Writing Fearless Javascript

for portlets, widgets, and portals...

Fluid

Fearless Javascript

Colin Clark
Fluid Project Technical Lead, Adaptive Technology Resource Centre, University of Toronto

Eli Cochran
user experience developer
University of California, Berkeley

Antranig Basman
CARET, Cambridge University

Fluid

http://fluidproject.org

Fluid

Douglas Crockford, senior JavaScript Architect at Yahoo!, Best known as one of the developers of JSON

John Resig, JavaScript Evangelist for the Mozilla Corporation, creator of jQuery, author of Pro Javascript Techniques

jQuery

Fluid

The jQuery Javascript Library. We love jQuery and JS toolkits in general. Please don't be turned off by our focus on jQuery. There are a lot of great JS Toolkits and libraries out there and the techniques we discuss today are applicable across toolkits.

In-class examples...

The jQuery Javascript Library. We love jQuery and JS toolkits in general. Please don’t be turned off by our focus on jQuery. There are a lot of great JS Toolkits and libraries out there and the techniques we discuss today are applicable across toolkits.

```
svn co https://source.fluidproject.org/
 svn/sandbox/javascript-workshop/trunk
```

Fluid

Firebug

- Debugger
- Profiling tool
- DOM inspector
- Interactive console
- Every JS programmer's best friend!

Download it at: http://www.getfirebug.com/

Fluid

The jQuery Javascript Library. We love jQuery and JS toolkits in general. Please don’t be turned off by our focus on jQuery. There are a lot of great JS Toolkits and libraries out there and the techniques we discuss today are applicable across toolkits.

Setting Up FireBug

1. In Firefox 2, go to
   http://www.getfirebug.com/
2. Click the big Install Firebug 1.0 button
3. Restart Firefox after installing
4. Enable Firebug by clicking the green checkmark
5. Select the Console tab
6. Choose Options > Larger Command Line

Fluid

The jQuery Javascript Library. We love jQuery and JS toolkits in general. Please don't be turned off by our focus on jQuery. There are a lot of great JS Toolkits and libraries out there and the techniques we discuss today are applicable across toolkits.

Fluid

Ridiculous question?
After all this is a Web 2.0 world, everywhere we turn, it's AJAX this, DHTML that, social networks, but it is a reasonable question

Why Javascript?

Fluid

Ridiculous question?
After all this is a Web 2.0 world, everywhere we turn, it's AJAX this, DHTML that, social networks, but it is a reasonable question

Why Javascript?

Why DHTML?

Fluid

Ridiculous question?
After all this is a Web 2.0 world, everywhere we turn, it's AJAX this, DHTML that, social networks, but it is a reasonable question

Why Javascript?

Why DHTML?

Why AJAX?

Fluid

Ridiculous question?
After all this is a Web 2.0 world, everywhere we turn, it's AJAX this, DHTML that, social networks, but it is a reasonable question

Why Javascript?

Why DHTML?

Why AJAX?

Why bother?

Fluid

Ridiculous question?
After all this is a Web 2.0 world, everywhere we turn, it's AJAX this, DHTML that, social networks, but it is a reasonable question

Web 2.0

aka. Web 3.0 alpha

Fluid

Definitions

DHTML = **D**ynamic **HTML**

AJAX = **A**synchronous **J**avascript **a**nd **X**ML

RIA = **R**ich **I**nternet **A**pplications

DHTML = Dynamic HTML = client-side interactivity
AJAX = Asynchronous Javascript and XML = small, responsive, transactions between client and server
the XML part of AJAX is used very rarely these days, being replaced by other data formats, such as JSON
RIA = Rich Internet Applications = rich interactions

Examples

Fluid

Flash?
Silverlight?

- Great technologies
  - powerful
  - fast
  - cross-platform (well, at least Flash has so far proven to be so)
  - Great for animation and media, I mean really great!
  - for some functionality Flash is better, at least for now, such as animation.

I've seen some awesome things done with DHTML based animations, especially using some of the new technologies that are being adopted such as Canvas and SVG, but for now Flash is much easier to author and has better results. But we're not talking about animation or video here, we're talking web apps, data, information, funcitonality and DHTML and AJAX can do anything that Flash and Flex, and Silverlight can do.

- Political
  - Open vs. Closed
- Practical
  - Also Open vs. Closed

Jump out to Fluid Layout Customizer in uPortal

DHTML/AJAX

- open-standards
- transparent
- works within a web page, not against it
- accessible

not just to screen readers and adaptive devices but to other components and other technologies, HTML and DOM are the currency of the web, components built using DOM can talk to each other in DOM, the HTML document object model becomes the API that allows us to build along side and on top of each other.

Javascript

- breezy little scripting language?
- industrial-strength programming language?
- both?

