DHTML Techniques for an Inclusive Web 2.0
Or
How to Write JavaScript That Doesn’t Suck
Techniques We'll Talk About

• Play nice with others: mashups and portals
  – Don't override built-in types
  – Namespacing and uniqueness
  – Addressing the DOM

• Accessibility
  – Keyboard Handling
  – Supporting assistive technologies
  – State of the standards

• Using toolkits
Mashups

• What's a mashup?
  – A combination of data and markup from different sources
  – Aggregating content from Web feeds, services, etc.

• Chunks of markup and JavaScript that:
  – Share the same DOM
  – Are mutually ignorant of the others' existence

• The same technical challenges as in a portal
Playing Nice With Others
Don't Modify Built-in Types

- JavaScript is wickedly dynamic… but use it carefully
- Our changes can effect other programs.
- In JavaScript, you can easily augment an object with simple assignment.

```javascript
myObject.myMethod = function() { }
```
Namespace Everything

- Namespaces will help avoid collisions.
- Encapsulate functionality nicely.
- Provides some documentation.
- How do we do this?
- JavaScript has a global object which holds top level functions and global variables.
  - Create an object in the global space and put everything in it.
Sample Code:

Defining Namespaces

```javascript
var myNameSpace = myNameSpace || { }

myNameSpace.foo = function () { alert("bar"); }

myNameSpace.foo();
```
...Even Your Markup

- In a portal we don’t control the whole page
- Markup can show up in multiple places on the page
- Unique ids are the key to addressing particular elements
Sample Markup: Semantic IDs

Example:

tool.context.widget.element

becomes

announcement.list.navToolBar.edit
Sample Code: Finding By ID

// Find an element using an explicit ID
var elm =
document.getElementById('announcement.list.navToolBar.new');
Accessibility
DHTML & Accessibility

- Just when we thought we had Web accessibility in hand...
  - Not enough information: opaque user interface markup
  - Non-mouse usage is often overlooked completely
  - Dynamically updated information can be challenging
Assistive Technologies

• Used by people with disabilities to perceive and control the user interface.
  – Screen reader
  – Screen magnifier
  – On-screen keyboard

• Most assistive technologies use built-in operating system APIs for reflecting the user interface:
  – Windows: MSAA/IAccessible2
  – Linux: ATK
  – Mac: Universal Access for Carbon and Cocoa
Opaque Markup

- HTML has limited semantics:
  - Forms, links, buttons, lists, tables
- Dynamic UIs are built from generic HTML tags
  - For example, `<div>` and `<span>`
  - No `<slider>` or `<menu>` tags available
- Assistive technologies attempt to read the underlying document markup
- Problem: how do assistive technologies represent DHTML interfaces to the user?
Example of Opaque Markup:

- A DHTML menu bar without semantics:

```html
<ol id="menubar">
  <li id="editMenu">Edit
    <ol>
      <li>Cut</li>
      <li>Copy</li>
      <li>Paste</li>
    </ol>
  </li>
</ol>
```
Opaque Markup: Solution

- Provide additional semantics or metadata that describe the role, function, and states of DHTML user interfaces. How?
- ARIA (Accessible Rich Internet Application)
  - http://www.w3.org/TR/aria-roadmap/
  - http://www.w3.org/TR/aria-role/
  - http://www.w3.org/TR/aria-state/

- Working standard from the W3C, led by Fluid partner Rich Schwerdtfeger
ARIA

• Attributes added to your HTML markup that describe the function and states of your UI components
• These map to all your familiar types of UI widgets:
  – Dialog
  – Slider
  – Progress Bar
  – Tab Panel
  – Menu bar
Sample Code: ARIA Roles

- A DHTML menu bar with ARIA semantics:

```html
<ol id="menubar" role="wairole:menubar">
  <li id="editMenu" role="wairole:menuitem" haspopup="true">Edit
    <ol>
      <li role="wairole:menuitem">Cut</li>
      <li role="wairole:menuitem">Copy</li>
      <li role="wairole:menuitem">Paste</li>
    </ol>
  </li>
</ol>
```
The Value of ARIA

• DHTML accessibility is a short-term problem
• Long-term, it has the potential to make web accessibility much better
• Assistive technology developers have had a decade to get desktop GUI accessibility right
• By mapping rich-client interfaces with ARIA, web interfaces can leverage this support
Non-mouse accessibility

- Most rich Web interactions *require* the mouse.
- Standard tabbing strategy in browsers is tedious
- Keyboard bindings will enable lots of non-mouse control strategies, including:
  - On-screen keyboard
  - Single switch
  - Voice control
Tabbing and tabindex

- Browsers used to only allow you to use tab to focus form elements and links
- There is an HTML attribute called “tabindex” that allows you to tell the browser how to handle tabbing
- Strategy:
  - allow the user to tab to user interface widgets
  - use the arrow keys allow selection within
  - Add JavaScript handlers for arrow keys
Sample Markup: Tabindex

<ol id="menubar" tabindex="0">
  <li id="editMenu">Edit
    <ol>
      <li><a href="/cut" tabindex="-1">Cut</a></li>
      <li><a href="/copy" tabindex="-1">Copy</a></li>
      <li><a href="/paste" tabindex="-1">Paste</a></li>
    </ol>
  </li>
</ol>
jQuery(elmRef).keydown( function(event) { 
  switch(event.keyCode){
    case 40: // 40 = Arrow Down
      // highlight the next element
      jQuery(elmRef).removeClass('highlight');
      var nextElm = jQuery(elmRef).next();
      jQuery(nextElm).addClass('highlight');
    case 38: // 38 = Arrow Up
      // highlight the prev element
  }
});
DHTML Accessibility Advice

• Out of date accessibility standards and legislation
  – Technology-specific standards go out of date easily
  – Current standards impede innovation

• Strategy:
  – Embrace JavaScript
  – Use emerging standards: ARIA, tabindex, etc.
  – Degrade gracefully
  – Think about the use case for accessibility
  – Start with accessibility, don’t add it at the end
Accessibility Meta Concepts

1. Label everything
   • Design for variable font and screen sizes
   • It has to work with the keyboard
JavaScript Toolkits
JavaScript is Painful

• Four points of pain:
  – Browser bugs and inconsistencies
  – DOM traversal and selection
  – Event management
  – AJAX
Why use a JS Framework?

• Leverage someone else's hell
• Someone else wrote it...
• ...and someone's already tested it
• The framework handles the fundamentals
Summary

• Key techniques:
  – Don't poke around with the built in types
  – Namespace everything
  – Be careful of vacuuming up the DOM
  – Make it work with the keyboard
  – Add ARIA roles and states

• Toolkits will save you time. We like:
  – jQuery for just about everything: “jQuery is the DOM”
  – Dojo for black-box, accessible widgets (for now)
Join us for a JavaScript BOF

- JavaScript Birds of a Feather
- 3:40 pm in the Laguna room
- More time to talk about JavaScript
- Bring your ideas and questions!