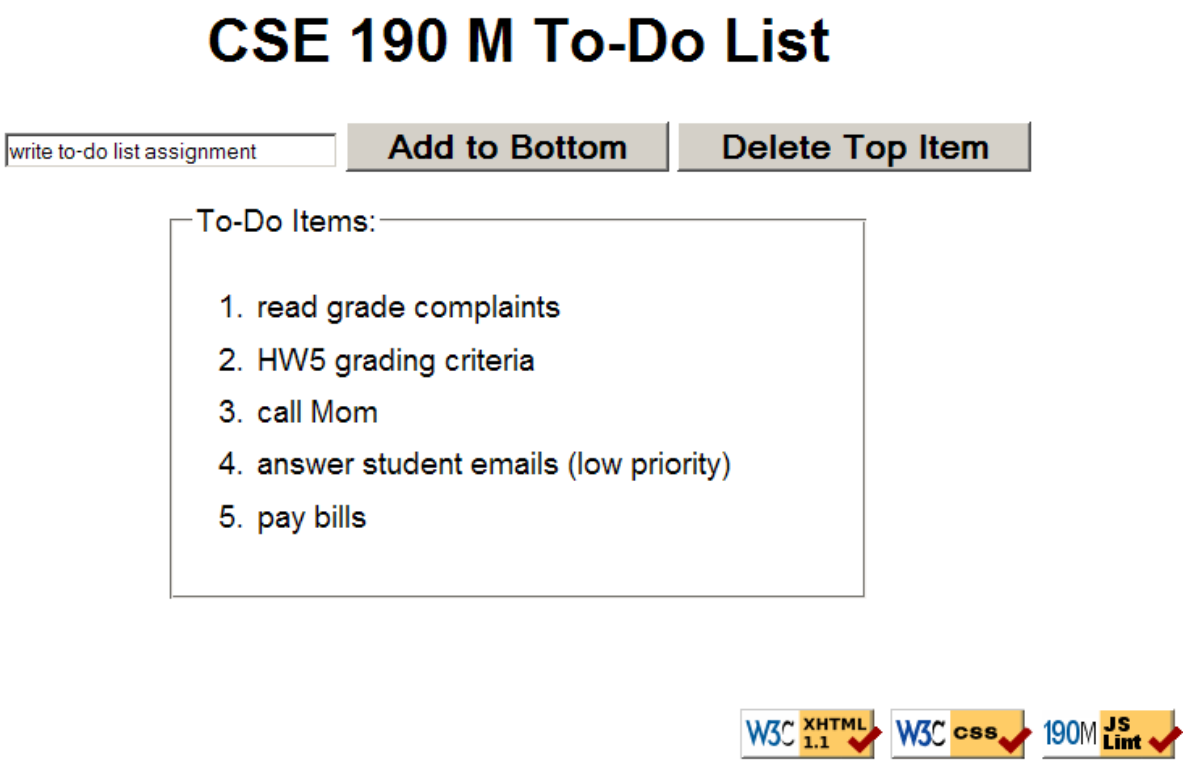
University of Washington, CSE 190 M
Homework Assignment 8: To-Do List

This assignment is about writing a complete Ajax-enabled "Web 2.0" application, using the Scriptaculous JavaScript library and PHP web services.

In this assignment you will create a web page for an online to-do list. The user can add an item to the end of the to-do list, delete the first item from the list, or drag the items into a different order. Any of these changes made to the list will be saved to the web server using Ajax, so that if the user leaves the page and returns later, the current state of the list will be remembered.

The following image shows a rough idea of how your page should look, but many aspects of the appearance are not rigidly specified. See the following pages for more details.



Unlike in many of the past assignments, we will not be providing you with any starter code whatsoever for this assignment. While this will make a bit more work for you, the goal is that you will gain experience creating an entire web application from scratch. Turn in the following files:

- **todolist.html**, the XHTML file holding the content of your web page
- **todolist.css**, the style sheet describing the appearance of your web page
- **todolist.js**, the JavaScript code implementing the behavior of your web page
- **todolist.php**, the PHP web service for remembering the state of the to-do list

This program uses Ajax to communicate with a PHP web service. In order for Ajax and PHP to work correctly, you must **upload your files to the Webster server** and test them there.

Page Behavior:

Unlike in past assignments, many aspects of the exact appearance of the page is not specified. Beyond the aspects on these pages, the rest is up to you, so long as it does not conflict with what is required here. If you want to use other resources such as image files, place them on your Webster space and link to them using **absolute URLs**. If you are unsure whether an aspect of your page is acceptable, ask your TA or the instructor. The goal is to be **creative** and personalize your page. You are should not match the screenshot in this document exactly; in fact, if you do copy its appearance, you will lose points for lack of creativity.

