Writing Fearless Javascript

for portlets, widgets, and portals...

http://fluidproject.org

Fluid

Outline:
1. Introductions
2. Why Javascript?
3. Javascript 101
4. Learning jQuery
5. JavaScript in Sakai
6. Javascript for Portals and Mashups
8. Accessibility
9. sData
10. Putting it all together
11. Where to go next

Fearless Javascript

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

Eli Cochran
User Interaction Developer
ETS, University of California, Berkeley

Nicolaas Matthijs
Researcher
CARET, University of Cambridge

Fluid

In-class examples...

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


All the presentation example code is available from the Fluid svn repository at this address.

Please try out the examples, steal from them, etc. Most of the examples are designed so that you can copy and paste them into the console of a debugger like the wonderful Firebug for Firefox and then view the results.

The best way to learn any language is to play around with it.

Firebug

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

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

Fluid

Any language needs a good debugger. Our debugger of choice is Firebug which runs primarily inside Firefox. There is a Firebug Lite version that runs in Internet Explorer but it lacks the tight integration of Firebug in Firefox.

Microsoft has a pretty good script debugger available with the free version of Visual Studio for the Web. And if you want to play on the bleeding edge, the beta of Internet Explorer 8 has an excellent debugger.

Setting Up FireBug

1. In Firefox 2, go to hyperlink
   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

This is our recommended way to configure Firebug if you would like to follow along with us during the presentation.

Web 2.0

aka. Web 3.0 alpha

Fluid

These technologies and the web applications built on top of them are oft-labeled as **Web 2.0.** The term is applied to everything from technology, to user experience, to marketing and business plans, so it's pretty much meaningless by now. But it's the term we've got, so **Web 2.0** it is.

Definitions

DHTML: **D**ynamic **HTML**

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

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

Fluid

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

And what we're really talking about when we build web sites and applications with these technologies are:
RIAs = Rich Internet Applications
= web sites with all the interactivity, dynamic content, ease of use, and accessibility of applications on the desktop

But why bother?

Why use Javascript at all?

Fluid

So... why Javascript? why DHTML? why AJAX? why bother at all?
What is wrong with our server-side rendered pages and our post some data and refresh the entire page web application model? After all this model served us well for years, and was the model that the world wide web was built on.

But did this model serve us well. It certainly didn't serve users well.

improved user experience

Fluid

improved

performance

Fluid

ease of development and deployment

Fluid

ubiquity =

expectation

Fluid

Flash?

Silverlight?

Most of you are familiar with Flash. Silverlight is a Microsoft competitor. Very new the game. So far what's been released is very nice but it's still really knew. I'm mostly going to talk Flash here, but the same rules apply...

Great technology

- powerful
- fast
- cross-platform (well, at least Flash has so far proven to be so)
- and for some things, such as complex animations, Flash is clearly better than DHTML, and Flash's video engine is second to none.

[Aside]
DHTML based animations are really coming along and shouldn't be dismissed. When viewing really rich animations or tricky-cool UI on the web, don't always assume that it's Flash. Some times it's just a really good Javascript programmer showing off.

But we're not talking about animation or video here, and for content and data, and rich interactivity, DHTML and AJAX easily hold their own over Flash and Silverlight.

Political

- Open vs. Closed

Practical

- Also Open vs. Closed

Fluid

And there are distinct advantages...

It comes down to open vs. closed systems

On the political level....
Javascript and DHTML are built to open-standards, and are completely transparent in their implementations. Flash is primarily a proprietary technology, Silverlight more so. To Adobe's credit they are actively releasing the Flash data formats as Open Source and engaging in open-standards and community building. But not completely, and at least for now the tools required to develop Flash are not open, and are not inexpensive.

On the practical level...
still a case of open and closed.

The Flash model, while filled with lots of interesting APIs to interact with your pages, back-end server technologies and with other Flash components, manages that communication opaquely. The only introspection that you get into a Flash object is what the Flash object want's you to see. Communication is explicit. Now some of you are saying, Yeah! Right On! As it should be! But this is not the web and it's not the strength of the web.

One of the things that has made the web so popular and easy for users and developers to work with is its open nature. The DOM, the structure of the page, with all it's content is available to users and functionality alike. Common data structures and methods make communication between elements easy, and has spawned whole applications which mash together data from multiple places into new and exciting services and content.

And while common namespaces and transparent data can and does cause headaches. If we really believe in open standards and an open web, we can't lock our content and our code up in private structures and tools.

