COMP284 Scripting Languages

Lecture 17: JavaScript (Part 4)
Handouts (8 on 1)

Ullrich Hustadt

Department of Computer Science
School of Electrical Engineering, Electronics, and Computer Science
University of Liverpool

Contents

Window and Document objects

JavaScript provides two objects that are essential to the creation of dynamic web pages and interactive web applications:

window object

- a JavaScript object that represents a browser window or tab
- automatically created whith every instance of a <body> or <frameset> tag
- allows properties of a window to be accessed and manipulated
  ⇝ JavaScript provides methods that allow window objects to be created and manipulated
  Example: window.open('http://www.csc.liv.ac.uk','Home')
- whenever an object method or property is referenced in a script without an object name and dot prefix it is assumed by JavaScript to be a member of the window object
  Example: We can write `alert()` instead of `window.alert()`

Window object

- A window object represents an open window in a browser.
- If a document contain frames, then there is
  - one window object, `window`, for the HTML document
  - and one additional window object for each frame, accessible via an array `window.frames`
- A window object has properties including

`document`	document object for the window
`history`	history object for the window
`location`	location object (current URL) for the window
`navigator`	navigator (web browser) object for the window
`opener`	reference to the window that created the window
`innerHeight`	inner height of a window's content area
`innerWidth`	inner width of a window's content area
`closed`	boolean value indicating whether the window is (still) open

Navigator object

Properties of a navigator object include

`navigator.appName`	the web brower's name
`navigator.appVersion`	the web brower's version

Example: Load different style sheets depending on browser

```
<html><head><title>Navigator example</title>
<script type="text/javascript">
if (navigator.appName == 'Netscape') {
  document.writeln('<link rel=stylesheet type="text/css" '+
                   href="Netscape.css">')
} else if (navigator.appName == 'Opera') {
  document.writeln('<link rel=stylesheet type="text/css" '+
                    href="Opera.css">')
} else {
  document.writeln('<link rel=stylesheet type="text/css" '+
                    href="Others.css">')
}
</script></head>
```


Window object

Methods provided by a window object include

- `open(`*url*`, `*name*` [, `*features*`])`
  - opens a new browser window/tab
  - returns a reference to a window object
  - *url* is the URL to access in the new window; can be the empty string
  - *name* is a name given to the window for later reference
  - *features* is a string that determines various window features

The standard sequence for the creation of a new windows is not:

```
// new instance of 'Window' class
var newWin = new Window(...)
newWin.document.write('<html>...</html>')
```

instead it is

```
// new window created by using 'open' with an existing one
var newWin = window.open(...)
newWin.document.write('<html>...</html>')
```


Window object

Methods provided by a window object include

- `close()`
  - closes a browser window/tab
- `focus()`
  - give focus to a window (bring the window to the front)
- `blur()`
  - removes focus from a window (moves the window behind others)
- `print()`
  - prints (sends to a printer) the contents of the current window

Window object: Example

```
<html><head><title>Window handling</title>
<script type="text/javascript">
function Help() {
  var OutputWindow = window.open('','Help','resizable=1');
  with (OutputWindow.document) {
    open()
    writeln("<!DOCTYPE html><html><head><title>Help</title>\
    </head><body>This might be a context-sensitive help\
    message, depending on the application and state of the
    page.</body></html>");
    close()
  }
}
</script></head><body>
<form name="ButtonForm" id="ButtonForm" action="">
<p>
  <input type="button" value="Click for Help"
         onclick="Help();">
</p>
</form></body></html>
```

Window object: Dialog boxes

- Often we only want to open a new window in order to
  - display a message
  - ask for confirmation of an action
  - request an input
- For these purposes, the window object in JavaScript provides pre-defined methods for the handling of dialog boxes (windows for simple dialogs):
  - `null alert(message_string)`
  - `bool confirm(message_string)`
  - `string prompt(message_string, default)`

Window object: Dialog boxes

- `null alert(message_string)`
  - creates a message box displaying *message_string*
  - the box contains an 'OK' button that the user will have to click (alternatively, the message box can be closed) for the execution of the remaining code to proceed

Example:

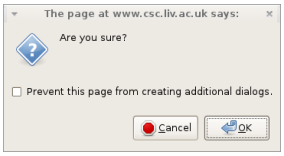
```
alert("Local time: " + (new Date).toString())
```

Window object: Dialog boxes

- `bool confirm(message_string)`
  - creates a message box displaying *message_string*
  - the box contains two buttons 'Cancel' and 'OK'
  - the function returns `true` if the user selects 'OK', `false` otherwise

