Net-Centric Computing

Javascript: Ajax I

Dr. Simon Fan, Associate Professor of CSSE

xfan@psu.edu

Objectives

- Scripted HTTP
- Ajax and Ajax transport
- **XMLHttpRequest** API
  - Steps for sending requests
  - Encoding requests

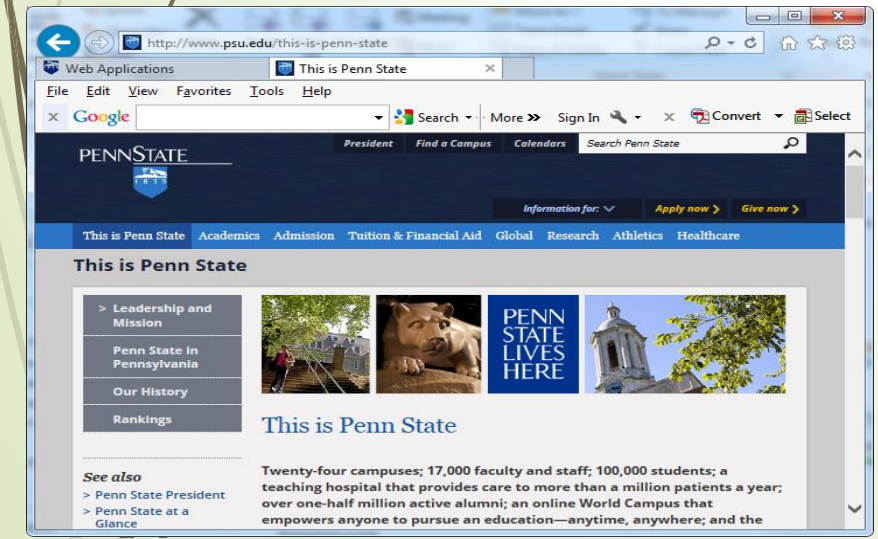
Scripted HTTP

- The Hypertext Transfer Protocol (HTTP) specifies how web browsers send **HTTP requests** to web servers, and how web servers respond to those requests.

1) **User initiated requests**: anchors (links), forms, and the URL field of a web browser have default actions. HTTP transports occur when a user clicks on a link, submits a form, or types a URL.

2) **Scripted HTTP requests:** HTTP requests are initiated by javascript code

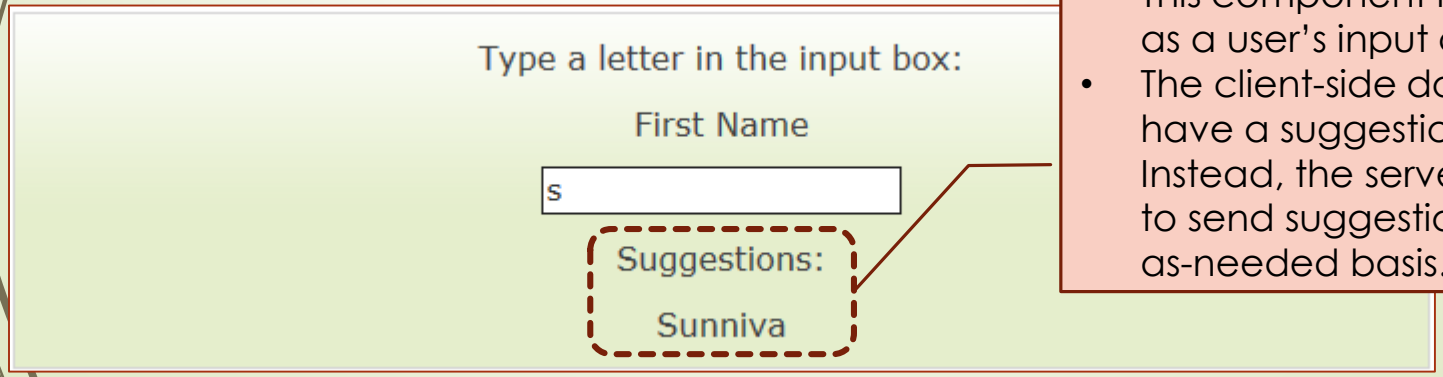
   - **Trivial Scripting** (**Lab 4**): a web browser loads a new page when
     - A script sets the location property of a window object or
     - A script calls the submit() method of a form object.



