# HTML 5

Lakshman M N
Tech Evangelist / Mentor
Lakshman.mn@gmail.com

## What is HTML5?

- Next generation of HTML superseding HTML 4.01, XHTML 1.0, XHTML 1.1
- Standardizes features of the web platform *(window object has never been formally documented)*
- Designed to be cross-platform like its predecessors.
- Latest versions of Safari, Opera, Firefox, Chrome support ***many*** HTML5 features. *(IE 9 will support some HTML5 functionality)*

## Design Principles: Compatibility

- Support Existing Content
- Degrade Gracefully
- Don't Reinvent the Wheel
- Evolution, not Revolution

## Design Principles: Utility

- Solve Real Problems
- Media Independence
- Universal Access
- Support World Languages
- Secure By Design

## Design Principles: Interoperability

- Well-Defined Behaviour
- Avoid Needless Complexity
- Handle Errors

## HTML5 Detection Library

- Modernizr (http://www.modernizr.com)
  - An open source, MIT-licensed JavaScript library
  - Detects support for many HTML5 and CSS3 features
  - Runs automatically
  - Creates a global object called **`Modernizr`** that contains a set of Boolean properties for each feature it can detect.

ModernizrDemo.htm

# Forms

# Forms

- HTML4 supports form controls, some of them implemented using the <Input> element.
- HTML5 defines quite a few input types that can be used in forms.
- Only a few browsers currently support these features.
  - Firefox, Safari, Chrome, Opera

# Placeholder Text

- This provides the ability to set placeholder text in an input field.
- It is displayed in the field as long as the field is empty and not focused.

```
<form>
   <label for="txtName">Name : </label>
   <input id="txtName"
          placeholder="Enter Name"/>
       <input type="submit" value="Check"/>
</form>
```

Name : Enter Name Check

HTMLForm01.htm

# Autofocus Fields

- JavaScript has been the choice to focus an input field on a form.
- HTML5 introduces an autofocus attribute on all form controls.
- Unlike scripts this is markup and therefore will be consistent across all sites.

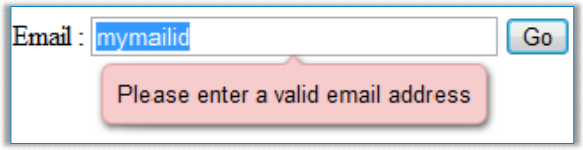
```
<form>
   <label for="txtName">Name : </label>
   <input id="txtName" type="text"

     autofocus/>
    <input type="submit" value="Check"/>
</form>
```

HTMLForm02.htm

# Email addresses

- "email" is regarded as one the types of input in HTML5.

```
<form>
    <label for="txtEMail">EMail : </label>
    <input id="txtEmail"
                        type="email"/>
    <input type="submit" value="Go"/>
</form>
```

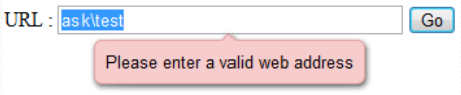



HTMLForm03.htm

# Web Addresses

- The syntax of a web address is constrained by relevant Internet standards.
- "url" is one of the additions in HTML5.

```
<form>
        <label for="txtURL">URL : </label>
        <input name="txtURL" type="url"/>
        <input type="submit" value="Go"/>
</form>
```




HTMLForm04.htm

# Dealing with Numbers

- Numbers can be trickier than email or web addresses since we may need them in a range.
- We may need numbers of a certain kind in a range.
- HTML5 caters to these numbering needs!

```
<input type="number" min="0" max="10"
                    step="2" value="4"/>
```


HTMLForm05.htm

# Dealing with Numbers…

- Slider controls can be used in forms.
- The type of input is "range".
- The available attributes are the same as those for type="number".
- The difference is in the UI.

```
<input type="range" min="0" max="10"
              step="2" value="4"/>
```

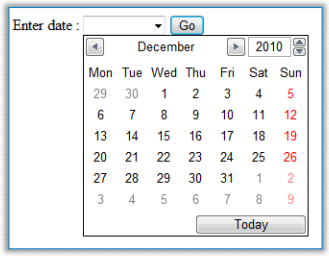

HTMLForm06.htm

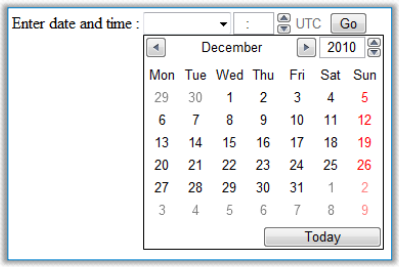
# Date Pickers

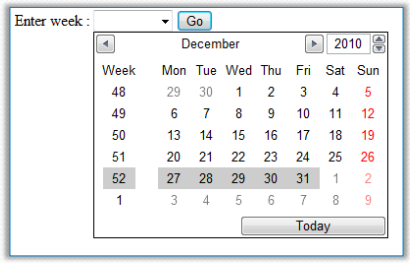
- Date picker control was sorely lacking in HTML4.
  - This was worked around with the help of JavaScript frameworks
- HTML5 defines a way to include a native date picker.
- Options include
  - date, month, week, time, date + time

