Developing Apps with the BlackBerry HTML5 WebWorks SDK

HTML5 WebStorage

Objective

The objective of this lab is to examine the development of an application which demonstrates how to use the HTML5 WebStorage API and JSON for serializing objects, and how it can be optimized for use on mobile devices. You, the student, will go through the process of learning how to use the new HTML5 Web Storage interface, using the Blackberry WebWorks SDK.

Application Requirements

The example application that you will be reviewing during the course of this lab is designed to create a form of text areas which can be filled out by a user and then saved as a template for informational forms to be created later. This application should meet the following requirements:

1. It should allow the user to design a list, and save the template.
2. The list should be used as a template for forms which they wish to fill out later.
3. The user should be able to save the results of any lists that they have designed.
4. The template should be the base design for any events/relevant activities the user is recording information for.
5. The results which have been filled out should also be available in long-term storage.
6. It should be possible to clear out all saved data which have been stored by the application on the device.
7. In order to ensure that the user will not lose all their saved results or have to recreate their templates each time they start our application, we have to find a way to store our data between application executions. We can do this using an HTML5 API, the WebStorage API. This interface is defined by the W3C, and can be found at: http://www.w3.org/TR/webstorage/#the-localstorage-attribute

Figure 1 shows what the main screen of the application. Note that the application will run on both the BlackBerry PlayBook and Smartphone.

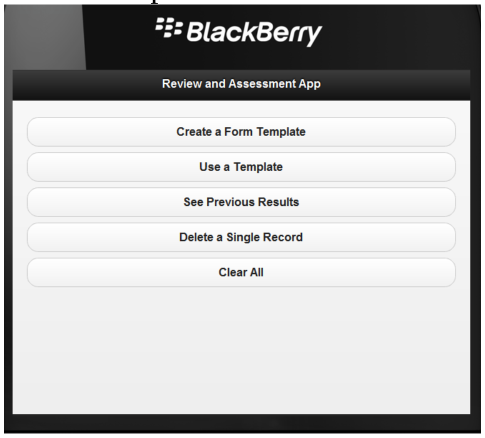

Figure 1: *The main screen of the application as seen on a smartphone*

In order to complete this lab, open the archive and extract the source code into a new folder. This application was designed and tested using the Ripple Emulator .

Using the localStorage object

The **localStorage** object works like a dictionary, using a key/value pair to store data. The **localStorage** object has two functions handling values: **localStorage.setItem(***key***,** *string***) for** storing data, **and localStorage.getItem(***key***)** for retrieving it. Examine ***line 18*** in the file ***builderPage.html***. Here we have a link which executes the **saveValue(***key*, *value***)** when clicked. This function can be found in the ***script.js*** file, on ***line 2***. Our program needs to save an object representing the form which is being created by the user(as seen in Figure 2). It converts serializes the object into a string, and then, using the title of the current template being designed, saves the object in the **localStorage** object.

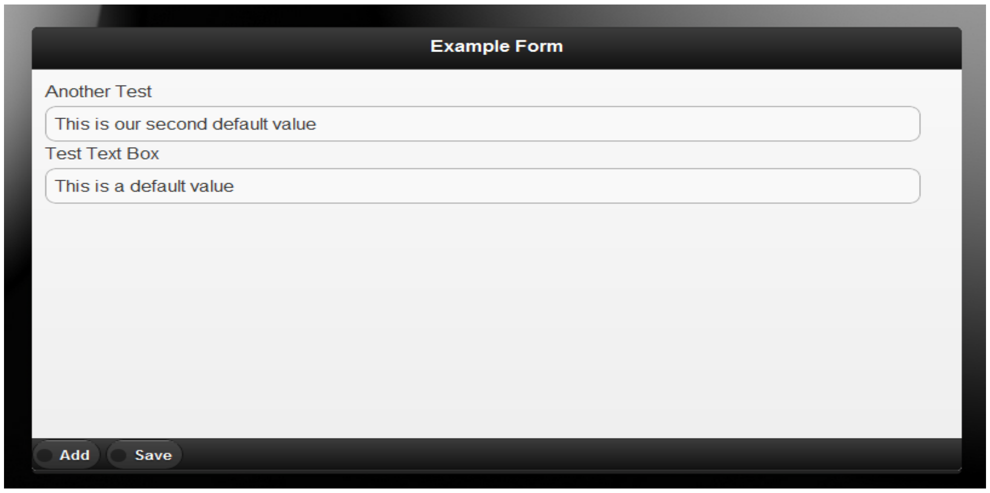

Figure 2: *A form which has been created but must be saved.*

localStorage has a corresponding function**, getValue(***key***)**, which will retrieve a string based on a given a key. We need this in order to retrieve and use the forms which have been created, as shown in Figure 3.

