Getting Started with HTML5 using BlackBerry WebWorks

Lab # 1: *Using New Options in the Latest Web Technologies*

The objective of this lab is to introduce some of the new concepts added in HTML5 by creating a WebWorks application for the BlackBerry Smartphone. The intent of this application is to simply show some basic features that are new in HTML5 by building a simple note taking and to-do list application for BlackBerry 7 devices. In labs 2 and 3, we will experiment with more advanced functions of recent web technologies by adding more features to this application. In this lab, we'll be changing some tags to newer more advanced versions, using local storage and checking network status.

Before attempting this lab, please be sure to download and install the BlackBerry WebWorks SDK for Smartphones available at https://bdsc.webapps.blackberry.com/html5/download/sdk. Make sure you that you have Java Software Development Kit installed and configured on your computer. Please also install the Ripple Emulator available at: https://bdsc.webapps.blackberry.com/html5/download/ripple. The Ripple emulator is a mobile environment emulator that is custom-tailored for mobile HTML5 application development and testing. You do not need to worry about signing your application as we'll be testing it on the emulator. Instructions on how to setup and start using the emulator can be found here: https://bdsc.webapps.blackberry.com/html5/documentation/ww_getting_started/getting_started_with_ripple_1866966_11.html. This lab also assumes that you have knowledge of basic HTML, JavaScript and CSS.

Working with HTML5 Tags

- Open the startup archive Lab1.zip extract it to a folder of your choice, then open index.html in your favorite HTML editor.

  You will notice that some code has already been provided to you. Open it in the Ripple emulator and see how it looks. There is HTML and CSS files provided that make this simple interface.

- In the index.html file, you will notice it begins with the code

```
<!DOCTYPE html PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN"
"http://www.w3.org/TR/html4/loose.dtd">
```

  This was necessary to identify the type of file and coding used in the past but HTML5 allows us to simplify it. On older browsers, the browser will still automatically identify correctly. Change that line simply to

```
<!DOCTYPE html>
```

- There is another simplification for tags when adding stylesheets and JavaScript files. No longer is type= "text/css" and type = "text/javascript" necessary to identify the files. You can go ahead and remove them from the two statements in the header.

We added a JavaScript document called script.js to the project which you can see in the startup files. This is a blank document currently and we will add more to it later. In the scripts folder, there is also a file called jquery.js, add a reference to this in your index.html before the importing of the script.js file. JQuery is a JavaScript library that provides simple functions to make app development much easier without having to re-write them each time.

- Next we will see on our page that there is already a simple bulleted list created for items to do. In your browser, you will notice that this is not editable. We would like to change this. In the <ul> tag, add the content editable line which will allow the user to make changes to the text. This text will be similar in look to how it was previously, but the user will be able to edit it. This will not be a textbox as in traditional HTML, but normal text.

```
<ul contenteditable=true>
```

Going back to Ripple, you will notice that the text can now be changed.

- For a useful app, this text should not go back to default every time we load it. In Ripple, if you refresh the page, all of your edits will disappear. To change this, we will make use of the new local storage feature. This is a new and much simpler method than using a database or cookies. This feature is not supported across all browsers but will work in our BlackBerry 7 application.

  The first step will be to give our unordered list an ID. Let's call it "items"

  Next we will open our script.js file and add a listener for when the user leaves the to-do box. In order to do this, we will use blur events and JQuery. Add the following function into your script.js file

```
$(function() {

	var items = document.getElementById('items');

	$(items).blur(function() {
		localStorage.setItem('itemData',this.innerHTML);

	});

});
```

Testing now would not show any difference, however the content will be saved. If you refresh the page, you will still see the defaults. To combat this, we need to check if there exists previous data when the page loads and if there is, insert that instead. Before the final closing tags in the above function, we will add a check to see if there is any previous data and insert it. This will be checked every time the app is loaded.

```
if( localStorage.getItem('itemData') ){
      items.innerHTML = localStorage.getItem('itemData');
}
```

Now refreshing your app in Ripple you should see that you have data that you previously entered into the list available. This is stored locally on the BlackBerry using the HTML5 local storage technique.

- In a later lab we'll be adding some functionality that will use the phones internet connection. For now, we'll only be adding a small notifier that tells the network connection status. This new addition to HTML5 allows web applications to check and confirm network status before performing actions. Usually this information does not need to be displayed, but we will show it as a demonstration.

