# HTML - Web Workers API

The **HTML Web Workers API** is a JavaScript feature that is used to run computationally intensive tasks in background in a separate thread without interrupting the user interface.

In this **Web Workers API** chapter, you will learn the following topics:



## What Are Web Workers?

- **Web workers** allows long tasks to be executed without yielding to keep the page unresponsive.
- **Web workers** are background scripts, and they are relatively heavy-weight and are not intended to be used in large numbers. For example, it would be inappropriate to launch one worker for each pixel of a four-megapixel image.
- When a script is executing inside a Web worker, it cannot access the web page's window object (window.document).
- **Web workers** don't have direct access to the web page and the DOM API. Although Web workers cannot block the browser UI, they can still consume CPU cycles and make the system less responsive.

## Need of Web Workers

JavaScript was designed to run in a single-threaded environment, meaning multiple scripts cannot run at the same time. Consider a situation where you need to handle UI events, query and process large amounts of API data, and manipulate the DOM. JavaScript will hang your browser in situations where CPU utilization is high.

### Example

Let us take a simple example where JavaScript goes through a big loop. Your browser will become unresponsive when you run this code:

```
<!DOCTYPE html>
<html>
<head>
   <title>Big for loop</title>
   <script>
      function bigLoop() {
         for (var i = 0; i <= 10000000000; i += 1) {
            var j = i;
         }
         alert("Completed " + j + "iterations");
      }
      function sayHello() {
         alert("Hello sir....");
      }
   </script>
</head>

<body>
   <input type="button"
          onclick="bigLoop();"
          value="Big Loop" />

   <input type="button"
          onclick="sayHello();"
          value="Say Hello" />
</body>
</html>
```

The situation explained above can be handled using **Web Workers**, which will do all the computationally expensive tasks without interrupting the user interface and typically run on separate threads.

Explore our **latest online courses** and learn new skills at your own pace. Enroll and become a certified expert to boost your career.

## How Web Workers Work?

Web Workers are initialized with the URL of a JavaScript file, which contains the code the worker will execute. This code sets event listeners and communicates with the script that spawned it from the main page. Following is the simple syntax:

```
var worker = new Worker('bigLoop.js');
```

If the specified JavaScript file exists, the browser will spawn a new worker thread, which is downloaded asynchronously. If the path to our worker returns a 404 error, the worker will fail silently.

If our application has multiple supporting JavaScript files, we can import them using the **importScripts()** method, which takes file name(s) as argument separated by comma as follows:

```
importScripts("helper.js", "anotherHelper.js");
```

Once the Web Worker is spawned, communication between the web worker and its parent page is done using the **postMessage()** method. Depending on the browser/version, the **postMessage()** method can accept either a string or JSON object as its single argument.

The message passed by Web Worker is accessed using **onmessage** event in the main page. Below is the main page (hello.htm) which will spawn a web worker to execute the loop and to return the final value of variable j:

## Example

```
<!DOCTYPE html>
<html>
<head>
   <title>Big for loop</title>
   <script>
      var worker = new Worker('bigLoop.js');
      worker.onmessage = function(event) {
         alert("Completed " + event.data + "iterations");
      };
      function sayHello() {
         alert("Hello sir....");
      }
   </script>
</head>

<body>
   <input type="button"
          onclick="sayHello();"
          value="Say Hello" />
</body>
</html>
```

Following is the content of bigLoop.js file. This makes use of postMessage() API to pass the communication back to main page:

```
for (var i = 0; i <= 1000000000; i += 1){
   var j = i;
}
postMessage(j);
```

The j variable from bigLoop.js is published using function `postMessage()`, Which then received at hello.htm using event attribute of `worker.onmessage = function(event) {}`

## Stopping Web Workers

Web Workers don't stop by themselves, but the page that started them can stop them by calling the **terminate()** method.

```
worker.terminate();
```

A terminated Web Worker will no longer respond to messages or perform any additional computations. We cannot restart a worker; instead, we need to create a new worker using the same URL.

## Handling Web Workers Errors

You can handle errors that occur in the Web Worker from the main thread by using the **worker.onerror()** method.

## Example

The following shows an example of an error handling function in a Web Worker JavaScript file that logs errors to the console. With error handling code, the above example would become as follows:

```
<!DOCTYPE html>
<html>
<head>
   <title>Big for loop</title>
   <script>
      var worker = new Worker('bigLoop.js');
      worker.onmessage = function(event) {
         alert("Completed " + event.data + "iterations");
      };
      worker.onerror = function(event) {
         console.log(event.message, event);
      };
      function sayHello() {
         alert("Hello sir....");
      }
   </script>
</head>

<body>
   <input type="button"
          onclick="sayHello();"
          value="Say Hello" />
```

```
</body>
</html>
```

You can use the following code snippets as an error handler for the worker on the main page:

```
worker.onerror = function(event) {
   console.error("Error in web worker: " + event.message, event);
   alert("Error occurred in the web worker.");
   // Prevents the default handling of the error
   event.preventDefault();
};
```

## Checking for Browser Support

To check the browser support for Web Workers, you can use Modernizr's **webworkers** feature test.

## Example

Following is the syntax to detect a Web Worker feature support available in a browser:

```
<!DOCTYPE html>
<html>
<head>
   <title>Big for loop</title>
   <script src="/js/modernizr-1.5.min.js">
   </script>

   <script>
      if (Modernizr.webworkers) {
         alert("You have web workers support.");
      } else {
         alert("You do not have web workers support.");
      }
   </script>
</head>

<body>
   <p>
      Checking for Browser Support for web workers
   </p>
</body>
</html>
```