To escape the PlayBook simulator and begin interacting with Windows again, press Ctrl+Alt.
- Next enable Development Mode on the PlayBook by clicking on this icon: .
- Click on the slider, you should see that development mode is now on, and the device IP address is visible.



Development Mode ON

IP Address: 192.168.72.130

- Click on the slider, you should see that development mode is now on, and the device IP address is visible. Write this IP address down. You may also be asked to set a password once you activate development mode. Use the password 'playbook'. The newer versions of the PlayBook simulator do not require a password to be set once development mode is activated, so do not be alarmed if you are not prompted.
- Now compile and deploy the application on the PlayBook.
- You'll first need to zip the folder that all the lab files are in first. Assuming you've named the zip file PlayBookLab2.zip, use the following commands to compile and deploy the application:

```
cd C:\Program Files (x86)\Research In Motion\BlackBerry WebWorks SDK
for TabletOS 2.0.0.4\bbwp

bbwp [path to folder]\PlayBookLab2\build\PlayBookLab2.zip
```

- After this command, the application is compiled into a .bar file. This file is stored in the \bin folder where the zip file was stored.
- Now, load the application onto the PlayBook using the following command:

```
cd C:\Program Files (x86)\Research In Motion\BlackBerry WebWorks SDK
for TabletOS 2.0.0.4\bbwp\blackberry-tablet-sdk\bin

blackberry-deploy -installApp -password playbook -device
192.168.72.130 -package [path to
folder]\PlayBookLab2\build\bin\PlayBookLab2.bar
```

- Click the application on the PlayBook to launch it. Now try clicking on the button that the event was just created for. You'll see that the link opens in a new browser window. Congratulations!
- Now you'll need to do a similar process for the other buttons. Let's work on the 'About the application' button next. For this button, we'll need the help of jQuery. jQuery is a fast and concise JavaScript Library that simplifies HTML document traversing, event handling, animating, and Ajax interactions for rapid web development. It will allow us to create fantastic UI elements with a really small amount of code. First, include the 'jquery.tools.min.js' file in the index.html file.
- You'll need to run a jQuery command as soon as the page loads which assigns the jQuery overlay command to an input field. We'll be using the 'rel' attribute to specify which inputbox invokes which overlay. Add another script tag below where you've imported all your other JavaScript commands with the following code:

```
<script type="text/javascript">
 $(document).ready(function() {
       $("input[rel]").overlay();
       });
 </script>
```

- The `$(document).ready` command is called when the page is finished loading. The function inside this command is called an inner function. These types of functions are useful if we only have 1 or two JavaScript commands to invoke. Instead of making an entire function, we simply add the code we want inside the inner function.
- Within the head section of your index.html file, you should have all of the following script tags:

```
<script type="text/javascript" src="scripts/jquery.js"></script>
<script type="text/javascript"
src="scripts/jquery.tools.min.js"></script>
<script type="text/javascript" src="scripts/script.js"></script>
<script type="text/javascript">
 $(document).ready(function() {
       $("input[rel]").overlay();
       });
```

```
</script>
```

- Now, add the code for the overlay. Another DIV container will be required for this. You can place this new container under the 'loading' DIV container. Create a DIV with the id 'mies1' and the class attribute set to 'simple_overlay'. Inside the 'mies1' container, create another container with the class attribute set to 'details'. The two containers should look something close to:

```
<div class="simple_overlay" id="mies1">
      <div class="details">

      </div>
</div>
```

- Inside the details DIV container, type some text which describes the program you're building.
- Next, CSS will be needed to create the overlay as right now the text you typed will appear on the page without having to press the button. Add the following code in your style.css file:

```
/* the overlayed element */
.simple_overlay {

      /* must be initially hidden */
      display:none;

      /* place overlay on top of other elements */
      z-index:10000;

      /* styling */
      background-color:#333;

      width:675px;
      min-height:350px;
      border:1px solid #666;

      /* CSS3 styling for latest browsers */
      -moz-box-shadow:0 0 90px 5px #000;
      -webkit-box-shadow: 0 0 90px #000;
}

/* close button positioned on upper right corner */
.simple_overlay .close {
      background-image:url(../images/close.png);
      position:absolute;
      right:-15px;
      top:-15px;
      cursor:pointer;
      height:35px;
      width:35px;
}

/* styling for elements inside overlay */
.details {
      position:absolute;
      top:15px;
      left:15px;
      font-size:14px;
      color:#fff;
      width:650px;
}
```

- Next, to make the button work, add the ‘rel’ attribute to the ‘About the application’ with the value ‘#mies1’. The code for your button should look like the following:

```
<div id="aboutbutton">
        <input type='button' rel="#mies1" value='About the
        application'/>
</div>
```

- Afterwards, try loading the application in the PlayBook simulator again and click on the button. You’ll see we have a nice looking overlay appear. Feel free to resize and reposition the overlay as you like by editing the CSS file.
- Now it is time to add an event handler to the search button in the application. For now, we’ll simply handle the even when the user clicks the button and make something happen on the screen.
- Start by creating a method in the JavaScript called ‘getScholarResults’ which takes in an argument called ‘authorname’. Inside this method, check if the object authorname is equal to “” or null. If so, you should alert the user of this and exit the method with a ‘return’ statement. Your function definition should look like the following:

```
function getScholarResults(authorname)
{
        if(authorname=="" || authorname == null)
        {
                alert("Please enter a name");
                return;
        }
}
```

