# Fetch API - Headers

Fetch API provides a special interface known as the Headers interface to perform various operations like setting, adding, retrieving and removing headers from the request and response's headers list. The Headers objects are initially empty or may contain zero or more name-value pairs. You can add header names in the headers object using the append() method. This interface provides various methods to perform actions on the Headers object.

## Constructor

To create a headers object we can use the Headers() constructor along with a new keyword. This constructor may or may not contain parameters.

## Syntax

```
const newHeader = New Headers()
Or
const newHeader = New Headers(init)
```

The Headers() constructor contains only one optional parameter that is init. It is an object which contains HTTP headers that you want to pre-populate your headers object. The value of this parameter is a string value or an array of name-value pairs.



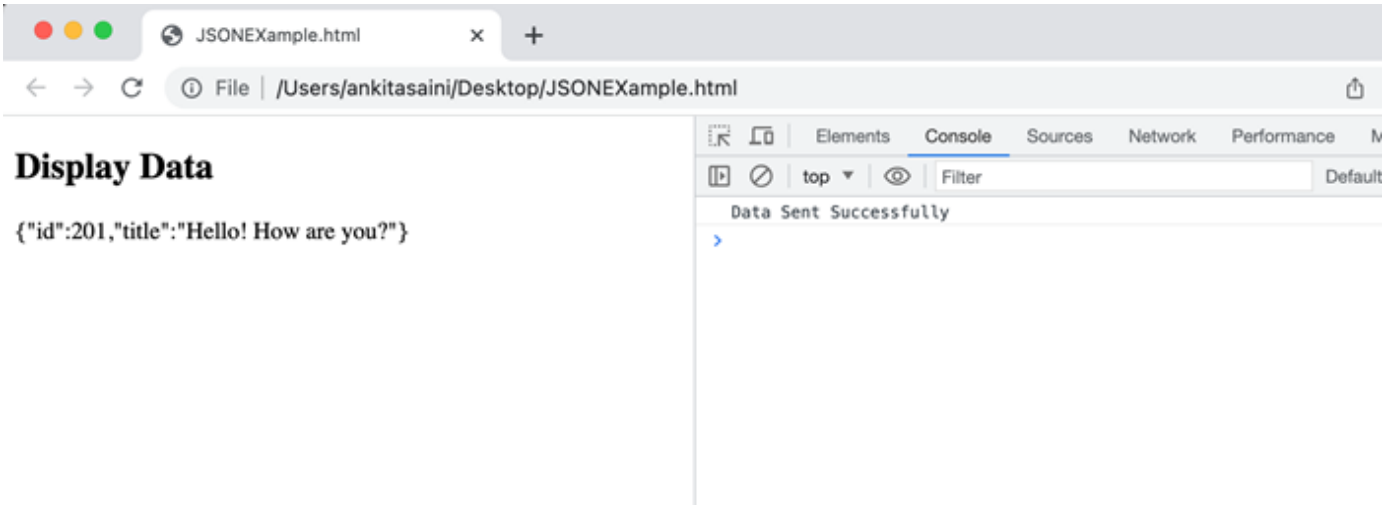
## Example 1

In the following program, we are sending data to the server. So for that, we create a new headers object using the Header() constructor and then add name-value pairs using the append() function. After that, we make a fetch() request with the fetch() function which includes the POST method, the headers object that we created earlier to add headers to the request, and the body of the request. Now, after sending the request to the server now we use the then() function to handle the response. If we encounter an error, then that error is handled by the catch() function.

```
<!DOCTYPE html>
<html>
<body>
<script>
   // Creating Headers object
   const myheaders = new Headers();

   // Adding headers to the Headers object
   myheaders.append('Content-Type', 'application/json');
   myheaders.append('Authorization', 'Bearer token123');

   // Sending data using POST request
   fetch("https://jsonplaceholder.typicode.com/todos", {
      // Adding POST request
      method: "POST",

      // Adding headers
      headers:myheaders,

      // Adding body which we want to send
      body: JSON.stringify({
         id: 32,
         title: "Hello! How are you?",
      })
   })

   // Converting received information into JSON
   .then(response => response.json())
   .then(myData => {
      // Display the sent data
      console.log("Data Sent Successfully");

      // Display output
      document.getElementById("manager").innerHTML = JSON.stringify(myData);
   });
</script>
   <h2>Display Data</h2>
   <div>
      <!-- Displaying retrevie data-->
      <p id = "manager"></p>
   </div>
```

```
</body>
</html>
```

## Output




## Methods

The following are the commonly used method of Header interface –

| Sr.No. | Method Name & Description |
|---|---|
| 1 | **Headers.append()**<br>This method is used to append a new value inside the existing headers object. Or it can add a header if it does not exist. |
| 2 | **Headers.delete()**<br>This method is used to delete a header from the Headers object. |
| 3 | **Headers.enteries()**<br>This method provides an iterator which allows us to iterate through all the key/value pairs present in the given object. |
| 4 | **Headers.forEach()**<br>This method executes once for each key/value pair present in the Headers object. |
| 5 | **Headers.get()**<br>This method is used to find all the string sequence of all the values of the header present inside the Header object. |
| 6 | **Headers.getSetCookie()** |

| | |
|---|---|
| | This method returns an array which contains all the values of Set-Cookie headers related to the response. |
| 7 | **Headers.has()**<br>This method returns a boolean value which checks if the current Headers object contains the specified header or not. |
| 8 | **Headers.keys()**<br>This method is used to iterate through all the keys of the key-value pairs present in the given object. |
| 9 | **Headers.set()**<br>This method is used to set a new value for the existing Headers object. Or can add a header if it doesn't exist. |
| 10 | **Headers.values()**<br>This method is used to iterate through all the values of the key-value pairs present in the given object. |

