**Dedicated WebWorker:**

var myWorker = new Worker(aURL);

aURL : path of shared-worker.js file. It must obey the same-origin policy.

Connecting :

* Dedicated workers use MessagePort objects behind the scenes, and thus support all the same features, such as sending structured data and transferring binary data.
* To receive messages from a dedicated worker, use the onmessage and addEventListener’s message event handler IDL attribute on the Worker object:
* The implicit MessagePort used by dedicated workers has its port message queue implicitly enabled when it is created, so there is no equivalent to theMessagePort interface's start() method on the Worker interface.
* To send data to a worker, use the postMessage() method.
* Structured data can be sent over this communication channel.
* To send ArrayBuffer objects efficiently (by transferring them rather than cloning them), list them in an array in the second argument.

For e.g. :

worker.postMessage({

operation: 'find-edges',

input: buffer, // an ArrayBuffer object

threshold: 0.6,

}, [buffer]);

* the data is provided in the event object's data attribute.
* To send messages back, you again use postMessage(). It supports the structured data in the same manner.

For e.g. :

postMessage(event.data.input, [event.data.input]); // transfer the buffer back

**Main JS:**

if(window.Worker){

var worker = new Worker("src/js/worker.js");

worker.addEventListener("message", function(evt){

console.log("main thread : " + evt.data);

}, false);

worker.addEventListener('error', function(e){

throw new Error(' Error: could not open Worker', e);

}, false);

worker.postMessage("worker");

}

**worker.js**

self.addEventListener("message",function(evt){

self.postMessage(evt.data + " message from dedicated worker");

},false);