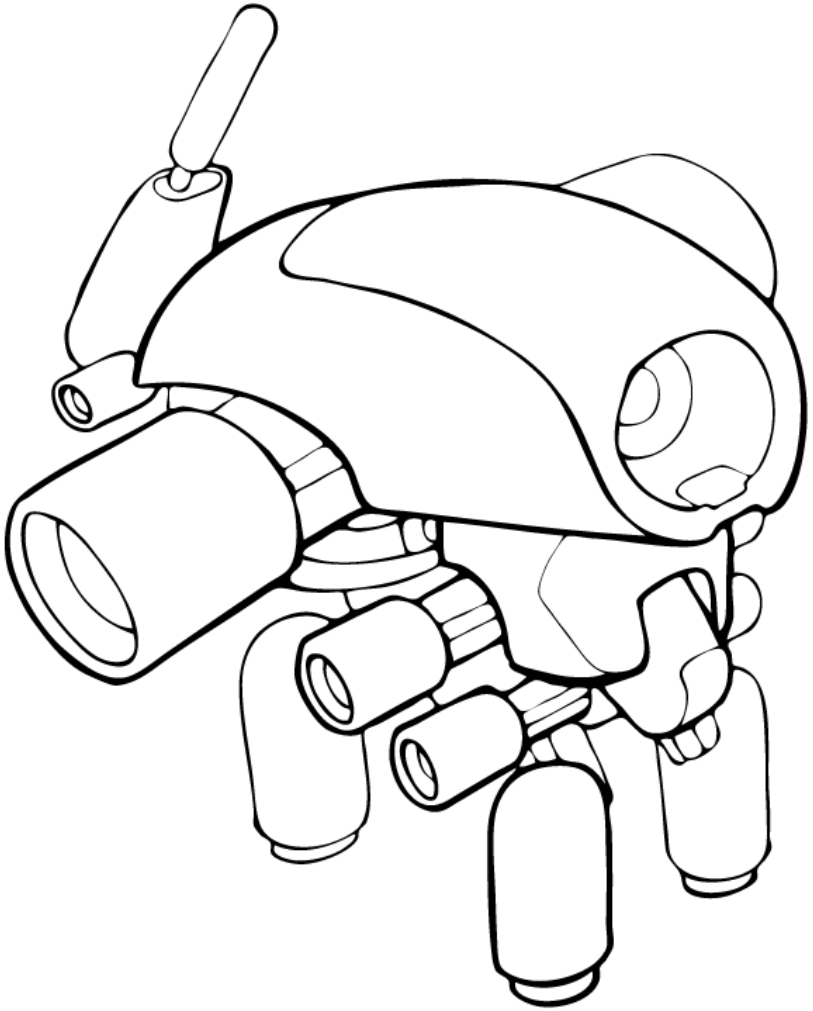
WEB Sockets

WEB Workers

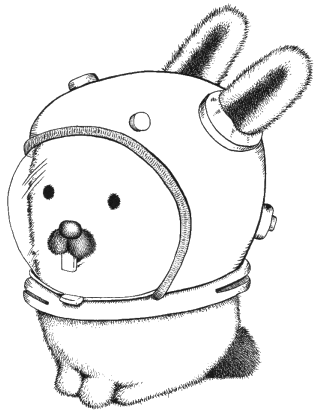
Genadi Samokovarov
genadi@vmware.com

Radoslav Georgiev
radoslav@game-craft.com

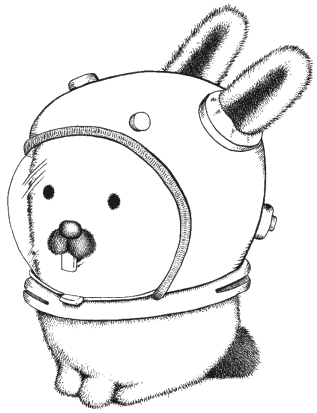
Web Sockets



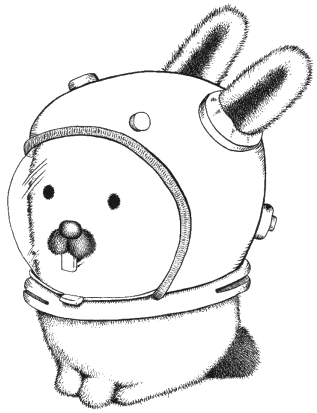
The WebSockets defines an API establishing "socket" connections between a web browser and a server.