# Date Pickers…

```
<input type="date"/>
<input type="datetime"/>
<input type="week"/>
<input type="time"/>
<input type="month"/>
```



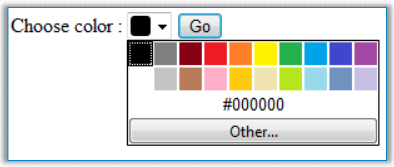




HTMLForm07.htm

# Color Picker

- HTML5 defines an <input> element for color.
- It helps pick a color and returns the hexadecimal representation of the chosen color.

```
<input type="color"/>
```




HTMLForm08.htm

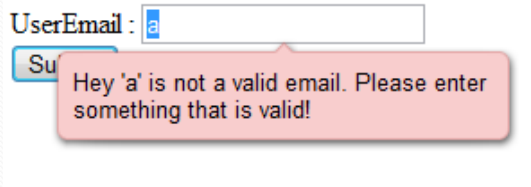
# Required

- "required" attribute in HTML5 makes a field mandatory.
- <input type="text" required>


HTMLForm10.htm

# Custom Validation Messages

- HTML5 shows validation messages in line with the field which fails validation.
- Custom messages can be shown instead of standard messages using **constraint validation API**
- The validation message can be set using **`element.setCustomValidity(message)`**

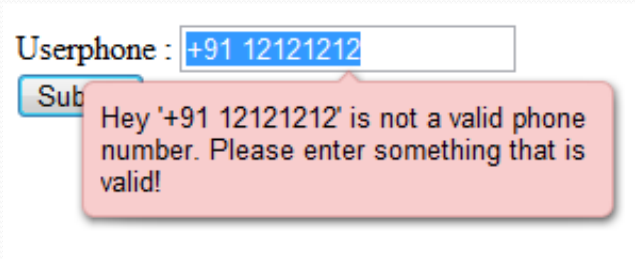



HTMLForm11.htm

# Pattern

- The **`Pattern`** attribute is used for field validation and accepts values if they match a specific format.

```
<input type="text" name="Tel"
 pattern="[+]\d{1,3} \d{3}-\d{8}" />
```



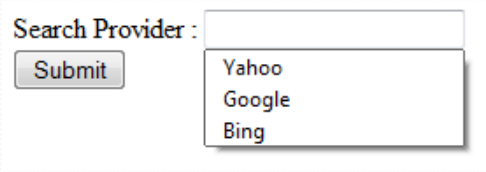

HTMLForm12.htm

# DataList

- **`<datalist>`** specifies a list of pre-defined options for an **`<input>`** element.
- **`<datalist>`** can be used to provide a drop down from a text input. (auto-complete)
- The **`list`** attribute of the **`<input>`** element can be used to bind it with a **`<datalist>`** element.

# DataList…

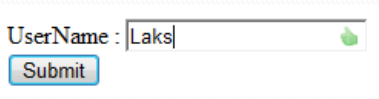
```
<input type="text" list="search_list"/>
<datalist id="search_list">
   <option value="http://www.yahoo.com"
  label="Yahoo"/>
   <option value="http://www.google.com"
  label="Google"/>
   <option value="http://www.bing.com" label="Bing"/
  >
</datalist>
```




HTMLForm13.htm

# Styling inputs using CSS

- `input:required:invalid, input:focus:invalid`
  - Apply to inputs that are required but empty or
  - To inputs that have a required format that has not yet been met
- `input:required:valid`
  - Apply to inputs that are both required and valid


HTMLForm14.htm

# Communication APIs

# Background

- Communication between frames, tabs and windows in a running browser has been restricted due to security concerns.
- It may open up the possibility for malicious attacks.
- Information could be stolen if programmatic access to content across tabs or frames is permitted.

# Background…

- There are legitimate cases for communication.
- "Mashups" are a combination of different applications such as messaging and news from different sites.
- These applications can be combined together to form a new meta-application.
- These applications would be served by direct communication channels inside the browser.

# Cross Document Messaging

- A new feature, Cross Document Messaging enables secure cross-origin communication across
  - iframes
  - Tabs
  - Windows
- It defines the **`postMessage`** API as a standard way to send messages.