Ajax: **A**synchronous **Ja**vaScript and **X**ML

- **Ajax**: An architecture where the client uses scripted HTTP to initiate data exchange with a web server without causing pages to reload.
- The client application is able to continue serving the user while at the same time the **Ajax engine** is submitting HTTP requests to the server and waiting for responses.
- A web application might use Ajax technologies to log user interaction data to the server or to improve its start-up time by displaying only a simple page at first and then downloading additional **data** and **page components on an as-needed basis**.

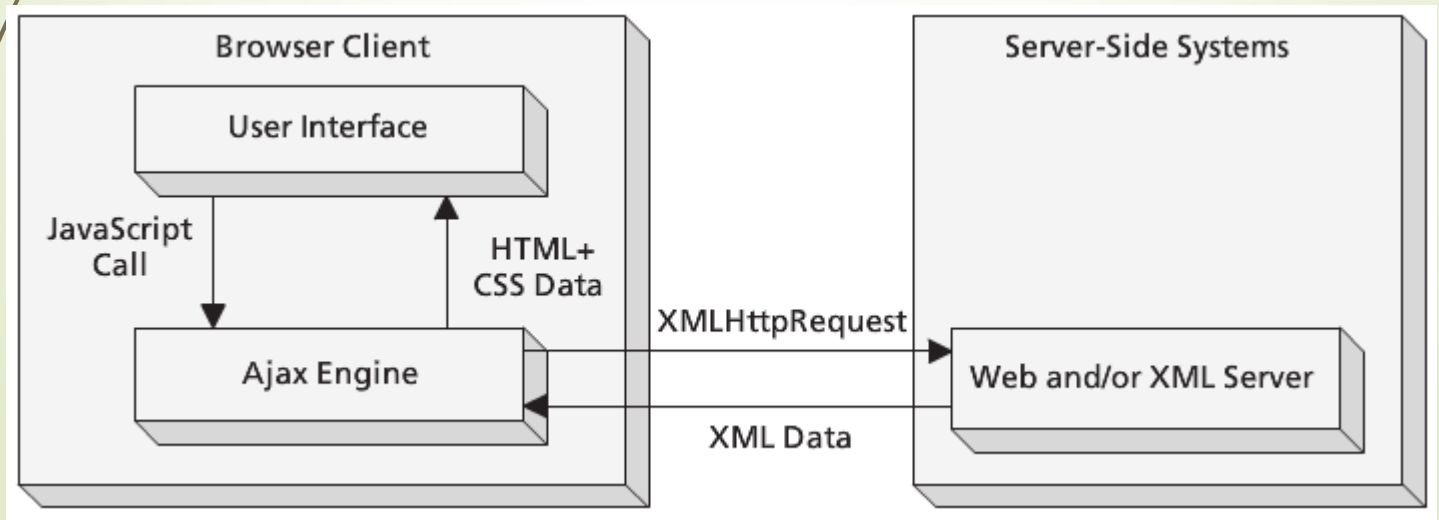
Note: Not downloading a new page!



- This component is updated as a user's input changes
- The client-side does not have a suggestion DB. Instead, the server needs to send suggestions on an as-needed basis.

