7.9 XMLHttpRequest (Ajax)

Ajax introduction

A normal HTTP request is triggered by clicking on a hyperlink or submitting a form, after which the browser may appear non-responsive while the browser waits for the server response. When the browser receives the response, the entire webpage is replaced with the HTML in the response. This delay may be undesirable for some web applications and may annoy users if the delay is long.

Ajax (***Asynchronous JavaScript and XML***) is a technique to asynchronously communicate with a server and update a webpage once the response is received, without reloading the whole webpage. An ***asynchronous request*** occurs when the web application sends a request to the server and continues running without waiting for the server response. Although the "x" in Ajax stands for "XML", Ajax is used to transmit plain text, HTML, XML, and JSON.

XMLHttpRequest is an object for communicating with web servers using Ajax. Using the XMLHttpRequest object allows web browsers to hide the communication latency and continue to provide a responsive user interface while waiting for a server response. The XMLHttpRequest object defines handlers for events that occur during the request/response cycle. Ex: A response arrives at the browser, an error occurs during a request, etc. Using event-driven programming, the web application can continue providing a responsive interface and does not need to wait for a response from the server. The web application later updates the page once the response is received.

PARTICIPATION ACTIVITY 7.9.1: Asynchronous HTTP request.

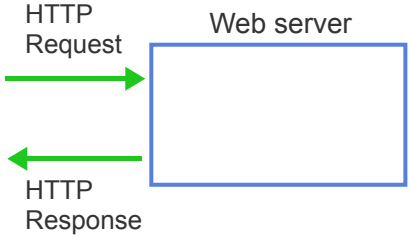
1 2 3 4 5 2x speed

```
<h1>Movie Information</h1>
<p id="movieinfo">

   <cite>Star Wars</cite>: Rated PG, released 1977

</p>
```

Movie Information

Star Wars: Rated PG, released 1977

```
let movieinfo = document.getElementById("movieinfo");
let xhr = new XMLHttpRequest();
xhr.addEventListener("load", function() {
   movieinfo.innerHTML = xhr.response;
});
xhr.open("GET", "starwars.html");
xhr.send();
```



The browser displays the Star Wars information.

Captions

1. The browser initially renders the webpage with no movie information.
2. The XMLHttpRequest object sends a GET request for starwars.html to the web server. The request is sent asynchronously.
3. The web server asynchronously responds to the browser with the contents of starwars.html.
4. The load event handler is called when starwars.html is loaded. The paragraph's inner HTML is replaced with the HTML contents of starwars.html.
5. The browser displays the Star Wars information.

Feedback?

Note

For security reasons, browsers limit Ajax requests to the web server from which the JavaScript was downloaded. Ex: JavaScript downloaded from http://instagram.com may only make Ajax requests to instagram.com. A ***cross-origin HTTP request*** *is a request made to another domain. Ex: An Ajax request from JavaScript downloaded from instagram.com to yahoo.com is a cross-origin HTTP request. Browsers can make cross-origin HTTP requests using a number of techniques including proxy servers, Cross-Origin Resource Sharing (CORS), and JSON with Padding (JSONP).*

Using XMLHttpRequest

The steps for using the XMLHttpRequest API are:

1. Create a new XMLHttpRequest object.
2. Assign handlers to the desired events via the `addEventListener()` method. The `addEventListener()` method takes two arguments: the event name and the event handler, code that should execute when the event occurs. If the handlers are not set up prior to calling the `open()` method, the progress events will not execute.
3. Initialize a connection to a remote resource using the `open()` method. The ***open()*** method takes two arguments: the HTTP request type and the URL for the resource. Most browsers only support "GET" and "POST" request types.
4. Modify the default HTTP request headers if needed with the ***setRequestHeader()*** method. Ex:
   `xhr.setRequestHeader("Content-Type", "application/x-www-form-urle`

sets the `Content-Type` header so a URL-encoded string may be sent in a POST request.

5. Send the HTTP request via the ***send()*** method. For POST requests, the data to be sent with the request is passed as the argument to the `send()` method.

PARTICIPATION ACTIVITY 7.9.2: Identify steps in making an Ajax request.

Match the JavaScript code with the description.

```
(a)  function responseReceivedHandler() {
       console.log("handling response: " + this.responseText);
     }

(b)  let xhr = new XMLHttpRequest();
(c)  xhr.addEventListener("load", responseReceivedHandler);
(d)  xhr.open("GET", "http://www.example.org/example.html");
(e)  xhr.send();
```

If unable to drag and drop, refresh the page.