# postMessage

```
otherWindow.postMessage(message,targetOrigin);
```

- **`otherWindow`**
  - A reference to another window; such a reference may be obtained
    - Using the **`contentWindow`** property of an iframe element
    - The object returned by **`window.open`**
    - By named or numeric index on **`window.frames`**
- **`message`**
  - String data to be sent to the other window.
- **`targetOrigin`**
  - Specifies what the origin of the **`otherWindow`** must be for the event to be dispatched, either as
    - a literal string "*" (indicating no preference)
    - Or a URI

# The dispatched event

- **otherWindow** can listen for dispatched messages by executing a script

```
window.addEventListener("message",receiveMessage,true);
function receiverMessage(event)
{
  if (event.origin != "http://mysite.org")
  return;
}
```

- **data**
  - A string holding the message passed from the other window
- **origin**
  - The origin of the window that sent the message at the time **postMessage** was called.
- **source**
  - A reference to the window object that sent the message
    - Used to establish two-way communication between the two windows.

http://localhost/cdm/postmessage_main.htm
http://localhost/cdm/postmessage_rcv.htm

# Origin Security

- An origin is a subset of an address used for modeling trust relationships on the web.
- Origins are made up of a scheme, a host and a port.
- Example
  - A page at https://www.mysite.com has a different origin than one at http://www.mysite.com because the scheme differs (https vs http)
- The path is not considered in the origin value.
  - A page at http://www.mysite.com/index.htm has the same origin as a page at http://www.mysite.com/page2.htm as only the paths differ.

# Origin Security…

- Security rules for **`postMessage`** ensure that messages cannot be delivered to pages with unexpected origins.
- When sending a message, the sender specifies the receiver's origin.
- When receiving a message, the sender's origin is included as a part of the message.
  - The message's origin is provided by the browser and cannot be spoofed.
- This allows the receiver to decide which messages to process and which ones to ignore.

# XMLHttpRequest

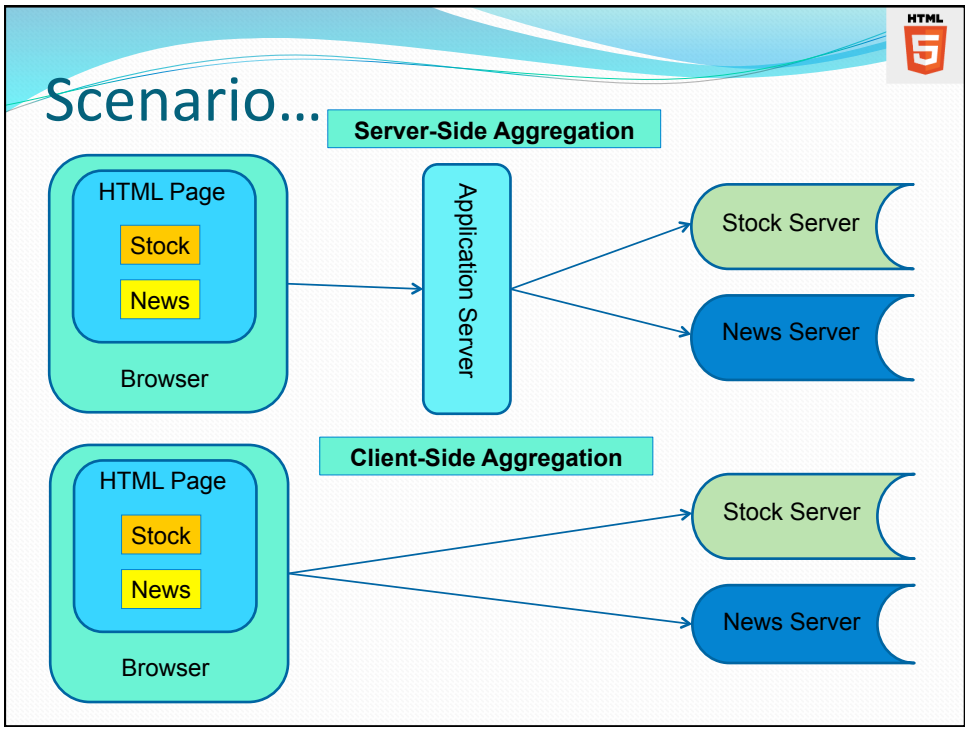
- XMLHttpRequest is the API that made Ajax possible.
- It is a JavaScript object that interacts with a Web server in order to submit or retrieve information in the background.

# Limitations

- The lack of cross-browser solutions that enables retrieval of multiple responses from the HTTP server for the same AJAX request.
- XMLHttpRequest cannot upload files or communicate with different domain names.

# Cross Origin XMLHttpRequest

- XMLHttpRequest Level 2 allows for cross-origin XMLHttpRequests using Cross Origin Resource Sharing (CORS)
- CORS uses the origin concept as in Cross Document Messaging.
- Cross-origin HTTP requests have an Origin header.
- The header provides the server with the request's origin.
  - The header is protected by the browser and cannot be changed.




