Objective

- **Geolocation**
- **HTML5 Drag and Drop**
- **HTML5 Web Storage**
- **HTML5 Application Cache**
- **HTML5 Web Workers**

Geolocation

HTML5 Geolocation is used to locate a user's position

Locate the User's Position

- The HTML5 Geolocation API is used to get the geographical position of a user.
- Since this can compromise user privacy, the position is not available unless the user approves it.

Browser Support

Internet Explorer 9+, Firefox, Chrome, Safari and Opera support Geolocation.

HTML5 - Using Geolocation

- Use the getCurrentPosition() method to get the user's position.
- The example below is a simple Geolocation example returning the latitude and longitude of the user's position:

Example

```
<script>
var x=document.getElementById("demo");
function getLocation()
  {
  if (navigator.geolocation)
    {
    navigator.geolocation.getCurrentPosition(showPosition);
    }
  else{x.innerHTML="Geolocation is not supported by this browser.";}
  }
function showPosition(position)
  {
  x.innerHTML="Latitude: " + position.coords.latitude +
  "<br>Longitude: " + position.coords.longitude;
  }
</script>
```

Example

```
<!DOCTYPE html>
<html>
<body>
<p id="demo">Click the button to get your coordinates:</p>
<button onclick="getLocation()">Try It</button>
<script>
var x=document.getElementById("demo");
function getLocation()
  {
  if (navigator.geolocation)
    {
    navigator.geolocation.getCurrentPosition(showPosition);
    }
  else{x.innerHTML="Geolocation is not supported by this browser.";}
  }
function showPosition(position)
  {
  x.innerHTML="Latitude: " + position.coords.latitude +
  "<br>Longitude: " + position.coords.longitude;
  }
</script>
</body>
</html>
```

Example explained:

Check if Geolocation is supported
If supported, run the getCurrentPosition() method. If not, display a message to the user
If the getCurrentPosition() method is successful, it returns a coordinates object to the function specified in the parameter (showPosition)
The showPosition() function gets the displays the Latitude and Longitude
The example above is a very basic Geolocation script, with no error handling.

Handling Errors and Rejections

The second parameter of the getCurrentPosition() method is used to handle errors. It specifies a function to run if it fails to get the user's location:

Example

```
function showError(error)
  {
  switch(error.code)
    {
    case error.PERMISSION_DENIED:
      x.innerHTML="User denied the request for Geolocation."
      break;
```

```
    case error.POSITION_UNAVAILABLE:
      x.innerHTML="Location information is unavailable."
      break;
    case error.TIMEOUT:
      x.innerHTML="The request to get user location timed out."
      break;
    case error.UNKNOWN_ERROR:
      x.innerHTML="An unknown error occurred."
      break;
    }
  }
```

Example

```
<!DOCTYPE html>
<html>
<body>
<p id="demo">Click the button to get your coordinates:</p>
<button onclick="getLocation()">Try It</button>
<script>
var x=document.getElementById("demo");
function getLocation()
  {
  if (navigator.geolocation)
    {
    navigator.geolocation.getCurrentPosition(showPosition,showError);
    }
  else{x.innerHTML="Geolocation is not supported by this browser.";}
  }
function showPosition(position)
  {
  x.innerHTML="Latitude: " + position.coords.latitude +
  "<br>Longitude: " + position.coords.longitude;
  }
function showError(error)
  {
  switch(error.code)
    {
    case error.PERMISSION_DENIED:
      x.innerHTML="User denied the request for Geolocation."
      break;
    case error.POSITION_UNAVAILABLE:
      x.innerHTML="Location information is unavailable."
```

```
    break;
  case error.TIMEOUT:
    x.innerHTML="The request to get user location timed out."
    break;
  case error.UNKNOWN_ERROR:
    x.innerHTML="An unknown error occurred."
    break;
  }
 }
</script>
</body>
</html>
```

Error Codes:
Permission denied - The user did not allow Geolocation
Position unavailable - It is not possible to get the current location
Timeout - The operation timed out

Displaying the Result in a Map

- To display the result in a map, you need access to a map service that can use latitude and longitude, like Google Maps:

