A Closer Look at Events

Overview

In this lab, you'll enhance the "product manager" Web application so that it uses subtle event-handling techniques to improve the user experience. For example, you'll handle `focus` and `blur` events on text boxes so that the current text box is easily identifiable via a different font colour. You'll also handle `mouseover` and `mouseout` events on HTML table elements, to highlight table cells as the user moves across them with the mouse.

We've already added some new features to the Web application, to demonstrate common HTML document manipulation. We'll explain the free new features in Exercise 1.

Roadmap

There are 5 exercises in this lab, of which the last exercise is "if time permits". Here is a brief summary of the tasks you will perform in each exercise; more detailed instructions follow later:

1. Understanding the free application enhancements
2. Handling `focus` and `blur` events in the capture phase
3. Dynamically adding and removing event handlers
4. Handling `mouseover` and `mouseout` events on table cells
5. Additional mouse-handling techniques (if time permits)

Server

HTML files can be opened from the file system, but for security reasons browsers will limit the possibilities. It is better to open a server in the project folders. Here are two ways to do that.

Visual Studio Code

1. Open Visual Studio Code
2. Choose menu "File | Open…" and open the root folder of the course material. After opening you should see a folder called ".vscode" at the top of the tree in the Explorer on the left.
3. Choose menu "Tasks | Run Task…" and choose task "npm: install".
4. Select/Open a file in the folder that you want to be the root of your server. This will generally be the homepage of your app.
5. Choose menu "Tasks | Run Build Task…" or choose the shortcut.
6. To close the server again, choose menu "Tasks | Terminate Task…".

Command line

1. Globally install “live-server” with command “npm install -g live-server” (see: https://www.npmjs.com/package/live-server)
2. Open the command line (Windows) or terminal (macOS) in the folder that you want to be the root of your server. This will generally be the homepage of your app.
3. Run “live-server” to open the server. Choose Ctrl+c to close the server again.