# Headers

- ***Request header***

```
POST /main HTTP/1.1
Host: www.mysite.com
User-Agent: Mozilla 5.0
Referer: http://www.mysite.com/
Origin: http://www.mysite.com
Cache-Control: no-cache
```

- ***Response header***

```
HTTP 1.1 201 Created
Transfer-Encoding: chunked
Date: Mon, 24 Jan 2011 07:00:00 GMT
Content-type: text/plain
Access-Control-Allow-Origin: http://www.mysite.com
Access-Control-Allow-credentials: true
```

# Progress Events

- Previously XMLHttpRequest supported only a **`readystatechanged`** event.
- XMLHttpRequest Level 2 introduces progress events with meaningful nomenclature
  - **`loadstart`**
  - **`progress`**
  - **`abort`**
  - **`error`**
  - **`load`**
  - **`loadend`**
- The old **`readyState`** property and the **`readystatechange`** events will be retained for backward compatibility.

http://localhost/CDM/XHR-ProgressEvents.htm

# Checking for Browser support

- XMLHttpRequest Level 2 has varying levels of browser support.
- It is good to check for XMLHttpRequest Level 2 support before using its functionality.
- Accomplished by checking whether the new **`withCredentials`** property is available on an XMLHttpRequest object.

# Checking Browser support

```
var xhr = new XMLHttpRequest();
if (typeof xhr.withCredentials === undefined)
 {
   document.getElementById("support").innerHTML  =
  "Your browser <b>does not</b> support cross-origin
  XMLHttpRequest";
 }
else
{
  document.getElementById("support").innerHTML =
  "Your browser <b>does</b> support cross-origin
  XMLHttpRequest";
}
```

CheckXHR-Support.htm
http://localhost/CDM/CrossOriginXHR.htm
http://localhost/CDM/CrossOriginXHR-Events.htm

# Server Sent Events

# Polling

- A traditional technique used by a majority of AJAX applications.
- The application repeatedly polls a server for data.
- Fetching data revolves around a request/response format.
- Client makes a request and waits for the server to respond with data.
- In case of no data an empty response is returned.
- *Extra polling creates HTTP overhead.*

# Long Polling

- A variation of polling in which if the server does not have data available, it holds the request open until new data is available.
- This technique is also known as "Hanging GET".
- When information is available, the server responds, closes the connection.
- The effect is that the server is constantly responding with new data as it becomes available.

# Server-Sent Events

- Server-Sent Events have been designed from scratch.
- When communicating using SSEs, a server can push data to the application whenever it wants.
- This does not require an initial request.
- Updates can be streamed from the server to the client as they happen.
- SSEs open a uni-directional channel between the server and client.
- Unlike long-polling, SSEs are handled directly by the browser.

# The API

- To subscribe to a new event stream, start by creating a new EventSource object and pass in the entrypoint

```
var source = new
  EventSource("myEvents.php");
```

- The referenced URL must be on the same origin (scheme, domain and port) as the page in which the object is created.

# The API…

- The **`EventSource`** instance has a **`readyState`** property with values
  - 0 : indicates it is connecting to the server
  - 1 : indicates an open connection
  - 2 : indicates a closed connection
- Three events are associated with the **`EventSource`**
  - **`open`** : fired when the connection is established
  - **`message`** : fired when a new event is received from the server
  - **`error`** : fired when no connection can be made

# The API…

```
source.addEventListener("message",getData,false);

function getData(e)
{
  var data = e.data;
}
```

- Information sent back from the server is returned via **`event.data`** as a string.
- The **`EventSource`** object will attempt to keep the connection alive with the server.
- The object can be forced to disconnect immediately by calling the **`close`**() method.

```
source.close();
```

# The event stream

- Server events are sent along a long-lasting HTTP request with a MIME type of **`text/event-stream`**
- The format of the response is plain text.
- It is made up of the prefix **`data`**: followed by text.
- When there are two or more consecutive lines beginning with data:
  - it is interpreted as a multiline piece of data
  - The values are concatenated with a newline character.

```
data: sometext
data: somemoretext
```

# The event stream

- The message event is never fired until a blank line is encountered after a line containing **`data`**:
- An ID can be associated with a particular event by including and **`id`**: line before or after the **`data`**:
- With the ID, the **`EventSource`** object keeps track of the last event fired.
- If the connection is dropped
  - A special HTTP header called **`Last-Event-ID`** is sent along with the request
  - The server can determine which event is appropriate to fire next.

http://localhost/CDM/ServerEvents.htm
http://localhost/CDM/myEvents.php

## Summary

- SSEs are sent over traditional HTTP.
- SSEs are handled directly by the browser.
- SSEs provide features such as automatic reconnection, eventIDs and the ability to send arbitrary events.