Fluid

Old-school JS

- Namespace pollution
- Obtrusive scripting: placing code and event handlers into markup
- Attempts to map a class-based system onto a class-less language
- Poor encapsulation: data and functions scattered about

Fluid

```
var currentTab = "";
var currentHighlight = "";

function showTab(tabKey) {
  // show tab stuff
}

function hideTab(tabKey) {
  // hide tab stuff
}

function init(tabSet) {
  // init block
}
```

Fluid

Javascript 101

Fluid

JavaScript is Different

- Everything is an object
- Extremely loose type system
- No classes
- Functions are first class
- Lots of annoying quirks

Fluid

Fun History

- Written at Netscape by Brendan Eich
- Original goal: kinda like LISP, but without all the brackets
- Rushed to market, bugs and all
- Microsoft reverse-engineered it
- ECMA was a browser war battleground

Fluid

Part 1: The Basics

- Variables
- Numbers
- Strings
- null
- undefined
- Objects and Arrays

Fluid

Defining Variables

- Define variables with var
- No need to declare types

```
var mango = "yum";
mango = 12345;
mango = false;
```

Fluid

Defining Variables

- If you omit var, it will be defined as a global variable. Bug!
- Accident prone; JavaScript won't warn you!

```
rottenTomato = "gross!"; // This is global
```

Fluid

Numbers and Strings

- Numbers
  - lots of precision
  - no distinction between floats and ints
  - NaN !== NaN
- Strings
  - Nearly Unicode
  - Immutable
  - No character type

Fluid

Null vs. Undefined

- null is the "nothing" value
- undefined is extremely nothing
  - Default parameter for defined variables
  - Also the value of undefined or missing members of objects

Fluid

Null vs. Undefined

think of undefined as "I've never heard of this thing"

and null as "nothing here"

Fluid

Null vs. Undefined

```
var foo;
foo === undefined

console.debug(someRandomVariable); // undefined
```

Fluid

Truthy and Falsey

- JavaScript does a lot of automatic type coercion
- This is scary, but helpful
- Shades of true and false
- Helpful when evaluating arguments

Fluid

Falsey Values

- `false`
- `null`
- `undefined`
- `""`
- `0 (zero)`
- `NaN`

Fluid

Truthy Values

Everything else is truthy...

- `true`
- Objects
- Non-zero numbers
- Non-empty Strings

Careful...

`-1`,`"false"`,`"0"` are all true

Fluid

Type Coercion

JavaScript does automatic type coercion in several scenarios

- This can be very dangerous!
- The + operator, for example:

```
1 + 2 === 3
"$" + 1 + 2 === "$12" not $3
+"12" === 12  // Implicit coercion
```

Fluid

Equal vs. Equivalent

Comparisons are coercive:

```
1 == "1" // true

0 == false // true
```

Non-coercive comparison:

```
0 === false // false

1 !== "1" // true
```

Fluid

Using Truthy and Falsey

- Checking for valid arguments before operating on them
- Substituting defaults
- Be careful with arguments that genuinely might be falsey, such as numbers

Fluid

Truthy/Falsey Example

```
function pie (apples, cinnamon) {
   // Only cut the apples if they aren't null or undefined.
   if (apples) {
      alert("Apples!");
      //apples.cut();
   }

   var nutmeg = "nutmeg";

   // If cinnamon is falsey (null or undefined),
   // assign spice to nutmeg instead
   var spice = cinnamon || nutmeg;

   if (apples && spice) {
      return "Tasty";
   } else {
      return "Crusty pie";
   }
}

pie(null, null); // "Crusty pie"
```

Fluid

Creating Objects

```
// Using the new keyword
var foo = new String("foo");
var answer = new Number(42);
var fiveThings = new Array(5);

// Using object literal notation
var myObject = {};

// Literals are more succinct
var foo = "foo";
var answer = 42;
var fiveThings = [1, 2, 3, 4, 5];
var basketOfFruit = {
   mangosteens: 5,
   kiwis: false,
   figs: "plenty"
};
```

Fluid

Objects Are Loose Containers

- At their core, objects are just maps
- `new Object()` or `{}` returns an empty container of key/value pairs
- Keys can be any string, values can be anything
- Two different ways to access members:

```
basketOfFruit.kiwis; // dot notation
basketOfFruit["figs"]; // subscript notation
```

- You can add new members to any object at any time

Fluid

Objects Are Modifiable

```
var basketOfFruit = {};

// New property
basketOfFruit.apples = "macintosh";

// New method
basketOfFruit.eat = function () {
    return “tasty”;
}
```

Fluid

No Classes

- JavaScript doesn't have any concept of classes
- Methods are just properties in a container:
  - pass them around
  - modify them
  - delete them

Fluid

No Classes

Duck typing:

If it walks like a duck and quacks like a duck, it's a duck.

Fluid

Part 2: Functions & Scope

- Functions are first class
- Ways to call a function
- Determining types
- Prototypal Inheritance
- Understanding this
- Closures

Fluid

First Class Functions

- Functions are data
- You can assign them
- You can pass them as arguments
- You can return them as results
- Functions can contain member variables

Fluid

Defining and Using Functions

```
var juicePuree = function (aFruit) {
    return puree(aFruit);
};
function squeezeJuice(aFruit) {
    return squeeze(aFruit);
}
function popsicle(juiceMakerFn, fruit) {
    var juice = juiceMakerFn(fruit);
    freeze(juice);
}
```

Fluid

What Does This Mean?