Example

```
function showPosition(position)
{
var latlon=position.coords.latitude+","+position.coords.longitude;

var img_url="http://maps.googleapis.com/maps/api/staticmap?center="
+latlon+"&zoom=14&size=400x300&sensor=false";

document.getElementById("mapholder").innerHTML="<img src='"+img_url+"'>";
}
```

Example

```
<!DOCTYPE html>
<html>
<body>
<p id="demo">Click the button to get your position:</p>
<button onclick="getLocation()">Try It</button>
<div id="mapholder"></div>
<script>
var x=document.getElementById("demo");
function getLocation()
  {
  if (navigator.geolocation)
```

```
  {
  navigator.geolocation.getCurrentPosition(showPosition,showError);
  }
 else{x.innerHTML="Geolocation is not supported by this browser.";}
 }

function showPosition(position)
 {
 var latlon=position.coords.latitude+","+position.coords.longitude;

 var img_url="http://maps.googleapis.com/maps/api/staticmap?center="
 +latlon+"&zoom=14&size=400x300&sensor=false";
 document.getElementById("mapholder").innerHTML="<img src='"+img_url+"'>";
 }

function showError(error)
 {
 switch(error.code)
  {
  case error.PERMISSION_DENIED:
    x.innerHTML="User denied the request for Geolocation."
    break;
  case error.POSITION_UNAVAILABLE:
    x.innerHTML="Location information is unavailable."
    break;
  case error.TIMEOUT:
    x.innerHTML="The request to get user location timed out."
    break;
  case error.UNKNOWN_ERROR:
    x.innerHTML="An unknown error occurred."
    break;
  }
 }
</script>
</body>
</html>
```

In the example above we use the returned latitude and longitude data to show the location in a Google map (using a static image).

Google Map Script
How to use a script to show an interactive map with a marker, zoom and drag option

Location-specific Information
This page demonstrated how to show a user's position on a map. However, Geolocation is also very useful for location-specific information.

Examples:

Up-to-date local information
Showing Points-of-interest near the user
Turn-by-turn navigation (GPS)

The getCurrentPosition() Method - Return Data

The getCurrentPosition() method returns an object if it is successful. The latitude, longitude and accuracy properties are always returned. The other properties below are returned if available.

Property	Description
coords.latitude	The latitude as a decimal number
coords.longitude	The longitude as a decimal number
coords.accuracy	The accuracy of position
coords.altitude	The altitude in meters above the mean sea level
coords.altitudeAccuracy	The altitude accuracy of position
coords.heading	The heading as degrees clockwise from North
coords.speed	The speed in meters per second
timestamp	The date/time of the response

Geolocation object - Other interesting Methods
watchPosition() - Returns the current position of the user and continues to return updated position as the user moves (like the GPS in a car).
clearWatch() - Stops the watchPosition() method.

The example below shows the watchPosition() method. You need an accurate GPS device to test this (like iPhone):

Example

```
<script>
var x=document.getElementById("demo");
function getLocation()
  {
  if (navigator.geolocation)
    {
    navigator.geolocation.watchPosition(showPosition);
    }
  else{x.innerHTML="Geolocation is not supported by this browser.";}
  }
function showPosition(position)
  {
  x.innerHTML="Latitude: " + position.coords.latitude +
  "<br>Longitude: " + position.coords.longitude;
  }
</script>
```

Example

```
<!DOCTYPE html>
<html>
<body>
<p id="demo">Click the button to get your coordinates:</p>
<button onclick="getLocation()">Try It</button>
<script>
var x=document.getElementById("demo");
function getLocation()
  {
  if (navigator.geolocation)
    {
    navigator.geolocation.watchPosition(showPosition);
    }
  else{x.innerHTML="Geolocation is not supported by this browser.";}
  }
function showPosition(position)
  {
  x.innerHTML="Latitude: " + position.coords.latitude +
  "<br>Longitude: " + position.coords.longitude;
  }
</script>
</body>
</html>
```

HTML5 Drag and Drop

Drag and drop is a part of the HTML5 standard.

Drag and Drop

Drag and drop is a very common feature. It is when you "grab" an object and drag it to a different location.

Browser Support

Internet Explorer 9+, Firefox, Opera, Chrome, and Safari support drag and drop.

