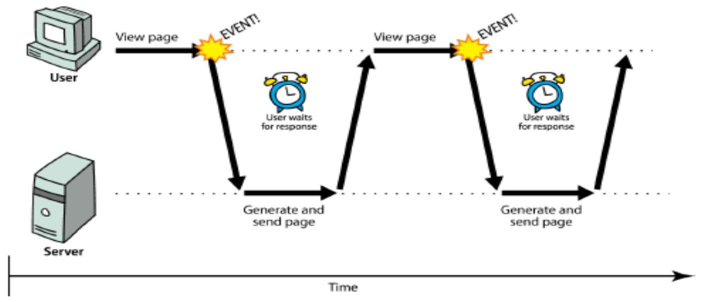
Ajax

Synchronous web communication



- synchronous: user must wait while new pages load
  - the typical communication pattern used in web pages (click, wait, refresh)

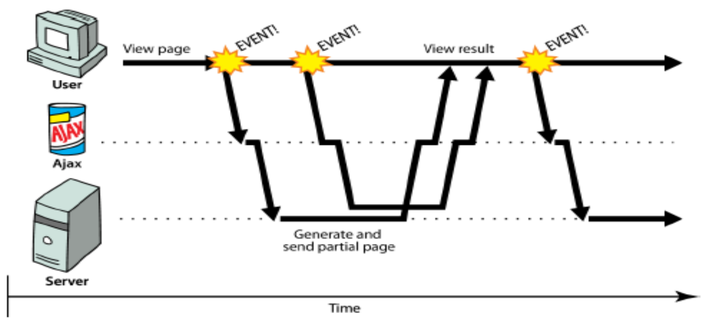
Web applications and Ajax

- **web application:** a dynamic web site that mimics the feel of a desktop app
  - presents a continuous user experience rather than disjoint pages
  - examples: Gmail, Google Maps, Google Docs and Spreadsheets, Flickr, A9

Web applications and Ajax

- **Ajax:** Asynchronous JavaScript and XML
  - not a programming language; a particular way of using JavaScript
  - downloads data from a server in the background
  - allows dynamically updating a page without making the user wait
  - avoids the "click-wait-refresh" pattern
  - Example: Google Suggest

Asynchronous web communication



- **asynchronous**: user can keep interacting with page while data loads
  - communication pattern made possible by Ajax

`XMLHttpRequest` (and why we won't use it)

- JavaScript includes an XMLHttpRequest object that can fetch files from a web server
  - supported in IE5+, Safari, Firefox, Opera, Chrome, etc. (with minor compatibilities)
- it can do this asynchronously (in the background, transparent to user)
- the contents of the fetched file can be put into current web page using the DOM

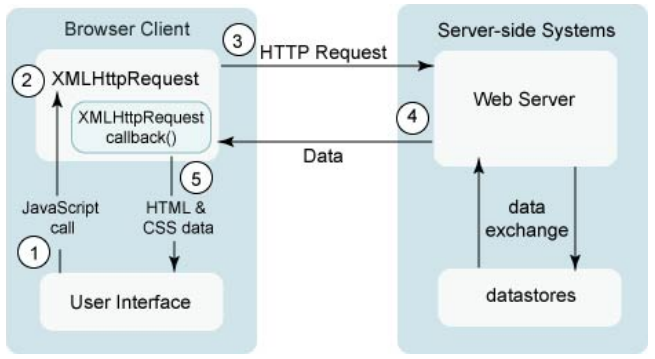
`XMLHttpRequest` (and why we won't use it)

- sounds great!...
- ... but it is clunky to use, and has various browser incompatibilities
- Prototype provides a better wrapper for Ajax, so we will use that instead

A typical Ajax request

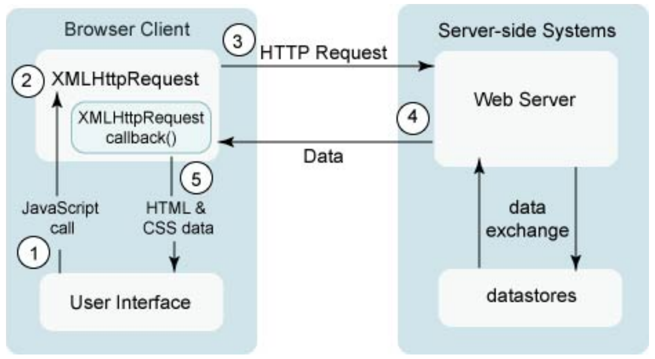
1. user clicks, invoking an event handler
2. handler's code creates an `XMLHttpRequest` object
3. `XMLHttpRequest` object requests page from server
4. server retrieves appropriate data, sends it back
5. `XMLHttpRequest` fires an event when data arrives
   - this is often called a callback
   - you can attach a handler function to this event
6. your callback event handler processes the data and displays it