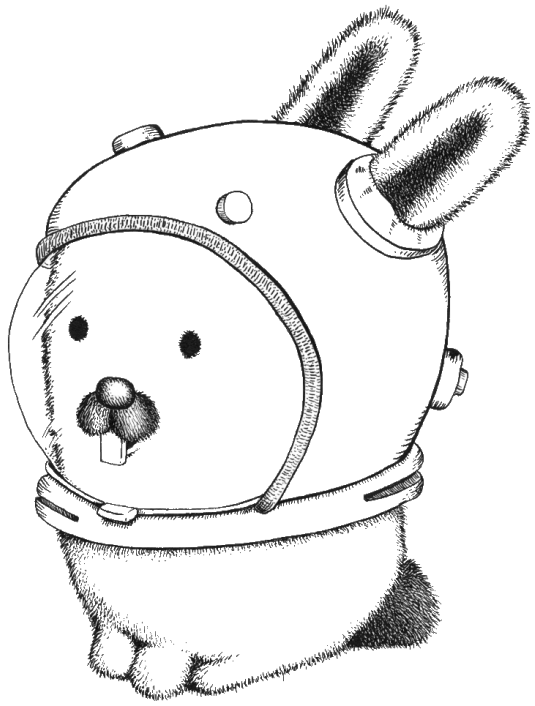
There is a persistent connection between the client and the server.



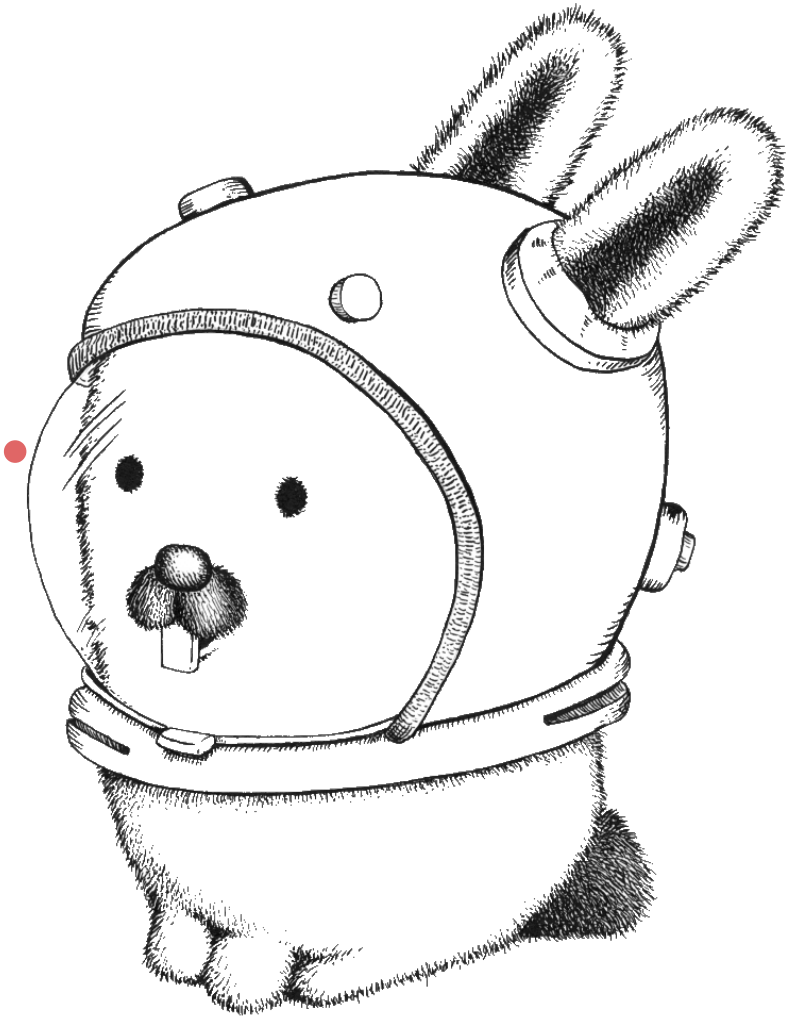
Both sides can talk.