Example

```
<!DOCTYPE HTML>
<html>
<head>
<script>
function allowDrop(ev)
{
ev.preventDefault();
```

```
}

function drag(ev)
{
ev.dataTransfer.setData("Text",ev.target.id);
}

function drop(ev)
{
ev.preventDefault();
var data=ev.dataTransfer.getData("Text");
ev.target.appendChild(document.getElementById(data));
}
</script>
</head>
<body>

<div id="div1" ondrop="drop(event)"
ondragover="allowDrop(event)"></div>

<img id="drag1" src="img_logo.gif" draggable="true"
ondragstart="drag(event)" width="336" height="69"

</body>
</html>
```

HTML5 Web Storage

What is HTML5 Web Storage?

- With HTML5, web pages can store data locally within the user's browser.
- Earlier, this was done with cookies. However, Web Storage is more secure and faster. The data is not included with every server request, but used ONLY when asked for. It is also possible to store large amounts of data, without affecting the website's performance.
- The data is stored in key/value pairs, and a web page cannonly access data stored by itself

Browser Support:
Web storage is supported in Internet Explorer 8+, Firefox, Opera, Chrome, and Safari.localStorage and sessionStorage

There are two new objects for storing data on the client:

localStorage - stores data with no expiration date
sessionStorage - stores data for one session

Before using web storage, check browser support for localStorage and session Storage:

```
if(typeof(Storage)!=="undefined")
  {
  // Yes! localStorage and sessionStorage support!
  // Some code.....
  }
else
  {
  // Sorry! No web storage support..
  }
```

The localStorage Object

The localStorage object stores the data with no expiration date. The data will not be deleted when the browser is closed, and will be available the next day, week, or year.

Example

```
localStorage.lastname="Smith";
document.getElementById("result").innerHTML="Last name: "
+ localStorage.lastname;
```

```
<!DOCTYPE html>
<html>
<body>
<div id="result"></div>
<script>
if(typeof(Storage)!=="undefined")
  {
  localStorage.lastname="Smith";
  document.getElementById("result").innerHTML="Last name: " + localStorage.lastname;
  }
```

Example explained:

Create a localStorage key/value pair with key="lastname" and value="Smith"

Retrieve the value of the "lastname" key and insert it into the element with id="result"

```
else
  {
  document.getElementById("result").innerHTML="Sorry, your browser does not support web
storage...";
  }
</script>
</body>
</html>
```

```
<!DOCTYPE html>
<html>
<body>
<div id="result"></div>
<script>
```

```
if(typeof(Storage)!=="undefined")
  {
  localStorage.lastname="Smith";
  document.getElementById("result").innerHTML="Last name: " + localStorage.lastname;
  }
else
  {
  document.getElementById("result").innerHTML="Sorry, your browser does not support web
storage...";
  }
</script>
</body>
</html>
```

Example explained:
Create a localStorage key/value pair with key="lastname" and value="Smith"
Retrieve the value of the "lastname" key and insert it into the element with id="result"

The following example counts the number of times a user has clicked a button. In this code the value string is converted to a number to be able to increase the counter:
Example

```
if (localStorage.clickcount)
  {
  localStorage.clickcount=Number(localStorage.clickcount)+1;
  }
else
  {
  localStorage.clickcount=1;
  }
document.getElementById("result").innerHTML="You have clicked the button " +
localStorage.clickcount + " time(s).";
```

```
<!DOCTYPE html>
<html>
<head>
<script>
function clickCounter()
{
if(typeof(Storage)!=="undefined")
  {
  if (localStorage.clickcount)
    {
    localStorage.clickcount=Number(localStorage.clickcount)+1;
    }
  else
```

```
  {
  localStorage.clickcount=1;
  }
 document.getElementById("result").innerHTML="You have clicked the button " +
localStorage.clickcount + " time(s).";
 }
else
 {
 document.getElementById("result").innerHTML="Sorry, your browser does not support web
storage...";
 }
}
</script>
</head>
<body>
<p><button onclick="clickCounter()" type="button">Click me!</button></p>
<div id="result"></div>
<p>Click the button to see the counter increase.</p>
<p>Close the browser tab (or window), and try again, and the counter will continue to count (is not
reset).</p>
</body>
</html>
```

The sessionStorage Object

The sessionStorage object is equal to the localStorage object, except that it stores the data for only one session. The data is deleted when the user closes the browser window.