## HTML5 Web Sockets

# Today's Requirements

- Today's Web applications demand reliable, real-time communications with near-zero latency
- Not just broadcast, but bi-directional communication
- Examples:
  - Financial applications
  - Social networking applications
  - Online games
  - Smart power grid

# About HTTP

- HTTP was originally designed for document transfer
- Until now, it has been cumbersome to achieve real-time, bi-directional web communication due to the limitations of HTTP
- HTTP is half-duplex (traffic flows in only one direction at a time)
- Header information is sent with each HTTP request and response, which can be an unnecessary overhead.

# Real-time and HTTP

- Current attempts to provide real-time web applications largely use polling and server-side push.
- Notable is "Comet", which delays the completion of a HTTP response to deliver messages to the client.
- In streaming,
  - the browser sends a complete request,
  - The server sends and maintains an open response that is continuously updated
  - The response is updated whenever a message is ready to be sent.

# HTML5 WebSocket

- W3C API and IETF Protocol
- Full-duplex text-based socket
- Enables web pages to communicate with a remote host
- Traverses firewalls, proxies, and routers seamlessly
- Leverages Cross-Origin Resource Sharing (CORS)
- Share port with existing HTTP content (80/443)

# WebSockets

- WebSockets provide bi-directional, full-duplex communication channels over a single TCP socket.
- HTML5 WebSockets provide an enormous reduction in unnecessary network traffic and latency.
- WebSockets account for network hazards like proxies and firewalls, making streaming possible over any connection.
- WebSocket-based applications place less-burden on servers .

# Basic WebSocket-based architecture

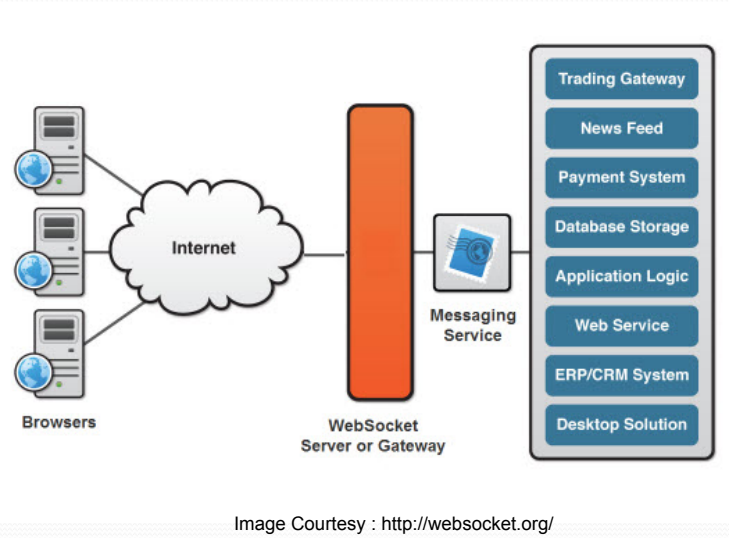



Image Courtesy : http://websocket.org/

# The WebSocket Handshake

- To establish a WebSocket connection, the client and the server upgrade from HTTP to the WebSocket protocol during their initial handshake.
- This process automatically sets up a tunnel through to the server
- Once established, the WebSocket is a full duplex channel between the client and the server.

# The WebSocket Handshake

**From client to server:**
GET /demo HTTP/1.1
Host: example.com
Connection: Upgrade
Sec-WebSocket-Key2: 12998 5 Y3 1 .P00
Sec-WebSocket-Protocol: sample
Upgrade: WebSocket
Sec-WebSocket-Key1: 4@1 46546xW%01 1 5
Origin: http://example.com

[8-byte security key]

**From server to client:**
HTTP/1.1 101 WebSocket Protocol Handshake
Upgrade: WebSocket
Connection: Upgrade
WebSocket-Origin: http://example.com
WebSocket-Location: ws://example.com/demo
WebSocket-Protocol: sample

[16-byte hash response]

# The WebSocket API

In order to use the WebSocket interface,

- Create a new WebSocket instance providing the new object with a URL representing the end-point

```
var myWebSocket = new
          WebSocket("ws://example.com");
```

- URL Scheme
  - The WebSocket protocol specifications defines two URI schemes
    - **ws:** - for unencrypted connections
    - **wss:** - for encrypted connections

# Check for Browser Support

- In order to use the HTML5 WebSocket API, browser support needs to be ensured.

```
if(window.WebSocket)
{
"supported"
}
else
{
"Not Supported"
}
```

CheckWebSocket-Support.htm

# WebSocket API…

- Once a connection is established WebSocket data frames can be sent back and forth between the client and server.
- Before connecting to an end-point and sending a message, event listeners can be associated to handle each phase

```
myWebSocket.addEventListener("open",openConn,false);
myWebSocket.addEventListener("message",getData,false);
myWebSocket.addEventListener("close",closeConn,false);
```