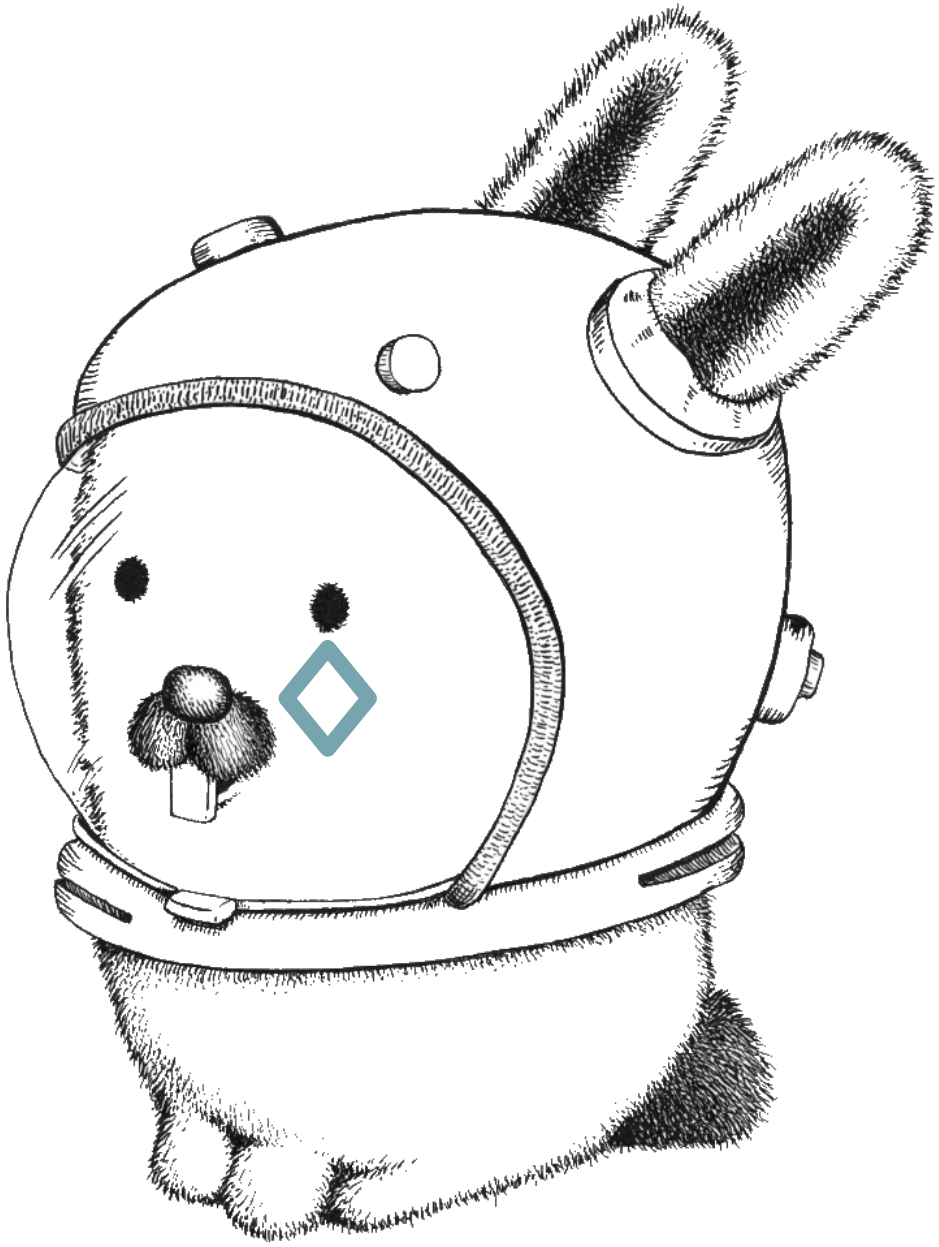
But I can't.



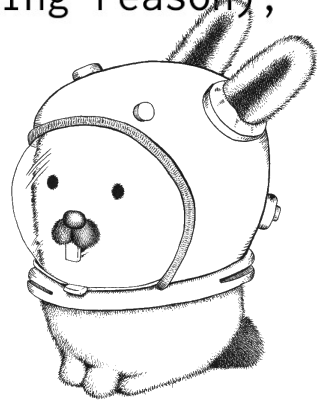
Cause I'm a dog.