DHTML/AJAX

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

... with a little effort on our part ...

Fluid

HTML and DOM are the currency of the web, components built using DOM can talk to each other with and through DOM.

The HTML document object model becomes the API that allows us to build along side and on top of each other. And this is a really good thing.

[aside] accessiblity is not a given with DHTML and Javascript, in fact, as you will see, it's hard even in the DHTML/JS domain. But there are proven techniques that developers have control over. Not so much with Flash and Silverlight.

Javascript 101

which brings us to

JavaScript is Different

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

Fluid

Super-Quick History

- Netscape rushed JS to market, bugs and all.
- Microsoft reverse-engineered it.
- Standards committee enforced the bugs.

Fluid

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

Part 1: The Basics

- Variables
- `null` vs. `undefined`
- Type coercion
- Objects and Arrays

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.
- This is 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
- “especially undefined:” error when the language encounters names it has never heard of before

Null vs. Undefined

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

and null as "nothing here"

Null vs. Undefined

```
var foo;
foo === undefined

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

Truthy and Falsey

- JavaScript does a lot of automatic type coercion
- Shades of true and false
- Helpful when evaluating arguments
- Use with care

Fluid

Falsey Values

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

- Everything else is truthy. Careful...

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

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
Number(12) // Explicit is better
```

Equal vs. Equivalent

Comparisons are coercive:

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

0 == false // true
```

Non-coercive comparison:

```
0 === false // false

1 !== "1" // true

1 === Number("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

```
// Very succinct, but be careful
// if your argument could be 0.
if (apples) {
  alert("Apples!");
  apples.cut();
}
```

Fluid

Objects

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

Objects Are Modifiable

```
var basketOfFruit = {};

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

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

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.

Part 2: Functions & Scope

- Functions are first class
- Determining types
- 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 puree = function (aFruit) {
    return puree(aFruit);
};
function squeeze(aFruit) {
    return squeeze(aFruit);
}
function popsicle(juiceMakerFn, fruit) {
    var juice = juiceMakerFn(fruit);
    freeze(juice);
}
```

Fluid

Notice the two different styles of defining a function.
I prefer the explicit assignment style: reminds us that functions are just data, and avoids hoisting issues.

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
- Functions are our basic building block

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 Inaccurate

```
// 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

Quack, quack

- The best way to check for types: **don't**.
- Check for behaviour:

```
function countChickens (eggs) {
    if (eggs.length &&
        typeof eggs.length === "number") {
        // Count your chickens.
    }
}
```

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];
```

this

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");
```

Object Instances

- The `new` keyword instantiates a new object
- `this` pointer is magically assigned to the new instance

Constructors are Just Functions

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

Fluid

Context and `this`

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

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";
```

Function Scope this

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

```
bake("cherry");
pie === "cherry"
window.pie === "cherry"
```

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

Closures

- Functions can be defined inside other functions
- 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); // result is an “add 3” Function
add(3); // Result is 6
add(5); // Result is 8
```

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(“Today's pie is: ");
$(element).click(clickHandler);

// Shows an alert: "Today's pie is: Banana creme pie"
$(element).click()
```

that

Coping With Bugs

- `this` can be confusing and unstable
- Constructor functions can accidentally clobber the global namespace
- Prototypal inheritance can easily cause existing code to break
- Can we simplify things?

Fluid

Plain Old Functions & Objects

```
// Just use plain old functions and objects.
function orange () {
    // Stable pointer to the current instance.
    var that = {};
    // Anything private stays inside here.

    // For public methods, just add properties.
    that.squeeze = function () {...}

    return that;
}
```

Fluid

Javascript Toolkits

Fluid

thousands of toolkits?

- single problem solutions
- widgets

there are many other kinds of toolkits
- single problem solutions such as a set of scripts which nail the AJAX or event problem
- or widget tool kits like the impressive EXTjs toolkit which are filled with bits of pre-built functionality for you to drop into your applications

these have their place but are not as indispensable as foundational toolkit

foundational toolkit

enabling and enhancing

Fluid

what's the problem?

As was stated in the previous section...

Javascript is tricky?

No

Javascript is easy

What is hard?

What is hard?

- browser inconsistencies and bugs
- complex data and user interfaces in web applications
- subtle and varied high-quality user interactions
- the call and response of asynchronous client-server interaction

Fluid

Frameworks can help!

