Web Storage

Web storage is key/value JavaScript API on web browsers to

- store *user data* locally on host running browser
- put it in *silos* correlated with web server *domain* origins

Only web pages from domain D can access D's *silo* via web storage.

localStorage stores data persistently for each web server

sessionStorage stores data sessionally per server per browser instance

These two associative arrays support the functions

clear()	remove all key/value pairs
getItem(key)	get value of *key*
key(i)	get *i*th element of array of keys
removeItem(key)	delete *key* and its value
setItem(key, value)	set *key* to *value*

These forms of web storage can be exercised by the form

```
<body onload="displayWebStore()">
 <h3>Web Storage</h3>
 <div id="WSData" style="margin-left: 1em"></div>
 <form action=""> <p>
   Key <input id="key" value=""/>
   Value <input id="val" value=""/>
   <input type="button" value="update" onclick="updateWebStore()">
  </p> <p>
   <input type="button" value="clear store" onclick="clearWebStore()">
   <input type="reset" value="reset form">
  </p>
 </form>
 <script type="text/javascript" src="storage.js"></script>
</body>
```

Web storage stores data sessionally or persistently.

Session data in browser's volatile memory is lost when it dies.

Web Storage

JavaScript file *storage.js* supports web storage use

```
function displayWebStore() {
  db = localStorage;
  if ( window.location.href.indexOf("?session") != -1 )
    db = sessionStorage;
  if ( typeof(db) == "undefined" ) {
    alert("no web storage available");
    return;
  }
  var s = "";
  for (var i=0; i<db.length; i++) {
    var key = db.key(i);
    s += key + " = " + db.getItem(key) + "<br/>";
  }
  document.getElementById("WSData").innerHTML = s;
}

function updateWebStore() {
  var key = document.getElementById("key").value;
  if ( key != "" )
    db[key] = document.getElementById("val").value;
  displayWebStore();
}

function clearWebStore() {
  db.clear();
  displayWebStore();
}
```

Web storage is sessional here if "?session" in URL otherwise it's not.

Example defines 3 functions for origin web server of web page

displayWebStore()	display keys and values in web storage
updateWebStore()	updates web storage with key/value from form
clearWebStore()	removes key/value pairs from web storage

Web storage avoids

- *size limits* of cookies
- *security issues* of cookie traffic with web server

Cookies

Cookies are small packets of data stored in file space of web users.

Cookies contain

- name
- value
- domain
- path
- duration

Web servers ask web clients to store cookies.

Compliant web clients supply *name* and *value* of cookie

- within that *duration*
- when requesting web pages for that *domain* on that *path*

Domain, *path*, *duration* are optional. Defaults apply if missing.

Cookies let web server applications connect HTTP requests together.

Cookies

- beat HTTP's statelessness by storing data on client
- are passed in *Set-Cookie* reply and *Cookie* request headers
- are stored in user's filespace or in browser's volatile memory

Cookies can be manipulated by

- meta directive in HTML
- JavaScript in web page
- web server programs - CGI, servlets, JSP, Node.js etc.

Cookie Standards

There used to be 2 cookie standards - Netscape's and RFC 2965.

Combined reformed standard RFC 6265 now governs cookie usage.

Form of major fields to set a cookie with *Max-age* is:

```
Set-Cookie: name=value [; Path=path] [; Domain=domain]
            [; Max-age=seconds] [; Secure] [; HttpOnly]
```

Optional attributes, which may occur once in any order, are:

Domain	domain of validity - defaults to server's hostname
Expires	RFC 1123 date time when cookie expires
HttpOnly	cookie is only to be sent over HTTP
Max-age	how many seconds from now onwards to keep cookie
Path	defaults to path of document creating cookie
Secure	cookie is only to be sent on secure channel (HTTPS)

Max-age trumps *Expires* header if both are present.