- No more anonymous inner classes!
- You can pass bits of logic around and have them be invoked later
- Callbacks are easy to write and ubiquitous

Fluid

Ways to Call a Function

```
// Plain old function call
popsicle(arguments);

// Calling a method on an object
var meal = {appetizer: "kiwi", dessert: function popsicle() { ..}};
meal.dessert(arguments);
meal["dessert"](arguments);

// Tricky context substition
var dessert = meal.dessert;
dessert.apply(thisObject, [arg1, arg2]);
dessert.call(thisObject, arg1, arg2);

// As a constructor...
```

Fluid

Constructor Functions

- No classes in JavaScript, so how do we define new objects?
- Instantiate a function using the new keyword
- Any function can be used as a constructor
- Conventional to use `CamelCaseLikeThis`.

Fluid

Constructor Functions

```
function Apple(type, colour) {
  this.type = type;
  this.colour = colour;
};

var macintosh = new Apple("macintosh", "red");
```

Fluid

.constructor

- All objects have a `.constructor` property
- It points to the function that created the instance

```
var macintosh = new Apple("macintosh", "red");
macintosh.constructor === Apple

var plainOldObject = {foo: "foo"};
plainOldObject.contructor === Object
```

Fluid

Determining Types

- JavaScript has a typeof keyword for determining type

```
var plum = "yum";
if (typeof plum === "string") {
    alert("Plum is  a String!");
}
```

Fluid

Typeof is Innacurate

```
// Inaccurate results for some built-in types
typeof(new Object()) // 'object'
typeof(new Array()) // 'object'
typeof(new Function()) // 'function'
typeof(new String()) // 'string'
typeof(new Number()) // 'number'
typeof(Boolean()) // 'boolean'
typeof(null) // 'object'
typeof(undefined) // 'undefined'
```

Fluid

Typeof is broken

```
// typeof is useless for custom types
function Apple(type, colour) {};
typeof(new Apple()) // 'object'
```

Fluid

Better Ways to Check Types

```
function DragonFruit() {};

// Check the .constructor property
var myFruit = new DragonFruit();
myFruit.constructor === DragonFruit;

// Use instanceof
(myFruit instanceof DragonFruit)
```

Fluid

Prototypal Inheritance

- Did I mention that JavaScript doesn't have classes?
- Inheritance is based on prototypes:

> "Give me an object that is like that one over there."

- Constructor functions have a .prototype property
- It points to an object that provides base functionality

Fluid

Setting Prototypes

```
var Animal = function() {
      this.sound = "growl";
      this.species = "mammal";
};

var Cat = function(colour) {
      this.colour = colour;
      this.sound = "meow";
};
Cat.prototype = new Animal();

var felix = new Cat(black);
felix.sound === "meow"; // Resolved directly at Cat.
felix.species === "mammal"; // Resolved at Cat.prototype.species

// Felix now has a species property, and the value defined by Animal will be ignored.
felix.species = "insect";
```

Fluid

Dynamically Modifying Types

```
Cat.prototype.claws = function () {
    return "Ouch!";
}

var garfield = new Cat();
garfield.claws(); // "Ouch!"

// Felix is automatically updated with the claws function.
felix.claws(); // Also "Ouch!"
```

Fluid

Don't Extend Built-in Types

- Dynamic objects are awesome. But dangerous.
- Looseness allows us to change contracts for everyone
- Different scripts share the same browser window
- They all share the basic types
- Modifying built-in functionality will break things

Fluid

Breaking Built-in Types

```
Object.prototype.keys = function () {
    var keys = [];
    for (prop in this) {
       keys.push(prop);
    }
    return keys;
}

var myKeys = {foo: "foo", bar: "bar"}.keys();
console.debug(myKeys); // [foo, bar, keys];
```

Fluid

this

Fluid

Context and `this`

- JavaScript this pointer seems wild and unpredictable
- It points to different objects depending on the context
- Subtle, confusing, and powerful

Fluid

Global `this`

- In the global space, this points to the Global object
- In a browser, `this === window`

```
this.pie = "apple";
pie === "apple";
window.pie === "apple";
```

Fluid

Function Scope `this`

```
function bake(aPie) {
  this.pie = aPie;
};
bake("cherry");
pie === "cherry"
window.pie === "cherry"
```

Fluid

Object Instances

- The new keyword instantiates a new object
- *this* pointer is automatically assigned to the new instance

Fluid

Object this

```
function Pie(fruit, flour) {
    this.fruit = fruit;
    this.bottomCrust = flour;
    this.topping = "chocolate";
};

Pie.prototype.showMeThis = function () {
    return this;
}

var applePie = new Pie("apple", "wholeWheat");
applePie.showMeThis() === applePie;
```

Fluid

Constructors are Just Functions

```
Pie("cherry", "flax");
window.fruit === "cherry";
window.bottomCrust === "flax";
window.topping === "chocolate";
```

Fluid

Borrowing Functions and *this*

```
var applePie = new Pie("apple", "wholeWheat");
var showPie = applePie.showMeThis;
      // Note the lack of brackets.

// We've detached the method from its context
// so 'this' reverts back to the Global object
showPie() === window;
```

Fluid

Substituting Contexts

```
var cake = new Cake("peach", "chocolate");
var showPie = applePie.showMeThis;

var cakePie = showPie.call(cake, null);
cakePie === cake;
```

