Web Programming Step by Step

Lecture 21
Scriptaculous
Reading: 12.1 - 12.2

Visual Effects

- **Visual Effects**
- Drag and Drop; Sortable Lists
- Auto-completing Text Fields
- Other Features

Scriptaculous overview

Scriptaculous : a JavaScript library, built on top of Prototype, that adds:

- visual effects (animation, fade in/out, highlighting)
- drag and drop
- Ajax features:
  - Auto-completing text fields (drop-down list of matching choices)
  - In-place editors (clickable text that you can edit and send to server)
- some DOM enhancements
- other stuff (unit testing, etc.)

Downloading and using Scriptaculous

```
<script src="http://www.cs.washington.edu/education/courses/cse190m/09sp/prototype
 type="text/javascript"></script>

<script src="http://www.cs.washington.edu/education/courses/cse190m/09sp/scriptacu
 type="text/javascript"></script>
```
JS

- option 1: link to Scriptaculous on the CSE 190 M web site
  - notice that you must still link to Prototype before linking Scriptaculous
- option 2: download the .zip file from their downloads page, and extract the 8 `.js` files from its `src/` folder to the same folder as your project
- documentation available on their wiki
- Scriptaculous Effects Cheat Sheet

Visual effects (12.2.1)

appear blindDown grow slideDown (appearing)

blindUp dropOut fade fold puff
shrink slideUp squish switchOff (disappearing)

highlight pulsate shake morph
Effect.Move Effect.Scale Effect.toggle (blind) (Getting attention)

script.aculo.us

Click effects above

Adding effects to an element

```
element.effectName();    // for most effects

// some effects must be run the following way:
new Effect.name(element or id);
```

```
$("sidebar").shake();

var buttons = $$("results > button");
for (var i = 0; i < buttons.length; i++) {
  buttons[i].fade();
}
```

- the effect will begin to animate on screen (asynchronously) the moment you call it
- six core effects are used to implement all effects on the previous slides:
    - `Effect.Highlight`, `Effect.Morph`, `Effect.Move`, `Effect.Opacity`, `Effect.Parallel`, `Effect.Scale`

Effect options

```
element.effectName(
  {
    option: value,
    option: value,
    ...
  }
);
```
JS

```
$("my_element").pulsate({
  **duration: 2.0,**
  **pulses: 2**
});
```
JS

- many effects can be customized by passing additional options (note the `{}`)
- options (wiki): `delay`, `direction`, `duration`, `fps`, `from`, `queue`, `sync`, `to`, `transition`
- Q: How would we show two effects in a row on the same element?

Effect events

```
$("my_element").fade({
  duration: 3.0,
  afterFinish: displayMessage
});

function displayMessage(effect) {
  alert(effect.element + " is done fading now!");
}
```
JS

- all effects have the following events that you can handle:
  - `beforeStart`, `beforeUpdate`, `afterUpdate`, `afterFinish`
- the `afterFinish` event fires once the effect is done animating
  - useful do something to the element (style, remove, etc.) when effect is done
- each of these events receives the `Effect` object as its parameter
  - its properties: `element`, `options`, `currentFrame`, `startOn`, `finishOn`
  - some effects (e.g. `Shrink`) are technically "parallel effects", so to access the modified element, you write `effect.effects[0].element` rather than just `effect.element`

Drag and Drop; Sortable Lists

Drag and drop (12.2.2)

Scriptaculous provides several objects for supporting drag-and-drop functionality:

- **`Draggable` : an element that can be dragged**
- `Draggables` : manages all `Draggable` objects on the page
- `Droppables` : elements on which a `Draggable` can be dropped
- **`Sortable` : a list of items that can be reordered**

- Shopping Cart demo

Draggable

```
new Draggable(element or id,
  { options }
);
```
JS

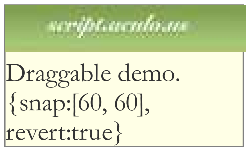
- specifies an element as being able to be dragged
- options: `handle`, `revert`, `snap`, `zindex`, `constraint`, `ghosting`, `starteffect`, `reverteffect`, `endeffect`
- event options: `onStart`, `onDrag`, `onEnd`
  - each handler function accepts two parameters: the `Draggable` object, and the mouse event

Draggable example

```
<div id="draggabledemo1">Draggable demo. Default options.</div>
<div id="draggabledemo2">Draggable demo.
  {snap: [40,40], revert: true}</div>
```
HTML

```
document.observe("dom:loaded", function() {
  new Draggable("draggabledemo1");
  new Draggable("draggabledemo2", {revert: true, snap: [40, 40]});
});
```
JS

Draggable demo.
Default options.