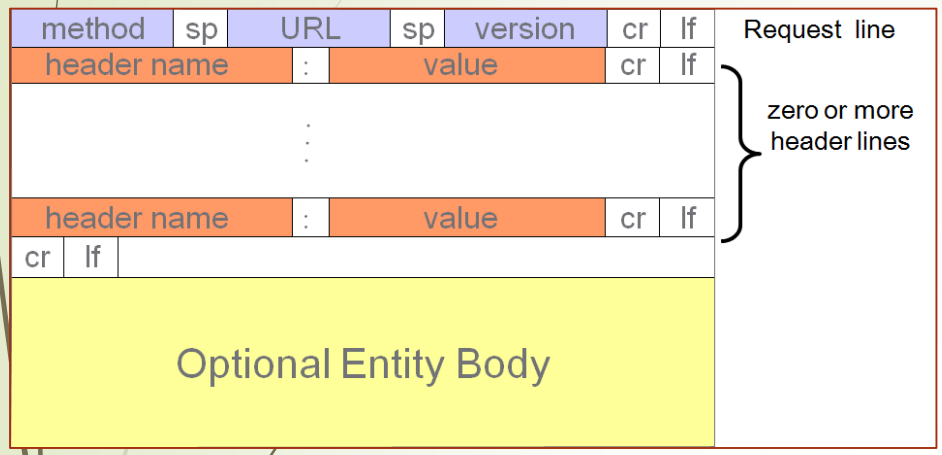
Use the **XMLHttpRequest** API

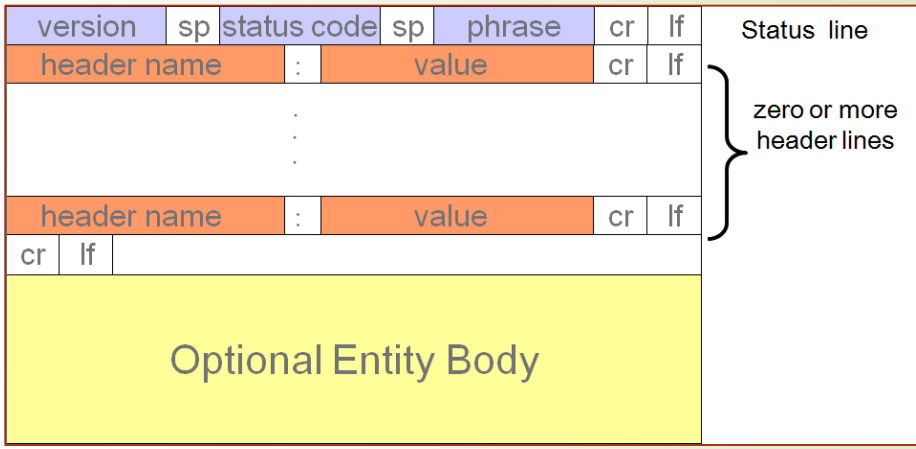
- The XMLHttpRequest object is **a developer's dream**, you can:
  - Send data to a server in the background
  - Request data from a server after the page has loaded
  - Receive data from a server after the page has loaded
  - Update portion of a web page without reloading the page
- XMLHttpRequest is a **browser-level API**, which includes
  - the ability to make **POST** requests, in addition to regular **GET** requests,
  - the support for any text based format, including XML, JSON.