The page should contain a **heading** that identifies the course and the fact that this is a to-do list program. The page should contain **images** somewhere that link to the W3C XHTML/CSS **validators** and JSLint. Your page should have a **"favorites icon"** ("favicon"). One acceptable favicon would be the image **todolist_icon.gif**  from the course web site. (Whatever image you use, please link to it using an **absolute URL**.)

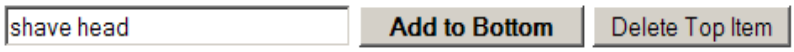
When the page first loads, it should show the current contents of the to-do list as an ordered list. Since the items are retrieved from the server, the page will not show any items until the server has been contacted. (The W3C XHTML validator complains if you have a list with no elements, so you may need to add an empty hidden element for it to validate.) Once the items arrive, they are added to your page using the DOM.

Apply a **Scriptaculous appearing effect** to the items, such as making the items "fade in" or "shake" when they arrive. You can make items initially invisible by calling hide on them or by setting their display style to none. An invisible element can be shown using Scriptaculous effects such as appear or grow. To put an effect on an element being deleted, consult Ch. 12 or the Scriptaculous slides on the `afterFinish` event.

Once the list appears, the user should be able to change its contents in at least the three following ways:

1. **add** an item to the end of the list
2. **delete** the item from the front of the list
3. **drag** the list items to reorder them

The UI for adding and deleting is up to you. For example, you could have a text field and "Add" and "Delete" buttons. Your page should not use alert or prompt boxes for this. You should reject empty strings as to-do items but otherwise can accept any text. If the user tries to delete when there are no items in the list, no JavaScript errors should occur.



Any change to the to-do list should be accompanied by a Scriptaculous effect of your choice, along with any other visual cues you want. For example, an added item could highlight or fade into view.

The reorderability of the list items should be done using Scriptaculous. Give the list an id and make it into a Sortable list. See the textbook or slides about subtleties of sortable lists, such as the ids on elements or their event properties. Also note that the sortability of a list breaks if you add or remove elements after you've made it sortable. To fix such a case, re-specify the list to be sortable after each modification.

When any of the three preceding actions (add, delete, reorder) has been made on the page, the page should immediately send an Ajax request to a server-side PHP web service **todolist.php** to inform the server of the change. The exact behavior of this service is described in the next section. If you've done this properly, at any point the user should be able to refresh the browser and still see the to-do list in its current state.

Any updates to the to-do list should appear instantly on the page. The instant that the user adds, deletes, or reorders items, the page should update to reflect this action. In the background, the page may be contacting the server to inform it of the change, but the page UI should not be out of date or locked up during this. You do not need to worry about multiple rapid updates overloading the server or arriving to the server out of order. Minimize the amount of heavy changes you make to the page's DOM; you should not destroy and recreate every list item in the DOM if only one item changes (i.e. on an add or remove).

Page Styling:

No CSS code is being provided to you, so the exact styling of your page is up to you. You should, however, set a non-trivial number of CSS styles including, but not limited to, the following:

- Set the page's text to use non-default font families and sizes. Don't forget to supply a list of possible fonts including a generic font family to make sure the page looks good on all computers.
- Supply at least some colors and/or images to your page to enhance its appearance.
- Lay out the page in a non-trivial way, such as by adjusting the positions, sizes, floating, borders, padding, margins, etc. of elements to give them a pleasant appearance.
- Indicate to the user that the to-do list items are rearrangeable by making their style change when the mouse hovers over them. We suggest that you do this using the CSS `:hover` selector.

PHP web service:

Your page's Ajax code should communicate with a PHP web service must support four types of web queries. To help you develop your program incrementally, we have placed a working version of the PHP web service at the following URL. You can initially write your page's code to talk to this working version if you like, but eventually you should re-implement the service yourself and connect your page to your own service.

https://webster.cs.washington.edu/cse190m/todolist.php

The following are the four query types. Some queries use a request type of GET and others use POST. Your code should respect this distinction and should always send its queries using the proper request type.

1. Getting the current contents of the to-do list: Every time the browser requests your PHP service, either as a GET or POST request, regardless of what parameters (if any) are passed to it, the service's output should always be a text/plain representation of the current to-do list, with one item per line. If the browser requests your service using a GET request, you should disregard any query parameters that may have been passed, and you should simply output the to-do list's current contents.