Example:

```
var answer = confirm("Are you sure?")
```



Window object: Dialog boxes

- `string prompt(message_string, default)`
  - creates a dialog box displaying *message_string* and an input field
  - if a second argument *default* is given, *default* will be shown in the input field
  - the box contains two buttons 'Cancel' and 'OK'
  - if the user selects 'OK' then the current value entered in the input field is returned as a string, otherwise `null` is returned

Example:

```
var userName =
  prompt("What is your name?",
         "")
```

Window object: Dialog boxes

- `prompt()` always returns a string, even if the user enters a number
- To convert a string to number the following functions can be used:
  - `number parseInt(string [,base])`
    - converts *string* to an integer number wrt numeral system *base*
    - only converts up to the first invalid character in *string*
    - if the first non-whitespace character in *string* is not a digit, returns `NaN`
  - `number parseFloat(string)`
    - converts *string* to a floating-point number
    - only converts up to the first invalid character in *string*
    - if the first non-whitespace character in *string* is not a digit, returns `NaN`
  - `number Number(string)`
    - returns `NaN` if *string* contains an invalid character

Dialog boxes: Example

```
<html>
 <head><title>Interaction example</title></head>
<body>
<script type="text/javascript">
do {
  string   = prompt("How many items do you want to buy?")
  quantity = parseInt(string)
} while (isNaN(quantity) || quantity <= 0)
do {
  string = prompt("How much does an item cost?")
  price  = parseFloat(string)
} while (isNaN(price) || price <= 0)
buy = confirm("You will have to pay "+
              (price*quantity).toFixed(2)+
              "\nDo you want to proceed?")
if (buy) alert("Purchase made")
</script>
</body></html>
```

http://cgi.csc.liv.ac.uk/~ullrich/COMP284/examples/jsPrompt.html

User input validation

- A common use of JavaScript is the validation of user input in a HTML form before it is processed:
  - check that required fields have not been left empty
  - check that fields only contain allowed characters or comply to a certain grammar
  - check that values are within allowed bounds

```
<form method="post" action="process.php"
      onSubmit="return validate(this)">
  <label>User name:     <input type="text" name="user"></label>
  <label>Email address: <input type="text" name="email"></label>
  <input type="submit" name="submit">
</form>
<script>
function validate(form) {
  fail  = validateUser(form.user.value)
  fail += validateEmail(form.email.value)
  if (fail == "") return true
  else { alert(fail); return false } }
</script>
```


User input validation

```
1  function validateUser(field) {
2    if (field == "") return "No␣username␣entered\n"
3    else if (field.length < 5)
4      return "Username␣too␣short\n"
5    else if (/[^a-zA-Z0-9_-]/.test(field))
6      return "Invalid␣character␣in␣username\n"
7    else return ""
8  }
9
10 function validateEmail(field) {
11   if (field == "") return "No␣email␣entered\n"
12   else if (!((field.indexOf(".") > 0) &&
13              (field.indexOf("@") > 0)) ||
14            /[^a-zA-Z0-9.@_-]/.test(field))
15     return "Invalid␣character␣in␣email\n"
16   else return ""
17 }
```

http://cgi.csc.liv.ac.uk/~ullrich/COMP284/examples/jsValidate.html

Window and Document objects

JavaScript provides two objects that are essential to the creation of dynamic web pages and interactive web applications:

document object

- an object-oriented representation of a web page (HTML document) that is displayed in a window
- allows interaction with the Document Object Model (DOM) of a page

  Example: `document.writeln()` adds content to a web page

Document Object Model

A platform- and language-neutral interface that allows programs and scripts to dynamically access and update the content, structure and style of HTML, XHTML and XML documents

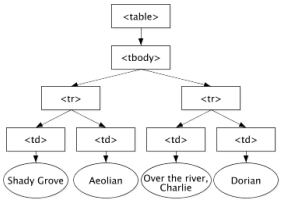
Document Object Model

Example:

The HTML table below

```
<table>
  <tbody>
    <tr>
      <td>Shady Grove</td>
      <td>Aeolian</td>
    </tr>
    <tr>
      <td>Over the River, Charlie</td>
      <td>Dorian</td>
    </tr>
  </tbody>
</table>
```

is parsed into the following DOM



Arnaud Le Hors, et al, editors: Document Object Model (DOM) Level 3 Core Specification, Version 1.0, W3C Recommendation 07 April 2004. World Wide Web Consortium, 2004. https://www.w3.org/TR/DOM-Level-3-Core/ [accessed 9 January 2017]

