SUPINFO International University

Sign of success

HTML 5

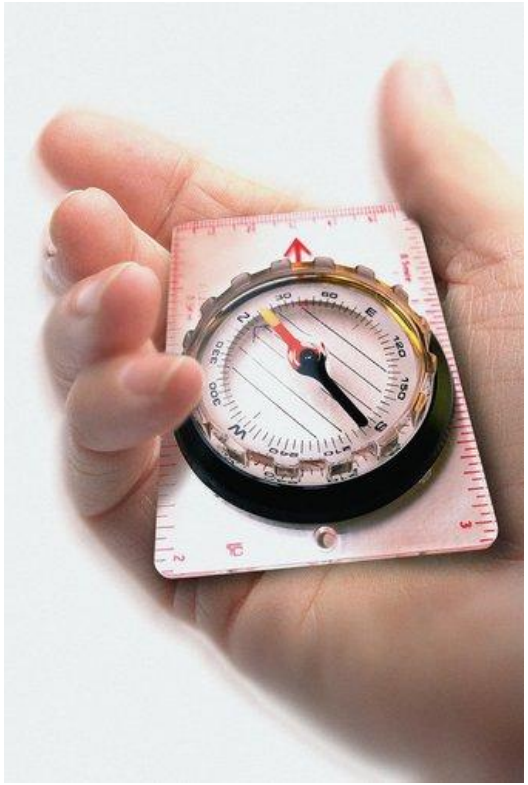
New APIs - Part 1

Course objectives

By completing this course, you will be able to:

- Develop multi-threaded JS applications
- Use Drag & Drop features in your web apps
- Geolocalize a user of your web apps

Course topics



Course's plan:

- Web Workers
- Drag & Drop
- Geolocation

HTML5 - New APIs

WEB WORKER

Presentation

- Web Workers API allows you to run scripts in the background
  - Independently of any user interface scripts !
- Useful for long-running scripts
  - Don’t need to manage yielding to keep the page responsive

Thread safety

- Web Workers use real OS-level threads
  - As in other technologies, bad concurrency code can cause starvations, dead locks and other side effects

Thread safety

- To limit these issues, the API provides :
  - A carefully controlled communication between threads
  - No access to non-thread safe components or the DOM

Worker interface

- The core of the API is the Worker interface
- It provides the following constructor :
  - *Worker(scriptUrl)* :
    - Creates a web worker that executes the script at the specified URL

Worker interface

- It also provides the following methods :
  - *postMessage(message)* :
    - Sends a message to the worker's inner scope
  - *terminate()* :
    - Immediately terminates the worker without offers the worker an opportunity to finish its operations

Worker interface

- And the following properties :
  - *onmessage* :
    - An event listener called when the worker return a message
  - *onerror* :
    - An event listener called when the worker return an error

Worker scope

- Scripts executed by Workers have access to :
  - *postMessage(message)* function
    - Returns a message to the *message* handler of the main thread
  - *onmessage* property
    - Function which will receive messages sent when the worker object's postMessage() is called

Worker principle

Main thread

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

worker.postMessage(message);


// Do something

worker.onmessage = function(event) {
      // Do something
};
```

worker.js

```
…

onmessage = function(event) {

     // Do something else

     postMessage(message);
}
```

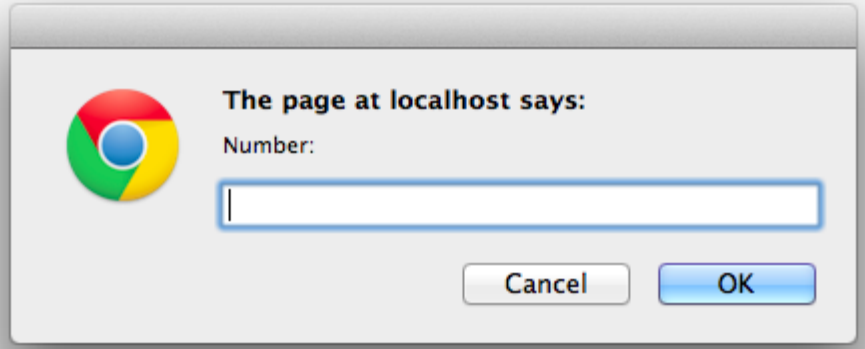
Example - *index.html*

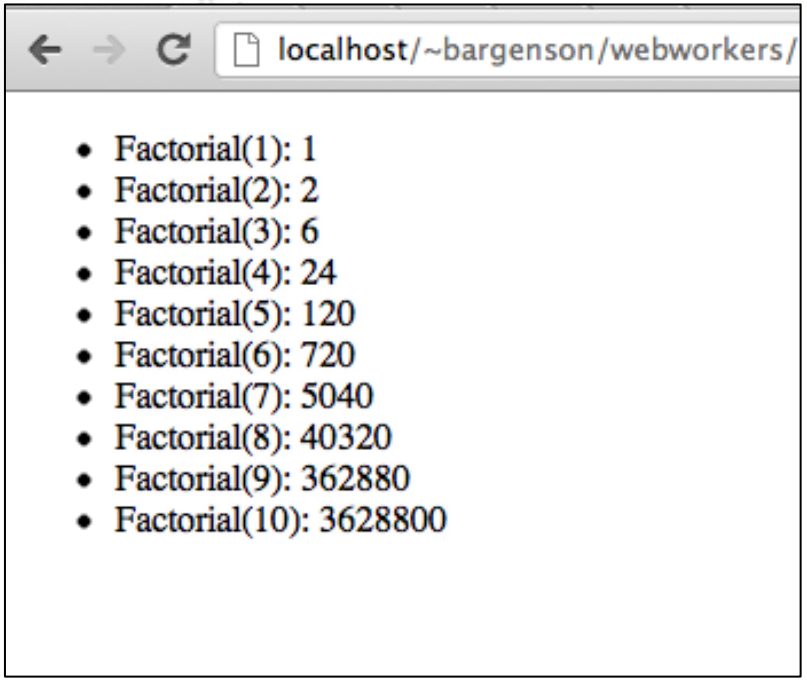
```
<ul id="result"></ul>