Parameter Name	Value
(none)	(none)

The items in the list are separated by `\n`. For example, if the to-do list contains the following items:

1. buy meatballs from Ikea
2. shave head
3. watch Flight of the Conchords

then the web service's output would be the following plain text:

```
buy meatballs from Ikea
shave head
watch Flight of the Conchords
```

Your PHP code should store the current to-do list as a text file named **todolist.txt**. Your service should read that file's contents as a string and print them as output. If the service is being used for the first time and no **todolist.txt** file exists, no output is produced.

2. Adding an element to the end of the to-do list: If the browser sends a POST request to your PHP service and passes the following two parameters, the web service should add an item to the end of the list:

Parameter Name	Value
`action`	`"add"`
`item`	the new to-do item's text as a string, such as `"go to the store"`

Do this in your PHP code by reading the to-do list's current contents into your program, concatenating the new item to the end of the contents, and then writing these new longer contents to the file. As stated previously, the service's output should always be the current state of the to-do list, so your response to this query should be to print the list's contents after the item is added.

3. Deleting the first element from the to-do list: If the browser sends a POST request to your PHP service and passes the following parameter, the web service should delete the first item from the front of the list:

Parameter Name	Value
`action`	`"delete"`

Do this in your PHP code by reading the to-do list's current contents into your program, removing the first line's text from the contents, and then writing the new shorter contents to the file. If the to-do list is currently empty or the **todolist.txt** file does not exist, this query has no effect; the list remains empty, no output is produced, and the program should not crash or display an error.

As stated previously, the service's output should always be the current state of the to-do list, so your response to this query should be to print the list's contents after the item is deleted.

4. Setting (replacing) the entire contents of the to-do list: If the browser requests your PHP service as a POST request and passes the following two parameters, then the web service should completely replace the items in the to-do list with a new set of items:

Parameter Name	Value
`action`	`"set"`
`items`	a string representing to-do items separated by `\n` line breaks, such as: `"buy meatballs from Ikea\nshave head\nwatch Flight of the Conchords"`

This third query replaces the entire contents of the to-do list with a new list. You can use this query to implement the rearranging of the list when the user drags an element into a new position. In your JavaScript code, use the DOM to gather the text of all elements of the to-do list into a large string, then send this string as a parameter named items in your Ajax request. In your PHP code, write this string's contents into the **todolist.txt** file on the server, replacing any previous contents of the file.

The `set` query is powerful, but it is also inefficient since all items must be sent from the client to the server. It would be possible to implement the `add` and `delete` operations by having the page's JavaScript code re-send the entire list of elements to the server using the `set` action. But this would be inefficient for large lists of items, so you should not implement those operations in that way.

In this query it is most important for you to understand the difference between GET and POST requests. Since the request you're sending holds a large amount of query data including `\n` line breaks, it potentially would not submit successfully as a GET request (as part of a URL query string). Make sure you are submitting this data as a POST request so that it will reach the server successfully.

As stated previously, the service's output should always be the current state of the to-do list, so your response to this query should be to print the list's contents after the items have been set.

While attempting to write to the file **todolist.txt**, you may see an error of, "failed to open stream: Permission denied". This can occur when the web server's PHP process doesn't have proper permission to write to the file. Try deleting the file on Webster and letting the PHP code recreate it. You might also need to give additional write or execute permissions on your overall `hw8` folder.

We recommend that you debug your queries in Firebug. You can see each Ajax query request in the Console tab. Expand it with the + sign to view the query parameters passed and the web service's response.

Extra Features:

In addition to the previous requirements, you must also complete at least **one of the following additional features**. If you want to complete more than one, that is fine, but only one is required.

- **Loading feedback:** The page as currently described does not give the user very good feedback while it is loading its data from the server, both initially (loading the to-do list) and subsequently (when changes to the list are saved). Add code to display a "Loading" message and/or image on the page while any request to the server is pending. Make this message appear and disappear using Scriptaculous effects.

  To help you test this functionality, the provided **todolist.php** service can accept an optional query parameter named `delay`, whose value should be an integer representing a number of seconds of delay that the server should wait before sending its response.

- **Ability to delete any item:** The currently specified behavior allows you to delete the top item from the to-do list. Modify your page so that it's possible to delete any item from the list, not just the first item. We suggest that you do this in the following way:

  - Add some UI to your page to allow the user to delete a given item. (For example, put a Delete button or icon in each item.) If you add such a UI, you don't need to also retain the Delete Top Item button, but your PHP service should still support the `delete` query as specified earlier that deletes the top item.