# Sending Messages

- To send a message to the server, call the **`send()`** method and provide the content as the argument.
- After sending the message, optionally invoke the **`close()`** to terminate the connection.

```
myWebSocket.send("Hello WebSocket World");
myWebSocket.close();
```

WebSocket.htm

## Securing WebSocket Traffic

- WebSocket defines the `ws://` and `wss://` schemes
- WSS is WS over TLS (Transport Layer Security), formerly known as SSL (Secure Socket Layer) support (similar to HTTPS)
- An HTTPS connection is established after a successful TLS handshake (using public and private key certificates)
- HTTPS is not a separate protocol, but it is HTTP running on top of a TLS connection (default port is 443)

## Server-Sent Events vs. WebSockets

- WebSockets are bi-directional while Server-Sent Events are not.
- Server-Sent Events are sent over plain old HTTP without any modification.
- WebSockets require new WebSocket servers to handle the protocol.
- SSEs have features such as automatic reconnection, event IDs that WebSockets lack.

## SSEs vs. WebSockets

- A two-way channel as in a WebSocket is more useful for applications like games, messaging apps etc.
- Cases where data does not need to be sent from the client as in SSEs include friend's status updates, stock tickers, news feeds etc.

## Summary

- WebSockets simplify authoring interactive real-time web applications.
- WebSocket API is simple to understand and use.
- WebSockets fit well into the existing infrastructure as they use the same ports as standard HTTP.
- The default port for WebSocket is 81 and the default port for secure WebSocket is 815.

# Web Workers

# JavaScript

- A number of limitations prevent interesting applications from being ported to client-side JavaScript.
- Some of these constraints include
  - Browser compatibility
  - Accessibility
  - Performance

# JavaScript Concurrency

- One significant hindrance for JavaScript is the single-threaded environment.
  - Multiple scripts cannot run concurrently
- "Concurrency" is mimicked by using techniques like setTimeout() and event handlers.
- Asynchronous events are processed after the current executing script has yielded.

# Web Workers

- HTML5 Web Workers provide background processing capabilities to web applications.
- They typically run on separate threads so that JavaScript applications can take advantage of multi-core CPUs.
- Web Workers can be used to handle computationally intensive tasks without blocking the UI.
- They are ideal for UI related tasks, performant and responsive to users.

## Use Cases

- Applications that could utilize Web Workers
  - Spell Checker
  - Background I/O or polling of Web services
  - Analyzing data
  - Code syntax highlighting

## Stepping In

- Web Workers run in an isolated thread.
- The code they execute needs to be contained in a separate file.
- Create a new Worker object in the main page.

```
var worker = new
            Worker("task1.js");
```

## Stepping In…

- If the specified file exists, the browser will spawn a new worker thread.
  - The file is downloaded asynchronously
  - The worker will not begin execution until the file has completely downloaded.
  - If the path to the worker returns 404, the worker will fail silently.
- The worker is started by calling the **postMessage**() method

```
worker.postMessage();
```

## Check for browser support

- Before using the Web Workers API functions, browser support for the same needs to be ascertained.

```
function loadDemo()
{
  if (typeof(Worker) != "undefined")
  {
     "Supports Web Workers";
  }
}
```

WebWorker01.htm

# Communicating with a Worker

- Communication between a worker and its parent is accomplished using an event model and the **`postMessage()`** method.

```
myworker.postMessage("Hello Web Worker!");
```

- An event listener is added to listen for messages from the Web Worker.

```
myworker.addEventListener("message",workerMsg,false);
function workerMsg(e)
 {
   document.getElementById("divInfo").innerHTML +=
                        "Worker said : " + e.data;
 }
```

# Communicating with a Worker

- The Web Worker JavaScript file must be setup to process incoming messages.

```
self.addEventListener("message",handler,true
 )
```

- *Messages passed between the main page and workers are copied, not shared.*
- The handler for the event

```
function handler(e)
{
  self.postMessage("worker says : " +
 e.data);
}
```

WebWorker02.htm
dotask.js

# Stopping Web Workers

- Web Workers don't stop by themselves; but the page that started them can stop them.
- Calling **`worker.terminate()`** stops the Web Worker.
- A terminated Web Worker will no longer respond to messages.
- A Web Worker cannot be restarted.

WebWorker03.htm
dotask1.js

# Features available to Workers

- Due to their multi-threaded behaviour, Web Workers only have access to a subset of JavaScript's features
  - **`navigator`** object
  - **`location`** object (read-only)
  - **`XMLHttpRequest`**
  - **`setTimeout() / clearTimeout()`**

## Features not available to Workers

- Workers do NOT have access to
  - The DOM (it's not thread-safe)
  - **`window`** object
  - **`document`** object
  - **`parent`** object

## Summary

- Web Workers can be used to create Web applications with background processing.
- Web Worker APIs help create new workers and facilitate communication between the worker and its context.