RFC 6265 example request with *Max-age* would be

```
Set-Cookie: Customer="WILE E COYOTE"; Path=/acme;
             Max-age=31536000; Domain=dod.gov
```

It specifies cookie should last for one year in seconds.

Form of request to set cookie with *Expires* date-time is

```
Set-Cookie: name=value [; Expires=date] [; Domain=domain]
            [; Path=path] [; Secure] [; HttpOnly]
```

Example request would be

```
Set-Cookie: Customer=WILE_E_COYOTE; Path=/acme; Domain=dod.gov;
            Expires="Tue, 25 Dec 2018 21:00:00 GMT"
```

Date time must use 4 digit year required by RFC 1123 section 5.2.14.

Cookie Properties

Session cookies last only as long as client's session

- have no *Max-age* or *Expires* setting
- are usually stored in browser's volatile memory
- aren't stored in cookie file(s) on client

Saving cookie with exactly same *Path*, *Domain* and *Name* values

- *overwrites* existing one
- *deletes* it if expiry date is past

Path matching is done by string comparison

- path "/ban" matches "/ban/bomb.html" and "/banana.html"
- most general path is "/"
- default path is URL path of associated document

Browsers often allow users to control whether to

- accept *all* cookies
- accept only cookies from web server's *domain*
- do *not* accept or send cookies
- *warn* and let user control acceptance of cookies
- *limit* maximum lifetime of cookies

Cookie controls

Mozilla	Edit, Preferences, Privacy and Security, Cookies
Explorer	Tools, Internet Options, Privacy, Advanced...

RFC 6265 Cookie Dialogue

Wile E Coyote fills in *Arms Inc* login form and sends to server:

```
POST /login HTTP/1.1
 ...
```

Server replies to him with arms purchase form

```
HTTP/1.1 200 OK
Set-Cookie: Customer="Wile E Coyote"; Path=/
 ...
```

He submits form asking to buy *rocket launcher* and sends:

```
POST /arms/buy HTTP/1.1
Cookie: Customer="Wile E Coyote"; Path=/
 ...
```

Server replies to him with arms shipping form and sends:

```
HTTP/1.1 200 OK
Set-Cookie: Part_Number="rocket launcher"; Path=/arms
 ...
```

He submits form asking for USAF air freight and sends:

```
POST /arms/shipping/freight HTTP/1.1
Cookie: Part_Number="rocket launcher";
        Path=/arms; Customer="Wile E Coyote"; Path=/;
 ...
```

Server confirms order and shipping details and sends:

```
HTTP/1.1 200 OK
Set-Cookie: Shipping=USAF; Path=/arms/shipping
 ...
```

Wile E Coyote requests details of shipping schedules and sends:

```
GET /arms/shipping/schedules.html HTTP/1.1
Cookie: Shipping=USAF; Path=/arms/shipping;
        Part_Number="rocket launcher"; Path=/arms;
        Customer="Wile E Coyote"; Path=/
```

Handling Cookies in Web Pages

Cookie can be set by adding directive to header of HTML page

```
<meta http-equiv="Set-Cookie" content="name=Harry; Max-age=86400">
```

It requests browser to act as if it had received the HTTP header

```
Set-Cookie: name=Harry; Max-age=86400
```

It is removed with a

```
<meta http-equiv="Set-Cookie" content="name=; Max-age=-1">
```

Cookies are set in JavaScript in following manner:

```
document.cookie = "name=value; Expires=...; Path=...; Domain=...";
```

Consider the Javascript function *cookie.js*

```
function Cookie(name, value, days) {
  d = new Date();
  if ( days < 0 ) { d.setTime(0); }
  else { d.setTime(d.getTime() + days * 24*60*60*1000); }
  return name+"=\""+value+"\"; Expires=\""+d.toUTCString()+\";
}
```

It can be invoked to set a cookie for a year by

```
<body onload="go()">
 <script src="cookie.js"></script>
 <script>
   function go() {
    c = Cookie("Skill", "basic", 365) + "; Domain=hw.ac.uk; Path=/";
    document.cookie = c;
    document.writeln("Cookie set: " + c); }
 </script>
</body>
```

