**HTML5 (Server-Send-Events)**

Server-Sent-Event is one way messaging.

* A server-sent event is when a web page automatically gets updates from a server.
* SSE is a server push technology enabling a browser to receive automatic updates from a server via HTTP connection.
* Using SSE you can push DOM events continuously from your web server to the visitor's browser.

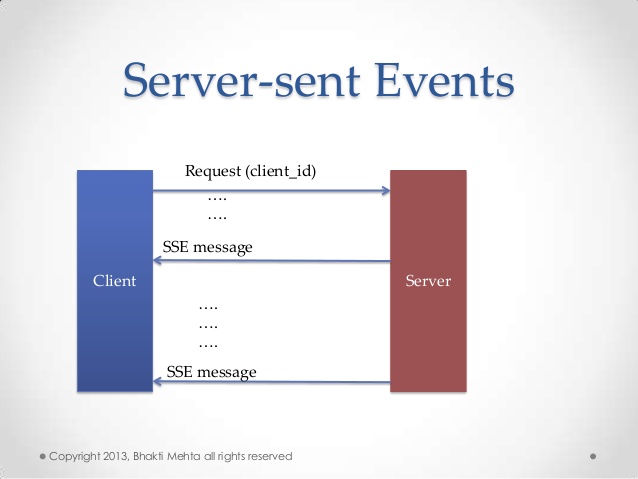
**What is SSE?**

* Server-sent events (SSE) is a technology enabling a browser to receive automatic updates from a server via HTTP connection. The Server-Sent Events EventSource API is standardized as part of HTML5 by the W3C.

**What is HTML SSE?**

* The type of events which are flowing from web browser to the web server may be called client-sent events. Along with HTML5, WHATWG Web Applications 1.0 introduces events which flow from web server to the web browsers and they are called Server-Sent Events (SSE).

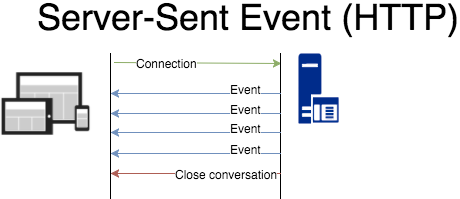
For example, a simple click on a link requests a new page from the server. The type of events which are flowing from web browser to the web server may be called client-sent events.



-> In case of server sent event the communication is carried out from server to browser only and browser cannot send any data to the server.

-> This kind of communication is mainly used when the need is only to show the updated data, then the server sends the message whenever the data gets updated.

-> For instance a one-way communication between the Server to Browser. This protocol is less complicated, so no need to rely on the external library JAVASCRIPT itself provides the EventSource interface to receive the server sent messages.



**What is the different between the Web Scoket and the SSE in HTML5?**

**Websockets** and SSE (**Server Sent Events**) are both capable of pushing data to browsers, however they are not competing technologies.

**1 -> Websockets** connections can both send data to the browser and receive data from the browser. A good example of an application that could use **websockets** is a chat application.

**2 ->** SSE connections can only push data to the browser. Online stock quotes, or twitters updating timeline or feed are good examples of an application that could benefit from SSE.

In practice since everything that can be done with SSE can also be done with Websockets, Websockets is getting a lot more attention and love, and many more browsers support Websockets than SSE.

**Advantages of SSE over Websockets:**

* Transported over simple HTTP instead of a custom protocol
* Can be poly-filled with javascript to "backport" SSE to browsers that do not support it yet.
* Built in support for re-connection and event-id
* Simpler protocol

**Advantages of Websockets over SSE:**

* Real time, two directional communication.
* Native support in more browsers

**Web Application for SSE**

* To use Server-Sent Events in a web application, you would need to add an <eventsource> element to the document.
* The src attribute of <eventsource> element should point to an URL which should provide a persistent HTTP connection that sends a data stream containing the events.

**Example of web application which would expect server time.**

<!DOCTYPE HTML>

<html>

<head>

<script type = "text/javascript">

/\* Define event handling logic here \*/

</script>

</head>

<body>

<div id = "sse">

<eventsource src = "/cgi-bin/ticker.cgi" />

</div>

<div id = "ticker">

<TIME>

</div>

</body>

</html>

## Handle Server-Sent Events

## Let us modify our web application to handle server-sent events. Following is the final example

<!DOCTYPE HTML>

<html>

<head>

<script type = "text/javascript">

document.getElementsByTagName("eventsource")[0].addEventListener("server-time",

eventHandler, false);

function eventHandler(event) {

// Alert time sent by the server

document.querySelector('#ticker').innerHTML = event.data;

}

</script>

</head>

<body>

<div id = "sse">

<eventsource src = "/cgi-bin/ticker.cgi" />

</div>

<div id = "ticker" name = "ticker">

[TIME]