A typical Ajax request



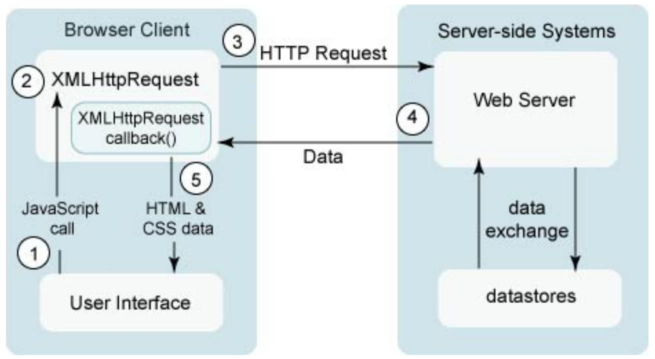
1. user clicks, invoking an event handler

A typical Ajax request



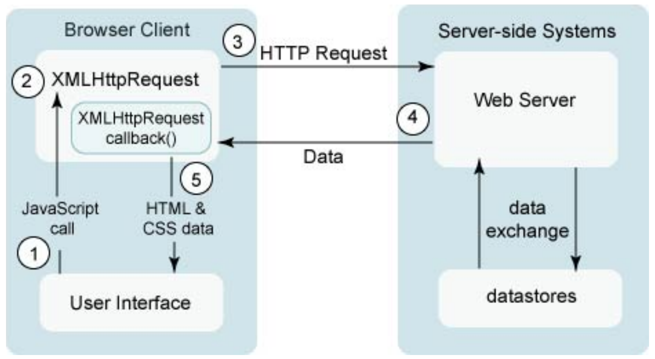
2. handler's code creates an XMLHttpRequest object

A typical Ajax request



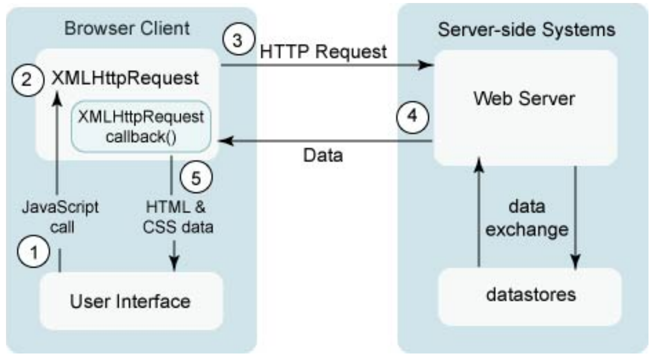
3. XMLHttpRequest object requests page from server

A typical Ajax request



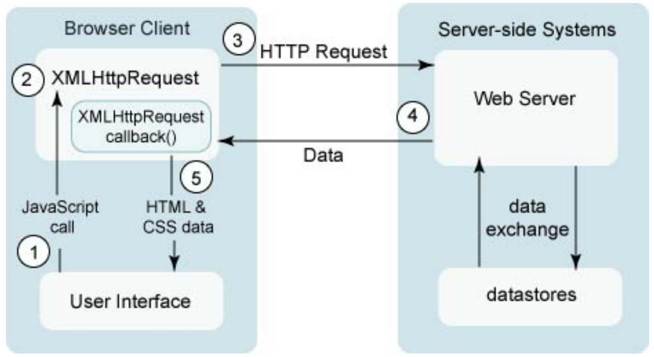
4. server retrieves appropriate data, sends it back

A typical Ajax request



5. XMLHttpRequest fires an event when data arrives
   - A callback to which you can attach a handler function

A typical Ajax request



6. your callback event handler processes the data and displays it

XMLHttpRequest methods

- the core JavaScript object that makes Ajax possible

Method	Description
open(*method*, *url*, *async*)	specifies the URL and HTTP request method

send()
send(*postData*)
	sends the HTTP request to the server, with optional POST parameters

abort()	stops the request
getAllResponseHeaders(), getResponseHeader(*name*), setRequestHeader(*name*,*value*)	for getting/setting raw HTTP headers

XMLHttpRequest properties

Property	Description
responseText	the entire text of the fetched page, as a string
responseXML	the entire contents of the fetched page, as an XML document tree (seen later)
status	the request's HTTP status code (200 = OK, etc.)
statusText	HTTP status code text (e.g. "Bad Request" for 400)
timeout	how many MS to wait before giving up and aborting the request (default 0 = wait forever)
readyState	request's current state (*0 = not initialized, 1 = set up, 2 = sent, 3 = in progress, 4 = complete*)