Create load event handler	(a) An event handler is created by defining a function that will be called when a particular event occurs.	Correct
Create new XMLHttpRequest object	(b) A new XMLHttpRequest object is created using `new XMLHttpRequest()`.	Correct
Register load event handler	(c) `addEventListener()` registers `responseReceivedHandler()` as the function to be called when the load event occurs.	Correct
Initialize server connection	(d) The server connection is initialized when the XMLHttpRequest object's `open()` method is called.	Correct
Send HTTP request	(e) The HTTP request is sent by calling the XMLHttpRequest's	Correct

`send()` method.

Reset

Feedback?

XMLHttpRequest result handlers

Good practice is to use a result handler for each specific result to separate functionality for each Ajax event. Ex: Error handling, progress bars, updating the user interface on success, etc. The XMLHttpRequest result handlers are:

- The ***load*** handler is called when the exchange between the browser and server has completed. From the browser's perspective, the server received the request and responded. However, the request might not have been successful because of a problem such as a non-existent webpage. The HTTP status code must be examined to check which type of response was received. Ex: 200 vs. 404. The load, error, and abort handlers are mutually exclusive and are called after any progress handlers.
- The ***error*** handler is called when the browser does not receive an appropriate response to a request. Ex: The browser is unable to connect to the server, the connection between browser and server is cut in the middle of a response, etc.
- The ***abort*** handler is called when the browser is told to stop a request/response that is still in progress. Ex: The user closes the webpage that made the request.
- The ***timeout*** handler is called if the browser takes too much time to fully receive a response to a request. The timeout is an optional value that can be provided before the request is made. By default, the browser does not provide a timeout for a request.

Note

*The **readystatechange** handler relates to any change in the XMLHttpRequest. When XMLHttpRequest was originally defined, readystatechange was the only handler defined. As a result, many Ajax examples on the Internet only use readystatechange and do not include other handlers. Load, error, abort, and timeout are replacements for readystatechange.*

PARTICIPATION ACTIVITY 7.9.3: Match the event handlers to their descriptions.

If unable to drag and drop, refresh the page.

load	Response received successfully. Load event occurs when the response was received and no error occurred.	Correct
error	Sending request failed. An error could occur because the URL is invalid or the web server crashed.	Correct
abort	Browser request was stopped. Calling the abort( ) method is one way to cause an abort event.	Correct
timeout	Request took too long to complete. Programmers frequently specify a time limit for completing a request. If the request is not completed within that time period, the timeout event handler is called. The timeout handler might then abort the request.	Correct

Reset

Feedback?

XMLHttpRequest progress handlers

XMLHttpRequest progress handlers are:

- The ***loadstart*** handler is called when the browser begins to send a request. The loadstart handler is called before any other XMLHttpRequest handler.
- The ***loadend*** handler is called after the browser receives the response. The loadend handler is called upon both response success and failure, and is called after all other XMLHttpRequest handlers.
- The ***progress*** handler is called one or more times while a response is being received by the client. Progress handlers are called before result handlers. The progress handler can be used to provide a data download progress indicator to the user. A similar handler is available to provide an indicator for uploaded data.

PARTICIPATION ACTIVITY 7.9.4: XMLHttpRequest event handler order.

Arrange the handlers in the order the handlers are called.

If unable to drag and drop, refresh the page.

loadstart	First Called when the browser begins to send a request.	Correct
progress	Second Called one or more times while a request response is being received.	Correct
result handler	Third Called after the progress handler to indicate the result status of the response.	Correct
loadend	Fourth Called when the browser terminates a response with either success or failure.	Correct

Reset

Feedback?

Attributes for determining XMLHttpRequest success

The XMLHttpRequest object has attributes for checking the status of a response, which are usually used in the load handler and used to update the DOM.

- The ***status*** attribute is the numeric status code returned in the response.
- The ***statusText*** attribute is the descriptive text describing the status attribute.

Checking the `status` attribute of a response is important because the status code identifies the specific reason for a failure response. Ex: 403 means the requestor does not have permission to access the requested resource, and 404 means the requested resource was not found.

A common error is to assume that a failure response causes the error handler to be called. If the server properly sends the failure response to the browser, the browser will treat the response as successful and call the load handler. The error handler is only called if the response is not fully received by the browser.

Table 7.9.1: Common HTTP response status codes.

