HTML5 & Javascript: Lab 3

Objectives of this lab session:

- Create a user-friendly user-interface in HTML
- Update the html page using Javascript
- Create event handlers for the UI

Resources required:

- A current web browser (ideally Google Chrome)
- WebStorm IDE
- A completed Lab 2, containing the Appointment type, the appointments array and the existing showTable() and window.onload() function definitions.

Part 1: Creating an HTML5 User-Interface

Now we have an Appointment type that does much of what we need to, we need to devise some way for the end-user to add appointments and display them. As you might imagine, this will be mostly a matter of adding HTML elements to the web page to act as a user-interface, but when it comes to displaying a list of appointments, Javascript code will do much of the heavy lifting:
To create a HTML user-interface that can be used to make new appointments:

1. Add some HTML code to the <body> of Lab2.html:

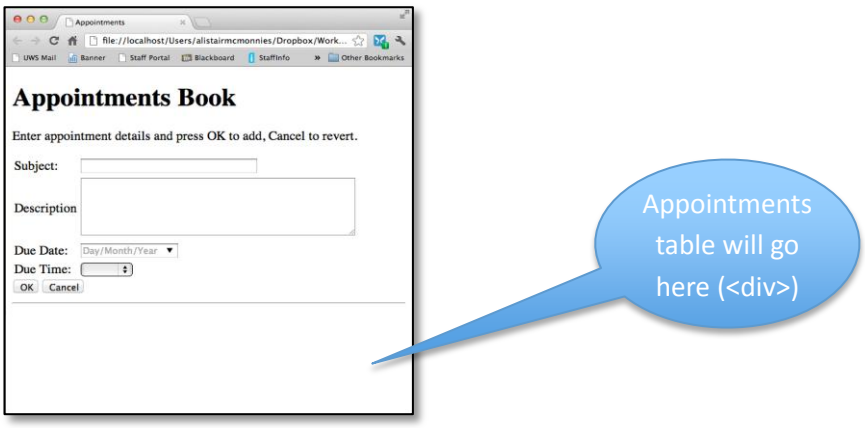
```
<header>
    <h1>Appointments Book</h1>
    <p>Enter appointment details and press OK to add, Cancel to revert.</p>
</header>
<table>
    <tr>
        <td>Subject:</td><td><input type="text" size="40" id="subject" /></td>
    </tr>
    <tr>
        <td>Description</td>
        <td><textarea rows="5" cols="50" maxlength="200" id="description" >
            </textarea></td>
    </tr>
    <tr>
        <td>Due Date:</td><td><input type="date" id="duedate" /></td>
    </tr>
    <tr>
        <td>Due Time:</td><td><select id="duetime" /></td>
    </tr>
</table>
<button id="ok">OK</button><button id="cancel">Cancel</button>
<hr/>
```

Note that there are four HTML input fields – a Text element, a Textarea element, in Input element whose type is set to “date and an empty <select> element. To display them neatly on the page, they have been entered into a table element with two columns – the first

column contains the captions describing the input controls, the second the actual controls. After the table a pair of Button elements have been added – for OK and Cancel. As yet, none of these have been wired up to any code.

2. We'll generate the actual list of appointments as an html table in Javascript, but we still need to provide a place for this table to go on the page. Add the following element immediately after the <hr/> (horizontal rule) element (it is deliberately empty):

```
<div id="table">
</div>
```



3. Save the file, reload the page and verify that it appears as shown above:

Coding the User-Interface

In the interests of keeping the code as clean as possible, a good strategy is to create the general functions that will be needed to manage the page. We'll need several functions:

1. Code to access all of the User-Interface elements on the form. Since all of these have id attributes, that should be easy.
2. A function to pre-fill the Due Time selection box with valid times of day = e.g. 12:00, 12:30 etc. For convenience, we'll use 30-minute intervals from midnight till 11:30pm.
3. A function to add a new appointment (when the OK button is pressed)
4. A function to clear the U-I elements (when the Cancel button is pressed)
5. A function to get the list of appointments and render them as an HTML table – note we've done this already in last week's lab
6. Some method of alerting the user to when an appointment becomes due

Implementing this code ought to be instructive:

1. Accessing the Input elements:

The standard Javascript pattern for assigning an HTML element to a variable (provided it has a unique id attribute) is:

```
var element = document.getElementById("id");
```

Our six UI elements have id's "subject", "description", "duedate", "duetime", "ok" and "cancel". Generally, I make sure I use lower case for these because it saves me having to check. We will want to store references to these elements in variables called subjectField, descriptionField, dateField, timeField, okButton and cancelButton.

1. At the top of the JS file, add a var statement that creates the list of variables given above

2. At the bottom of the JS file, replace the current code in the window.onload event handler (from last week) with a sequence of statements that assigns the appropriate html input control to each of the variables.

