



Clipcode's Guide to the Web Workers

Specification

The latest Web Workers spec is available here:

<http://dev.w3.org/html5/workers/>

Compatibility

<http://caniuse.com/#feat=webworkers>

Overview

Web workers brings multithreading to the browser.

A web worker is a separate flow of execution that is spawned from another web worker or the main window (UI) script. A distinct script (*.js) runs in the web worker and allows for background processing without having to block the user interface. Messages can be posted between web workers.

Message ports exist between different executing scripts – one port exists for DedicatedWorkerGlobalScope and multiple ports for SharedWorkerGlobalScope.

Messages are posted to ports via postMessage function and received from ports via onMessage event handler.

Web workers are different to typical multithreading in a typical operating system (e.g. pthreads) or runtime layer (e.g. .NET's System.Threading). Web workers are build around message passing. Web workers do not directly support synchronization constructs such as mutexes, semaphores or locks. Instead, web workers can coordinate activities via the shared worker global scope.

To learn more about use of message ports, see section 8 of the Html5 spec:

<http://www.w3.org/TR/html5/comms.html#crossDocumentMessages>

Web Workers Concepts

The following concepts are part of the Web Workers (APIs are from W3C/WHATWG spec, and copyright by them):

WorkerNavigator – Identity and online status of user agent

```
interface WorkerNavigator {};  
WorkerNavigator implements NavigatorID;  
WorkerNavigator implements NavigatorOnLine;
```

WorkerUtils – provides access to WorkerNavigator & importScript for loading extra scripts

```
[Supplemental, NoInterfaceObject]
interface WorkerUtils {
  void importScripts(in DOMString... urls);
  readonly attribute WorkerNavigator navigator;
};
WorkerUtils implements WindowTimers;
WorkerUtils implements WindowBase64;
```

The main script that runs in a web worker can use importScripts to loading additional scripts.

Worker Location – Absolute URL

```
interface WorkerLocation {
  // URL decomposition IDL attributes
  readonly attribute DOMString href;
  readonly attribute DOMString protocol;
  readonly attribute DOMString host;
  readonly attribute DOMString hostname;
  readonly attribute DOMString port;
  readonly attribute DOMString pathname;
  readonly attribute DOMString search;
  readonly attribute DOMString hash;
};
```

WorkerGlobalScope – Base for dedicated and shared scopes

```
interface WorkerGlobalScope {
  readonly attribute WorkerGlobalScope self;
  readonly attribute WorkerLocation location;

  void close();
  attribute Function? onerror;
  attribute Function? onoffline;
  attribute Function? ononline;
};
WorkerGlobalScope implements WorkerUtils;
WorkerGlobalScope implements EventTarget;
```

Can be used to set functions for onerror, onoffline and ononline.

DedicatedWorkerGlobalScope – A dedicated worker (that cannot be shared)

```
[Supplemental, NoInterfaceObject]
interface DedicatedWorkerGlobalScope : WorkerGlobalScope {
  void postMessage(in any message, in optional sequence<MessagePort> ports);
  attribute Function? onmessage;
};
```

There is a single message port, and calls to postMessage puts the message on this port.

SharedWorkerGlobalScope – A worker that can be shared

```
[Supplemental, NoInterfaceObject]
interface SharedWorkerGlobalScope : WorkerGlobalScope {
  readonly attribute DOMString name;
  readonly attribute ApplicationCache applicationCache;
  attribute Function? onconnect;
};
```

The event parameter to the onconnect handler has a ports array, and this is used to access the

message port for a particular connection.

AbstractWorker – base for worker (used by creators)

```
[Supplemental, NoInterfaceObject]
interface AbstractWorker {
    attribute Function? onerror;
};
AbstractWorker implements EventTarget;
```

Worker – Used by the creator of a dedicated worker

```
[Constructor(in DOMString scriptURL)]
interface Worker : AbstractWorker {
    void terminate();

    void postMessage(in any message, in optional sequence<MessagePort> ports);
    attribute Function? onmessage;
};
```

(Perhaps should be called dedicatedWorker).

SharedWorker – Used by the creator of shared workers

```
[Constructor(in DOMString scriptURL, in optional DOMString name)]
interface SharedWorker : AbstractWorker {
    readonly attribute MessagePort port;
};
```

Messages are sent via the port attribute.

ErrorEvent – Used for error information

```
interface ErrorEvent : Event {
    readonly attribute DOMString message;
    readonly attribute DOMString filename;
    readonly attribute unsigned long lineno;
    void initErrorEvent(in DOMString typeArg, in boolean canBubbleArg,
                        in boolean cancelableArg, in DOMString messageArg,
                        in DOMString filenameArg, in unsigned long linenoArg);
};
```