Status code	Meaning
200	HTTP request successful
3XX	General form for request redirection errors
301	Resource permanently moved, the new URL is provided
4XX	General form for client errors
400	Bad request. Ex: Incorrect request syntax
401	Unauthorized request. Ex: Not properly authenticated.
403	Request forbidden. Ex: User does not have necessary permissions.
404	Not found. Ex: Requested resource does not exist.
5XX	General form for server error codes
500	Internal server error. Ex: Server-side code crashed.
503	Service unavailable. Ex: Webpage is temporarily unavailable due to site maintenance.

Feedback?

Accessing Ajax response data

The XMLHttpRequest object provides multiple ways to access the response data.

- The ***response*** attribute is the response body, which is parsed by the browser according to the `responseType` attribute.
- The ***responseText*** attribute is the plain text version of the response.
- The ***responseXML*** attribute is the XML DOM version of the response. The `responseXML` attribute is only available as a DOM object if the response is a valid and

correctly formatted XML document.

The ***responseType*** attribute is set by the programmer to let the browser know the expected response data format.

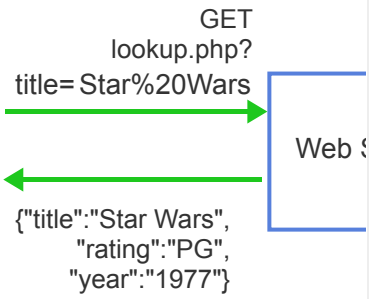
- If the `responseType` attribute is set to `"json"`, then the browser parses the entire response as a JSON object and sets the `response` attribute to the JSON object.
- If the `responseType` attribute is either `""` or `"text"`, the browser leaves the response unprocessed, and the `response` attribute contains the same value as `responseText`.
- If the `responseType` attribute is `"document"`, the browser assumes the response is an XML document, and the `response` attribute contains the same value as `responseXML`.

PARTICIPATION ACTIVITY 7.9.5: Creating a query string and loading JSON.

1 2 3 4 5 6 7 2x speed

```
<body>
  <label for="title">Title:</label>
  <input type="text" id="title"><br>
  <button id="search">Search</button>
  <p id="movieinfo">
    <cite>Star Wars</cite>: Rated PG,
    released in 1977
  </p>
</body>
```


```
let searchBtn = document.getElementById("search");
searchBtn.addEventListener("click", function() {
   let xhr = new XMLHttpRequest();
   xhr.addEventListener("load", responseReceivedHandler);
   xhr.responseType = "json";
   let title = document.getElementById("title");
   let queryString = "title=" +
                     encodeURIComponent(title.value);
   xhr.open("GET", "lookup.php?" + queryString);
   xhr.send();
});

function responseReceivedHandler() {
   let movieInfo = document.getElementById("movieinfo");
   if (this.status === 200) {
      let movie = this.response;
      movieInfo.innerHTML = "<cite>" + movie.title +
                  "</cite>: Rated " + movie.rating +
                  ", released in " + movie.year;
   } else {
      movieInfo.innerHTML = "Movie data unavailable.";
   }
}
```



The movie information is placed in the paragraph, and the browser renders the HTML.

Captions

1. The user types the title "Star Wars" and presses the Search button, causing the Search button's click handler to execute.
2. xhr.responseType is set to "json" so that the JSON sent to the browser in the Ajax response will be automatically converted into a JavaScript object.
3. A query string is constructed using the text from the text box. encodeURIComponent() converts "Star Wars" into a string with no spaces.
4. An asynchronous HTTP request to lookup.php with a query string is sent to the web server.
5. Web server looks up "Star Wars" in a database and sends back a JSON response with information about the movie.
6. The load handler verifies the response's status code is 200 and accesses the movie object from this.response, which was created from the JSON response.
7. The movie information is placed in the paragraph, and the browser renders the HTML.

Feedback?

PARTICIPATION ACTIVITY 7.9.6: Ajax and JSON.

Refer to the animation above.

1) If the `responseType` attribute is set to "json", what attribute contains the parsed JSON object when the response is received?

- `response`
- `responseText`
- `responseXML`

Correct

The `response` attribute contains the parsed JSON object.

2) What query string is created for the Ajax request when the user enters *Pride & Prejudice*?

- `title=Pride & Prejudice`
- `title=Pride%20&%20Prejudice`
- `title=Pride%20%26%20Prejudice`

Correct

The ASCII hex value of a space is 20, and ampersand is 26, so `encodeURIComponent()` converts the space characters to %20 and ampersand to %26.

3) What does the webpage display if lookup.php was accidentally

Correct

The `this.status` property in `responseReceivedHandler()` is 404 when the Ajax

misspelled lookup.html, and no lookup.html file exists?