The following example counts the number of times a user has clicked a button, in the current session:

Example

```
if (sessionStorage.clickcount)
 {
 sessionStorage.clickcount=Number(sessionStorage.clickcount)+1;
 }
else
 {
 sessionStorage.clickcount=1;
 }
document.getElementById("result").innerHTML="You have clicked the button " +
sessionStorage.clickcount + " time(s) in this session.";
```

Example

```
<!DOCTYPE html>
<html>
<head>
```

```
<script>
function clickCounter()
{
if(typeof(Storage)!=="undefined")
  {
  if (sessionStorage.clickcount)
    {
    sessionStorage.clickcount=Number(sessionStorage.clickcount)+1;
    }
  else
    {
    sessionStorage.clickcount=1;
    }
  document.getElementById("result").innerHTML="You have clicked the button " +
sessionStorage.clickcount + " time(s) in this session.";
  }
else
  {
  document.getElementById("result").innerHTML="Sorry, your browser does not support web
storage...";
  }
}
</script>
</head>
<body>
<p><button onclick="clickCounter()" type="button">Click me!</button></p>
<div id="result"></div>
<p>Click the button to see the counter increase.</p>
<p>Close the browser tab (or window), and try again, and the counter is reset.</p>
</body>
</html>
```

HTML5 Application Cache

With HTML5 it is easy to make an offline version of a web application, by creating a cache manifest file.

What is Application Cache?

HTML5 introduces application cache, which means that a web application is cached, and accessible without an internet connection.

Application cache gives an application three advantages:

- Offline browsing - users can use the application when they're offline
- Speed - cached resources load faster

- Reduced server load - the browser will only download updated/changed resources from the server

Browser Support:Internet Explorer 10, Firefox, Chrome,Safari and Opera support Application cache.

HTML5 Cache Manifest Example

The example below shows an HTML document with a cache manifest (for offline browsing):
Example:

```
<!DOCTYPE HTML>
<html manifest="demo.appcache">
<body>
The content of the document.....
</body>
</html>

<!DOCTYPE html>
<html manifest="demo_html.appcache">
<body>
<script src="demo_time.js">
</script>
<p id="timePara"><button onclick="getDateTime()">Get Date and Time</button></p>
<p><img src="img_logo.gif" width="336" height="69"></p>
<p>Try opening <a href="tryhtml5_html_manifest.htm" target="_blank">this page</a>, then go offline, and reload the page. The script and the image should still work.</p>
</body>
</html>
```

Cache Manifest Basics
To enable application cache, include the manifest attribute in the document's <html> tag:

```
<!DOCTYPE HTML>
<html manifest="demo.appcache">
...
</html>
```

Every page with the manifest attribute specified will be cached when the user visits it. If the manifest attribute is not specified, the page will not be cached (unless the page is specified directly in the manifest file).
The recommended file extension for manifest files is: ".appcache"

The Manifest File

The manifest file is a simple text file, which tells the browser what to cache (and what to never cache).

The manifest file has three sections:

CACHE MANIFEST - Files listed under this header will be cached after they are downloaded for the first time

NETWORK - Files listed under this header require a connection to the server, and will never be cached

FALLBACK - Files listed under this header specifies fallback pages if a page is inaccessible

CACHE MANIFEST

The first line, CACHE MANIFEST, is required:

```
CACHE MANIFEST
/theme.css
/logo.gif
/main.js
```

The manifest file above lists three resources: a CSS file, a GIF image, and a JavaScript file. When the manifest file is loaded, the browser will download the three files from the root directory of the web site. Then, whenever the user is not connected to the internet, the resources will still be available.

NETWORK

The NETWORK section below specifies that the file "login.asp" should never be cached, and will not be available offline:

```
NETWORK:
login.asp
```

An asterisk can be used to indicate that all other resources/files require an internet connection:

```
NETWORK:
*
```

FALLBACK

The FALLBACK section below specifies that "offline.html" will be served in place of all files in the /html/ catalog, in case an internet connection cannot be established:

```
FALLBACK:
/html/ /offline.html
```

Note: The first URI is the resource, the second is the fallback.