Same cookie can be deleted by giving it a date in the past:

```
<body onload="go()">
 <script src="cookie.js"></script>
 <script>
  function go() {
   c = Cookie("Skill", "anything", -1) + "; Domain=hw.ac.uk; Path=/";
   document.cookie = c;
   document.writeln("Cookie set: " + c); }
 </script>
</body>
```

Handling Cookies in JavaScript

Useful JavaScript in *cookiefunctions.js* for handling cookies:

```
function getCookieNames() {
  if ( document.cookie == "" ) return new Array();
  var c = document.cookie.split(";");
  var names = new Array(c.length);
  for (var i=0; i<c.length; i++)
    names[i] = c[i].substr(0, c[i].indexOf("="));
  return names;
}

function getCookieValue(name) {
  var cs = document.cookie.split(";");
  for (var k=0; k<cs.length; k++) {
    Nam = cs[k].substr(0, cs[k].indexOf("="));
    Val = cs[k].substr(cs[k].indexOf("=") + 1);
    if ( Nam == name )
      return unescape(Val);
  }
  return null;
}
```

getCookieNames() returns array of cookie names.

getCookieValue() takes a string argument and returns its cookie value.

All cookies delivered with a page can be listed by listcookies.html

```
<html>
 <body>
  <script type="text/javascript" src="cookiefunctions.js"></script>
  <script type="text/javascript">
    var arr = getCookieNames();
    for (var j=0; j<arr.length; j++) {
      var val = getCookieValue(arr[j]);
      if ( val != null ) {
        document.writeln(arr[j] + " = " + unescape(val));
        document.writeln("<br>");
      }
    }
  </script>
 </body>
</html>
```

It uses 2 functions to list cookie names and values given with page.

Handling Cookies with Node.js

Module cookie-parser uses *req.cookies* to get cookie details.

Module set-cookie lets *setCookie()* set a cookie's name, value etc.

Install both modules in addition to the Express module.

Script *node_cookie.js* updates and reports on stored data

```
var express = require('express');
var cookieParser = require('cookie-parser');
var setCookie = require('set-cookie');
var http = require('http');
var app = express();
app.use(cookieParser());
var head = ["<!DOCTYPE html>", "<html>", "<head>",
            "<title>Cookies</title>", "</head>", "<body>"];
var url = "http://linux11.macs.hw.ac.uk:8080/cookie";
var tail = ["<form action='" + url + "'>",
            "<p>", "Key <input name='name' value=''>", "</p>",
            "<p>", "Value <input name='value' value=''>", "</p>",
            "<p>", "Age <input name='age' value=''>", "</p>",
            "<p>", "<input type='submit'/>", "</p>",
            "</form>", "</body>", "</html>"];
var eol = "\n";

app.get("/cookie", function (req, res) {
  var s = "";
  for (var i in head) s += head[i] + eol;
  if ( req.query.name != undefined && req.query.name != "" &&
       req.query.value != undefined ) {
    if ( isNaN(req.query.age) ) req.query.age = 0;
    setCookie(req.query.name, req.query.value,
              {maxAge: req.query.age, res: res});
  }
  s += "<p>" + eol;
  for (var i in req.cookies)
    s += i + " = " +  req.cookies[i] + "<br>" + eol;
  s += "</p>" + eol;
  for (var i in tail) s += tail[i] + eol;
  res.send(s); // send web page
});

var svr = http.createServer(app);
svr.on('error', function(err) { console.log('Server: ' + err); });
svr.listen(8080, function() {console.log("Node: linux11 port 8080");});
```

Web app can be run (and killed) remotely on *linux11*.

In HWU access *node_cookie.js* at URL.