Draggable demo.
{snap:[60, 60],
revert:true}

Draggables

- a global helper for accessing/managing all Draggable objects on a page
- (not needed for this course)
- properties: `drags`, `observers`
- methods: `register`, `unregister`, `activate`, `deactivate`, `updateDrag`, `endDrag`, `keyPress`, `addObserver`, `removeObserver`, `notify`

Droppables

```
Droppables.add(element or id,
  { options }
);                                                                    JS
```

- specifies an element as being able to be dragged
- options: `accept`, `containment`, `hoverclass`, `overlap`, `greedy`
- event options: `onHover`, `onDrop`
  - each callback accepts three parameters: the `Draggable`, the `Droppable`, and the event
  - Shopping Cart demo

Drag/drop shopping demo

```html
<img id="product1" src="images/shirt.png" alt="shirt" />
<img id="product2" src="images/cup.png" alt="cup" />
<div id="droptarget"></div>
```
HTML

```js
document.observe("dom:loaded", function() {
  new Draggable("product1");
  new Draggable("product2");
  Droppables.add("droptarget", {onDrop: productDrop});
});

function productDrop(drag, drop, event) {
  alert("You dropped " + drag.id);
}
```
JS

Sortable

```js
Sortable.create(element or id of list,
  { options }
);
```
JS

- specifies a list (`ul`, `ol`) as being able to be dragged into any order
- implemented internally using `Draggables` and `Droppables`
- options: `tag`, `only`, `overlap`, `constraint`, `containment`, `format`, `handle`, `hoverclass`, `ghosting`, `dropOnEmpty`, `scroll`, `scrollSensitivity`, `scrollSpeed`, `tree`, `treeTag`
- to make a list un-sortable again, call `Sortable.destroy` on it

Sortable demo

```
<ol id="simpsons">
  <li id="simpsons_0">Homer</li>
  <li id="simpsons_1">Marge</li>
  <li id="simpsons_2">Bart</li>
  <li id="simpsons_3">Lisa</li>
  <li id="simpsons_4">Maggie</li>
</ol>
```
HTML

```
document.observe("dom:loaded", function() {
  Sortable.create("simpsons");
});
```
JS

1. Homer
2. Marge
3. Bart
4. Lisa
5. Maggie

Sortable list events

event	description
onChange	when any list item hovers over a new position while dragging
onUpdate	when a list item is dropped into a new position (more useful)

```
document.observe("dom:loaded", function() {
  Sortable.create("simpsons", {
      onUpdate: listUpdate
  });
});
```
JS

- onChange handler function receives the dragging element as its parameter
- onUpdate handler function receives the list as its parameter

`Sortable` list events example

```
document.observe("dom:loaded", function() {
  Sortable.create("simpsons", {
      onUpdate: listUpdate
  });
});

function listUpdate(list) {
  // can do anything I want here; effects, an Ajax request, etc.
  list.shake();
}
```
JS

1. Homer
2. Marge
3. Bart
4. Lisa
5. Maggie

Subtleties of `Sortable` events

- for `onUpdate` to work, each `li` **must** have an `id` of the form *listID_index*

```
<ol id="simpsons">
  <li id="simpsons_0">Homer</li>
  <li id="simpsons_1">Marge</li>
  <li id="simpsons_2">Bart</li>
  <li id="simpsons_3">Lisa</li>
  <li id="simpsons_4">Maggie</li>
</ol>
```
HTML

- if the elements of the list change after you make it sortable (if you add or remove an item using the DOM, etc.), the new items can't be sorted
  - must call `Sortable.create` on the list again to fix it

-->

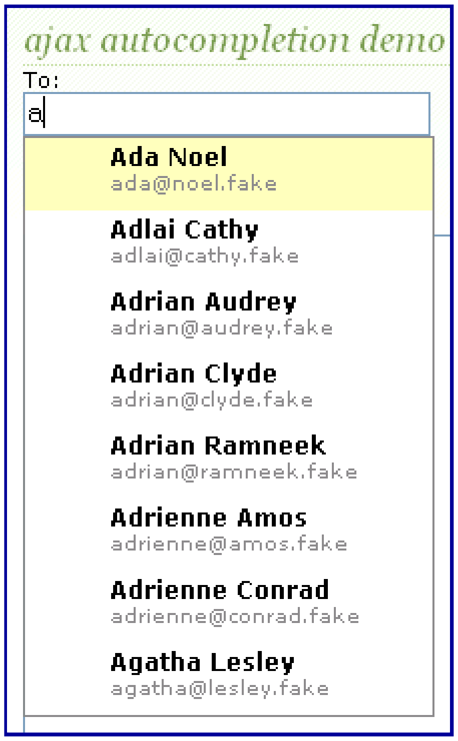
Auto-completing Text Fields

- Visual Effects
- Drag and Drop; Sortable Lists
- **Auto-completing Text Fields**
- Other Features