- Now, let’s create the listener which will call this function when the search button is clicked. To do that, add the ‘onclick’ attribute to the ‘button’ tag that is used for the search button. The value of this attribute should be the ‘getScholarResults’ function name with the value in the search box as the argument you are sending. Your button tag should look something similar to the following:

```
<button
onclick="getScholarResults(document.getElementById('searchbox').v
alue);">Search</button>
```

- If you load the program in the simulator or web browser now after saving and re-compiling, you should see an alert box appear if you click search without typing something.
- When the user searches, you should also show a loading icon to tell the user something is happening. To do that, you’ll need to change the style of the DIV container where we

have our loading image. Add the following code after the IF statement you created to check if the 'authorname' field is empty.

```
document.getElementById("loading").style.display="";
```

- Your 'getScholarResults' function should now look like:

```
function getScholarResults(authorname)
{
        if(authorname=="" || authorname == null)
        {
                alert("Please enter a name");
                return;
        }
        document.getElementById("loading").style.display="";
}
```

- The 'searchinfo' field which we'll use in the future should also be hidden. Therefore, you'll need to add the following JavaScript code as well which will hide the element on the HTML page.

```
document.getElementById("searchinfo").style.display="none";
```

- The text in the 'results' field should also be cleared with each search, so also include the following JavaScript code as well.

```
document.getElementById("results").innerHTML = "";
```

- Run the program again, but this time, have something in the text field. You should see the loading icon.
- In the next lab, we'll handle communication with Google in order to get the results that we need. Let's start using JavaScript to format the text the user inputs in a way that Google understands.
- Below what you've just added let's start manipulating the string sent with the search request to the function. This is the author's name that the user would like to get statistics about. The steps required are:
    - Split the 'authorname' variable into an array using a space (' ') as the delimiter. Call the array 'authorarray'.
    - For each element in 'authorarray', append the string 'author%3A' to the front of it.
- Create a global variable called 'resultquery' and assign it initially to an empty string. After the code to make the required changes to the array in the previous step, again assign 'resultquery' to an empty string. The reason we do this again is to make sure our old query has been erased before creating a new one.
- After resetting our 'resultquery' variable, we'll need to use the 'authorarray' array to create our query. For each element in the 'authorarray', append it to the 'resultquery' string with a '+' before the 'authorarry' element you are appending. If you are appending the first element in the 'authorarray', you don't need to append a '+' before the element. Your for loop should look something like the following:

```
for(var i=0; i<authorarray.length; i++)
```

```
{
        if(i==0)
        {
                resultquery = resultquery + authorarray[i];
        }
        else
        {
                resultquery = resultquery + "+" + authorarray[i];
        }
}
```

- The next step is creating the URL for communicating with Google Scholar. Before the URL is created, you'll first need to create a variable which tells Google how many results we want returned from our request. Make this a global variable at the top of the program and call the variable 'ret_results'. Assign this variable to 100. This variable will allow us to have at most 100 results per page. This variable is also going to used future parts of the program.
- Also create another global variable called 'openpageurl' and assign it to an empty string.
- Now we'll create our request URL. Below the for-loop you wrote to construct the result query string, create the following 'url' variable.

```
var url =
"http://scholar.google.ca/scholar?q="+resultquery+"&num="+ret_res
ults+"&hl=en&btnG=Search&as_sdt=1%2C5&as_sdtp=on";
```

- Next, assign the 'url' variable to the 'openpageurl' variable. The 'openpageurl' variable will be used in another function in the future to show all the results of the search.
- Finally, lets allow the user to simply hit enter in the application instead of clicking the search button. To do this, we'll need to add a listener to the button.
- In the HTML page, add an attribute to the input box. The attribute we need to add is 'onkeyup.' Set this variable equal to a JavaScript function called 'handleKeyPress' which accepts the arguments 'event' and 'this.form.' Your HTML input box should look like the following:

```
<input type="text" id="searchbox"
onkeypress="handleKeyPress(event, this.form)"/>
```

- Now create a JavaScript function called 'handleKeyPress(e, form)' where 'e' is the event, and 'form' is the form which triggered the call to the function. In this case, we aren't using a form so 'form' would be equal to 'null.'
- The that was pressed can be found by using the attribute 'keyCode' of the event variable 'e.'
- In the function, assign this attribute to a variable, then check whether or not the variable is equal to 13 (the code for enter). If this variable is equal to 13, we should call the 'getScholarResults' with 'document.getElementById('searchbox').value' as the parameter of the function. The DOM command 'document.getElementById('searchbox').value' will obtain the text that is currently in the input box with the ID of 'searchbox.' Your 'handleKeyPress' function should look something similar to the following:

```
function handleKeyPress(e,form)
{
      var key=e.keyCode;
      if (key==13)
      {

      getScholarResults(document.getElementById('searchbox').valu
e);
      }
}
```

- Now test out what you have just done in the PlayBook simulator or the web browser. You should notice that now you can simply press enter and the loading icon will appear.
- You now have an application that is able to handle events, and take input from the user. The next lab will focus on communication with Google in order to get the results. Keep all your work from this lab for the next lab.