Barf.



```
[Constructor(DOMString url, optional (DOMString or DOMString[]) protocols)]
interface WebSocket : EventTarget {
  readonly attribute DOMString url;

  // ready state
  const unsigned short CONNECTING = 0;
  const unsigned short OPEN = 1;
  const unsigned short CLOSING = 2;
  const unsigned short CLOSED = 3;
  readonly attribute unsigned short readyState;
  readonly attribute unsigned long bufferedAmount;

  // networking
  [TreatNonCallableAsNull] attribute Function? onopen;
  [TreatNonCallableAsNull] attribute Function? onerror;
  [TreatNonCallableAsNull] attribute Function? onclose;
  readonly attribute DOMString extensions;
  readonly attribute DOMString protocol;
  void close([Clamp] optional unsigned short code, optional DOMString reason);

  // messaging
  [TreatNonCallableAsNull] attribute Function? onmessage;
  attribute DOMString binaryType;
  void send(DOMString data);
  void send(ArrayBufferView data);
  void send(Blob data);
};
```



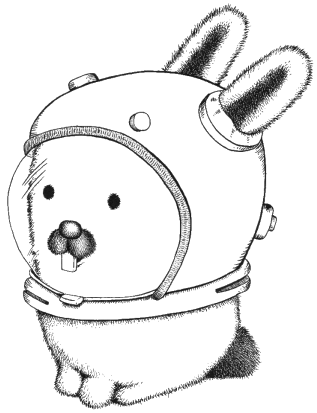
Create a connection with:

```
var connection =
  new WebSocket(url, proto)
```

The url should be absolute and can point cross domains.

```
var connection =
  new WebSocket(url, proto)
```

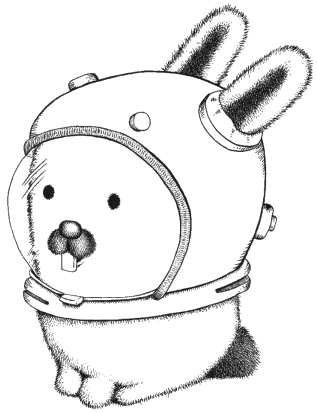
The WebSockets use their own transportation protocol and not HTTP.



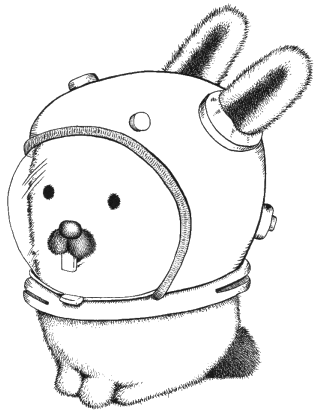
The protocol can be SOAP or undefined.

```
var connection =
  new WebSocket(url, proto)
```

The protocol argument is the messaging protocol. The only supported by the spec is SOAP.



```
[Constructor(DOMString url, optional (DOMString or DOMString[]) protocols)]
interface WebSocket : EventTarget {
  [...]

  // ready state
  const unsigned short CONNECTING = 0;
  const unsigned short OPEN = 1;
  const unsigned short CLOSING = 2;
  const unsigned short CLOSED = 3;
  readonly attribute unsigned short readyState;
  readonly attribute unsigned long bufferedAmount;

  [...]
};
```



The connection can be in the following states:

connection.readyState

CONNECTING (numeric value 0)

The connection has not yet been established.

OPEN (numeric value 1)

The WebSocket connection is established and communication is possible.

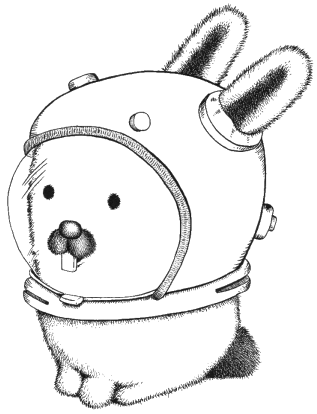
CLOSING (numeric value 2)

The connection is going through the closing handshake.

CLOSED (numeric value 3)

The connection has been closed or could not be opened.

```
[Constructor(DOMString url, optional (DOMString or DOMString[]) protocols)]
interface WebSocket : EventTarget {
  [...]

  // networking
  [TreatNonCallableAsNull] attribute Function? onopen;
  [TreatNonCallableAsNull] attribute Function? onerror;
  [TreatNonCallableAsNull] attribute Function? onclose;
  readonly attribute DOMString extensions;
  readonly attribute DOMString protocol;
  void close([Clamp] optional unsigned short code, optional DOMString reason);

  [...]
};
```