## Example 2

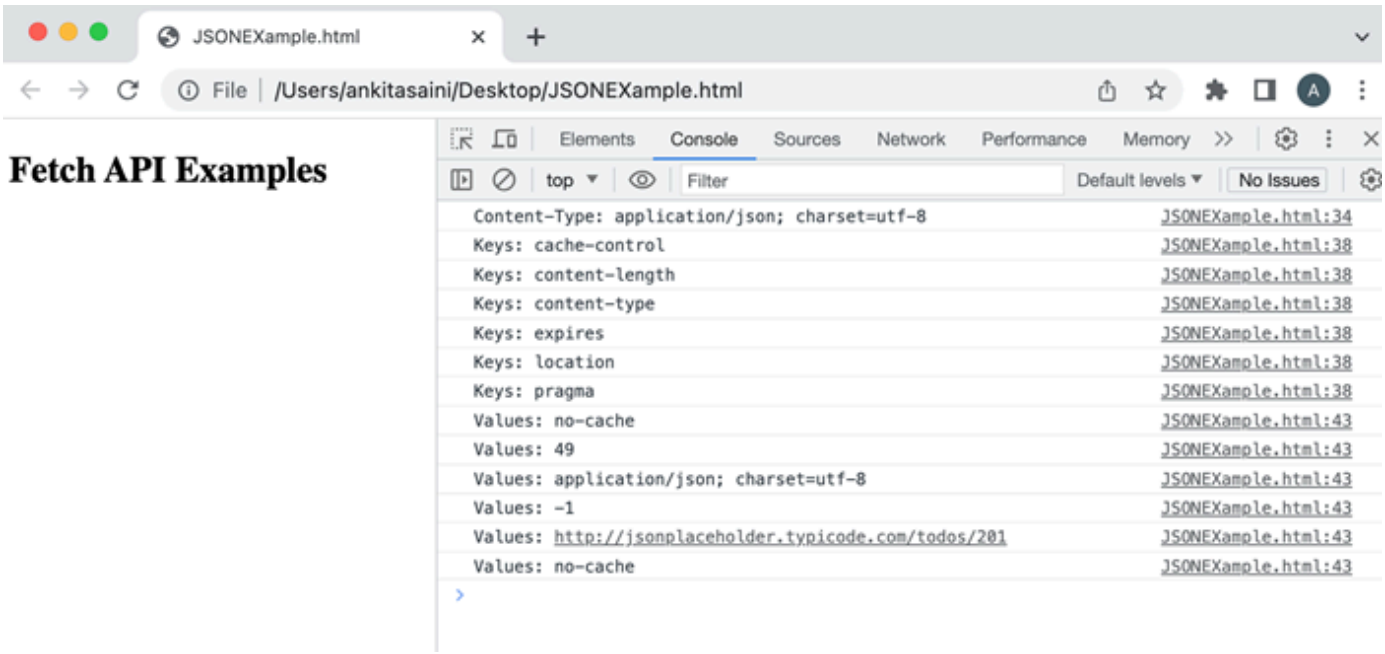
In the following program, we use the methods such as append(), get(), keys() and values() provided by the Headers interface.

</> Open Compiler

```
<!DOCTYPE html>
<html>
<body>
<script>
   // Creating Headers object
   const myheaders = new Headers();

   // Adding headers to the Headers object
   myheaders.append('Content-Type', 'application/json');
   myheaders.append('Authorization', 'Bearer token123');

   // Sending data using POST request
   fetch("https://jsonplaceholder.typicode.com/todos", {
      // Adding POST request
      method: "POST",

      // Adding headers
      headers:myheaders,
```

```
      // Adding body which we want to send
      body: JSON.stringify({
         id: 32,
         title: "Hello! How are you?",
      })
   })
   // Converting received information into JSON
   .then(response => {
      // Header also returned in response
      // Accessing response header
      const resHeader = response.headers;

      // Getting content type value of the response header
      // Using get() function
      const contentTypeValue = resHeader.get("Content-Type");
      console.log("Content-Type:", contentTypeValue);

      // Getting all the keys present in the
      //  key-value pairs in response header
      // Using keys() function
      const headerKeys = resHeader.keys();
      for(const res of headerKeys){
      console.log("Keys:", res);
      }
      // Getting all the values present in the
      //  key-value pairs in response header
      // Using Values() function
      const headerValues = resHeader.values();
         for(const resVal of headerValues){
         console.log("Values:", resVal);
      }
   });
</script>
   <h2>Fetch API Examples</h2>
</body>
</html>
```

## Output




## Conclusion

So this is how we use the Header interface in Fetch API. It provides various methods to manipulate, access and iterate over the headers. We can also retrieve Header objects from the Request and response using Request.headers and Response.headers properties. Now in the next article, we will learn about the Request interface.