<script type="text/javascript">
   var worker = new Worker('factorial.js');
   worker.onmessage = function(event) {
     var result = event.data;
     var li = "<li>Factorial(" + result.n + "): ";
     li += result.factorial + "</li>";
     document.getElementById("result").innerHTML = li;
   };
   worker.postMessage(prompt("Number: "));
</script>
```

Example - *factoral.js*

```
function factorial(n, callback) {
  if ((n == 0) || (n == 1)) callback(1);
  else {
    factorial(n - 1, function(result) {
      postMessage({ n: (n - 1), factorial: result });
      result *= n;
      setTimeout(function() { callback(result) }, 2000);
    });
  }
}

self.onmessage = function(event) {
  factorial(event.data, function(result) {
    postMessage({ n: event.data, factorial: result });
  });
}
```

Example





Questions ?

Exercise (1/4)

- During this exercise, we will develop a simple page which will
  - Fetch tweets about SUPINFO every 10 seconds
  - Display them

Exercise (2/4)

Exercise (3/4)

- The script fetching the tweets must be execute by a *WebWorker*

- You can retrieve the tweets about SUPINFO thanks to this URI :

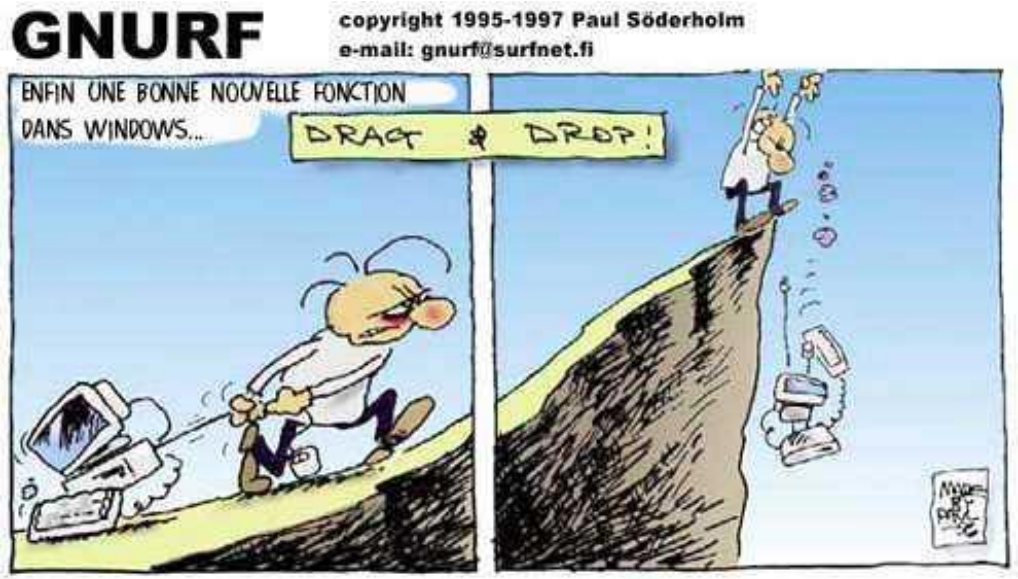
http://search.twitter.com/search.json?q=SUPINFO&since_id=<LAST_TWEET_ID>

Exercise (4/4)

- Your script must check every 10 seconds if new tweets are available
  - If there are, you must send them to the UI thread to display them

HTML5 - New APIs

DRAG & DROP



Presentation

- Drag and drop is a very common feature
  - "grab" an object and drag it to a different location

- In HTML5, drag and drop is part of the standard, and any element can be draggable

Draggable

- To make an element draggable is very simple :
  - Give the element a *draggable* attribute
  - Set an event listener for *dragstart* event