Auto-completing text fields (12.2.3)

Scriptaculous offers ways to make a text box that auto-completes based on prefix strings:

- `Autocompleter.Local` : auto-completes from an array of choices
- `Ajax.Autocompleter` : fetches and displays list of choices using Ajax



Ada Noel
ada@noel.fake
Adlai Cathy
adlai@cathy.fake
Adrian Audrey
adrian@audrey.fake
Adrian Clyde
adrian@clyde.fake
Adrian Ramneek
adrian@ramneek.fake
Adrienne Amos
adrienne@amos.fake
Adrienne Conrad
adrienne@conrad.fake
Agatha Lesley
agatha@lesley.fake

Using `Autocompleter.Local`

```
new Autocompleter.Local(
  element or id of text box,
  element or id of div to show completions,
  array of choices,
  { options }
);
```
JS

- you must create an (initially empty) `div` to store the auto-completion matches
  - it will be inserted as a `ul` that you can style with CSS
  - the user can select items by pressing Up/Down arrows; selected item is given a `class` of `selected`
- pass the choices as an array of strings
- pass any extra options as a fourth parameter between { }
  - options: `choices`, `partialSearch`, `fullSearch`, `partialChars`, `ignoreCase`

`Autocompleter.Local` demo

```
<input id="bands70s" size="40" type="text" />
<div id="bandlistarea"></div>
```
HTML

```
document.observe("dom:loaded", function() {
  new Autocompleter.Local(
    "bands70s",
    "bandlistarea",
    ["ABBA", "AC/DC", "Aerosmith", "America", "Bay City Rollers", ...],
    {}
  );
});
```
JS

Autocompleter styling

```html
<input id="bands70s" size="40" type="text" />
<div id="bandlistarea"></div>
```
HTML

```css
#bandlistarea {
  border: 2px solid gray;
}
/* 'selected' class is given to the autocomplete item currently chosen */
#bandlistarea .selected {
  background-color: pink;
}
```
CSS

Using `Ajax.Autocompleter`

```js
new Ajax.Autocompleter(
   element or id of text box,
   element or id of div to show completions,
   url,
   { options }
);
```
JS

- when you have too many choices to hold them all in an array, you can instead fetch subsets of choices from the server using Ajax
- instead of passing choices as an array, pass a URL from which to fetch them
  - the choices are sent back from the server as an HTML `ul` with `li` elements in it
- options: `paramName`, `tokens`, `frequency`, `minChars`, `indicator`, `updateElement`, `afterUpdateElement`, `callback`, `parameters`

Ajax.InPlaceEditor

```
new Ajax.InPlaceEditor(element or id,
   url,
   {  options  }
);
```

- options: okButton, okText, cancelLink, cancelText, savingText, clickToEditText, formId, externalControl, rows, onComplete, onFailure, cols, size, highlightcolor, highlightendcolor, formClassName, hoverClassName, loadTextURL, loadingText, callback, submitOnBlur, ajaxOptions
- event options: onEnterHover, onLeaveHover, onEnterEditMode, onLeaveEditMode

Ajax.InPlaceCollectionEditor

```
new Ajax.InPlaceCollectionEditor(element or id,
   url,
   {
      collection:  array of choices,
      options
   }
);
```

- a variation of Ajax.InPlaceEditor that gives a collection of choices
- requires collection option whose value is an array of strings to choose from
- all other options are the same as Ajax.InPlaceEditor

Other Features

- Visual Effects
- Drag and Drop; Sortable Lists
- Auto-completing Text Fields
- **Other Features**

Playing sounds (API)

method	**description**
`Sound.play("`*url*`");`	plays a sound/music file
`Sound.disable();`	stops future sounds from playing (doesn't mute any sound in progress)
`Sound.enable();`	re-enables sounds to be playable after a call to `Sound.disable()`

```
Sound.play("music/java_rap.mp3");
Sound.play("music/wazzaaaaaap.wav");
```
PHP

- to silence a sound playing in progress, use `Sound.play('', {replace: true});`
- cannot play sounds from a local computer (must be uploaded to a web site)

Other neat features

- slider control:

```
new Control.Slider("id of knob", "id of track", {options});
```

- `Builder` - convenience class to replace `document.createElement`:

```
var img = Builder.node("img", {
  src: "images/lolcat.jpg",
  width: 100, height: 100,
  alt: "I can haz Scriptaculous?"
});
$("main").appendChild(img);
```

- Tabbed UIs