Use the **XMLHttpRequest** API

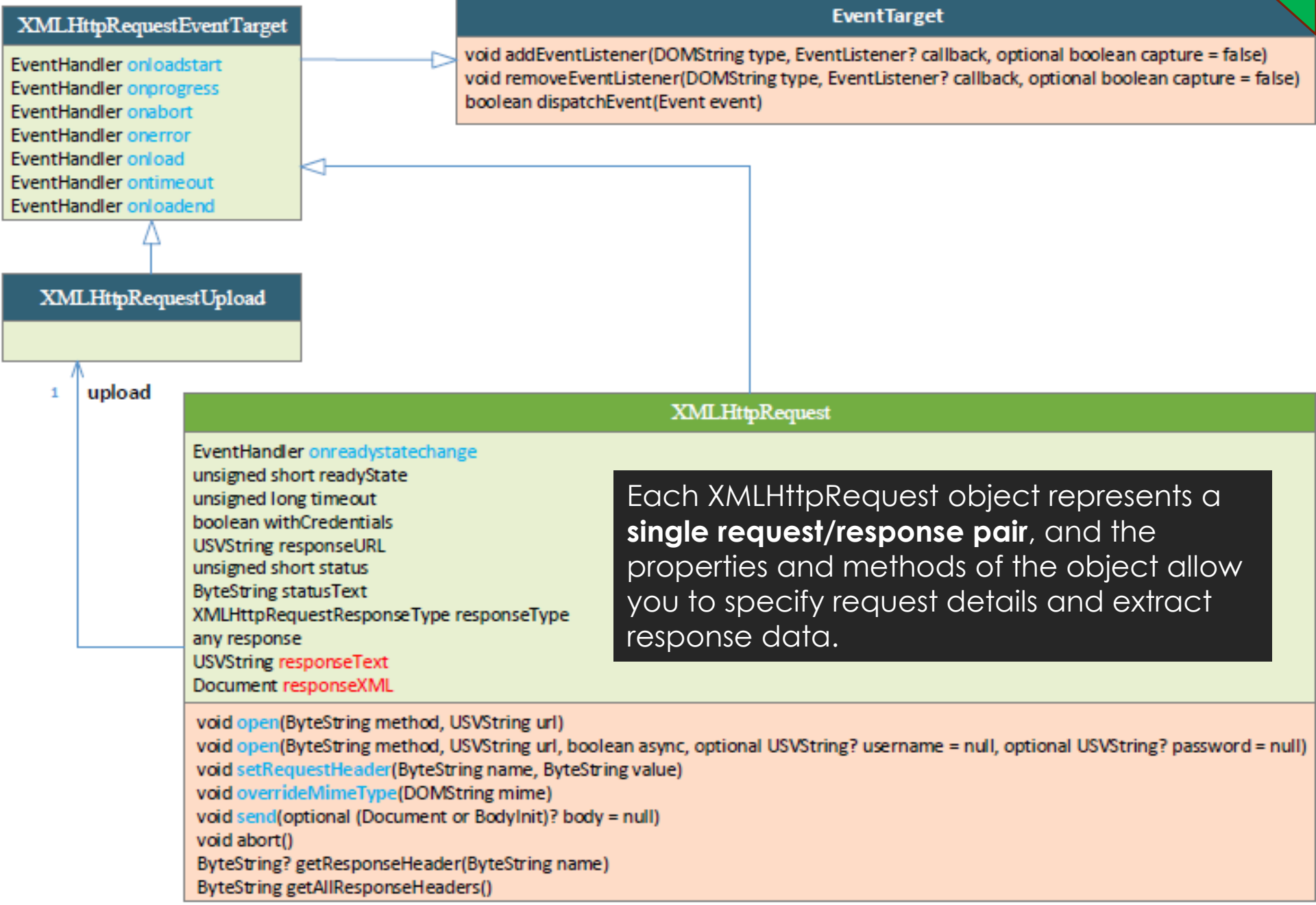
- XMLHttpRequest is designed to work with the HTTP and HTTPS protocols.





- "**GET**" is used for most "regular" requests, and it is appropriate when
  - the URL completely specifies the requested resource,
  - the request has **no side effects** on the server, and
  - The server's response is **cacheable**.
- "**POST**" is typically used by HTML forms.
  - It includes additional data (the form data) in the request body and
  - that data is often stored in a database on the server (**a side effect**).
  - **Repeated POSTs to the same URL** may result in different responses from the server. Thus, requests that use this method should not be cached.
- Other verbs: DELETE, HEAD, OPTIONS, and PUT

Use the **XMLHttpRequest** API



Each XMLHttpRequest object represents a **single request/response pair**, and the properties and methods of the object allow you to specify request details and extract response data.