  In the index file, decide where you would like the network status to be shown, it is recommended that it is below the unordered list. Add in the following code for us to modify using JavaScript and display the status.

```
<p><span id="netState"></span></p>
```

  Next we will go to the script.js file and add in some JavaScript to check the network status. Before the final closing tag, add the following functions that will allow us to add window events

```
var addEvent = (function () {
  if (document.addEventListener) {
    return function (el, type, fn) {
      if (el && el.nodeName || el === window) {
        el.addEventListener(type, fn, false);
      } else if (el && el.length) {
        for (var i = 0; i < el.length; i++) {
          addEvent(el[i], type, fn);
        }
      }
    };
  } else {
    return function (el, type, fn) {
      if (el && el.nodeName || el === window) {
        el.attachEvent('on' + type, function () { return fn.call(el,
window.event); });
      } else if (el && el.length) {
        for (var i = 0; i < el.length; i++) {
          addEvent(el[i], type, fn);
        }
      }
    };
  }

      })();
```

Now we will move to checking the status of the network by using this code directly below what we just added

```
var statusElem  = document.getElementById('netState');

function online(event) {
	statusElem.className = navigator.onLine ? 'online' : 'offline';
	statusElem.innerHTML = navigator.onLine ? 'Network status: Online' : '
	Network status: Offline';
}

addEvent(window, 'online', online);
addEvent(window, 'offline', online);

online({ type: 'ready' });
```

If we test our app in Ripple or a standard browser, we will see the network status shown as offline or online. Now go back through the function you added and modify the text so if the network is online, it displays in green font and red for offline. If you are unable to determine how to change the text color, take a look at this example: http://html5demos.com/offline, examine the code and see how you can implement it into your application.

- Next we will add a form at the top of our page to add structured elements to our to-do list. We will use new validation techniques in this form.

  Create a new form element with 3 standard text fields for Name, Email and Website. Since on mobile devices we have limited screen real-estate, we do not want to waste space with labels for these form elements. Instead we can use the new placeholder element inside the input tag to give some example text so the user knows what to enter. By adding the placeholder= "TEXT HERE" tag to our elements, we will find something similar to this:

  This text disappears when the user goes to enter information. This method allows us to save quite a bit of space without confusing users. Previous web technologies did allow this but it was difficult and cumbersome to do.

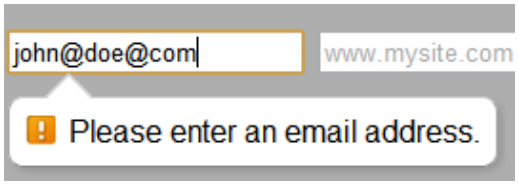
- Next let's change the form element for the email box from a general text type to an email type. Where we had

```
<input type="text" name="email" id="email" placeholder=" my@email.com ">
```

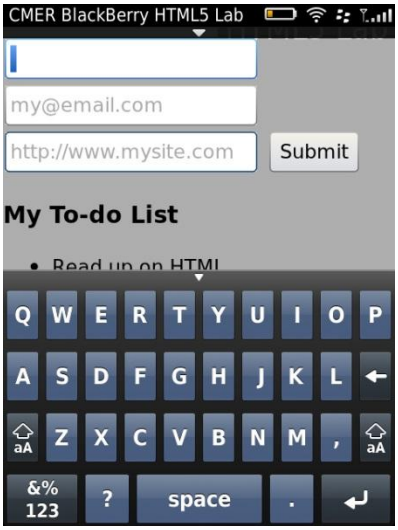
  Would now become

```
<input type="email" name="email" id="email" placeholder="my@email.com">
```

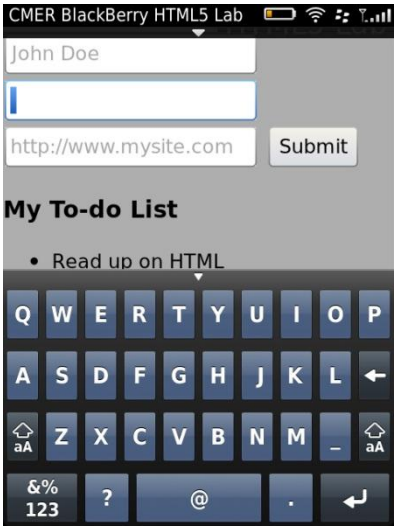
This small change makes the box a new email type field adding two important things. On many browsers, if you incorrectly enter an email using invalid syntax, an error message will automatically be shown. Note that you will not see this currently on the BlackBerry browser or in Ripple but there is a second benefit for this change we will see.



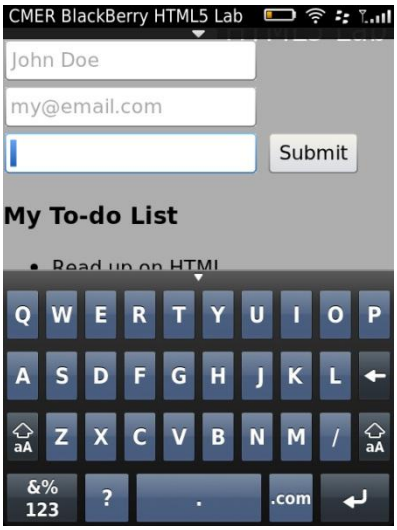
On mobile devices, this field will give the OS a hint of what information is required, changing the virtual keyboard to fit. On a BlackBerry we can see that the keyboard changes to fit the needs for email, URL and number type fields rather than showing just a general keyboard. While entering an email address on a physical keyboard, it will also correctly place the "@" symbol and period in the email address by using the space button. Try also now changing your website field to a URL type. The URL type also offers the same validation options.



Standard Keyboard



Keyboard for Email



Keyboard for URL

This is the end of Lab 1, in Lab 2 we will make use of the form fields we have created; add geolocation and many other enhancements.