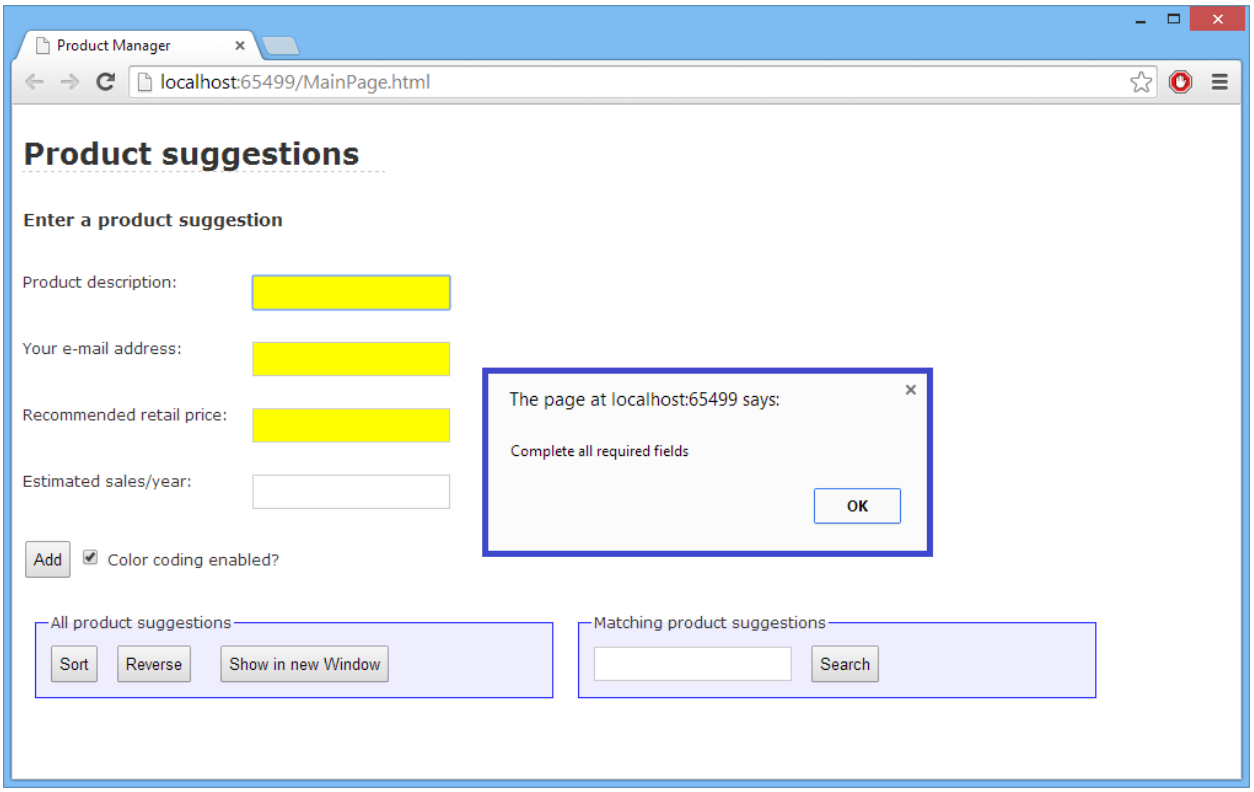
Familiarization

Start Visual Studio and open the *Solution* project for this lab, to get an idea of what you're aiming for in this lab (and to review some additional functionality we've added). Here are the details of the project:

- *Folder:*
  `\start`

The project contains two HTML pages – `MainPage.html` as before, plus a new page named `ProductSuggestionsPage.html` that will open in a new window to show a snapshot of product suggestions so far.

Run the application. The home page appears as usual. Notice that the first text box automatically receives input focus. Now see what happens if you click *Add* before entering any data. The Web page has validation rules built-in, forcing the user to enter values for the required fields (we've made the first 3 text boxes "required"):



Also notice the following new features in the home page, which you'll implement in this lab:

- Whenever you tab into a text box anywhere on the page, the font colour changes to orange. When you tab out again, the font color reverts to black. This enhancement makes it easier for the user to identify which text box currently has the input focus.
- The main form has a check box that allows the user to enable or disable the colour coding feature dynamically. If you uncheck this check box, text boxes no longer change orange when they get input focus. If you re-check the check box, you're back to square one and colour coding is re-enabled.
- Add some product suggestions, and then click the *Show in new Window* button (this is new in this lab). A new browser window (or tab) opens, showing the product suggestions as a list and as a table. Move your mouse over the table; the current row is highlighted in light grey, and the current cell is highlighted in dark grey:

Product description	User's email address	Recommended retail price	Estimated sales/year
Dummy description 1	Dummy email address 1	Dummy retail price 1	Dummy estimated sales/year 1
Dummy description 2	Dummy email address 2	Dummy retail price 2	Dummy estimated sales/year 2
Dummy description 3	Dummy email address 3	Dummy retail price 3	Dummy estimated sales/year 3
Swans jersey	andy@example.com	49.99	22
Cardiff jersey	dai@example.com	22.00	5

When you're happy with how all this works, close the browser window, return to Visual Studio, and close the *Solution* project.

Exercise 1: Understanding the free application enhancements

In Visual Studio, open the *Student* project for this lab as follows:

- *Folder:*
  `\start`

Take a look in `MainPage.html`. We've enhanced this page so that when the page is loaded, it assigns input focus to the first input control in the main form. Here's a quick summary of how we did this:

- To ensure input focus is set as soon as the page is loaded, we set the `onload` property on the `<body>` tag.
- In the "onload" JavaScript handler function, we used the HTML5 selector API to get the first input control in the main form:

```
var firstInputField = document.querySelector("#mainForm input");
```

- To set focus to this field, we called the `focus()` method on the field.

We also improved the integrity of `MainPage.html` so that it forces the user to enter a value in all the "required" text boxes. Any empty boxes will be highlighted in a different colour. Here's what we did:

- We needed a mechanism to indicate which text boxes are required and which can be left blank. One way to achieve this is to set the `class` property on the required fields. For example:

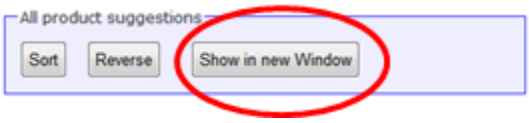
```
<input id="description" … class="required" />
```

- We enhanced the `doAdd()` function so that it validates all required text boxes in the main form, by using the HTML5 selector API as follows:

```
var requiredFields =
     document.querySelectorAll("#mainForm input[type='text'].required");
```

- We looped through all the required fields and checked the `value` property for each one. If the `value` is empty, we set the field's background colour to yellow, otherwise reset its background colour to white.