Fluid

No Block Scope

```
// Blocks don't have scope in JavaScript
var fruits = ["apples", "oranges", "lemons"];
for (var i = 0; i < fruits.length; i++) {
  var currentFruit = fruits[i];
}
// The incrementor and currentFruit variables
are still in scope!
i === 3;
currentFruit === "lemons";
```

Fluid

Functions Have Scope

```
function cherryPicker(cherryTree) {
    function picker(branch) {
        var cherries = branch.pickCherries();
        cherries.wash();
        return cherries;
    }

    var basketOfCherries = [];
    for (branch in cherryTree) {
        var someCherries = picker(branch);
        basketOfCherries.push(someCherries);
    }

    return basketOfCherries;
}

var cherries = cherryPicker(hugeTree);
picker === undefined;
```

Fluid

Closures

- You can define a function inside another function
- Inner functions have access to the outer function's variables
- A closure is formed by returning the inner function from the outer function
- The inner function will still have access to all the variables from the outer function

Fluid

A Simple Closure

```
function addNumbers (a, b) {

    function addEmUp (c) {
       return a + b + c;
    }

    return addEmUp;
}
var add = addNumbers(1, 2); // add is a Function
add(3); // Result is 6.
```

Fluid

Closures Simplify Event Handlers

```
function showMessage(messageToDisplay) {
     var todaysPie = "Banana creme pie";

     return function(event) {
        alert(messageToDisplay + " " + todaysPie);
        showPictureOfPie(event.target, todaysPie);
     }
}
```

```
var clickHandler = showMessage("Welcome to my pie shop. Today's pie is:")
$(element).click(clickHandler);

$(element).click() // Shows an alert that reads "Welcome to my pie shop. Today's pie is: Banana creme pie"
```

Fluid

How Is This Possible?

- Each invocation of a function runs in its own execution context
- The execution context holds helpful information:
  - arguments to the function call
  - a scope chain, which stores variable declarations
  - the `this` pointer
- Identifiers are resolved against the context's scope chain
- This means that inner scopes can access variables in the outer scope

Fluid

Global

Variables[]

- messageClickHandler

showMessage()

Variables[]

- messageToDisplay
- todaysPie

function()

Variables[]

- event

Fluid

Javascript Toolkits

Fluid

what's the problem?

Fluid

Javascript is tricky?

Fluid

No

Fluid

Javascript is easy

Fluid

What is hard?

Fluid

Example: Events

Fluid

Example: Events

```
Microsoft Internet Explorer
   = element.attachEvent('onclick',doSomething);

W3C Way (everyone else)
   = element.addEventListener('click',doSomething,false);
```

Fluid

Example: Events

```
if (myBtn.addEventListener) {
    myBtn.addEventListener('click',doSomething(),false);
} else {
    myBtn.attachEvent('onclick',doSomething());
}
```

Fluid

Example: Events

```
function addEvent(element, type, handler) {
    if (element.addEventListener) {
       element.addEventListener(type, handler, false);
    } else {
       element.attachEvent('on'+type, handler);
    }
}
```

Fluid

Example: Events

Fluid

Example: Events

```
        }
}
// a counter used to create unique IDs
addEvent.guid = 1;

function removeEvent(element, type, handler) {
        if (element.removeEventListener) {
                element.removeEventListener(type, handler, false);
        } else {
                // delete the event handler from the hash table
                if (element.events && element.events[type]) {
                        delete element.events[type][handler.$$guid];
                }
        }
}

function handleEvent(event) {
        var returnValue = true;
        // grab the event object (IE uses a global event object)
        event = event || fixEvent(((this.ownerDocument || this.document || this).parentWindow || window).event);
        // get a reference to the hash table of event handlers
        var handlers = this.events[event.type];
        // execute each event handler
        for (var i in handlers) {
                this.$$handleEvent = handlers[i];
                if (this.$$handleEvent(event) === false) {
                        returnValue = false;
                }
        }
        return returnValue;
}

function fixEvent(event) {
        // add W3C standard event methods
        event.preventDefault = fixEvent.preventDefault;
        event.stopPropagation = fixEvent.stopPropagation;
        return event;
}

fixEvent.preventDefault = function() {
        this.returnValue = false;
};

fixEvent.stopPropagation = function() {
        this.cancelBubble = true;
};
```