- Nothing.
- The movie information for the given movie title.
- "Movie data unavailable."

request attempts to load a non-existing resource.

4) If lookup.php expects the movie title to be POSTed, the calls to `open()` and `send()` must be modified, and one more line of code must be added. What is the missing line of code?

```
xhr.open("POST", "lookup.php");
// Missing line
xhr.send(queryString);
```

- `xhr.responseType = "text";`
- `xhr.setRequestHeader("Content-Type", "application/x-www-form-urlencoded");`
- `xhr.addEventListener("error", errorHandler);`

Correct

The `Content-Type` request header must be set properly for the URL-encoded data to be POSTed to lookup.php.

Feedback?

PARTICIPATION ACTIVITY 7.9.7: Ajax practice.

The URL `https://wp.zybooks.com/weather.php?zip=XXXXX`, where XXXXX is a five digit ZIP code, returns JSON containing a randomly produced forecast for the given ZIP code. If the ZIP code is not given or is not five digits, the JSON response indicates the ZIP code is not found.

Successful request	Unsuccessful request

```
{
   "success": true,
   "forecast": [
      { "high": 90, "low": 72, "desc":
"sunny" },
      { "high": 92, "low": 73, "desc":
"mostly sunny" },
      { "high": 87, "low": 64, "desc":
"rain" },
      { "high": 88, "low": 65, "desc":
"cloudy" },
      { "high": 90, "low": 68, "desc":
"partly cloudy" }
   ]
}
```

```
{
   "success": false,
   "error": "ZIP code
not found"
}
```

Enter any ZIP code in the webpage below, and press the Search button. When Search is pressed, an Ajax request is made to the URL above using the ZIP code entered in the form. The raw JSON response is displayed in the webpage, which is not ideal.

Make the following changes:

1. Modify `getForecast()` to specify that a JSON response is expected before calling `xhr.send()`:

   ```
   xhr.responseType = "json";
   ```

2. Replace the code that appends the raw JSON to the `html` string with code that loops through the `forecast` array and produces a numbered list with each day's forecast:

   ```
   //html += this.response;
   html += "<ol>";
   for (let day of this.response.forecast) {
      html += `<li>${day.desc}: high is ${day.high}, low is
   ${day.low}</li>`;
   }
   html += "</ol>";
   ```

3. Render the webpage, and verify the changes you have made work correctly to show the weather for the ZIP code you enter.

4. Modify the code to display an appropriate error message if the Ajax response indicates the ZIP code is not found.

```
let html = "";
if (this.response.success) {
   html += "<h1>Forecast</h1>";
   html += "<ol>";
   for (let day of this.response.forecast) {
      html += `<li>${day.desc}: high is ${day.high}, low is
${day.low}</li>`;
   }
   html += "</ol>";
}
else {
   html = `<h1>Error: ${this.response.error}</h1>`;
}
```

5. Render the webpage, and verify that entering a bad ZIP code like "abc" produces an error message.

HTML | JavaScript

```
<!DOCTYPE html>
<html lang="en">
<head>
   <title>Weather Forecast</title>
</head>
<body>
   <p>
      <label for="zip">ZIP code:</label>
      <input type="text" id="zip" maxlength="5">
      <button id="search">Search</button>
   </p>
   <div id="forecast"></div>
</body>
</html>

```

Render webpage | Reset code

Your webpage

ZIP code: Search

Expected webpage

ZIP code: Search

▼ View solution

Explain

```
--- START FILE: HTML ---

<!DOCTYPE html>
<html>
<body>
   <p>
      <label for="zip">ZIP code:</label>
      <input type="text" id="zip" maxlength="5">
      <button id="search">Search</button>
   </p>
   <div id="forecast"></div>
</body>
</html>


--- END FILE: HTML ---

--- START FILE: JavaScript ---


function getForecast() {
   let zipcode = document.getElementById("zip").value;
   let xhr = new XMLHttpRequest();
   xhr.addEventListener("load", responseReceivedHandler);
   xhr.open("GET", "https://wp.zybooks.com/weather.php?zip="
+ zipcode);
   xhr.responseType = "json";
   xhr.send();
}

function responseReceivedHandler() {
   if (this.status !== 200) {
      alert("Error making HTTP request");
   }

   let html = "";
   if (this.response.success) {
      html += "<h1>Forecast</h1>";
      html += "<ol>";
      for (let day of this.response.forecast) {
         html += `<li>${day.desc}: high is ${day.high}, low
is ${day.low}</li>`;
      }
```

```
        html += "</ol>";
    }
    else {
        html = `<h1>Error: ${this.response.error}</h1>`;
    }

    document.getElementById("forecast").innerHTML = html;
}

document.getElementById("search").addEventListener("click",
getForecast);


--- END FILE: JavaScript ---
```

Feedback?