# HTML5 Geolocation

## You are Here!

- Geolocation is the art of figuring out our position in the world and optionally sharing that information.
- HTML5 Geolocation is an API that allows users to share their location with web applications.
- This facilitates location-aware services.

# Location Information - Sources

- A device can use any of the following sources
  - IP address
  - Coordinate triangulation
    - GPS
    - Wi-Fi with MAC addresses from RFID, Bluetooth
    - GSM or CDMA cell IDs
  - User defined

# IP Address Geolocation Data

- Previously, IP address-based geolocation was the only way to get a possible location.
- IP address-based geolocation works by looking up a user's IP address and then retrieving the registrant's physical address.
- **Pros**
  - Available everywhere
  - Processed on the server side
- **Cons**
  - Not very accurate
  - Costly operation

# GPS Geolocation Data

- GPS provides accurate location as long as there is line of sight with the satellites.
- A GPS fix is acquired by acquiring the signal from multiple GPS satellites.
- It can take a while to get a fix, therefore this task can be asynchronous
- **Pros**
  - Very accurate
- **Cons**
  - Takes a while and consumes power
  - Does not work well indoors
  - Additional hardware

# Wi-Fi Geolocation Data

- The information is acquired by triangulating the location based on the user's distance from a number of known Wi-Fi access points.
- **Pros**
  - Accurate
  - Works indoors
  - Can get a fix quickly
- **Cons**
  - Not good in rural areas

# Cell Phone Geolocation Data

- Information is acquired by triangulating the location based on user's distance from a number of cell phone towers.
- The location result is fairly accurate.
- This method is used in combination with Wi-Fi and GPS based geolocation information.
- **Pros**
  - Fairly accurate
  - Works indoors
- **Cons**
  - Requires a device with access to cell phone
  - Not good in rural areas.

# Privacy

- Geolocation is completely opt-in.
- HTML5 Geolocation specification mandates that location information should not be made available without user's consent.
- The browser can never automatically find the user's location.

# Check for Browser Support

- In order to use HTML5 Geolocation API functions browser support needs to be checked for.

```
if(navigator.geolocation)
 {
   "Geolocation supported";
 }
else
 {
   "Geolocation not supported";
 }
```

Geolocation01.htm

# Position Requests

- There are two types of position requests
- One-Shot Position Request
  - Retrieve the user's location only once or only by request.
- Repeated Position Request
  - Request and retrieve the user location at repeated intervals.

# One-Shot Position Request

- `getCurrentPosition(PositionCallback successCallback, optional PositionErrorCallback errorCallback, optional PositionOptions options);`
  - `successCallback` tells the browser the function to be called when the location data is made available.
    - Fetching location data may take a while to complete.
  - `errorCallback` can present the user with an explanation if the request for location information is not completed.
  - `options` object can be provided to the HTML5 geolocation service to fine-tune the way data is gathered

```
navigator.geolocation.getCurrentPosition(
      updateLocation,handleLocationError);
```

# successCallback()

- The successCallback function is provided with a position object as a parameter.
- The position object will contain coordinates as the attribute coords and a timestamp for when the location data was gathered.
- The coordinates have multiple attributes on them
  - latitude
  - longitude
  - accuracy

## successCallback()

- Other attributes of the coordinates are not guaranteed to be supported and will return a null value if they are not:
  - altitude – height of the user' s location
  - altitudeAccuracy – in meters
  - heading – direction of travel, in degrees relative to true north
  - speed – ground speed in meters per second

## errorCallback()

- Handling errors is important as there can be many possibilities for location calculation services to fail
- The API defines error codes for all the cases needed
  - UNKNOWN_ERROR (code 0) – an error that is not covered by other error codes.
  - PERMISSION_DENIED (code 1) – user chose not to let the browser access location information
  - POSITION_UNAVAILABLE (code 2) – user' s location was attempted, but failed
  - TIMEOUT (code 3) – attempt to determine the location exceeded the timeout value.

## Optional Geolocation Request Attributes

- There are three optional attributes that can be provided to the HTML5 Geolocation service in order to fine-tune its data gathering approach
- **`enableHighAccuracy`**
  - A message to the browser that, if available, use a higher accuracy detection mode.
- **`timeout`**
  - Provided in milliseconds, telling the browser the maximum amount of time it is allowed to calculate the current position
- **`maximumAge`**
  - Indicates how old a location value can be before the browser must attempt to recalculate.

```
navigator.geolocation.getCurrentPosition(updateLoc
  ation
            ,handleLocationError,{timeout:10000});
```

Geolocation02.htm

## Repeated Position Updates

- This will cause the Geolocation service to call the **`updateLocation`** handler repeatedly as the user's location changes

```
watchId =
  navigator.geolocation.watchPosition(
     updateLocation, handleLocationError);
```

- Turning off updates requires a call to the **`clearWatch()`** function

```
navigator.geolocation.clearWatch(watchId);
```

Geolocation03.htm

# Share Me on a Google Map

- One extremely common request for geolocation data is to show a user's position on a map, such as the Google Maps service.
- The Google Map API has been designed to take decimal latitude and longitude locations.
- Hence the results of the position lookup can be passed to the Google Map API.