- browser inconsistencies and bugs
- complex data and user interfaces in web applications
- subtle and varied high-quality user interactions
- the call and response of asynchronous client-server interaction

Fluid

Frameworks can help!

- Browser Abstraction
- complex data and user interfaces in web applications
- subtle and varied high-quality user interactions
- the call and response of asynchronous client-server interaction

Fluid

Frameworks can help!

- Browser Abstraction
- DOM traversal, selection, and manipulation
- subtle and varied high-quality user interactions
- the call and response of asynchronous client-server interaction

Fluid

Frameworks can help!

- Browser Abstraction
- DOM traversal, selection, and manipulation
- easy and dynamic event binding
- the call and response of asynchronous client-server interaction

Fluid

Frameworks can help!

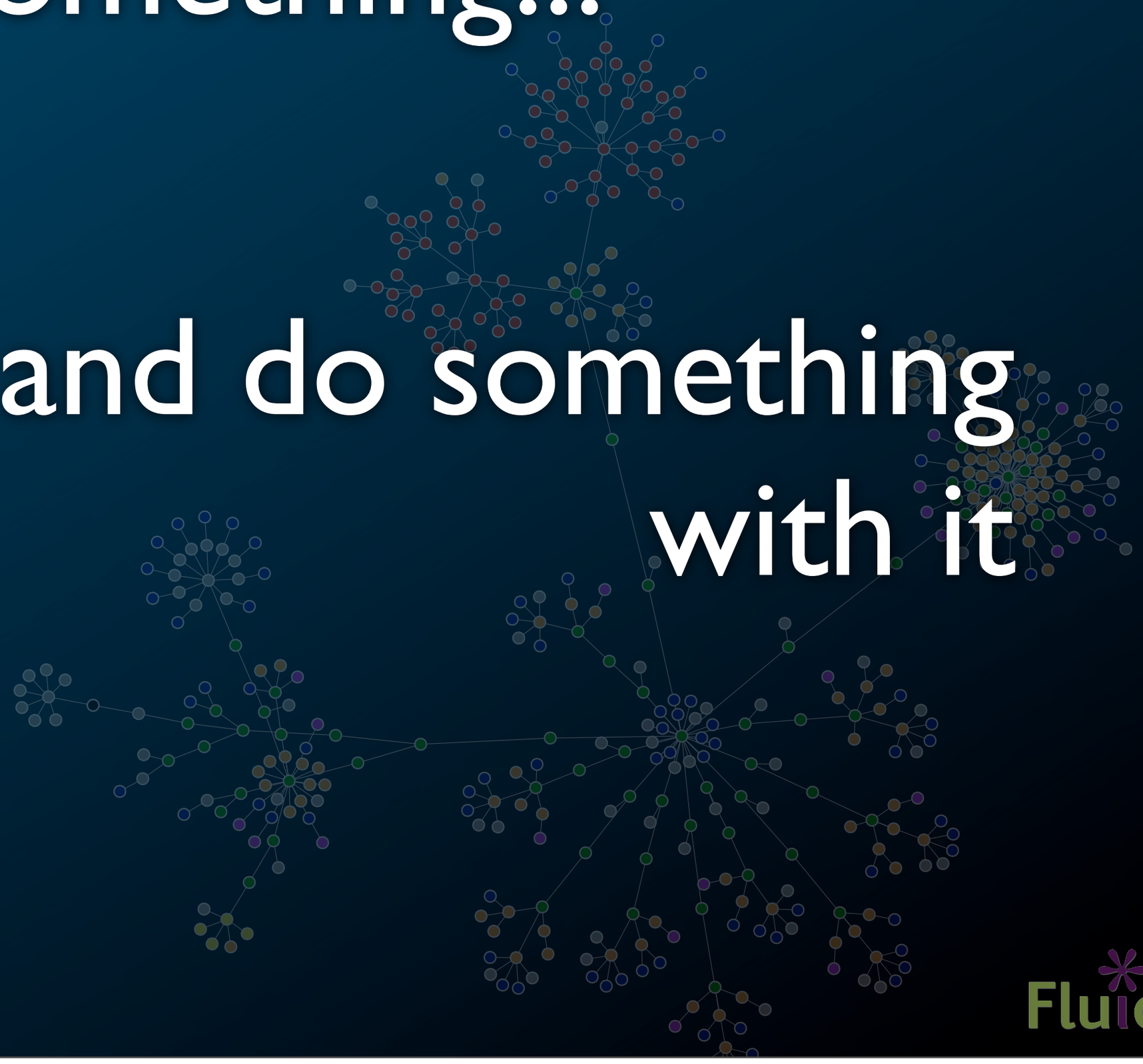
- Browser Abstraction
- DOM traversal, selection, and manipulation
- easy and dynamic event binding
- quick, responsive, bullet-proof AJAX functionality

Fluid

Find something...