```
<p>What fruits do you like?</p>
<ol>
   <li draggable="true"> Apples </li>
   <li draggable="true"> Oranges </li>
   <li draggable="true"> Pears </li>
</ol>
```

Draggable

- Handler :

```
var fruits = document.querySelectorAll("li[draggable]");
var fruit = null;
for (var i = 0; i < fruits.length; i++) {
  fruit = fruits[i];
  fruit.addEventListener("dragstart", function(event) {
    console.log("dragstart");
  });
};
```

Dropzone

- To make an element accept a drop :
  - Give the element a *dropzone* attribute
  - Set an event listener for *dragover* event and cancel the event
    - Otherwise, no *drop* event will be triggered
  - Set an event listener for *drop* event

Dropzone

- Example :

```
<ol dropzone="move string:text/x-example"
   ondragover="event.preventDefault()"
   ondrop="dropHandler(event)">
</ol>
```

Dropzone

- The *dropzone* attribute specifies
  - What kind of data to accept
    - *string:text/plain*
    - *file:image/png*
    - *…*
  - What kind of feedback to give
    - *move*
    - *copy*
    - *…*

Dropzone

- Instead of using the *dropzone* attribute, you can use :
  - A *dragenter* event handler
    - To report whether or not the drop target is to accept the drop
  - A *dragover* event handler
    - To specify what feedback is to be shown to the user

Dropzone

- Example:

```
var zone = document.querySelector("ol[dropzone]");

zone.addEventListener("dragover", function(event) { ... });
zone.addEventListener("dragenter", function(event) { ... });
zone.addEventListener("dragleave", function(event) { ... });
zone.addEventListener("drop", function(event) { ... });
```

Data Transfer

- *DataTransfer* object is used to transfer information from the *draggable* element to the *dropzone*
  - Accessible from the *event* parameter inside your handlers

```
fruit.addEventListener("dragstart", function(event) {
   var dataTransfer = event.dataTransfer;
});
```

Data Transfer

- The two main methods of the *DataTransfer* interface are :
  - *getData(type)*
    - Retrieves the data for a given type, or an empty string if does not exist
  - *setData(type, data)*
    - Sets the data for a given type

Data Transfer - Example

```
...
fruit.addEventListener("dragstart", function(event) {
   event.dataTransfer.setData("text", this.id);
});
...
zone.addEventListener("dragover", function(event) {
   event.preventDefault(); // allows us to drop
});
zone.addEventListener("drop", function(event) {
   var fruitId = event.dataTransfer.getData("text");
   // Append the fruit to the dropzone
});
```

Questions ?

Exercise (1/2)

- Update your SUPINFO Tweets page :
  - Add a new area for your favorite tweets
  - Make it a drop zone
  - Make the timeline tweets draggable

Drag & Drop

Exercise (2/2)

SUPINFO Tweets

Timeline

RT @SUPINFO_Rennes: Samedi 20 octobre: Journée Portes-Ouvertes - SUPINFO Rennes http://t.co/htfsiK8S

Created the Tue, 16 Oct 2012 17:06:40 +0000 by @Gwennin.

"This was a triumph // I'm making a note here: HUGE SUCCESS" #supinfo #soutenance

Created the Tue, 16 Oct 2012 16:42:44 +0000 by @Jognu.

Petite question pour la soutenance orale : Faut-il apporter une copie papier du mémoire? #supinfo

Created the Tue, 16 Oct 2012 16:03:41 +0000 by @_G4ry.

Favorites

Le livre #Centreon Maitrisez la supervision de votre système d'information est dispo pour les étudiants de #SUPINFO sur libraries ;)

Created the Tue, 16 Oct 2012 17:20:37 +0000 by @lolokai.

HTML5 - New APIs

GEOLOCATION

Presentation

- The new Geolocation API defines a high-level interface to location information

- The API itself is agnostic of the underlying location information sources
  - Can be GPS, location inferred from network signals as well as user input

Presentation

- The API is designed to enable
  - "one-shot" position requests
  - repeated position updates

- Location information is represented by latitude and longitude coordinates

One-shot position request

- The *Geolocation* interface expose the following method to do that:
  - *getCurrentPosition(callback)*

- The callback function take the user position as argument

One-shot position request

- Example :