Geolocation04.htm

# Summary

- Geolocation has gained in popularity over the last few years.
- Many web services add location into their apps.
- HTML5 Geolocation APIs can be used to create compelling, location-aware web applications.
- Privacy concerns however need to be considered.

# HTML5 Web Storage

## Background

- Browser cookies have been a way of sending text values back and forth from server to client.
- Servers can use the values in these cookies to track user information across web pages.
- Cookie values are transmitted back and forth every time a user visits a domain.
- Cookies can also be used for targeted advertising.

# Background…

- Cookies have some well-known drawbacks:
  - Extremely limited in size. (generally about 4KB)
  - Cookies are transmitted back and forth from the server to browser. This implies
    - Cookie data is visible on the network
    - Data persisted as cookies will consume network bandwidth

# The Need and the Solution

- In many cases data need not be transmitted repeatedly over a network to a remote server.
- HTML5 Web Storage provides API that
  - allows developers to store values in easily retrievable JavaScript objects that persist across page loads.
  - store large values as high as a few megabytes.
- Stored data is not transmitted across the network and is accessed on return visits to a page.
- Using **`sessionStorage`** or **`localStorage`**, data can survive across page loads or across browser restarts respectively.

# Check for Browser Support

- The storage database for a given domain is accessed directly from the window object.

```
function checkStorageSupport()
 {
    if(window.sessionStorage)
    {
       "Browser supports sessionStorage"
    }
    else
    {
       "Browser does not support sessionStorage"
    }
 }
```

# Check for Browser Support

```
    if(window.localStorage)
    {
       "Browser supports localStorage"
    }
    else
    {
       "Browser does not support
localStorage"
    }
```

WebStorage01.htm

# Setting and Retrieving Values

- **`setItem()`** method associated with **`window.sessionStorage`** takes a key string and a value string.

```
window.sessionStorage.setItem("myFirstKey",
                              "myFirstValue");
```

- **`getItem()`** method helps retrieve the value for a particular key string.

```
alert(window.sessionStorage.getItem("myFirstKey"));
```

WebStorage02.htm

# sessionStorage Characteristics

- All pages served from the same origin (scheme + host + port) can retrieve values set on **`sessionStorage`** using the same keys.
- Objects set into **`sessionStorage`** will persist as long as the browser window/tab is not closed.
- The **`sessionStorage`** API solves the problem of scoping of values.
  - For example, a shopping application that allows users to purchase air tickets.
  - The preference data such as the departure date and return date can be persisted instead of using cookies and still be accessible across pages.

## Other Web Storage API Attributes

- Both **`sessionStorage`** and **`localStorage`** implement the **`Storage`** interface.
- Properties include
  - **`length`** – specifies how many key-value pairs are currently stored in the session object.
    - Storage objects are specific to their origin
  - **`key(index)`** – function allows retrieval of a given key.
    - Keys are zero-based.
    - Once a key is retrieved, it can be used to fetch its corresponding value
  - **`removeItem(key)`** – function that removes a value currently in storage under the specified key.

## Summary

- HTML5 Web Storage can be used as an alternative to browser cookies
- Network traffic is reduced
  - User information is stored locally in the browser.
- Transient Storage
  - Data that is not required for a longer period of time can be stored in **`sessionStorage`**.
- Persistence
  - Data can be stored across browser restarts in **`localStorage`**.

WebStorage04.htm

## Session Summary

- HTML5 is based on various design principles
  - Compatibility
  - Utility
  - Interoperability
  - Universal Access
- HTML5 provides native support for many features that used to be possible only with plug-ins.

## Strategies for implementing HTML5 today

- Progressive enhancement
- Accessibility > validation
- Detect support for HTML5
- When can I use .... http://caniuse.com

# HTML5 WYSIWYG Editor / Tools

- Adobe Dreamweaver CS5 +
- NetBeans 7.3 +
- Aptana Studio
  - Its available as stand-alone application and also available as eclipse IDE plug-in.
- Sublime Text
- WebStorm
- Maqetta
  - Maqetta is a browser based online development tool.
- Topstyle 4
  - Top style4 is also a good, powerful and rich functionality based tool for developing HTML5 and CSS3.
- Aloha Editor
  - The Aloha Editor is a browser-based rich text editor framework that was created in JavaScript.