Accessing HTML elements: Object methods

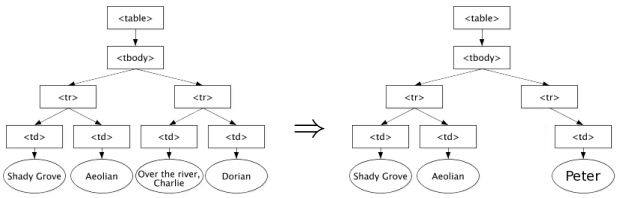
Example:

```
// access the tbody element from the table element
var myTbodyElement = myTableElement.firstChild;

// access its second tr element; the list of children starts at 0 (not 1).
var mySecondTrElement = myTbodyElement.childNodes[1];

// remove its first td element
mySecondTrElement.removeChild(mySecondTrElement.firstChild);

// change the text content of the remaining td element
mySecondTrElement.firstChild.firstChild.data = "Peter";
```



Accessing HTML elements: Names (1)

Instead of using methods such as `firstChild` and `childNodes[n]`, it is possible to assign names to denote the children of a HTML element

Example:

```
<form name="form1" action="">
<label>Temperature in Fahrenheit:</label>
<input type="text" name="fahrenheit" size="10" value="0"><br>
<label>Temperature in Celsius:</label>
<input type="text" name="celsius"    size="10" value="">
</form>
```

Then
- `document.form1`
  Refers to the whole form
- `document.form1.celsius`
  Refers to the text field named `celsius` in `document.form1`
- `document.form1.celsius.value`
  Refers to the attribute `value` in the text field named `celsius` in `document.form1`

Accessing HTML elements: Names (2)

Accessing HTML elements by giving them names and using paths within the Document Object Model tree structure is still problematic

⇝ If that tree structure changes, then those paths no longer work

Example:

Changing the previous form to

```
<form name="form1" action="">
<div class="field" name="fdiv">
<label>Temperature in Fahrenheit:</label>
<input type="text" name ="fahrenheit" size=10 value="0" />
</div>
<div class="field" name="cdiv">
<label>Temperature in Celsius:</label>
<input type="text" name="celsius"     size="10" value="" />
</div>
</form>
```

means that `document.form1.celsius` no longer works as there is now a `div` element between form and text field, we would now need to use `document.form1.cdiv.celsius`

Accessing HTML elements: IDs

A more reliable way is to give each HTML element an ID (using the `id` attribute) and to use `getElementById` to retrieve a HTML element by its ID

Example:

```
<form id="form1" action="">
<label>Temperature in Fahrenheit:</label>
<input type="text" id="fahrenheit" size="10" value="0"><br>
<label>Temperature in Celsius:</label>
<input type="text" id="celsius"    size="10" value="">
</form>
```

Then
- `document.getElementById('celsius')`
  Refers to the HTML element with ID `celsius` document
- `document.getElementById('celsius').value`
  Refers to the attribute `value` in the HTML element with ID `celsius` in `document`

Manipulating HTML elements

It is not only possible to access HTML elements, but also possible to change them on-the-fly

```
<html><head><title>Manipulating HTML elements</title>
<style>
  td.RedBG { background: #f00; }
</style>
<script>
function changeBackground1(id) {
  document.getElementById(id).style.background = "#00f";
  document.getElementById(id).innerHTML = "blue";
}
function changeBackground2(id) {
  document.getElementById(id).cell.className = "RedBG";
  document.getElementById(id).cell.innerHTML = "red";
}
</script></head><body>
<table border="1"><tr>
 <td id="0" onclick="changeBackground1('0');">white</td>
 <td id="1" onclick="changeBackground2('1');">white</td>
</tr></table></body></html>
```

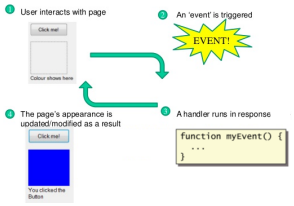
http://cgi.csc.liv.ac.uk/~ullrich/COMP284/examples/jsBG.html

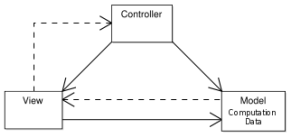
Event-driven JavaScript Programs

- The JavaScript programs we have seen so far were all executed sequentially
  - programs have a particular starting point
  - programs are executed step-by-step, involving control structures and function execution
  - programs reach a point at which their execution stops

Event-Driven JavaScript Programs

- Web applications are event-driven
  ⇝ they react to events such as mouse clicks and key strokes





nickywalters: What is Event Driven Programming?
SlideShare, 7 September 2014.
https://tinyurl.com/ya58xbs9 [accessed 5/11/2017]

- With JavaScript,
  – we can define event handler functions for a wide variety of events
  – event handler functions can manipulate the document object
  (changing the web page in situ)

Event Handlers and HTML Elements

- HTML events are things, mostly user actions, that happen to HTML elements
- Event handlers are JavaScript functions that process events
- Event handlers must be associated with HTML elements for specific events
- This can be done via attributes

```
<input type="button" value="Help"  onclick="Help()">
```