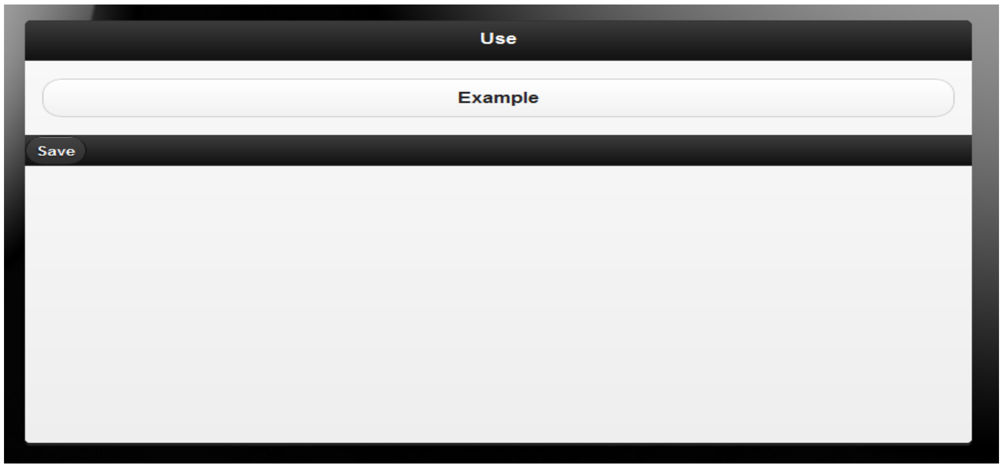

Figure 3: *The screen where the user chooses which template to use.*

Look again at the file script.js and you will find a wrapper on line 7 near the top of the page. There are also two other ways of accessing data in localStorage. Optionally, you can also treat **localStorage** like an associative array, (i.e. ***var bar = localStorage['foo'];***), or as an object, where your keys are now the properties the object, (i.e. ***var bar = localStorage.foo***).

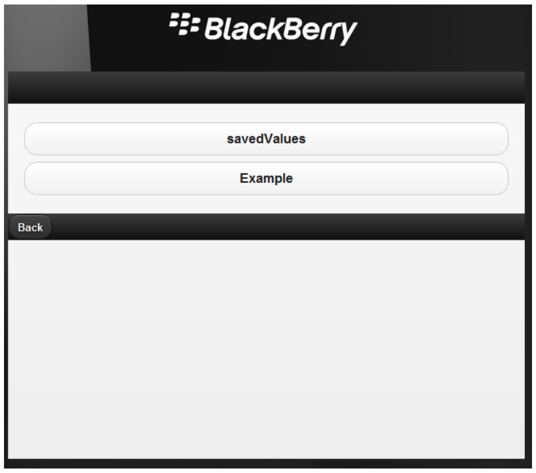

Figure 4: Screen for deleting templates

In addition to the two functions above**, localStorage** allows our program to remove forms from the data store (as seen in Figure 4), using the removeItem(key) function, permanently delete the form which was stored, the program uses a wrapper called **delForm(***string***).** :

```
function delForm(string)
{
        localStorage.removeItem(string); //Removes the string associated with this key
}
```

Note that writing a different value to a previously used key which has not been removed will simply overwrite and data previously associated with that key.

Lastly, one important factor every developer must remember is that of the limitations pertaining to storage space. The HTML5 specification for the **localStorage** object specifies a maximum of 5 megabytes for any given domain. In order to clear out the values, we can use the **localStorage.clear()** method. You can see from the example used on line 71 of the index.html file:

```
        <button onclick='localStorage.clear();'>Clear All</button>
```

JSON

One problem that arises using the localStorage interface is that it can only deal with string primitives. In order to store DOM Elements, we have to serialize them. In order to prevent the developer from having to select the pertinent pieces of an object to store, especially a complicated one such as an object referencing an HTML element, JavaScript provides an excellent mechanism to deal with this. A developer can use the JSON object to convert any

DOM element into a formatted string, using the **JSON.stringify(***object***)** method. The object can be recreated using the ***JSON.parse(**string**)*** method. This easy way of converting objects to strings will make your life much easier when you need to store data across application sessions.

An important point to remember is that any JavaScript objects which are converted to a JSON string will only have their properties converted. This means that any object which you restore with the **JSON.parse(***string*) **<u>will not</u>** have its methods restored. Thus, in order to restore an object and ensure we can call all the methods originally associated with the object's prototype, we have to create a new object and use a *for …. in* loop to copy the properties. Here is an example:

```
var bar = localStore.getItem('bar_1');  /* bar_1 was originally an Object of type FooBar as
well*/
var foo = new FooBar();
for (var x in bar) {
        foo[x] = bar[x];
}
foo.do(); //Calling bar.do() would have resulted in a runtime error indicating that the method
was undefined
```