Fluid

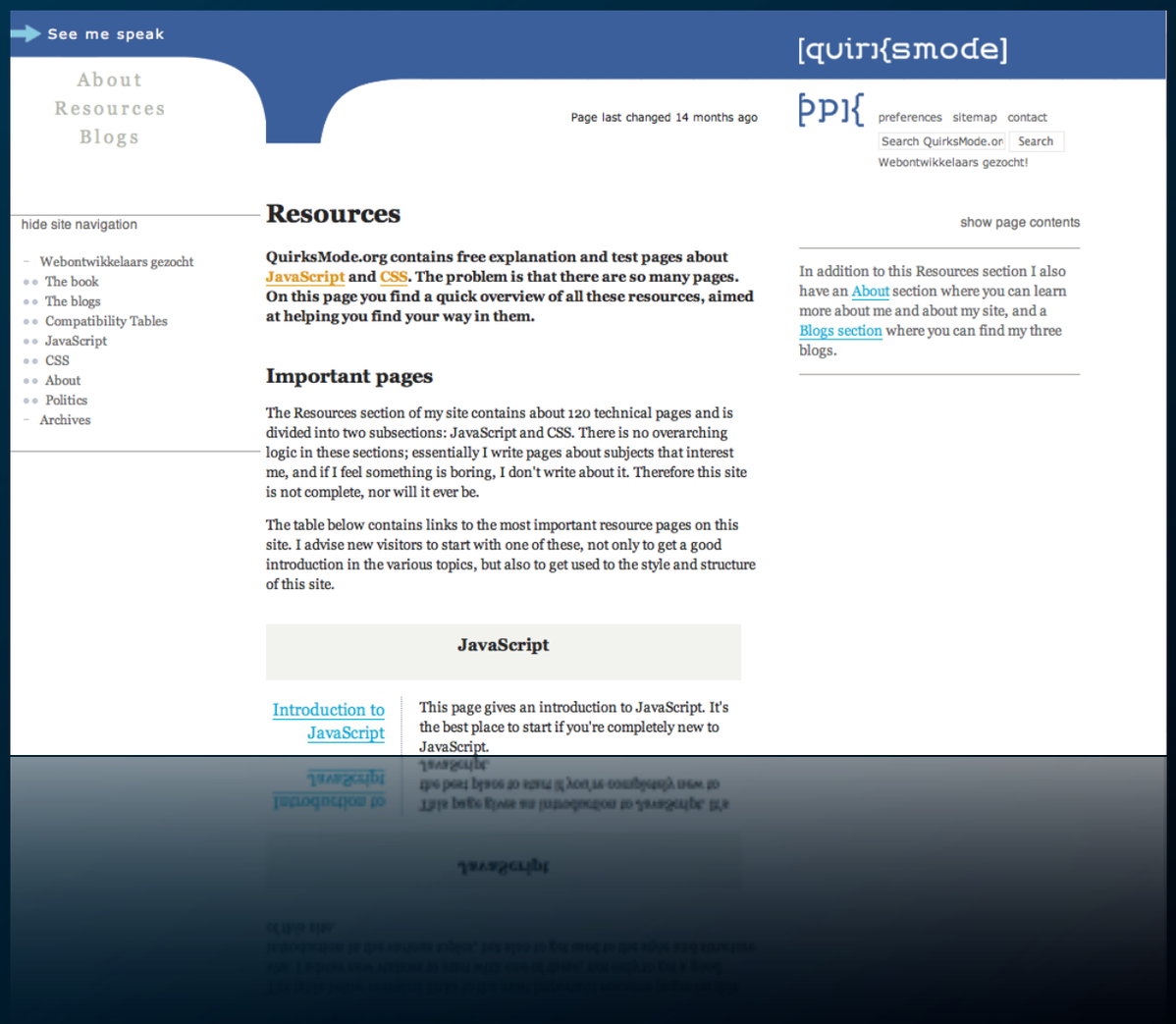
browser differences...

too many to list...

Fluid

QuirksMode

www.quirksmode.org



Fluid

Example: DOM Selection

Fluid

Example: DOM Selection

visual representation of the
DOM for http://uPortal.org

Fluid

Example: DOM Selection

Fluid

Example: DOM Selection

find something and do something with it

Fluid

Example: DOM Selection

```
var elm = document.getElementById('myButton');

var elms = document.getElementsByTagName('tr');
```

Fluid

Example: DOM Selection

```
var myItems =
  document.getElementById("myList")
    .getElementsByTagName("li");
```

Fluid

Example: DOM Selection

Fluid

Example: DOM Selection

```
by id
by class
by parent
by child
by sibling
by attribute
by function
```

Fluid

Example: DOM Selection

by a combination of any and all of these

Fluid

What is hard?

Fluid

What is hard?

- browser inconsistencies and bugs

Fluid

What is hard?

- browser inconsistencies and bugs
    - Browser Abstraction

Fluid

What is hard?

Fluid

What is hard?

- complex data and user interfaces in web applications

Fluid

What is hard?

- complex data and user interfaces in web applications
    - DOM traversal, selection, and manipulation

Fluid

What is hard?

Fluid

What is hard?

- subtle and varied high-quality user interactions

Fluid

What is hard?

- subtle and varied high-quality user interactions
    - event binding must be easy and dynamic

Fluid

What is hard?

Fluid

What is hard?

- the call and response of asynchronous client-server interaction

Fluid

What is hard?

- the call and response of asynchronous client-server interaction
    - quick, responsive, bullet-proof AJAX functionality

Fluid

foundational toolkits

Fluid

enabling and enhancing

Fluid

leveraging someone else's pain

Fluid

thousands of toolkits?

Fluid

thousands of toolkits?

- single problem solutions

Fluid

thousands of toolkits?

- single problem solutions
- widgets

Fluid

The Fluid Criteria

- Cross-browser support
- Easy debugging
- Event abstraction
- A solid DOM manipulation library
- A strong community and clear roadmap for improvements
- Accessibility

Fluid

Leading Toolkits

- Prototype / script.aculo.us
- Dōjō / dijit
- YUI YAHOO!
- GWT Google™
- MooTools
- MochiKit
- jQuery

Fluid

do more or less the same things, but they do them in very different ways
what I mean from by a philosophy is that the toolkits reflect the linguistic-bias of the people who wrote them
prototype feels like Ruby
Dojo attempts to bring to Javascript the classical inheritance of Java
in GWT you actually write Java which then is compiled into Javascript ... did half the room just wake up?