Updating the Cache

Once an application is cached, it remains cached until one of the following happens:

- The user clears the browser's cache
- The manifest file is modified (see tip below)
- The application cache is programmatically updated

With HTML5 it is easy to make an offline version of a web application, by creating a cache manifest file.

What is Application Cache?

HTML5 introduces application cache, which means that a web application is cached, and accessible

without an internet connection.

Application cache gives an application three advantages:

- Offline browsing - users can use the application when they're offline
- Speed - cached resources load faster
- Reduced server load - the browser will only download updated/changed resources from the server

Browser Support

Internet Explorer 10, Firefox, Chrome, Safari and Opera support Application cache.
HTML5 Cache Manifest Example

The example below shows an HTML document with a cache manifest (for offline browsing):
Example

```
<!DOCTYPE HTML>
<html manifest="demo.appcache">

<body>
The content of the document.....
</body>

</html>
```

Cache Manifest Basics

To enable application cache, include the manifest attribute in the document's <html> tag:

```
<!DOCTYPE HTML>
<html manifest="demo.appcache">
...
</html>
```

Every page with the manifest attribute specified will be cached when the user visits it. If the manifest attribute is not specified, the page will not be cached (unless the page is specified directly in the manifest file).

The recommended file extension for manifest files is: ".appcache"
Note A manifest file needs to be served with the correct MIME-type, which is "text/cache-manifest". Must be configured on the web server.

The Manifest File

The manifest file is a simple text file, which tells the browser what to cache (and what to never cache).

The manifest file has three sections:

CACHE MANIFEST - Files listed under this header will be cached after they are downloaded for the first time

NETWORK - Files listed under this header require a connection to the server, and will never be cached

FALLBACK - Files listed under this header specifies fallback pages if a page is inaccessible

CACHE MANIFEST

The first line, CACHE MANIFEST, is required:

```
CACHE MANIFEST
/theme.css
/logo.gif
/main.js
```

The manifest file above lists three resources: a CSS file, a GIF image, and a JavaScript file. When the manifest file is loaded, the browser will download the three files from the root directory of the web site. Then, whenever the user is not connected to the internet, the resources will still be available.

NETWORK

The NETWORK section below specifies that the file "login.asp" should never be cached, and will not be available offline:

```
NETWORK:
login.asp
```

An asterisk can be used to indicate that all other resources/files require an internet connection:

```
NETWORK:
*
```

FALLBACK

The FALLBACK section below specifies that "offline.html" will be served in place of all files in the /html/ catalog, in case an internet connection cannot be established:

```
FALLBACK:
/html/ /offline.html
```

Note: The first URI is the resource, the second is the fallback.

Updating the Cache

Once an application is cached, it remains cached until one of the following happens:

- The user clears the browser's cache
- The manifest file is modified (see tip below)
- The application cache is programmatically updated

Example - Complete Cache Manifest File

```
CACHE MANIFEST
```

#2012-02-21 v1.0.0
/theme.css
/logo.gif
/main.js

NETWORK:
login.asp

FALLBACK:

/html/ /offline.html

Notes on Application Cache
Be careful with what you cache.
Once a file is cached, the browser will continue to show the cached version, even if you change the file on the server. To ensure the browser updates the cache, you need to change the manifest file.

HTML5 Web Workers

A web worker is a JavaScript running in the background, without affecting the performance of the page.

What is a Web Worker?
When executing scripts in an HTML page, the page becomes unresponsive until the script is finished.

A web worker is a JavaScript that runs in the background, independently of other scripts, without affecting the performance of the page. You can continue to do whatever you want: clicking, selecting things, etc., while the web worker runs in the background.

Browser Support:Internet Explorer 10, Firefox, Chrome, Safari and Opera support Web workers.