XMLHttpRequest: sending request

- **Step 1**: instantiate an XMLHttpRequest object:

```
var request = new XMLHttpRequest();
```

- **Step 2**: call the **open**() method of the XMLHttpRequest object to specify the method and the URL

```
request.open("GET",          // Begin a HTTP GET request
             "data.csv");    // For the contents of this URL
```

- **Step 3**: set the request headers, if any.

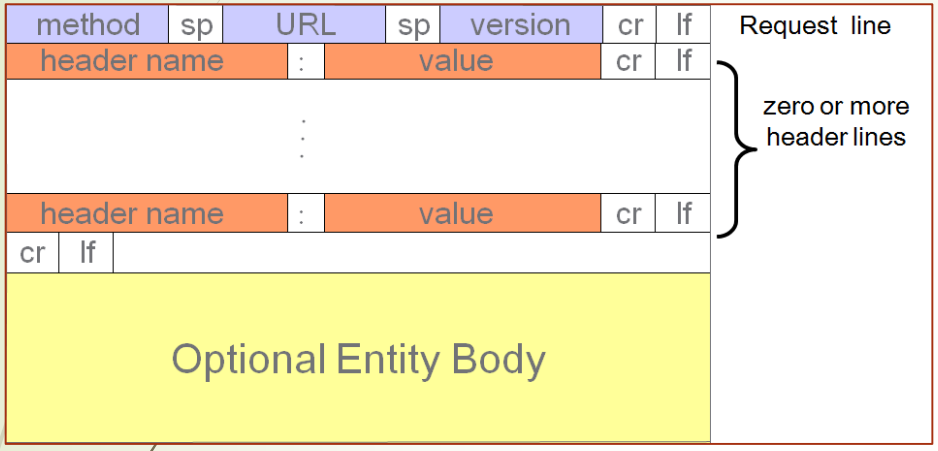
```
request.setRequestHeader("Content-Type", "text/plain");
```

  - You cannot specify the following headers yourself: XMLHttpRequest will handle those automatically for you.

Accept-Charset	Content-Transfer-Encoding	TE
Accept-Encoding	Date	Trailer
Connection	Expect	Transfer-Encoding
Content-Length	Host	Upgrade
Cookie	Keep-Alive	User-Agent
Cookie2	Referer	Via

- **Step 4**: specify the optional request body and **send** it off to the server.

XMLHttpRequest: sending request (example)




```
function postMessage(msg) {
    var request = new XMLHttpRequest();        // New request
    request.open("POST", "/log.php");          // POST to a server-side script
    // Send the message, in plain-text, as the request body
    request.setRequestHeader("Content-Type", // Request body will be plain text
                             "text/plain;charset=UTF-8");
    request.send(msg);                         // Send msg as the request body
}
```

Encoding Request: Form-encoded (1)

- The encoding scheme used for HTML form data (**URL encoding**):
  - URLs can only be sent over the Internet using the ASCII character-set
  - URL encoding replaces non ASCII characters with a "**%**" followed by hexadecimal digits. [JavaScript global function **encodeURIComponent**()]
  - URL encoding normally replaces a space with a plus (+) sign, or %20.
  - Separate the encoded name and value with an equals sign, and
  - Separate these name/value pairs with ampersands.

- This form data encoding format has a formal MIME type:
  - **application/x-www-form-urlencoded**.
- When HTML forms are POSTed to the server, the encoded form data is used as the body of the request. You must set the "Content-Type" request header to "**application/x-www-form-urlencoded**"

Encoding Request: Form-encoded (1)

URL encoding normally replaces a space with a plus (+) sign, or %20.