Handling Cookies with JSP

JSP page *cookies.jsp* sets, displays and captures cookies:

```
<!DOCTYPE html>
<html>
<head> <title>Cookies</title> </head>
<body>
<form action="http://www2.macs.hw.ac.uk:8080/demo/cookies.jsp">
<% String name  = request.getParameter("cookie");
   String value = request.getParameter("value");
   String domain, maxage, path, s = "";
   if ( name != null && value != null &&
        ! name.equals("") && ! value.equals("") ) {
     Cookie c = new Cookie(java.net.URLEncoder.encode(name, "UTF-8"),
                           java.net.URLEncoder.encode(value, "UTF-8"));
     if ( (domain = request.getParameter("domain")) != null )
       c.setDomain(domain);
     if ( (path = request.getParameter("path")) != null )
       c.setPath(path);
     if ( (maxage = request.getParameter("maxage")) != null )
       try { c.setMaxAge(Integer.parseInt(maxage));
       } catch (Exception e) {}
     response.addCookie(c);
   }
   Cookie cs[] = request.getCookies();
   if ( cs != null )
     for (int i=0; i<cs.length; i++) {
       s += "<br>" + java.net.URLDecoder.decode(cs[i].getName()) +
            " = " + java.net.URLDecoder.decode(cs[i].getValue());
       if ( cs[i].getPath() != null )
         s += " " + cs[i].getPath();
     }
   out.println(s); %>
   <h4>New cookie</h4>
     <table>
     <tr> <td>Name    <td> <input name=cookie size=20> </tr>
     <tr> <td>Value   <td> <input name=value  size=40> </tr>
     <tr> <td>Domain  <td> <input name=domain size=20> </tr>
     <tr> <td>Max Age <td> <input name=maxage size=10> </tr>
     <tr> <td>Path    <td> <input name=path   size=40> </tr>
     <tr> <td><input type=submit> </tr>
   </table>
 </form>
</body>
</html>
```

addCookie() adds a *Set-Cookie* request to reply headers.

getCookies() gets cookies supplied to web server by browser.

Cookie names and values are url-encoded to hide illegal characters.

Cookies, Web Storage and EU Law

In 2012 UK implemented clause 66 of EU directive 2009/136/EC to

- require apt consent for storage or access to information
- where it is stored on a user's terminal equipment

Change was motivated by concerns about

- *online tracking* of individuals and use of *spyware*
- on their devices without their knowledge and agreement

Directive applies to use of *cookies* and use of HTML 5 *Web Storage*.

Apt consent should be *explicit* and *prior* if possible and be

> a freely given *specific* and *informed* indication of a data subject's *wishes* by which he signifies his agreement to *personal data* relating to him being processed

i.e. it must be *specific*, *informed* and *express user's wishes*.

Consent isn't needed where personal data storage on user computer

(a) is solely to enable communication over electronic network

(b) is strictly necessary to provide an information society service requested by user.

So informed specific consent isn't (supposed to be) needed for

- a shopping cart
- security requirement like authentication to protect user's privacy

It is needed for advertising, analysing page usage, customisation.

UK Information Commissioner supplies guidance on compliance.

Things To Do

Read Web Storage (2nd ed) https://www.w3.org/TR/webstorage/
W3C Recommendation 19 April 2016

Read HTTP State Management Mechanism
RFC 6265, A. Barth, April 2011

Try *Quiz on Lecture*

Do *Exercise 12*

Key Points

- Web storage is a key/value JavaScript API supported by web browsers to store and retrieve user data. It is put in storage silos correlated with web server domain origins so only their HTML pages can manipulate it via the API. Session storage stores the data in the browser's volatile memory. Local storage stores it in the browser's configuration directory.

- Cookies are packets of data stored in the file space of web browser users. Web servers ask web clients via HTTP to store cookies. Compliant web clients supply name and value of cookie within that duration when requesting web pages for that domain on that path within HTTP requests.

- UK law protects computer users from covert tracking and spyware by requiring free, specific and informed consent to their personal data being stored on their own machine. It applies to web apps using cookies and local web storage in the EU.