**Serialization**

Serialization converts an in-memory data structure to a value that can be stored or transferred.

Let’s say we have a users object in a JavaScript script that contains a total user count and a list of users, as follows:

const usersObject = {  
 "total": 2,  
 "users": [  
 {  
 "name": "Patrick"  
 },  
 {  
 "name": "Michael"  
 }  
 ]  
}

When the JavaScript run time executes the above code, it’ll create a data structure in memory similar to the below. The engine will store each object and array at a separate location in memory and provide references to the places where they are used:

***object\_in\_memory\_a***  
{  
 name: "Patrick"  
}***object\_in\_memory\_b***  
{  
 name: "Michael"  
}***array\_in\_memory\_c***  
[  
 ***object\_in\_memory\_a***, ***object\_in\_memory\_b***]***object\_in\_memory\_d***  
{  
 total: 2,  
 users: ***array\_in\_memory\_c***  
}

The above isn’t something that can be easily transmitted to another machine or used by the same machine later after the program’s process ends. To do this, we must serialize the data. That is, we must convert the in-memory data structure into a series of bytes (usually ASCII characters) that record the data structure in a recoverable format.

In our JavaScript example, we can serialize the usersObject by passing it into the JSON.stringify() function, as we do below.

Command:

JSON.stringify(usersObject);

Result:

**"{"total":2,"users":[{"name":"Patrick"},{"name":"Michael"}]}"**

As we can see, the in-memory object has been converted to a string. This string can now be transferred to other machines or stored to the disk for later use.

**Deserialization**

In our example, we know we can pass the serialized result to another machine or retrieve it later ourselves if we saved it to disk.

Let’s say this serialization occurs on a server, and we want to pass it to a browser client when a request for system users is made. The client will receive the result of our serialization as follows:

**"{"total":2,"users":[{"name":"Patrick"},{"name":"Michael"}]}"**

The data is great, but it’s a string.

At this point, the browser client isn’t able to use this data to know how many users there are or their names.

To have this ability, we must take this serialized string and convert it into an in-memory data structure. The conversion from serialized string to an in-memory data structure is deserialization. We can accomplish this with the JavaScript JSON.parse() function.

const usersObject = JSON.parse(responseFromServer);

After the deserialization, usersObject will become an identical in-memory data structure to the one we saw on the server side:

***object\_in\_memory\_a***  
{  
 name: "Patrick"  
}***object\_in\_memory\_b***  
{  
 name: "Michael"  
}***array\_in\_memory\_c***  
[  
 ***object\_in\_memory\_a***, ***object\_in\_memory\_b***]***object\_in\_memory\_d***  
{  
 total: 2,  
 users: ***array\_in\_memory\_c***  
}

Now that the browser client has an in-memory data structure, we have the ability to navigate it.

We can now know the number of users along with their names.

console.log(usersObject.total); // 2  
console.log(usersObject.users[0].name); // Patrick  
console.log(usersObject.users[1].name); // Michael

**Summary**

Serialization takes an in-memory data structure and converts it into a series of bytes that can be stored and transferred.

Deserialization takes a series of bytes and converts it to an in-memory data structure that can be consumed programmatically.

‘