- **Example**: Form-encode the properties of a Javascript object

```
function postData(url, data, callback) {
    var request = new XMLHttpRequest();
    request.open("POST", url);                          // POST to the specified url
    request.onreadystatechange = function() {           // Simple event handler
        if (request.readyState === 4 && callback)       // When response is complete
            callback(request);                          // call the callback.
    };
    request.setRequestHeader("Content-Type",            // Set Content-Type
                             "application/x-www-form-urlencoded");
    request.send(encodeFormData(data));                 // Send form-encoded data
}
```

```
/**
 * Encode the properties of an object as if they were name/value pairs from
 * an HTML form, using application/x-www-form-urlencoded format
 */
function encodeFormData(data) {
    if (!data) return "";    // Always return a string
    var pairs = [];          // To hold name=value pairs
    for(var name in data) {                                     // For each name
        if (!data.hasOwnProperty(name)) continue;               // Skip inherited
        if (typeof data[name] === "function") continue;         // Skip methods
        var value = data[name].toString();                      // Value as string
        name = encodeURIComponent(name.replace(" ", "+"));      // Encode name
        value = encodeURIComponent(value.replace(" ", "+")); // Encode value
        pairs.push(name + "=" + value);   // Remember name=value pair
    }
    return pairs.join('&'); // Return joined pairs separated with &
}
```

Encoding Request: JSON-encoded

- The JSON format has gained popularity as a web interchange format.

```
{"menu": {
  "id": "file",
  "value": "File",
  "popup": {
    "menuitem": [
      {"value": "New", "onclick": "CreateNewDoc()"},
      {"value": "Open", "onclick": "OpenDoc()"},
      {"value": "Close", "onclick": "CloseDoc()"}
    ]
  }
}}
```

```
function postJSON(url, data, callback) {
    var request = new XMLHttpRequest();
    request.open("POST", url);                          // POST to the specified url
    request.onreadystatechange = function() {           // Simple event handler
        if (request.readyState === 4 && callback)       // When response is complete
            callback(request);                          // call the callback.
    };
    request.setRequestHeader("Content-Type", "application/json");
    request.send(JSON.stringify(data));
}

```

Encoding Request: XML-encoded

```
<menu id="file" value="File">
  <popup>
    <menuitem value="New" onclick="CreateNewDoc()" />
    <menuitem value="Open" onclick="OpenDoc()" />
    <menuitem value="Close" onclick="CloseDoc()" />
  </popup>
</menu>
```

```
// Encode what, where
// specified url, inv
function postQuery(url, what, where, radius, callback) {
    var request = new XMLHttpRequest();
    request.open("POST", url);                    // POST to the specified url
    request.onreadystatechange = function() {     // Simple event handler
        if (request.readyState === 4 && callback) callback(request);
    };

    // Create an XML document with root element <query>
    var doc = document.implementation.createDocument("", "query", null);
    var query = doc.documentElement;              // The <query> element
    var find = doc.createElement("find");         // Create a <find> element
    query.appendChild(find);                      // And add it to the <query>
    find.setAttribute("zipcode", where);          // Set attributes on <find>
    find.setAttribute("radius", radius);
    find.appendChild(doc.createTextNode(what)); // And set content of <find>

    // Now send the XML-encoded data to the server.
    // Note that the Content-Type will be automatically set.
    request.send(doc);
}
```

Cross-Origin HTTP Requests

- As part of the same-origin security policy, the XMLHttpRequest object can normally issue HTTP requests only to the server from which the document that uses it was downloaded.
- With **XMLHttpRequest** object, document contents are always exposed through the responseText property, so the same-origin policy **cannot** allow XMLHttpRequest to make cross-origin requests.
- the same-origin policy will be relaxed and cross-origin requests will work
  - If the browser supports **CORS** (Cross-Origin Resource Sharing) for XMLHttpRequest and
  - if the website allows cross-origin requests with CORS.
  - Testing for the presence of the withCredentials property is a way to test for CORS support in your browser.

```
// Is there any chance that cross-origin requests will succeed?
var supportsCORS = (new XMLHttpRequest()).withCredentials !== undefined;
```