  - When your Delete UI element is clicked for a given item, send a request to the server-side PHP service to delete the element. You can choose to implement this delete by sending back the entire list again using the `set` query, or you can implement a second variation of the `delete` query that accepts a second parameter for an index. The latter choice is more efficient, but the former may be easier for you to implement depending on your code. To help you test this functionality, the provided **todolist.php** service's `delete` query can accept a second optional query parameter named `index`, whose value should be an integer representing the 0-based index of the element to delete.

- **HTTP error codes:** Make your PHP web service emit proper HTTP error codes when it is used incorrectly or passed the wrong parameters. Specifically, if the client does any of the following, emit an HTTP error 400 to indicate an invalid request:

  - makes a POST request with no `action` parameter
  - makes a POST request with an `action` parameter of `add` but with no `item` parameter
  - makes a POST request with an `action` parameter of `set` but with no `items` parameter
  - makes a POST request with an `action` parameter whose value is neither `add`, `set`, nor `delete`

Near the top of your HTML file, put a comment saying which extra feature(s) you have completed. If you implement more than one, comment which one you want us to grade (the others will be ignored). Regardless of how many additions you implement, the main behavior should still work as specified. If you have a different idea for an addition to the program, please ask us and we may approve it. Beyond this, you can add any other functionality you like to your page. Such functionality will be ignored for grading as long as it does not interfere with your code's stylistic quality or our ability to test the specified functionality.

Development Strategy:

There is a lot of code to be written, and none of it is being provided to you. It can be challenging to know where to start or how to make the various pieces fit together. We suggest roughly the following development strategy:

- Write an initial page and CSS for its basic appearance. Set up non-dynamic content and an empty to-do list.
- Make your page able to read the current contents of the to-do list, from the instructor-provided PHP web service. Read the list's contents using Ajax and place them onto the page using the DOM.
- Implement the ability to add and delete from the list. Start with the XHTML/CSS code for any UI for doing this, then write the JavaScript to contact the instructor-provided web server to notify it of the action.
- Make the list rearrangeable using Scriptaculous' `Sortable` functionality.
- Add "bling" to your page with Scriptaculous effects.
- Write your own version of the PHP web service. Implement the basic `get` query first, then `add`, then `delete`, then `set`. Remember to look at the query requests and responses in Firebug to debug them.

For reference, our solution has roughly 55 lines of XHTML, 70 lines of CSS, 90 lines of JavaScript, and 40 lines of PHP including blank lines and comments. You do not need to match these totals.

Implementation and Grading:

Your XHTML and CSS code should be well-styled as in past assignments. For full credit, your page must pass the W3C XHTML and CSS **validators**. Express CSS concisely and without unnecessary or **redundant** styles. Format your code with proper whitespace and indentation. Do not place more than one block element on the same line or begin any block element past the 100th character on a line. Place a **comment** header in each XHTML and CSS file containing your name, section, and a brief description of the assignment and the file's contents.

Make extra effort to minimize **redundant code**. Capture common operations as functions to keep code size and complexity from growing. You can reduce your code size by using the `this` keyword in your event handlers.

For full credit, your JavaScript code should pass the provided **JSLint** tool with no errors reported. Use the HTML DOM appropriately. Follow proper style in your Ajax requests, including obeying the proper query request type (GET vs. POST). You should also follow reasonable style guidelines similar to those of a CSE 14x programming assignment. In particular, minimize global variables, avoid redundant code, and use parameters and return values properly. You should not use any other libraries besides Prototype and Scriptaculous, unless given explicit permission from the instructor.

A few **global variables** are allowed, but it is not appropriate to declare lots of them; values should be local as much as possible. If a particular constant value is used frequently throughout your code, declare it as a global "constant" variable named `IN_UPPER_CASE` and use the constant throughout your code.

You should separate content (XHTML), presentation (CSS), and behavior (JS). As much as possible, your JS code should **use styles and classes from the CSS** rather than manually setting each style property in the JS.

For full credit, you must write your code using **unobtrusive JavaScript**, so that no JavaScript code, `onclick` handlers, etc. are embedded into the XHTML code.

Your JavaScript code should have adequate **commenting**. The top of your file should have a descriptive comment header describing the assignment, and each function and complex section of code should be documented.

Your **PHP code** should generate no error or warning messages when run using reasonable sets of parameters.

Format your code similarly to the examples from class. Properly use whitespace and indentation. Use good variable and method names. Avoid lines of code more than 100 characters wide.

Do not place a solution to this assignment on a public web site. Upload your files to the **Webster** server at:

https://webster.cs.washington.edu/*your_uwnetid*/hw8/todolist.html