Remember the async execution?

```
connection.onopen =
  function(event) {}
```

Remember the async execution?

```
connection.onerror =
  function(event) {}
```

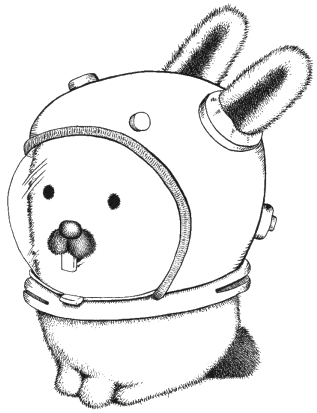
Remember the async execution?

```
connection.onclose =
  function(event) {}
```

Each function is called when the event happens.

```
connection.onclose =
  function(event) {}
```

```
[Constructor(DOMString url, optional (DOMString or DOMString[]) protocols)]
interface WebSocket : EventTarget {
  [...]

  // messaging
  [TreatNonCallableAsNull] attribute Function? onmessage;
  attribute DOMString binaryType;
  void send(DOMString data);
  void send(ArrayBufferView data);
  void send(Blob data);

  [...]
};
```

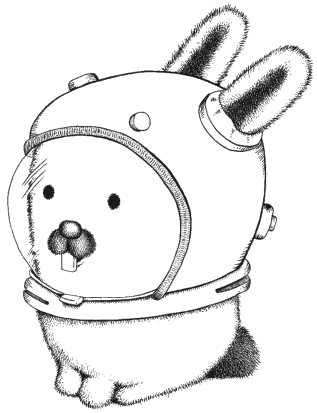


Sending messages to the server:

```
connection.send(message)
```

The messages can be binary and you will probably use JSON and not SOAP.

You are not obligated to follow the protocol.

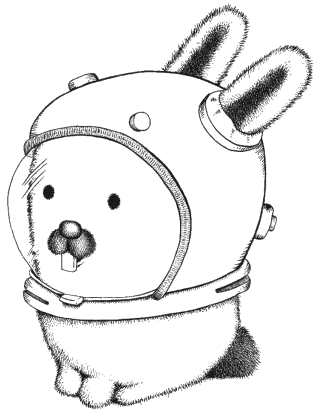


Receiving data from the server:

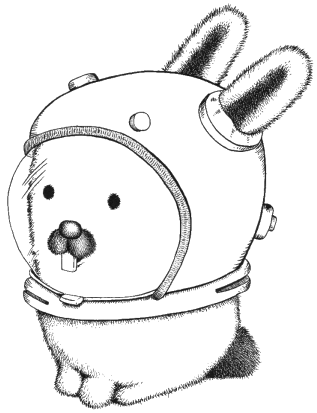
```
connection.onmessage =
  function(event) {}
```

Web Workers

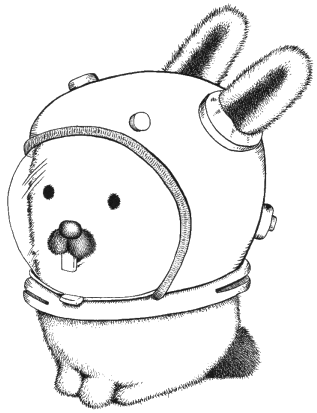
The Web Workers defines an API for running scripts in the background independently of any user interface scripts.



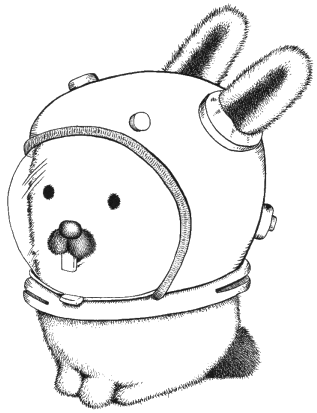
Workers allow long tasks to be executed without yielding to keep the page responsive.



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

  void postMessage(any message, optional sequence<Transferable> transfer);
  [TreatNonCallableAsNull] attribute Function? onmessage;
};
Worker implements AbstractWorker;

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



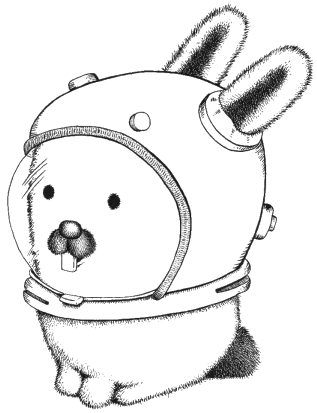
```
[Constructor(DOMString scriptURL)]
interface Worker : EventTarget {
  void terminate();

  void postMessage(any message, optional sequence<Transferable> transfer);
  [TreatNonCallableAsNull] attribute Function? onmessage;
};
Worker implements AbstractWorker;

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



Create a worker with:

```
var worker =
  new Worker(scriptURL)
```

The scriptURL should be relative and should point to a valid JavaScript.



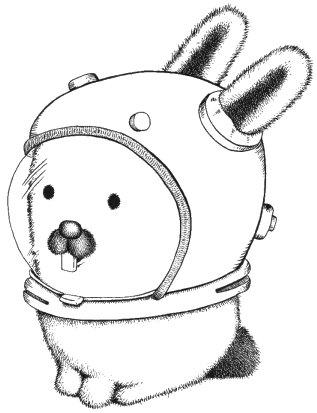
Communicate messages to the background script:

```
worker.postMessage(data)
```

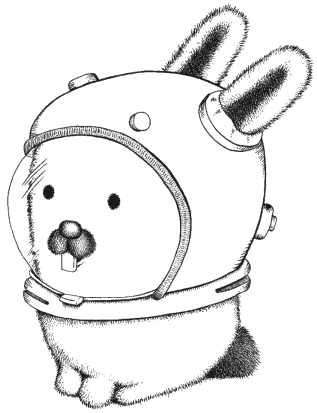
Receive messages from the background script:

```
worker.onmessage =
  function(event) {}
```

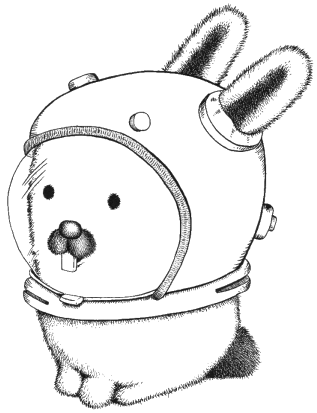
The data send between the worker and the background script and is serialized.



The serialization format is JSON for most of the current browsers.



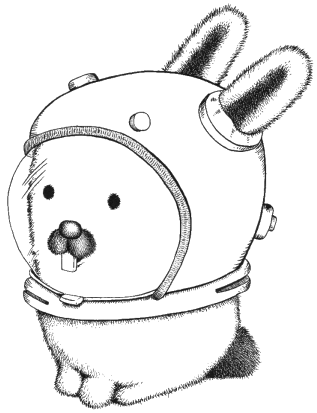
```
interface WorkerGlobalScope : EventTarget {
  readonly attribute WorkerGlobalScope self;
  readonly attribute WorkerLocation location;

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



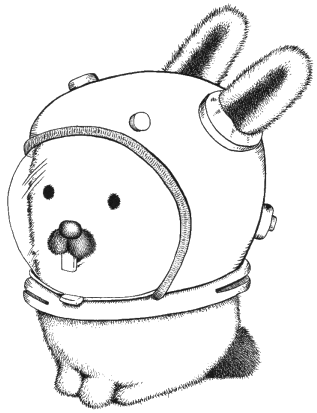
Receive messages in the background script:

```
self.addEventListener(type,
  function(event) {})
```

Because of the concurrent execution, some common JavaScript objects are not available.



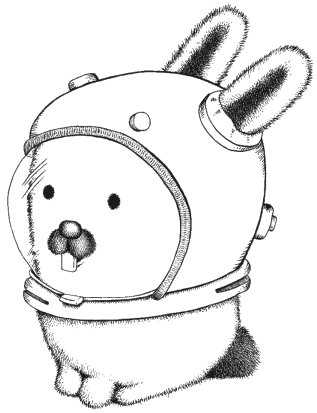
window & document & parent are not available, because they are not Thread Safe.



Post messages back to the worker:

```
self.postMessage(data)
```

In fact sef === this. That is the global context. Like window is in regular scripts.



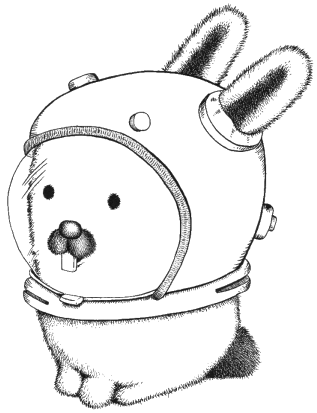
Receive messages in the background script:

```
addEventListener(type,
  function(event) {})
```

Post messages back to the worker:

```
postMessage(data)
```

Are equivalent to the ones we used before.



Terminate the background script execution:

```
self.close()
```

Terminate the background script execution:

```
worker.terminate()
```