1. Synchronized requests (bad)

```
// this code is in some control's event handler
var ajax = new XMLHttpRequest();
ajax.open("GET", url, false);
ajax.send();
do something with ajax.responseText;
                                                   JS
```

- create the request object, open a connection, send the request
- when send returns, the fetched text will be stored in request's `responseText` property

Why synchronized requests suck

- your code waits for the request to completely finish before proceeding
- easier for you to program, but ...
  - the user's entire browser LOCKS UP until the download is completed
  - a terrible user experience (especially if the page is very large or slow to transfer)
- **Better solution**
  - **Use an asynchronous request that notifies you when it is complete**
  - **This is accomplished by learning about the event properties of `XMLHttpRequest`**

XMLHttpRequest events

Event	**Description**
load	occurs when the request is completed
error	occurs when the request fails
timeout	occurs when the request times out
abort	occurs when the request is aborted by calling abort()
loadstart, loadend, progress, readystatechange	progress events to track a request in progress

2. Asynchronous requests, basic idea

```
var ajax = new XMLHttpRequest();
ajax.onload = functionName;
ajax.open("GET", url, true);
ajax.send();
...
function functionName() {
  do something with this.responseText;
}                                                           JS
```

- attach an event handler to the load event
- handler will be called when request state changes, e.g. finishes
- function contains code to run when request is complete
  - inside your handler function, this will refer to the ajax object
  - you can access its `responseText` and other properties

What if the request fails?

```
var ajax = new XMLHttpRequest();
ajax.onload = functionName;
ajax.open("GET", "url", true);
ajax.send();
...
function functionName() {
  if (this.status == 200) {   // 200 means request succeeded
    do something with this.responseText;
  } else {
    code to handle the error;
  }
}
```
JS

- web servers return status codes for requests (200 means Success)
- you may wish to display a message or take action on a failed request

Handling the error event

```
var ajax = new XMLHttpRequest();
ajax.onload = functionName;
ajax.onerror = errorFunctionName;
ajax.open("GET", "url", true);
ajax.send();
...
function functionName(e) {
  do something with e, this.status, this.statusText, ...
}
```
JS

- the graceful way to handle errors is to listen for the error event
- the handler is passed the error/exception as a parameter
- you can examine the error, as well as the request status, to determine what went wrong

Example Ajax error handler

```
var ajax = new XMLHttpRequest();
...
ajax.onerror = ajaxFailure;
...

function ajaxFailure(exception) {
  alert("Error making Ajax request:" +
        "\n\nServer status:\n" + this.status + " " + this.statusText +
        "\n\nServer response text:\n" + this.responseText);
  if (exception) {
    throw exception;
  }
}
```
JS

- for user's (and developer's) benefit, show an error message if a request fails

Passing query parameters to a request

- to pass parameters, concatenate them to the URL yourself
  - you may need to URL-encode the parameters by calling the JS encodeURIComponent(string) function on them
  - won't work for POST requests (see next slide)

```
var ajax = new XMLHttpRequest();
ajax.onload = functionName;
ajax.open("GET", "url?name1=value1&name2=value2&...", true);
ajax.send();
```
JS

Creating a POST request

```
var params = new FormData();
params.append("name", value);
params.append("name", value);

var ajax = new XMLHttpRequest();
ajax.onload = functionName;
ajax.open("POST", "url", true);
ajax.send(params);                                    JS
```

- use a FormData object to gather your POST query parameters
- pass the FormData to the request's send method
- method passed to open should be changed to "POST"

Ajax: The PrototypeJS Way!

```
new Ajax.Request("url",
{
        option : value,
        option : value,
        ...
        option : value
}
);                                                   JS
```

- construct a Prototype Ajax.Request object to request a page from a server using Ajax
- constructor accepts 2 parameters:
  1. the URL to 1. fetch, as a String,
  2. a set of options, as an array of key : value pairs in {} braces (an anonymous JS object)

Prototype Ajax methods and properties

option	description
method	how to fetch the request from the server (default "post")
parameters	query parameters to pass to the server, if any
asynchronous (default true), contentType, encoding, requestHeaders	

options that can be passed to the Ajax.Request constructor

Prototype Ajax methods and properties

event	description
onSuccess	request completed successfully
onFailure	request was unsuccessful
onException	request has a syntax error, security error, etc.