Why use jQuery?

Fluid

- The Fluid project had done extensive work in Dojo before settling on jQuery.
- We had high hopes for Dojo because it was the first library to really embrace accessibility... and we're all about accessibility on Fluid
- But we ran into issues with Dojo and their widget library Dijit which made it difficult to move forward...
    - So after a frantic week of rewriting one of our components in a handful toolkits we settled on...

- The Fluid project had done extensive work in Dojo before settling on jQuery.
- We had high hopes for Dojo because it was the first library to really embrace accessibility... and we're all about accessibility on Fluid
- But we ran into issues with Dojo and their widget library Dijit which made it difficult to move forward...
    - So after a frantic week of rewriting one of our components in a handful toolkits we settled on...

jQuery

jQuery was built with the skills of Web developers in mind. It doesn't try to be another language, it attempts to support and advance the inherent strengths of Javascript, prototypical inheritance, functions as first-class objects, and it's strengths are in how easy it is to find and work with the DOM - the HTML, the stuff of web sites and web applications

jQuery

Write Less, Do More

jQuery was built with the skills of Web developers in mind. It doesn't try to be another language, it attempts to support and advance the inherent strengths of Javascript, prototypical inheritance, functions as first-class objects, and it's strengths are in how easy it is to find and work with the DOM - the HTML, the stuff of web sites and web applications

jQuery

Fluid

jQuery

- DOM selection and manipulation
  - using CSS selectors
- Event management
- Effects and Animation

Fluid

jQuery

- DOM selection and manipulation
  - using CSS selectors
- Event management
- Effects and Animation
- AJAX

Fluid

jQuery

- DOM selection and manipulation
  - using CSS selectors
- Event management
- Effects and Animation
- AJAX
- jQuery UI

Fluid

finding something without jQuery

```
var myItems =
  document.getElementById("myList")
    .getElementsByTagName("li");
```

Fluid

finding something with jQuery

```
var myItems = jQuery('#myList li');
```

Fluid

finding something with jQuery

```
jQuery('#myList li.highlighted');

jQuery('table.foo tbody tr:even');

jQuery('div > p');
```

Fluid

jump out to 1st example

doing something

```
function stripeListElements(listID) {
    // get the items from the list
    var myItems =
    document.getElementById(listID).getElementsByTagName("li");
    // skip line 0 as it's the header row
    for(count = 0; count < myItems.length; count++) {
        if ((count % 2) === 0) {
            myItems[count].className = "odd";
        }
    }
}
```

Fluid

doing something with jQuery

```
function stripeListElements(listId) {
    jQuery('#' + listId + " li:even").addClass("odd");
}
```

Fluid

zero-based foo

doing something with jQuery

```
function stripeListElements(listId) {
    jQuery('#' + listId + " li:even").addClass("odd");
}
```

Fluid

zero-based foo

Learning jQuery

Fluid

online documentation

jQuery.com

Fluid

The basics

Fluid

Finding stuff in the DOM

```
jQuery('selector')
```

selectors can be

- tags: `jQuery('tr')`
- ids: `jQuery('#myId")`
- classes: `jQuery(".myClass")`
- and various combinations there of

Fluid

Doing something with it

```
$('.some-hidden-thing').show();

$('.some-hidden-thing').fadeIn('slow');

$('<li>A new list item</li>').appendTo('#myList');

$('#myList li:last').replaceWith('<li>A new list item</li>');

$('div.container').clone().appendTo(‘body’);
```

Fluid

Attaching Events

```
$('.button').click(function(){
     doSomething();
});

$('.button').hover(function(){
     jQuery(this).addClass('hilite');
}, function(){
     jQuery(this).removeClass('hilite');
});

$('.button').focus(function(){
     jQuery(this).addClass('hilite');
});

$('.button').blur(function(){
     jQuery(this).addClass('hilite');
});
```

Fluid

Attaching Events : chaining

```
$('.button').addClass('buttonClass);

$('.button').click(doSomething()});
```

Fluid

Attaching Events : chaining

```
$('.button').addClass('buttonClass).click( doSomething());
```

Fluid

One Special Event

```
$(document).ready();
```

```
$(document).ready(
    jQuery('#aList li:even").addClass("odd");
);
```

Fluid

jQuery

- Google (in Google Code)
- Bank of America
- Source Forge
- Sakai
- Drupal
- BBC
- Dell
- Slashdot
- Engadget

Fluid

How to Build a Portal-Friendly UI

Fluid

Portals Are Hard

- Multiple instances: namespacing is essential
- Lots of different JavaScript code running
- High chance of collisions
- Can't expect control of the document

Fluid

Writing Portal-Friendly JavaScript

- Put code in a unique namespace
- Use closures for privacy
- Be unobtrusive
- Support multiple instantiation
- Constrain selectors to a specific fragment

Fluid

Start With a Unique Namespace

```
// Add on to the fluid object if it exists
// otherwise initialize it as an empty object.

var fluid = fluid || {}
```