PARTICIPATION ACTIVITY 7.9.8: Updating the DOM via an Ajax request.

Match the JavaScript code with the description.

```
// Called when Ajax response is received
    function responseReceivedHandler() {
(a)     if (this.status === 200) {
            // code assumes response returns an ordered list that can
be inserted
            // into the list named myList, overwriting any previous
contents
            let list = document.getElementById("myList");
(b)         list.innerHTML = this.response;
        } else {
(c)         console.log("The request failed, status: " + this.status +
" " + this.statusText);
        }
    }

    let xhr = new XMLHttpRequest();
    xhr.addEventListener("load", responseReceivedHandler);
(d) xhr.responseType = "text";
    xhr.open("GET", "http://www.example.org/example.html");
    xhr.send();
```

If unable to drag and drop, refresh the page.

Check request success (a) **Correct**

	The response status code is accessed by using `this.status`.	
Access response's data	(b) Response data is accessed using `this.response`.	Correct
Print debugging information	(c) Console output helps a developer see what is happening in the application.	Correct
Indicate how to interpret response data	(d) By setting the `responseType` to "text", the browser does not parse the response, and the `response` and `responseText` attributes contain the same value.	Correct

Reset

Feedback?

CHALLENGE ACTIVITY 7.9.1: XMLHttpRequest (Ajax).

530096.4000608.qx3zqy7

Jump to level 1

If the response object's success property is true, log the final grade from the response to the console. Otherwise, log the error from the response to the console. Hint: this.response is the response.

```
10
11      Unsuccessful request:
12      {
13         "success": false,
14         "error": "..."
15      } */
16
17    /* Your solution goes here */
18    if (this.response.success) {
19      console.log(this.response.grades.final);
20    } else {
21      console.log(this.response.error);
22    }
23
24  }
25
```

1 2 3 4

1 2 3 4

Check Try again

Done. Click any level to practice more. Completion is preserved.

✔

✔ Testing console log for unsuccessful response

Yours `Name not found`

✔ Testing console log for successful response

Yours `99`

Feedback?

Monitoring uploads

The XMLHttpRequest object's ***upload*** attribute is an object for monitoring the status of the request being sent to the server. The `upload` attribute has the same handlers as the XMLHttpRequest object, but the progress handler is the only handler typically used for the `upload` attribute. The progress handler can be used to monitor the status of uploading large files, such as attaching a document to a Gmail message.

Example 7.9.1: Monitoring the progress of an uploaded file.

```
function uploadProgressHandler(event) {
  if (event.lengthComputable) {
    console.log(event.loaded + " bytes uploaded out of " +
event.total + " bytes total.");
  }
}

let file = document.getElementById("file_widget").files[0];
let xhr = new XMLHttpRequest();
xhr.upload.addEventListener("progress", uploadProgressHandler);
xhr.open("POST", "http://www.example.org/example.html");
xhr.setRequestHeader("Content-Type", file.type);
xhr.send(file);
```

Feedback?

PARTICIPATION ACTIVITY 7.9.9: Uploading files using XMLHttpRequest.

1) `xhr.addEventListener("progress", handler)` tracks the status of a request to the server.

- ○ True
- ◉ False

Correct

The code is missing the upload attribute, so the code monitors the status of the response, not the status of the upload.

2) The XMLHttpRequest object can use a GET or POST request to upload files to a server.

- ○ True
- ◉ False

Correct

Files must be sent using a POST request. A GET request can only be used to get data from the server, not upload files to a server.

3) When uploading files to a server, the Content-Type header indicates the type of file being uploaded.

- ◉ True
- ○ False

Correct

The Content-Type is used by both web servers and browsers to understand the file type. A web server keeps track of the file type in case an uploaded file is requested by another browser.

4) The XMLHttpRequest object's load handler takes an event object as an argument.

- ○ True
- ◉ False

Correct

The load handler has no parameters and is called on the XMLHttpRequest object. So, the `this` keyword is used to access the XMLHttpRequest object.

5) The XMLHttpRequest object's progress handler takes an event object as an argument.

- ◉ True
- ○ False

Correct

The progress handler is passed an event object so the handler can access information about the progress. Ex: The `loaded` and `total` attributes.

Feedback?

Exploring further:

- XMLHttpRequest from MDN
- HTTP access control (CORS) from MDN

How was this section?

Provide section feedback