The next addition in `MainPage.html` was a *Show in new Window* button as follows:



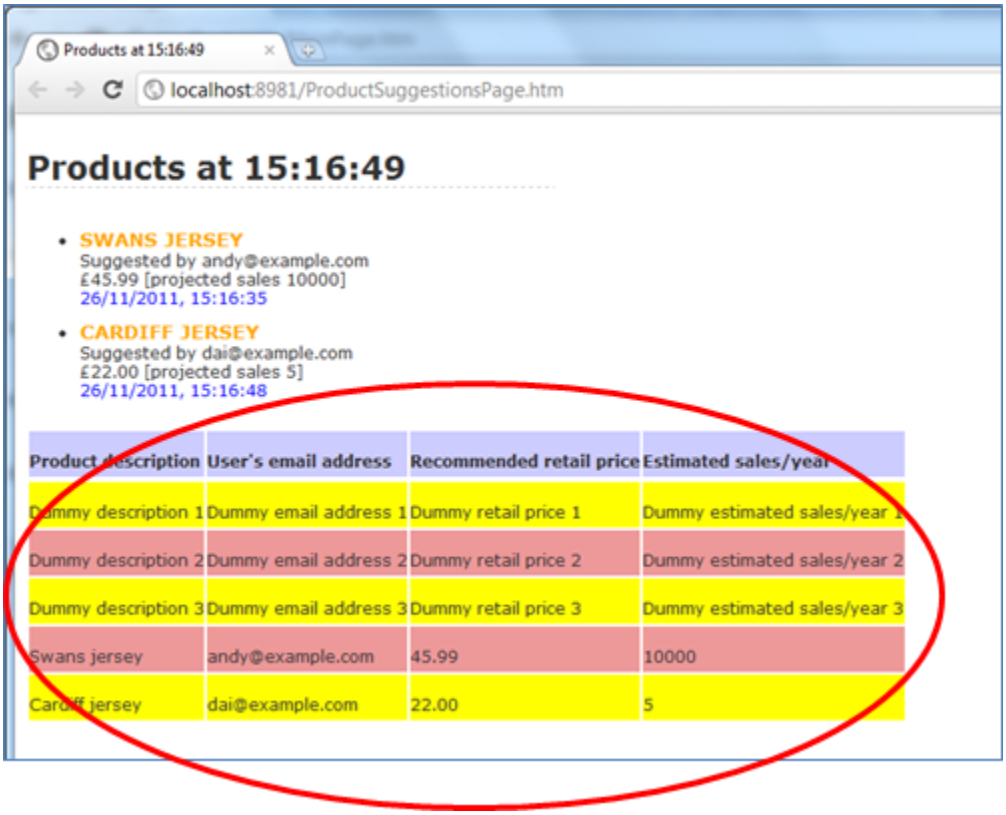
When the user clicks this button, we open the new `ProductSuggestionsPage.html` page in a new window. In `ProductSuggestionsPage.html`, we handle the `onload` event so that it displays product data as soon as the window opens. Here's how it works:

- The product data is held in the `allProducts` global variable in the main window. Global variables are actually properties on the `window` object. Furthermore, a window can access the window that launched it via the `window.opener` property. Therefore, the `ProductSuggestionsPage.html` page can access the product data from the main page as follows:

```
window.opener.allProducts
```

- `ProductSuggestionsPage.html` includes the `ProductSuggestionsFunctions.js` script file. Therefore, it can use `displayProducts()` to display all the products on this page (we specify "`currentProductsList`" as the target element name for the output).
- We display the current time in the window's caption bar (we access the caption bar via the `document.title` property). We also display the current time in the "`heading`" control on the page as well.

We also added some code in `ProductSuggestionsPage.html` to populate an HTML table dynamically, to show the current product data in tabular format:



Here's how we did this:

- `ProductSuggestionsPage.html` already contains a `<table>` with some dummy table rows and table columns, to show the basic syntax for HTML tables.
- The body of the table is designated by a `<tbody>` element with an `id` of "`currentProductsTableBody`".
- We wrote code in the `onload` handler function, to create a series of new `<tr>` elements (one per product) and append each `<tr>` to the `<tbody>`. For each `<tr>` element, we created 4 new `<td>` elements (one per property in a `Product` object) and appended each `<td>` to the `<tr>`. We used the following DOM APIs to achieve all this:

```
// To create a new element:
var newElem = document.createElement("newElemTagName");

// To set the value of an element:
newElem.innerHTML = elemValue;

// To append an element to a parent element:
parentElem.appendChild(newElem)
```

Exercise 2: Handling focus and blur events in the capture phase

Now it's time for you to add event-handling code for this lab.

In this exercise, you'll enhance `MainPage.html` so that it intercepts `focus` and `blur` events on all text boxes anywhere on the page. When a text box gains focus, you want its text to change to orange. When a text box loses focus, you want its text to revert to black.