HTML5 Web Workers Example

The example below creates a simple web worker that count numbers in the background:
Example

```
<!DOCTYPE html>
<html>
<body>
<p>Count numbers: <output id="result"></output></p>
<button onclick="startWorker()">Start Worker</button>
<button onclick="stopWorker()">Stop Worker</button>
<br><br>
<script>
var w;
```

```
function startWorker()
{
if(typeof(Worker)!=="undefined")
  {
  if(typeof(w)=="undefined")
  {
  w=new Worker("demo_workers.js");
  }
  w.onmessage = function (event) {
    document.getElementById("result").innerHTML=event.data;
    };
  }
else
  {
  document.getElementById("result").innerHTML="Sorry, your browser does not support Web
Workers...";
  }
}

function stopWorker()
{
w.terminate();
}
</script>
</body>
</html>
```

Check Web Worker Support

Before creating a web worker, check whether the user's browser supports it:

```
if(typeof(Worker)!=="undefined")
  {
  // Yes! Web worker support!
  // Some code.....
  }
else
  {
  // Sorry! No Web Worker support..
  }
```

Create a Web Worker File

Now, let's create our web worker in an external JavaScript.

Here, we create a script that counts. The script is stored in the "demo_workers.js" file:

```
var i=0;
```

```
function timedCount()
{
i=i+1;
postMessage(i);
setTimeout("timedCount()",500);
}

timedCount();
```

The important part of the code above is the postMessage() method - which is used to post a message back to the HTML page.

Note: Normally web workers are not used for such simple scripts, but for more CPU intensive tasks.
Create a Web Worker Object

Now that we have the web worker file, we need to call it from an HTML page.

The following lines checks if the worker already exists, if not - it creates a new web worker object and runs the code in "demo_workers.js":

```
if(typeof(w)=="undefined")
  {
  w=new Worker("demo_workers.js");
  }
```

Then we can send and receive messages from the web worker.

Add an "onmessage" event listener to the web worker.

```
w.onmessage=function(event){
document.getElementById("result").innerHTML=event.data;
};
```

When the web worker posts a message, the code within the event listener is executed. The data from the web worker is stored in event.data.
Terminate a Web Worker

When a web worker object is created, it will continue to listen for messages (even after the external script is finished) until it is terminated.

To terminate a web worker, and free browser/computer resources, use the terminate() method:

```
w.terminate();
```

Full Web Worker Example Code

We have already seen the Worker code in the .js file. Below is the code for the HTML page:

Example

```
<!DOCTYPE html>
<html>
<body>
<p>Count numbers: <output id="result"></output></p>
<button onclick="startWorker()">Start Worker</button>
<button onclick="stopWorker()">Stop Worker</button>
<br><br>
<script>
var w;
function startWorker()
{
if(typeof(Worker)!=="undefined")
{
  if(typeof(w)=="undefined")
    {
    w=new Worker("demo_workers.js");
    }
  w.onmessage = function (event) {
    document.getElementById("result").innerHTML=event.data;
  };
}
else
{
document.getElementById("result").innerHTML="Sorry, your browser does not support Web
Workers...";
}
}

function stopWorker()
{
w.terminate();
}
</script>
</body>
</html>

<!DOCTYPE html>
<html>
<body>
<p>Count numbers: <output id="result"></output></p>
<button onclick="startWorker()">Start Worker</button>
<button onclick="stopWorker()">Stop Worker</button>
<br><br>
<script>
var w;
```

```
function startWorker()
{
if(typeof(Worker)!=="undefined")
  {
  if(typeof(w)=="undefined")
  {
  w=new Worker("demo_workers.js");
  }
  w.onmessage = function (event) {
    document.getElementById("result").innerHTML=event.data;
    };
  }
else
  {
  document.getElementById("result").innerHTML="Sorry, your browser does not support Web
Workers...";
  }
}
function stopWorker()
{
w.terminate();
}
</script>
</body>
</html>

<!DOCTYPE html>
<html>
<body>
<p>Count numbers: <output id="result"></output></p>
<button onclick="startWorker()">Start Worker</button>
<button onclick="stopWorker()">Stop Worker</button>
<br><br>
<script>
var w;
function startWorker()
{
if(typeof(Worker)!=="undefined")
  {
  if(typeof(w)=="undefined")
  {
  w=new Worker("demo_workers.js");
  }
  w.onmessage = function (event) {
    document.getElementById("result").innerHTML=event.data;
    };
```

```
  }
else
  {
  document.getElementById("result").innerHTML="Sorry, your browser does not support Web
Workers...";
  }
}

function stopWorker()
{
w.terminate();
}
</script>
</body>
</html>
```