Fluid

Use Closures for Privacy

```
(function() {
    function myPrivateFunction () {
    }

    fluid.Tabs = function () {
        // Constructor function.
    };
}) ();
```

Fluid

Keep Common Aliases Private

Pass important dependencies in as an argument to the closure:

```
jQuery.noConflict();

(function ($) {
  // The $ variable is only visible inside our
  private space.
  $ === jQuery;
}) (jQuery);

$ === undefined;
```

Fluid

Support Multiple Instances on the Same Page

- Don't share global variables, encapsulate state
- Parameterize CSS class names so different instances can be styled differently
- Constrain your searches to a unique container

Fluid

Multiple Instance Support

```
function Tabs(parentContainerId, itemsSelector, cssClasses) {
     this.componentContainer = jQuery("#" + parentContainerId);
     this.items = jQuery(itemsSelector);
     this.currentlySelectedItem = items[0];
     this.classNames = cssClasses;
};

var portalTabs = new Tabs("portalTabs", "li", {selected:
"portalTabs-selected"});

var portletTabs = new Tabs("myPortlet-tabs", "li", {selected:
"myPortlet-selected"});
```

Fluid

Don't Wildly Scan by Class

```
jQuery(".highlighted", jQuery(this.componentContainer));
```

not

```
jQuery(".highlighted");
```

Fluid

Fluid Components

We use all these techniques and a few more:

- Unobtrusiveness
- DOM Agnosticism
- Highly configurable

Fluid

Fluid Design Goals

- Components should be customizable
- Skinnable with style sheets
- Customize the structure and appearance by modifying HTML
- Inject custom handlers and logic
- Accessible from the start

Fluid

Unobtrusiveness

Separation of code and content

```
jQuery("#myItem").click(
  function () { alert "foo";}
);
```

not

```
<div id="myItem" onclick="function()
{ alert('foo') };">
```

Fluid

Don't Make Assumptions About the DOM

- Don't make assumptions that will prevent customization...
- Flexible containment hierarchy
- Types of elements
- Class names

Fluid

Allow Users to Specify Their Own Selectors

- Components bind to a set of "interesting things"
- Use jQuery selectors to form these bindings, but make them configurable
- Otherwise, if the HTML changes, your hard-coded selectors will break
- Allow users to pass in their own alternate

Fluid

```
function Tabs(elementSelectors) {
  this.selectors = jQuery.extend({},
  elementSelectors, defaults.selectors);
}
```

Fluid

Good Defaults

```
defaults: {
    selectors: {
        tabContainer: "#tabList",
        tabs: "#tabList li",
        tabPanels: "#panels div",
    },
    styleNames: {
        selected: "key-highlight",
        focussed: "highlight",
        disabled: "dim"
    },
    activate: function () {
    alert("Doesn't do anything in this prototype. " +
              "Add your own activate function here!");
    },
    showDebugPane: false
}
```

Fluid

Provide Useful Extension

```
function Tabs() { ... };
Tabs.prototype.shouldSelect(element) { ... };
Tabs.prototype.willSelect(element) { ... };
Tabs.prototype.willDisplayPanel(forTab) { ... };
var myTabs = new Tabs();
myTabs.shouldSelect = function (tab) {
    return (!tab.is(".disabled"));
}
myTabs.willDisplayPanel = function (activeTab) {
    getContentFromServerForTab(activeTab);
}
```

Fluid

JavaScript Accessibility

Fluid

What is Accessibility?

Fluid

A New Definition

- Accessibility is the ability of the system to accommodate the needs of the user
- Disability is the mismatch between the user and the interface offered by the system
- We all experience disability

Fluid

Fluid's a11y Vision

- Embrace diversity: one size doesn't fit all
- Recognize different needs under different circumstances
- Build systems that can bend and adapt to meet those user needs

Fluid

Accessible is Better

- The **curb cut effect**: everyone benefits
- Interoperable
- Easier to reuse and repurpose
- Better future-proofing
- More robust
- Works on more devices

Fluid

Familiar Techniques

- Label images with alt text
- Label form fields with `<label>` tags
- Skip links and access keys
- Use semantic markup

Fluid

A New Can of Worms

- The shift from document to application
- The familiar techniques aren't enough
- Most DHTML is completely inaccessible
- New techniques are still being figured out

Fluid

Assistive Technologies

- Present and control the user interface in alternative ways
- Screen readers
- Screen magnifiers
- On-screen keyboards
- Use built-in operating system APIs to understand the user interface

Fluid

The Problem

- Custom widgets often look but don't act like their counterparts on the desktop
- HTML provides only simple semantics
- Not enough information for ATs
- Dynamic updates require new design strategies to be accessible

Fluid

The Solution

- Describe user interfaces with ARIA
- Add consistent keyboard controls
- Provide flexible styling and presentation

Fluid

Keyboard Accessibility

Fluid

Keyboard Navigation

- Everything that works with the mouse should work with the keyboard
- ... but not always in the same way
- Support familiar conventions

Fluid

Keyboard Conventions

- **Tab** key focuses the control or widget
- **Arrow keys** select an item
- **Enter** or **Spacebar** activate an item

- Tab is handled by the browser. For the rest, you need to write code.

