---

title: Set up web3.js to use the Ethereum blockchain in JavaScript description: How to use a smart contract to interact with a token using the Solidity language author: "jdourlens" tags: ["web3.js", "javascript"] skill: beginner lang: en published: 2020-04-11 source: EthereumDev sourceUrl: https://ethereumdev.io/setup-web3js-to-use-the-ethereum-blockchain-in-javascript/ address: "0x19dE91Af973F404EDF5B4c093983a7c6E3EC8ccE"

In this tutorial, we'll see how to get started withweb3.js to interact with the Ethereum blockchain. Web3.js can be used both in frontends and backends to read data from the blockchain or make transactions and even deploy smart contracts.

The first step is to include web3.js in your project. To use it in a web page, you can import the library directly using a CDN like JSDeliver.

```html

```

If you prefer installing the library to use in your backend or a frontend project that uses build you can install it using npm:

```bash npm install web3 --save```

Then to import Web3.js into a Node.js script or Browserify frontend project, you can use the following line of JavaScript:

```js const Web3 = require("web3")```

Now that we included the library in the project we need to initialize it. Your project needs to be able to communicate with the blockchain. Most Ethereum libraries communicate with a node through RPC calls. To initiate our Web3 provider, we'll instantiate a Web3 instance passing as the constructor the URL of the provider. If you have a node or ganache instance running on your computer it will look like this:

```js const web3 = new Web3("http://localhost:8545")```

If you'd like to directly access a hosted node you can find options onnodes as a service.

```js const web3 = new Web3("https://cloudflare-eth.com")```

To test that we correctly configured our Web3 instance, we'll try to retrieve the latest block number using the`getBlockNumber` function. This function accepts a callback as a parameter and returns the block number as an integer.

```js var Web3 = require("web3") const web3 = new Web3("https://cloudflare-eth.com")

web3.eth.getBlockNumber(function (error, result) { console.log(result) }) ```

If you execute this program, it will simply print the latest block number: the top of the blockchain. You can also use `await/async` function calls to avoid nesting callbacks in your code:

```js async function getBlockNumber() { const latestBlockNumber = await web3.eth.getBlockNumber() console.log(latestBlockNumber) return latestBlockNumber }

getBlockNumber() ```

You can see all the functions available on the Web3 instance inthe official web3.js documentation.

Most of Web3 libraries are asynchronous because in the background the library makes JSON RPC calls to the node which send backs the result.

If you are working in the browser, some wallets directly inject a Web3 instance and you should try to use it whenever possible especially if you plan to interact with the user's Ethereum address to make transactions.

Here is the snippet to detect if a MetaMask wallet is available and try to enable it if it is. It will later allow you to read the user's balance and enable them to validate transactions you'd like to make them do on the Ethereum blockchain:

```js if (window.ethereum != null) { state.web3 = new Web3(window.ethereum) try { // Request account access if needed await window.ethereum.enable() // Accounts now exposed } catch (error) { // User denied account access... } }```
```

Alternatives to web3.js like Ethers.js do exist and are also commonly used. In the next tutorial we'll see how to easily listen to new incoming blocks on the blockchain and see what they contain.