2. Pre-filling the time list:

Note we could have opted for the very tedious process of adding html <option> elements to the web page. Here, Javascript gives us a more convenient way:

```
var fillTimeList = function(timeList){          // timeList will be the <select> element
    var hours, minutes;
    for(hours=0; hours<24; hours+=1){           // Step through from midnight till 11pm
        var hh = hours.toString();              // Make an 'hour' string…
        if(hh.length < 2){                  // …and make sure it is 2 digits long
            hh = "0"+hh;
        }
        timeList.options[timeList.options.length] = new Option(hh+":00");// add the hour
        timeList.options[timeList.options.length] = new Option(hh+":30");// add half-past
    }
    selectNearestTime(timeList);    // When the page opens, select a suitable time
};

var selectNearestTime = function(timeField){
    // Now select the nearest time.
    var t = new Date(),
            n = t.getHours()*2 + Math.floor(t.getMinutes() / 30);
    timeField.options[n].selected = true;
};
```

Add this code to the Javascript file – I suggest placing it above the window.onload event, but it won't really matter where provided it does not get mingled with existing functions. Now add a call to the fillTimeList() function to the end of the window.onload event handler – pass the timeField variable as a parameter to it. Note. The selectNearestTime() method does the job of setting the selected time in the list to be near the current time – generally this will make it easier to use.

Now reload the page and check that the Due Time box now contains a list of times, and that the current time (to the nearest 30 minutes) is shown.

3. Adding a new Appointment

Adding a new appointment based on the values the user has entered into the various input elements is simple. Note that because of the style of controls we have used, there is no need for validation – the user will pick a date and time, and we have no basis for deciding that any text entered as input or description is valid or not (text is text). Code to create an appointment and add it to the list is refreshingly simple.

1. Add the following code:

```
var addAppointment = function(subjField, descField, dateField, timeField){
    var a = new Appointment(subjField.value, descField.value, dateField.value, timeField.value);
    appointments.push(a);                   // adds it to the end of the array
};
```

Note that the parameters we're passing into this function will be the values in the HTML input controls that you declared at the top of the file and set up in the window.onload method. Each of

these has a value property that refers to the control's visible content.

From here, we can add an event handler (i.e. a function) that will be executed when the user presses the OK button to create a new appointment. The two steps to this are:

2. Create the function

```
var pressOkButton = function(){
    addAppointment(subjectField, descriptionField, dateField, timeField);
    showTable();     // Display the whole list of appointments
};
```

3. Assign the handler to the button's onclick event. Add this statement to window.onload (at some point after the code that creates all of the references to the controls:

```
okButton.onclick  = pressOkButton;
```

4. Clearing the user-interface

If we make this into a single function call, it will be much easier to reset the state of the application when either details entered into the UI are to be cancelled or when a new appointment has just been added. One simple function can do this job:

```
var clearUI = function(){
    subjectField.value = "";
    descriptionField.value = "";
    dateField.valueAsDate = new Date();              // Today's date
    selectNearestTime(timeField);
};
```

This code removes any text from the subject and description boxes and sets the date and time boxes to show today's date and a time near the current time. There are two situations where we will want to do this.

1. When the user presses the cancel button – an exercise for you to do now
2. When a new appointment has just been made. Just insert a call to the function into the pressOkButton() function, after the appointment has been added of course.

5. Done – see the call to showTable() in 3

6. Alerting the user

Javascript contains functions to schedule a function call – either after a given time period (setTimeout()) or repeatedly as a set interval (setInterval()). setInterval() it the most useful one here, since we can arrange it to check the list of appointments for up-coming ones.

The format for setInterval() is:

```
intr = setInterval( function_to_call, milliSeconds_between_calls);
```

For example:

```
intr = setInterval( checkSchedule, 60000 );  // calls the function checkSchedule() every minute
```

The variable intr stores a reference to the schedule so that it can be cleared later with a call to clearInterval(). Normally we'd do this when the user navigated away form the page.

For our purposes, the checkSchedule() function would need to step through each appointment calling the isDue() method for each. If isDue() returned true, we could use an alert() call to tell the user (you might pass the appointment's subject or description in the alert() call.

Exercises

The app is more or less complete, at least until we revisit it later in the module to add persistence (so that appointments stay when we leave and return to the app or refresh the page) and ability for it to work offline. Both of these features come under the heading of new HTML5 features, dealt with later. Meanwhile, there are a number of enhancements you might consider to improve this app:

1. Implement point 6 above – you'll need to write the checkSchedule() function and pass it into a call to setInterval() (at the end of window.onload)
2. At the moment, if two appointments are made for the same time and date, the program will accept them. We can get around this by adding code to the addAppointment() function – this would have to first create the new appointment ,and then step through the list of existing appointments comparing the datetime value of the new one with the existing ones. If we found a match, an appropriate response would be to call the alert() function to let the user know and return from addAppointment() without adding the new one to the list. Try adding this feature.
3. As it is just now, the page is a boring black text on a white background. Although it does the job, most users would prefer some colour, possibly some nicer fonts and maybe even some graphic to brighten up the page. Use your imagination (or have a look at some other sites and site design web pages to get some ideas for this.

End of Lab 3