Fluid

Tabbing and Tabindex

- Each focusable item can be reached in sequence by pressing the Tab key
- Shift-Tab moves backwards
- The `tabindex` attribute allows you to customize the tab order
- `tabindex="1"` removes element from the tab order: useful for custom handlers

Fluid

Tabindex examples

```
<!-- Tab container should be focusable -->
<ul id="animalTabs" tabindex="0">
  <!-- Individual Tabs shouldn't be focusable -->
  <!-- We'll focus them with JavaScript instead -->
  <li id="tab1" tabindex="-1">Cats</li>
  <li id="tab2" tabindex="-1">Dogs</li>
  <li id="tab3" tabindex="-1">Alligators</li>
</ul>
```

Fluid

Setting Tabindex with jQuery

```
// Put the tab list in the tab order.
jQuery("#animalTabs").tabindex(0);

// Remove the individual tabs from the tab order.
// We'll focus them programmatically with the arrows.
jQuery("#animalTabs li").tabindex(-1);
```

Fluid

Handling Focus Events

```
// Make the tabList focusable with Tab.
var tabList = jQuery("#animalTabs").tabbable();

// Make the tabs selectable with the arrow keys.
var tabs = jQuery("li", tabList);
tabs.selectable(tabList, {
  willSelect: function(aTab) {
    aTab.addClass("highlight");
  }
});
```

Fluid

Adding Activation Handlers

```
// Make each tab activatable with Enter & Spacebar
tabs.activatable(function(aTab) {
  alert("You just selected: " + aTab.text());
});
```

Fluid

Supporting Assistive Technology

Fluid

Opaque Markup

```
// These are tabs. How would you know?
<ul>
   <li>Cats</li>
   <li>Dogs</li>
   <li>Gators</li>
</ul>
<div>
   <div>Cats meow.</div>
   <div>Dogs bark.</div>
   <div>Gators bite.</div>
</div>
```

Fluid

ARIA

- Accessible Rich Internet Applications
- W3C specification in the works
- Fills the semantic gaps in HTML
- Roles, states, and properties
- Live regions

Fluid

Roles

- Describe widgets not present in HTML 4
- `slider, menubar, tab, dialog`
- Applied using the `role` attribute

Fluid

States and Properties

- Added to elements within the DOM
- Properties describe characteristics:
  - `draggable, hasPopup, required`
- States describe what's happening:
  - `busy, disabled, selected, hidden`
- Applied using custom `aria-` attributes

Fluid

Using ARIA

```
// Now *these* are Tabs!
<ul id="animalTabs" role="tablist" tabindex="0">
  <!-- Individual Tabs shouldn't be focusable -->
  <!-- We'll focus them with JavaScript instead -->
   <li id="cats" role="tab" tabindex="-1">Cats</li>
   <li id="dogs" role="tab" tabindex="-1">Dogs</li>
   <li id="gators" role="tab" tabindex="-1">Gators</li>
</ul>
<div id="panels">
   <div role="tabpanel" labelledby="cats">Cats meow.</div>
   <div role="tabpanel" labelledby="dogs">Dogs bark.</div>
   <div role="tabpanel" labelledby="gators">Gators bite.</div>
</div>
```

Fluid

Setting ARIA with jQuery

```
var tabContainer = jQuery(“#animalTabs”);
tabContainer.ariaRole(“tablist”);

var tabs = jQuery(“li”, tabContainer);
tabs.each(function(idx, item) {
   jQuery(item).ariaRole(“tab”);
});
tabs.eq(0).ariaState(“selected”, “true”);
```

```
var panels = jQuery(“#panels > div”);
panels.each(function(idx, item) {
   jQuery(item).ariaRole(“tabpanel”);
};
```

Fluid

Live Regions

- Stock tickers, Ajax validation, etc.
- Need to identify areas that are updated
- Associate controls with live content
- Types of changes (add/remove/modify)
- Is it appropriate to interrupt the user?

Fluid

Be Polite

- `aria-live="polite"`: only announce if nothing else is going on.
- `aria-live="assertive"`: Announce ASAP, but don't interrupt.
- `aria-live="rude"`: Updates are extremely important. Interrupt immediately.

Fluid

Things to Think About

- What kind of UI are you building?
- Does it resemble something familiar?
- What states or modes does it have?
- Can you reuse an existing widget?

Fluid

Accessibility Resources

- http://wiki.fluidproject.org/display/fluid/DHTML+Developer+Checklist
- http://wiki.fluidproject.org/display/fluid/UX+Accessibility+Walkthrough+Protocols
- http://developer.mozilla.org/en/docs/Accessible_DHTML
- http://developer.mozilla.org/en/docs/Key-navigable_custom_DHTML_widgets
- http://developer.mozilla.org/en/docs/AJAX:WAI_ARIA_Live_Regions

Fluid

Where to go next?

Fluid

Resources

- Fluid DHTML Developers Checklist
- Fluid Javascript Resources
  - Links to our favorite JS Resources

Fluid

Tools

- Firefox
- Firebug
- JSLint in Eclipse or Aptana
- IE Debugging
  - Script Debugger in the free version of Visual Studio for the Web

Fluid

Q & A

Fluid