To create a proxy object, we use the Proxy constructor - new Proxy();. The proxy constructor takes two items:

* the object that it will be the proxy for
* an object containing the list of methods it will handle for the proxied object

The second object is called the **handler**.

**A Pass Through Proxy**

The simplest way to create a proxy is to provide an object and then an empty handler object.

**var** richard = {status: 'looking for work'};

**var** agent = **new** Proxy(richard, {});

agent.status; *// returns 'looking for work'*

The above doesn't actually do anything special with the proxy - it just passes the request directly to the source object! If we want the proxy object to actually intercept the request, that's what the handler object is for!

The key to making Proxies useful is the handler object that's passed as the second object to the Proxy constructor. The handler object is made up of a methods that will be used for property access. Let's look at the get:

**Get Trap**

The get trap is used to "intercept" calls to properties:

**const** richard = {status: 'looking for work'};

**const** handler = {

get(target, propName) {

console.log(target); *// the `richard` object, not `handler` and not `agent`*

console.log(propName); *// the name of the property the proxy (`agent` in this case) is checking*

}

};

**const** agent = **new** Proxy(richard, handler);

agent.status; *// logs out the richard object (not the agent object!) and the name of the property being accessed (`status`)*

In the code above, the handler object has a get method (called a "trap" since it's being used in a Proxy). When the code agent.status; is run on the last line, because the get trap exists, it "intercepts" the call to get the status property and runs the get trap function. This will log out the target object of the proxy (the richard object) and then logs out the name of the property being requested (the status property). *And that's all it does!* It doesn't actually log out the property! This is important - *if a trap is used, you need to make sure you provide all the functionality for that specific trap*.

**Accessing the Target object from inside the proxy**

If we wanted to actually provide the real result, we would need to return the property on the target object:

**const** richard = {status: 'looking for work'};

**const** handler = {

get(target, propName) {

console.log(target);

console.log(propName);

**return** target[propName];

}

};

**const** agent = **new** Proxy(richard, handler);

agent.status; *// (1)logs the richard object, (2)logs the property being accessed, (3)returns the text in richard.status*

Notice we added the return target[propName]; as the last line of the get trap. This will access the property on the target object and will return it.

**Having the proxy return info, directly**

Alternatively, we could use the proxy to provide direct feedback:

**const** richard = {status: 'looking for work'};

**const** handler = {

get(target, propName) {

**return** `He's following many leads, so you should offer a contract as soon as possible!`;

}

};

**const** agent = **new** Proxy(richard, handler);

agent.status; *// returns the text `He's following many leads, so you should offer a contract as soon as possible!`*

With this code, the Proxy doesn't even check the target object, it just directly responds to the calling code.

So the get trap will take over whenever any property on the proxy is accessed. If we want to intercept calls to *change* properties, then the set trap needs to be used!

The set trap is used for intercepting code that will *change a property*. The set trap receives: the object it proxies the property that is being set the new value for the proxy

**const** richard = {status: 'looking for work'};

**const** handler = {

set(target, propName, value) {

**if** (propName === 'payRate') { *// if the pay is being set, take 15% as commission*

value = value \* 0.85;

}

target[propName] = value;

}

};

**const** agent = **new** Proxy(richard, handler);

agent.payRate = 1000; *// set the actor's pay to $1,000*

agent.payRate; *// $850 the actor's actual pay*

In the code above, notice that the set trap checks to see if the payRate property is being set. If it is, then the proxy (the agent) takes 15 percent off the top for her own commission! Then, when the actor's pay is set to one thousand dollars, since the payRate property was used, the code took 15% off the top and set the actual payRate property to 850;

**Other Traps**

So we've looked at the get and set traps (which are probably the ones you'll use most often), but there are actually a total of 13 different traps that can be used in a handler!

1. [the get trap](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/Proxy/handler/get) - lets the proxy handle calls to property access
2. [the set trap](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/Proxy/handler/set) - lets the proxy handle setting the property to a new value
3. [the apply trap](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/Proxy/handler/apply) - lets the proxy handle being invoked (the object being proxied is a function)
4. [the has trap](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/Proxy/handler/has) - lets the proxy handle the using in operator
5. [the deleteProperty trap](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/Proxy/handler/deleteProperty) - lets the proxy handle if a property is deleted
6. [the ownKeys trap](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/Proxy/handler/ownKeys) - lets the proxy handle when all keys are requested
7. [the construct trap](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/Proxy/handler/construct) - lets the proxy handle when the proxy is used with the new keyword as a constructor
8. [the defineProperty trap](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/Proxy/handler/defineProperty) - lets the proxy handle when defineProperty is used to create a new property on the object
9. [the getOwnPropertyDescriptor trap](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/Proxy/handler/getOwnPropertyDescriptor) - lets the proxy handle getting the property's descriptors
10. [the preventExtenions trap](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/Proxy/handler/preventExtensions) - lets the proxy handle calls to Object.preventExtensions() on the proxy object
11. [the isExtensible trap](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/Proxy/handler/isExtensible) - lets the proxy handle calls to Object.isExtensible on the proxy object
12. [the getPrototypeOf trap](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/Proxy/handler/getPrototypeOf) - lets the proxy handle calls to Object.getPrototypeOf on the proxy object
13. [the setPrototypeOf trap](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/Proxy/handler/setPrototypeOf) - lets the proxy handle calls to Object.setPrototypeOf on the proxy object

As you can see, there are a lot of traps that let the proxy manage how it handles calls back and forth to the proxied object.