- Alternatively, a JavaScript function can be used to add a handler to an HTML element

```
// All good browsers
window.addEventListener("load", Hello)
// MS IE browser
window.attachEvent("onload", Hello)
```

More than one event handler can be added this way to the same element for the same event

Event Handlers and HTML Elements

- As our scripts should work with as many browsers as possible, we need to detect which method works:

```
if (window.addEventListener) {
   window.addEventListener("load", Hello)
} else {
   window.attachEvent("onload", Hello)
}
```

- Event handlers can also be removed

```
if (window.removeEventListener) {
   window.removeEventListener("load", Hello)
} else {
   window.detachEvent("onload", Hello)
}
```


Events: Load

- An (on)load event occurs when an object has been loaded
- Typically, event handlers for onload events are associated with the window object or the body element of an HTML document

```
<html>
  <head>
    <title>Onload Example</title>
    <script type="text/javascript">
      function Hello()   { alert("Welcome to my page!")  }
    </script>
  </head>
  <body onload="Hello()">
    <p>Content of the web page</p>
  </body>
</html>
```

http://cgi.csc.liv.ac.uk/~ullrich/COMP519/examples/jsOnload.html

Events: Focus / Change

- A focus event occurs when a form field receives input focus by tabbing with the keyboard or clicking with the mouse
  ⇝ onFocus attribute
- A change event occurs when a select, text, or textarea field loses focus and its value has been modified
  ⇝ onChange attribute

Example:

```
<form name="form1" method="post" action="process.php">
  <select name="select" required
          onChange="document.form1.submit();">
    <option value="">Select a name</option>
    <option value="200812345">Tom Beck</option>
    <option value="200867890">Jim Kent</option>
  </select>
</form>
```


Events: Focus / Change

- A focus event occurs when a form field receives input focus by tabbing with the keyboard or clicking with the mouse
  ⇝ onFocus attribute
- A change event occurs when a select, text, or textarea field loses focus and its value has been modified
  ⇝ onChange attribute

```
<form>
<label>Temperature in Fahrenheit:</label>
<input type="text" id="fahrenheit" size="10" value="0"
 onchange="document.getElementById('celsius').value =
 FahrenheitToCelsius(parseFloat(
 document.getElementById('fahrenheit').value)).toFixed(1);"
 ><br>
<label>Temperature in Celsius:</label>
<input type="text" id="celsius"
 size="10" value="" onfocus="blur();"></form>
```

http://cgi.csc.liv.ac.uk/~ullrich/COMP519/examples/jsOnchange.html

Events: Blur / Click

- A blur event occurs when an HTML element loses focus
  ⇝ onBlur attribute
- A click event occurs when an object on a form is clicked
  ⇝ onClick attribute

Example:

```
<html><head><title>Onclick Example</title></head><body>
<form name="form1" action="">
  Enter a number here:
  <input type="text" size="12" id="number" value="3.1">
  <br><br>
  <input type="button" value="Double"
    onclick="document.getElementById('number').value =
    parseFloat(document.getElementById('number').value)
    * 2;">
</form></body></html>
```

http://cgi.csc.liv.ac.uk/~ullrich/COMP284/examples/jsOnclick.html

Events: MouseOver / Select / Submit

- A keydown event occurs when the user presses a key
  ⇝ onkeydown attribute
- A mouseOver event occurs once each time the mouse pointer moves over an HTML element from outside that element
  ⇝ onMouseOver attribute
- A select event occurs when a user selects some of the text within a text or textarea field
  ⇝ onSelect attribute
- A submit event occurs when a user submits a form
  ⇝ onSubmit attribute

Events and DOM

- When an event occurs, an event object is created
  - ⇝ an event object has attributes and methods
  - ⇝ event objects can be created by your code independent of an event occurring
- In most browsers, the event object is passed to event handler functions as an argument
- In most versions of Microsoft Internet Explorer, the most recent event can only be accessed via `window.event`

```
<html><body onKeyDown="processKey(event)">
  <script>
    function processKey(e) {
      e = e || window.event
      document.getElementById("key").innerHTML =
       String.fromCharCode(e.keyCode)+' has been pressed'}
  </script>
  <!-- key code will appear in the paragraph below -->
  <p id="key"></p>
</body></html>
```


Revision

Read

- Chapter 17: JavaScript and PHP Validation and Error Handling
- Chapter 18: Using Ajax

of

R. Nixon:
Learning PHP, MySQL, and JavaScript.
O'Reilly, 2009.

- Mozilla Developer Network and individual contributors: Document Object Model (DOM), 18 March 2014. `https://developer.mozilla.org/en/docs/DOM` [accessed 18 March 2014].
- W3Schools: JavaScript and HTML DOM Reference, 18 March 2014. `http://www.w3schools.com/jsref/` [accessed 18 March 2014].