</div>

</body>

</html>

## Server-Sent Events Demo

## Index.html

|  |
| --- |
| <!DOCTYPE html> |
|  | <html> |
|  | <head> |
|  | <meta charset="utf-8" /> |
|  | <meta name="viewport" content="width=device-width, height=device-height" /> |
|  | <title>Server-Sent Events Demo</title> |
|  | <style type="text/css"> |
|  | body { |
|  | font-family: 'Open Sans', sans-serif; |
|  | } |
|  | </style> |
|  | </head> |
|  | <body> |
|  |  |
|  | <h1>Server-Sent Events Demo</h1> |
|  |  |
|  | <ul></ul> |
|  |  |
|  | <script> |
|  | (function() { "use strict"; |
|  | var ul = document.querySelector('ul'); |
|  | var es = new EventSource('/events'); |
|  | function li(text) { |
|  | var li = document.createElement('li'); |
|  | li.innerText = text; |
|  | ul.appendChild(li); |
|  | } |
|  | es.addEventListener('open', function() { |
|  | li('Server connected :)'); |
|  | }); |
|  | es.addEventListener('my-custom-event', function(event) { |
|  | li(event.data); |
|  | }); |
|  | es.addEventListener('error', function() { |
|  | li('Server unavailable :('); |
|  | }); |
|  | })(); |
|  | </script> |
|  |  |
|  | </body> |
|  | </html> |

## Serever.js

|  |
| --- |
| Var fs = require('fs'); |
|  | var http = require('http'); |
|  |  |
|  | var connectionCounter = 1; |
|  |  |
|  | http.createServer(function(request, response) { |
|  |  |
|  | if (request.url === '/') { |
|  |  |
|  | response.writeHead(200, { 'Content-Type': 'text/html' }); |
|  | response.write(fs.readFileSync('index.html')); |
|  | response.end(); |
|  |  |
|  | } else if (request.url === '/events') { |
|  |  |
|  | var thisConnection = connectionCounter++; |
|  | var thisEvent = 1; |
|  |  |
|  | console.log('Client connected to event stream (connection #' + thisConnection + ', Last-Event-Id: ' + request.headers['last-event-id'] + ')'); |
|  | response.writeHead(200, { |
|  | 'Content-Type': 'text/event-stream', |
|  | 'Cache-Control': 'no-cache' // let intermediaries know to NOT cache anything |
|  | }); |
|  |  |
|  | var ticker = setInterval(function() { |
|  | response.write('event: my-custom-event\n'); |
|  | response.write('id: ' + (thisConnection \* 1000 + thisEvent) + '\n'); |
|  | response.write('data: Server says hi! (event #' + thisEvent++ +' of connection #' + thisConnection + ')\n\n'); |
|  | }, 2500); |
|  |  |
|  | request.on('close', function() { |
|  | console.log('Client disconnected from event stream (connection #' + thisConnection + ')'); |
|  | response.end(); |
|  | clearInterval(ticker); |
|  | }); |
|  |  |
|  | } else { |
|  |  |
|  | response.writeHead(404); |
|  | response.end(); |
|  |  |
|  | } |
|  |  |
|  | }).listen(8888); |

## Server-Sent Events, Hello World Servlet

## we create an html page that has a button to start server-sent event. The server will send us a timestamp every second which we just display on page.

## *index.jsp*

<!DOCTYPE HTML>

<html>

<body>

Time: <span id="foo"></span>

<br><br>

<button onclick="start()">Start</button>

<script type="text/javascript">

function start() {

var eventSource = new EventSource("HelloServlet");

eventSource.onmessage = function(event) {

document.getElementById('foo').innerHTML = event.data;

};

}

</script>

</body>

</html>

## we created an event source for “/HelloServlet” path. On each message we just prints the data in span “foo”.

## checking Servlet code which sents the Server-Sent Events.

## *HelloServlet.java*

package net.viralpatel.servlets;

import java.io.IOException;

import java.io.PrintWriter;

import javax.servlet.ServletException;

import javax.servlet.http.HttpServlet;

import javax.servlet.http.HttpServletRequest;

import javax.servlet.http.HttpServletResponse;

public class TestServlet extends HttpServlet {

public void doGet(HttpServletRequest request, HttpServletResponse response)

throws ServletException, IOException {

//content type must be set to text/event-stream

response.setContentType("text/event-stream");

//encoding must be set to UTF-8

response.setCharacterEncoding("UTF-8");

PrintWriter writer = response.getWriter();

for(int i=0; i<10; i++) {

writer.write("data: "+ System.currentTimeMillis() +"\n\n");

try {

Thread.sleep(1000);

} catch (InterruptedException e) {

e.printStackTrace();

}

}

writer.close();

}

}

## The servlet is quite simple. Content type must be set to “text/event-stream” and the character encoding must be UTF-8.

## Also each message that we send must ends with \n\n.

## *web.xml*

<?xml version="1.0" encoding="UTF-8"?>

<web-app id="WebApp\_ID" version="2.4"

xmlns="http://java.sun.com/xml/ns/j2ee" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"

xsi:schemaLocation="http://java.sun.com/xml/ns/j2ee http://java.sun.com/xml/ns/j2ee/web-app\_2\_4.xsd">

<display-name>ServerSentEvent\_HttpServlet\_example</display-name>

<welcome-file-list>

<welcome-file>index.jsp</welcome-file>

</welcome-file-list>

<servlet>

<display-name>HelloServlet</display-name>

<servlet-name>HelloServlet</servlet-name>

<servlet-class>net.viralpatel.servlets.HelloServlet</servlet-class>

</servlet>

<servlet-mapping>

<servlet-name>HelloServlet</servlet-name>

<url-pattern>/HelloServlet</url-pattern>

</servlet-mapping>

</web-app>