The most obvious way to do this would be to find all the text boxes on the page and handle the `focus` and `blur` events on each text box individually. This approach would certainly work, but there is a smarter way…

JavaScript events have a "capture phase", which allows high-level elements (such as `<body>`) to intercept events triggered by lower-level elements (such as `<input>`). For example, you can define a single `focus` event-handler function on the `<body>` element, and it will see all the `focus` events triggered by descendent elements (ditto for `blur` events).

To implement this behavior, follow these steps:

- In `MainPage.html`, note that the `<body>` tag already has an `onload` event-handler function named `onLoad()`. You can find this function in `ProductSuggestionsFunctions.js`.
- In the `onLoad()` function, add code to handle the `focus` event on the `<body>` element. To do this, locate the `<body>` element and call `addEventListener()` to handle the "`focus`" event (remember to set the 3rd parameter to `true`, to ensure the event-handler intercepts events triggered by descendent elements).
- Implement the event-handler function for the `focus` event. The function should first test whether the actual target of the event is a text box (you're not interested in `focus` events from any other controls, such as buttons or check boxes). You can use the following code to see if the event's target is a text box:

```
if (e.target["type"] == "text") …
```

- If the actual target element really is a text box, set its colour to orange. You can use the following code to do this:

```
e.target.style.color = "orange";
```

- Repeat the previous 2 steps to handle the blur event, to set the text box color to black in this situation.

Run the Web application. Verify that text boxes display orange text when they have input focus, and black text otherwise.

Exercise 3: Dynamically adding and removing event handlers

In this exercise you'll implement the "Color coding enabled?" checkbox, to allow the user to enable or disable the colour-coding behavior dynamically:

- If the checkbox is checked, text boxes should change orange when they gain focus. This is the behviour you implemented just now in Exercise 2 ☺️.
- If the checkbox is unchecked, text boxes should not change colour when they gain focus.

In `MainPage.html`, locate the checkbox control and define a handler for its `change` event. For consistency, implement the event-handler function in `ProductSuggestionsFunctions.js`.

In the event-handler function for the `change` event, determine whether the checkbox is checked or unchecked:

- If the checkbox is checked, you should add event handlers for the `focus` and `blur` events on the `<body>` element. This is exactly the code you wrote in Exercise 1, so you might want to refactor your code to reuse the same logic here.
- If the checkbox is unchecked, you should remove event handlers for the `focus` and `blur` events on the `<body>` element.

Run the Web application again. Initially, the Web page should colour-code text boxes when they gain focus. Now uncheck the checkbox and verify that colour-coding of text boxes no longer takes place. Then re-check the checkbox and verify that colour-coding of text boxes is re-enabled again.

Exercise 4: Handling mouseover and mouseout events on table cells

In this exercise, you'll enhance the `ProductSuggestionsPage.html` page so that it highlights `<td>` table cells when the user moves over them with the mouse.

Follow these steps:

- Open `ProductSuggestionsPage.html` and locate the `onLoad()` function. This function initializes the contents of the page, including creating an HTML table dynamically to display product data in tabular format. Review the code in this function, and make sure you're happy with it before proceeding.
- Add code to find all `<td>` elements in the table body, and handle the `mouseover` and `mouseout` events on all the `<td>` elements.
- Implement the `mouseover` event-handler function so that it sets the target element's style to be white text on a light-grey background. The following code will do the trick:

```
e.target.style.cssText = "color:white; background-color:#888888";
```

- Implement the `mouseout` event-handler function so that it removes the styling.

Run the Web application. In the main page, add some project suggestions and then click the *Show in new Window* button. When the `ProductSuggestionsPage.html` page appears, move your mouse over some table cells and verify they change colour to grey. When you move away from a cell, it should revert to its original colour.

Exercise 5 (If time permits): Additional mouse-handling techniques

Enhance `ProductSuggestionsPage.html` so that it displays table rows in light grey when you move your mouse over any cell in that row. Think about how you're going to achieve this effect…

Product description	User's email address	Recommended retail price	Estimated sales/year
Dummy description 1	Dummy email address 1	Dummy retail price 1	Dummy estimated sales/year 1
Dummy description 2	Dummy email address 2	Dummy retail price 2	Dummy estimated sales/year 2
Dummy description 3	Dummy email address 3	Dummy retail price 3	Dummy estimated sales/year 3
Swans jersey	andy@example.com	49.99	22
Cardiff jersey	dai@example.com	22.00	5