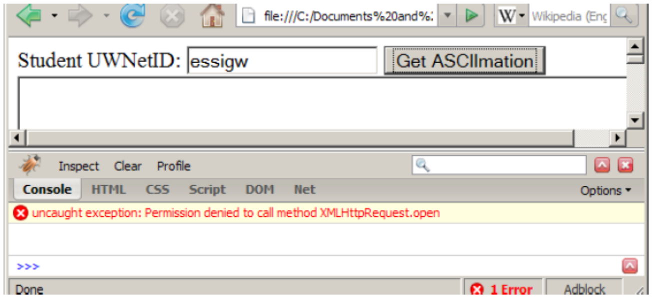
events in the Ajax.Request object that you can handle

Basic Prototype Ajax template

property	description
status	the request's HTTP error code (200 = OK, etc.)
statusText	HTTP error code text
responseText	the entire text of the fetched page, as a String
responseXML	the entire contents of the fetched page, as an XML DOM tree (seen later)

```
function handleRequest(ajax) {
        alert(ajax.responseText);
}                                                      JS
```

XMLHttpRequest security restrictions


Student UWNetID: essigw   Get ASCIImation

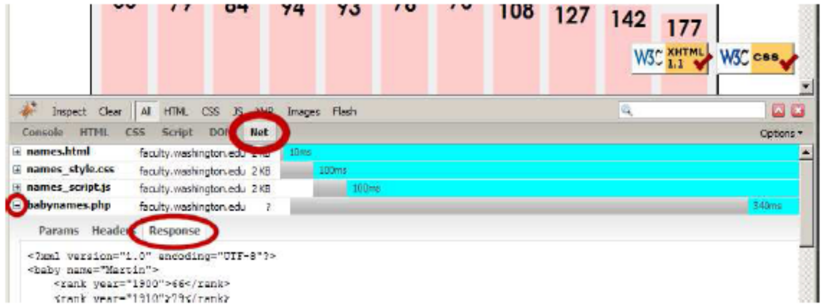
uncaught exception: Permission denied to call method XMLHttpRequest.open

- cannot be run from a web page stored on your hard drive
- can only be run on a web page stored on a web server
- can only fetch files from the same site that the page is on
  www.foo.com/a/b/c.html can only fetch from www.foo.com

Handling Ajax errors

```
new Ajax.Request("url",
{
        method: "get",
        onSuccess: functionName,
        onFailure: ajaxFailure,
        onException: ajaxFailure
}
);
...
function ajaxFailure(ajax, exception) {
        alert("Error making Ajax request:" + "\n\nServer
status:\n" + ajax.status + " " + ajax.statusText +
"\n\nServer response text:\n" + ajax.responseText);
        if (exception) {
                throw exception;
        }
}                                                      JS
```

Debugging Ajax code


```
names.html         faculty.washington.edu         10ms
names_style.css    faculty.washington.edu  2 KB   100ms
names_script.js    faculty.washington.edu  2 KB   100ms
babynames.php      faculty.washington.edu  ?      340ms
```

```
<?xml version="1.0" encoding="UTF-8"?>
<baby name="Martin">
    <rank year="1900">66</rank>
    <rank year="1910">79</rank>
```

- Net tab shows each request, its parameters, response, any errors
- expand a request with + and look at Response tab to see Ajax result

Creating a POST request

```
new Ajax.Request("url",
{
        method: "post", // optional
        parameters: { name: value, name: value, ..., name:
value },
        onSuccess: functionName,
        onFailure: functionName,
        onException: functionName
}
);                                                          JS
```

Creating a POST request

- Ajax.Request can also be used to post data to a web server
- method should be changed to "post" (or omitted; post is default)
- any query parameters should be passed as a parameters parameter
  - written between {} braces as a set of name : value pairs (another anonymous object)
  - get request parameters can also be passed this way, if you like

Prototype's Ajax Updater

```
new Ajax.Updater(
        "id",
        "url",
        {
                method: "get"
        }
);                                                          JS
```

- Ajax.Updater fetches a file and injects its content into an element as innerHTML
- additional (1st) parameter specifies the id of element to inject into

Example

```
<form method="get" action="greeting.php" id="greeting-form">
<div>
   <label for="greeting-name">Enter your name:</label>
   <input id="greeting-name" name="greeting-name" type="text" />
   <input id="greeting-submit" name="greeting-submit" type="submit" value="Greet me!" />
</div>
<div id="greeting"></div>
</form>
                                                                          HTML
```

Example

```
<?php
     $the_name = htmlspecialchars($_GET['greeting-name']);
     echo "<p>Season's Greetings, $the_name!</p>";
?>
                                                      php
```

Example

```
Event.observe(window, 'load', init, false);

function init() {
    $('greeting-submit').style.display = 'none';
    Event.observe('greeting-name', 'keyup', greet, false);
}

function greet() {
    var url = 'greeting.php';
    var pars = 'greeting-name=' + escape($F('greeting-name'));
    var target = 'greeting';
    var myAjax = new Ajax.Updater(target, url, {
        method: 'get',
        parameters: pars
    });
}
                                                     JavaScript
```