Events

Chapter 11

- Static webpages are useful for presenting information.
- If you wish to interact with the user or provide a more dynamic page, you'll need to capture and handle events.
- Mouse events, keyboard events, form events and timers are events you may want your page to respond to.
- Prototype is being used in this chapter so you must include that file.

- In order to capture an event, we need to assign a function to that element.

```
$("myTextBox").onmousemove = someFunction;
```

- As usual, browsers are not consistent with event handling. Luckily, Prototype takes care of many of these inconsistencies.
- To attach an event handler using Prototype, we would use the following code:

```
$("myTextBox").observe("mousemove", someFunction);
```

- When an event handler is called, the function is passed an argument representing the event that occurred.

```
function myFunc(event){
    //handle event here
}
```

- **type** - event property that explains what the event is, e.g. click or mouseup. Events can be found at http://www.w3schools.com/jsref/dom_obj_event.asp

- When a user clicks the mouse button, several events occur. First, a *mousedown* event happens. When the user lifts up the mouse button, a *mouseup* event happens and then a *click* event happens.
- It is possible for a mousedown and mouseup event to occur without a click event.
- *mouseover* and *mouseout* events are useful for hovering effects.

- The event passed to the function handling the event contains a lot of information. This is particularly important for mouse and key events.
- Many of the mouse properties are inconsistent across browsers. The following may or may not work depending on the browser: clientX, clientY, screenX, screenY, offsetX, offsetY, pageX, pageY, button, layerX, layerY, srcElement and target.
- Prototype events are found at http://api.prototypejs.org/dom/Event/ and you are encouraged to use them.

```
window.onload = function() {
    $("someElement").observe("click", someFunction);
};

function someFunction(event){
    if(event.pointerX() < 100 && event.pointerY() < 100){
        alert("You clicked on " + event.element());
    }
}
```

- There are 3 main keyboard events to handle: keydown, keyup, keypress.
- The difference between keydown and keypress is similar to the difference between mousedown and click.
- If a textbox or text area is able to be typed into, it has **focus**. When an element loses focus, there is a function called **blur**.

- Keyboard events have several important properties: keyCode, altKey, ctrlKey and shiftKey.
- **keyCode** returns the integer ASCII value of the key pressed. *String.fromCharCode* can convert it from an integer to a String.
- http://api.prototypejs.org/dom/Event/ contains several ways to check for other keystrokes like the down arrow, esc key, tab, etc.

- Let’s say we are trying to ensure the user types in numbers only for a telephone number textbox. Our HTML in some form:
  ...<input id=“phone” size=“10” maxlength=“10”/>...
- We then add the event to our element:

```
window.onload = function() {
    $("phone").observe("keydown", phonePress);
};
```

- Some browsers return different key codes for numbers entered via the numeric keypad. Therefore, this simple function may not always work.

```
function phonePress(event){
    var zero = "0".charCodeAt(0); //or var zero = 48;
    var nine = "9".charCodeAt(0); //or var nine = 57;
    if(event.keyCode < zero || event.keyCode > nine){
        event.stop();
    }
}
```

- Form events are useful to validate information **before** it is sent to the server.
- Let's modify our telephone verifier to ensure that the user typed in 10 numbers.
  <form id="phoneForm" . . . />
  . . . <input id="phone" size="10" maxlength="10" />. . .
- We modify our onload function:

```
window.onload = function() {
    $("phone").observe("keydown", phonePress);
    $("phoneForm").observe("submit", formSubmit);
};
```

- We add the following function to our code:

```
function formSubmit(event){
    if($("phone").value.length == 7){
        //forget area code, maybe?
        alert("Please enter full telephone number");
        event.stop();
    }
    else if($("phone").value.length != 10){
        //invalid phone number
        alert("Please enter valid telephone number");
        event.stop();
    }
}
```

- The **change** event is often used when the user selects a radio button, checkbox, etc.

Example

Create a page that has 2 dropdown boxes. One dropbox box has 2 states: Wisconsin and Iowa. The other dropbox box has valid area codes for those states. Only provide valid area code options depending on the state chosen. The area code box should be disabled if a state has not been chosen yet.

- Several events are related to the overall web page.
- We've already seen and used **onload**.
- Others include: abort, contextmenu and unload.
- Some unloads are useful if there is unsaved work. Most are obnoxious.

```
window.onunload = pleaseDontGo;

function pleaseDontGo(event){
    alert("You can't leave.");
    event.stop();
}
```

- window.onload can be a bit slow especially if one is showing/hiding elements in that function.
- A second option is to use the *dom:loaded* event instead.

```
window.onload = someFunction; //old way

//better way
document.observe("dom:loaded", function() {
    //someFunction implemented here instead
});
```

- **setTimeout(function, delay)** - calls function once after delay milliseconds and returns a timer ID.
- **setInterval(function, delay)** - calls function repeatedly every *delay* milliseconds and returns a timer ID.
- **clearTimeout(timerID)** - stops timer associated with the timerID.
- **clearInterval(timerID)** - stops timer associated with the timerID.

```
window.onload = someFunction; //old way

//better way
document.observe("dom:loaded", function() {
    //someFunction implemented here instead
});
```

Example

Write a page that asks the user how many multiplication problems he or she wishes to solve. Once the user enters that information, that part of the form disappears and the actual quiz appears. The user is given 3 seconds to solve a problem. Some type of countdown should be shown and the score should also be visible the entire time. After the last problem is solved, a finishing comment should be made about the time took and number of problems solved.