Web Programming Step by Step

Lecture 19
Ajax
Reading: 10.1 - 10.2

Except where otherwise noted, the contents of this presentation are Copyright 2009 Marty Stepp and Jessica Miller.

---

**Synchronous web communication (10.1)**

- **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
- **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
  - examples: UW's CSE 14x Diff Tool, Practice-It; 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

- 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
Prototype's Ajax model (10.2.4)

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

- construct a Prototype Ajax.Request object to request a page from a server using Ajax
- constructor accepts 2 parameters:
  1. the **URL** to fetch, as a String,
  2. a set of **options**, as an array of **key : value** pairs in {} braces (an anonymous JS object)
- hides icky details from the raw XMLHttpRequest; works well in all browsers

### Prototype Ajax methods and properties

<table>
<thead>
<tr>
<th>option</th>
<th>description</th>
</tr>
</thead>
<tbody>
<tr>
<td>method</td>
<td>how to fetch the request from the server (default &quot;post&quot;)</td>
</tr>
<tr>
<td>parameters</td>
<td>query parameters to pass to the server, if any</td>
</tr>
<tr>
<td>asynchronous</td>
<td>(default true), contentType, encoding, requestHeaders</td>
</tr>
</tbody>
</table>

**options** that can be passed to the `Ajax.Request` constructor

<table>
<thead>
<tr>
<th>event</th>
<th>description</th>
</tr>
</thead>
<tbody>
<tr>
<td>onSuccess</td>
<td>request completed successfully</td>
</tr>
<tr>
<td>onFailure</td>
<td>request was unsuccessful</td>
</tr>
<tr>
<td>onException</td>
<td>request has a syntax error, security error, etc.</td>
</tr>
<tr>
<td>onCreate, onComplete, on###</td>
<td>(for HTTP error code ###)</td>
</tr>
</tbody>
</table>

**events** in the `Ajax.Request` object that you can handle
Basic Prototype Ajax template

```javascript
new Ajax.Request("url",
{
    method: "get",
    onSuccess: functionName
}
);
...

function functionName(ajax) {
    do something with ajax.responseText;
}
```

- most Ajax requests we'll do in this course are GET requests
- attach a handler to the request's onSuccess event
- the handler takes an Ajax response object, which we'll name `ajax`, as a parameter

The **Ajax response object**

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

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

- most commonly property is `responseText`, to access the fetched page
XMLHttpRequest security restrictions

- 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

```javascript
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;
    }
}
```

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

- **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

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

- **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

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

- `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