Topics

- Web Messaging
  - Messaging API
- Web Workers
  - Threading
- Web Sockets
  - Server sent events

Web Messaging

- Messaging API specification
  - Part of the HTML5 specification
  - Supported by all major browser (Including IE8+)

Messaging API

- Messaging API
  - Messaging between frames
  - Client side messaging across domains
  - Message data passing
    - Current implementations only handle string data
    - Objects needs to be serialized (E.g. as JSON) to be passed with a message
    - All though the W3C specification specifies how to clone and handle other data types than strings

Messaging API Example

- Messaging API examples
  - postMessage to iframe across domains

```
var iframe = document.getElementById(“myIframe”);
iframe.contentWindow.postMessage( 'Hello frame', 'http://mydomain.no');
```

  - To receive messages an event listener needs to be implemented

```
window.addEventListener( 'message', function(event){
    //Handler function
}, false );
```

Threading

- Threading
  - Parallell processing in programmed applications (Multithreading)
  - Several processess running simultaneously and sharing resources (Such as memory etc.)
  - Processing:
    - Single processor: Multiplexing
    - Multiprocessor: Threads running at same time
- Threading in Javascript:
  - Web Workers

Web Workers

- A separate specification to HTML5
- A Web Worker is a script running in the background independently from the main application processing
- Possible usage scenarios
  - Processes that may be run independently of the main application
  - Heavy processes hogging system resources for a longer period of time

Web Workers Interaction

- Web Workers interact with each other via messaging

  (Through the postMessage method as in the Messaging API)

Web Workers Interaction

Example

```
//Send messages to the worker

var worker = new Worker("somescript.js");

worker.postMessage("Hello worker"); //Send message to the worker. Use this.onmessage
                                                    //in the worker as a message event handler


//Recieve messages from the worker by defining a message event handler

worker.onmessage = function( event ){
    //Handler function
    alert( event.data );
}
```

Detecting Web Worker Support

- Detecting support for Workers
  - Example 1:

    if(typeof Worker !=”undefined”){ //Do Worker stuff }
  - Example 2:

    if(!!window.Worker){ //Do Worker stuff }

Web Workers Example

Main script

```
<button onclick="sayHI()">Say HI</button>
<button onclick="stop()">Stop worker</button>
<output id="result"></output>
<script>
  function sayHI() {
    worker.postMessage({'cmd': 'start', 'msg': 'Hi'});
  }

  function stop() {
    //Calling worker.terminate() would also stop the worker.
    worker.postMessage({'cmd': 'stop', 'msg': 'Bye'});
  }

  var worker = new Worker('doWork2.js');

  worker.addEventListener('message', function(e) {
    document.getElementById('result').textContent = e.data;
  }, false);
</script>
```

doWork.js

```
self.addEventListener('message', function(e) {
  var data = e.data;
  switch (data.cmd) {
    case 'start':
      self.postMessage('WORKER STARTED: ' + data.msg);
      break;
    case 'stop':
      self.postMessage('WORKER STOPPED: ' + data.msg);
      self.close(); // Terminates the worker.
      break;
    default:
      self.postMessage('Unknown command: ' + data.msg);
  };
}, false);
```

Web Workers – Allowed Functionality

- Allowed functionality
  - Set event listeners
  - Ajax requests (XMLHttpRequest)
  - Timers (setTimeout, setInterval)
  - Core javascript functions (eval, isNaN etc.)
  - WebSockets
  - navigator and location objects
  - EventSource
  - WebSQL, IndexedDB
  - Web Workers
  - Import scripts: importScripts('someScript.js', 'someOtherScript.js');

Web Workers – Disallowed Functionality

- Disallowed functionality
  - DOM
  - window, document and parent objects

Shared Workers

- Multiple documents can access the same instance of a Worker
- E.g. frames sharing a Worker which serves these frames simultaneously
- May be used for syncing data between frames
- Implements an onconnect method in addition to onmessage of the plain Worker

Web Sockets

- Web Sockets provides a full duplex real time communication stream to a server from a web client
- Communication over a TCP socket
- The server is able to push messages directly to the client (ws -> push, http -> pull)
- Solve latency of real-time applications
  - Simulating push over http usually means AJAX polling or long polling
  - These methods results in high latency

Web Sockets – Streaming Data

- The web socket is always open an enable the client to send the server streaming data asynchronously over the same connection
- This opens for a wide range of real time applications with real time communication able to stream data between each other

The Web Sockets API

- Web Sockets API
  - Web Sockets work in almost the same way as Web Workers; application communication is based on message passing
  - Provides methods for
    - Creating connection
    - Sending data
    - Receiving data
    - Closing socket

The Web Sockets API

- Web Sockets API
  - Event handlers:
    - onmessage
    - onclose
    - onopen
    - onerror
  - Methods
    - send

Web Sockets Example

- Example

```
var connection = new WebSocket('ws://somedomain.com/echo');
// Sending String
connection.send('your message');

// Sending canvas ImageData as ArrayBuffer
var img = canvas_context.getImageData(0, 0, 400, 320);
var binary = new Uint8Array(img.data.length);
for (var i = 0; i < img.data.length; i++) {

  binary[i] = img.data[i];

}
connection.send(binary.buffer);

// Sending file as Blob
var file = document.querySelector('input[type="file"]').files[0];
connection.send(file);

//Receiving messages from server
connection.onmessage = function( e ){ //Handler function }
```

Server-Sent Events

- Push-messaging to the server over http (Based on the EventSource API)
- The EventSource object tries to maintain a persistent connection with the server
- Reestablishes the connection if the connection is lost

The EventSource API

- The EventSource object event handlers:
  - onopen
  - onmessage
  - onerror
- Message MIME type - “text/event-stream”

Server-Sent Events Example

Client

```
if (!!window.EventSource) {
  var source = new EventSource('stream.php');
} else {
  // Result to xhr polling :(
}

source.onmessage = function(e) {
  console.log(e.data);
};

source.onopen = function(e) {
  // Connection was opened.
};

source.onerror = function(e) {
  if (e.readyState == EventSource.CLOSED) {
    // Connection was closed.
  }
};
```

Server

```
<?php
header('Content-Type: text/event-stream');
header('Cache-Control: no-cache'); // recommended to
                                   //prevent caching of event data.

/**
 * Constructs the SSE data format and flushes that data to the
client.
 */
function sendMsg($id, $msg) {
  echo "id: $id" . PHP_EOL;
  echo "data: $msg" . PHP_EOL;
  echo PHP_EOL;
  ob_flush();
  flush();
}

$serverTime = time();

sendMsg($serverTime, 'server time: ' . date("h:i:s", time()));
```

References

- Lawson and Sharp
- W3C
- HTML5rocks.com (Examples)
- Wikipedia