```
navigator.geolocation.getCurrentPosition( function(position){
   console.log("Latitude: " + position.coords.latitude);
   console.log("Longitude: " + position.coords.longitude);
});
```

Repeated position updates

- The *Geolocation* interface expose also the following methods :
  - *watchPosition(callback, errorHandler)*
    - Request repeated updates and return a watcher ID
  - *clearWatch(watchId)*
    - Cancel the updates for a given watcher ID

Repeated position updates

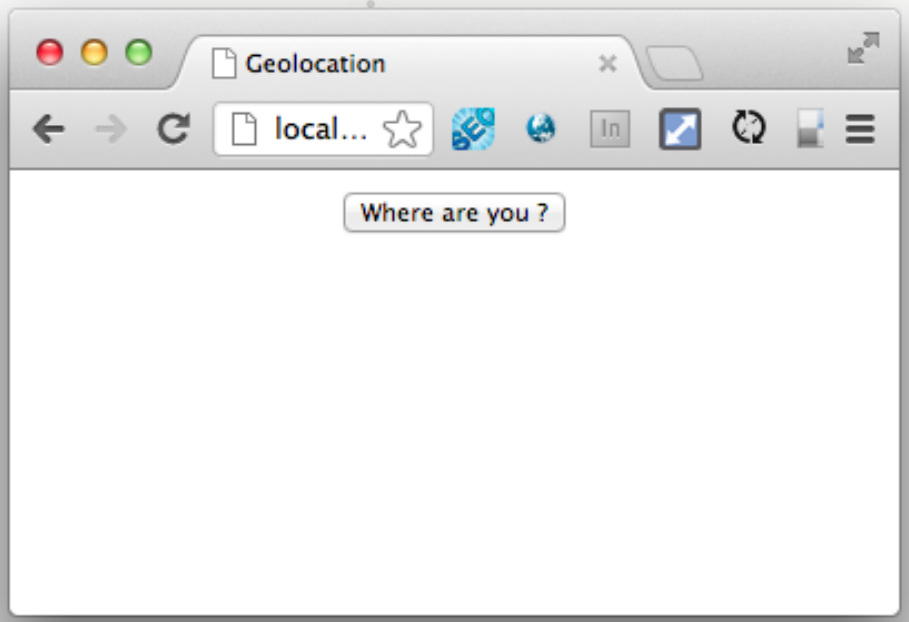
- Example :

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

function updateLocation(position) {   // Do something    }

function handleError(error) {    // Display an error message    }

document.getElementById("cancelButton").onclick = function() {
   navigator.geolocation.clearWatch(watchId);
}
```

Questions ?

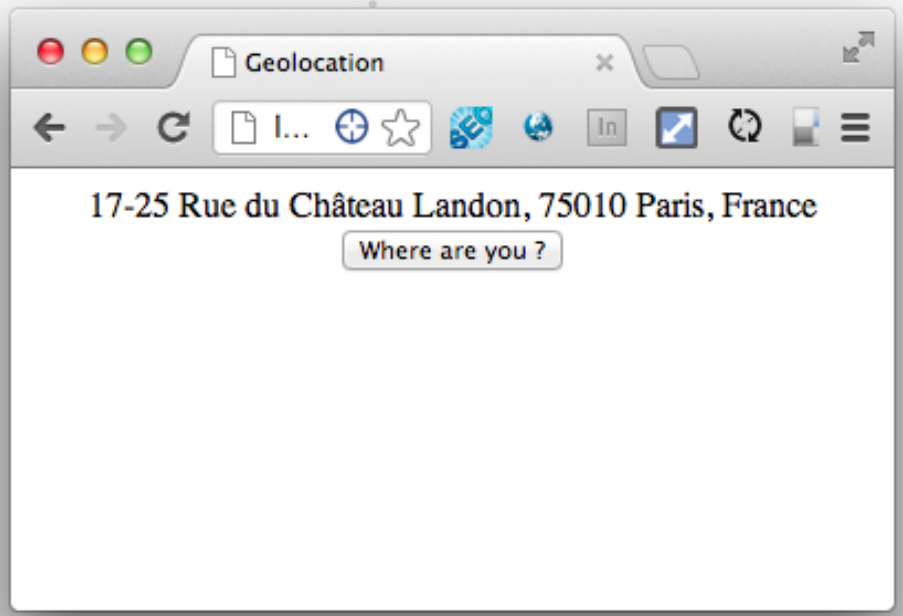
Exercise (1/3)

- Create a new page with a simple button as follow :



Exercise (2/3)

- When the user clicks on the button, you have to display his address :



Exercise (3/3)

- To convert coordinates to address, you can use the Google Maps Geocoder API:

https://developers.google.com/maps/documentation/javascript/geocoding