Find something...

and do something with it



Fluid

Choosing a toolkit

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

When we were selecting a toolkit for the Fluid project we used the following criteria...
- Cross-browser support
- Easy debugging
- Event abstraction
- A solid DOM manipulation library
we've covered these or they are pretty much self-explanatory

Choosing a toolkit with a strong community and a strong installed base is important for the same reasons that one doesn't want to roll your own toolkit.
Using a toolkit from a large community means that you are leveraging someone else's pain. The learning and input of the community makes for a stronger toolkit, one with better performance, fewer bugs, and broader functionality.

And, of course, accessibility is a big part of what Fluid is about. And, I admit, it was here that we had to make some compromises. With the exception of the Dojo toolkit, very few of the major toolkits have done much in this area, including jQuery. But that is changing.

Leading Toolkits

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

Fluid

To me, these are the major players. But it's an ever changing list. Every online pundit has their own list. You get mine.

And although they offer different features, each covers the basics that I outlined before, very well, even if one might offer better widgets or another might try to nail some bit of functionality that another deems on the margins.

The primary difference between them is one of style, programming style.

I believe that each of these toolkits reflect a philosophy or linguistic bias of there original authors.
- prototype feels a lot like Ruby, and is, in fact, baked into Rails
- Dojo attempts to bring to Javascript, the classical inheritance of Java
- and using GWT you actually write Java which then is compiled into Javascript ... imagine that?

jQuery

Write Less, Do More

Fluid

So what's the philosophy and bias of jQuery. jQuery focus is on the DOM and on CSS. Which are after all the core of what front-end web development are all about. jQuery makes it as easy as possible to find things in the DOM and to do something with them using CSS as the primary mechanism. CSS selectors make a ton of sense to someone already creating selectors to define the visual language of semantic markup. Which makes jQuery easy to learn and to use.

jQuery UI 1.5

- a relatively new set of UI widgets built by the jQuery development team.
  - Dialog
  - Drag and Drop
  - ...
  - Animation

jQuery

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

Fluid

finding something without a framework

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

Fluid

doing something without a framework

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

Fluid

finding something with jQuery

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

Fluid

Here's the same code using jQuery. And here is my nice little short hand for retrieving an array of all the list items inside an element with a certain id.

And this notation should look very familiar as it is exactly the kind of notation that we use for CSS to identify specific objects in the DOM for styling.

Short, sweet.

finding something with jQuery

```
jQuery("<selector>")
```

selectors =

tags: `jQuery("tr")`

ids: `jQuery("#myId")`

classes: `jQuery(".myClass")`

pseudo tags: `jQuery("div:first")`

finding something with jQuery

```
jQuery("<selector>")
```

more selectors = combining selectors

element by class: `jQuery("li.selected");`

relationships: `jQuery("tbody tr:even");`

children: `jQuery("div > p");`

siblings: `jQuery("div ~ p");`

etc, etc, etc...

Fluid

$('#myList li.highlighted');

$('div > p');

doing something with jQuery

```
jQuery("li");
```


zero-based foo

doing something with jQuery

```
jQuery("li:even");
```


zero-based foo

doing something with jQuery

```
jQuery("li:even").addClass("odd");
```


zero-based foo

doing something with jQuery

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


zero-based foo

jQuery === $

doing something with jQuery

```
$(".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");
```

doing something with jQuery

```
$("div.container").clone().appendTo("body");
```

chaining...

```
$("div#mytemplate").addClass('menu')
    .clone()
  .appendTo("body")
    .click(dosomething());
```

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

```
$(".button").click();
```

online documentation

jQuery.com

JavaScript Accessibility

What is Accessibility?

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 provided
- We all experience disability
- Accessible software = better software

Fluid

One size doesn't fit all
Different needs under different circumstances
Systems should bend and adapt to meet user needs

DHTML: A New Can of Worms

- The shift from documents to applications
- Familiar a11y techniques aren't enough
- Most DHTML is completely inaccessible
- New techniques are still being figured out

Fluid

Familiar techniques aren't enough:

- alt text, label tags, access keys, skip links
- semantic markup

Assistive Technologies

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

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

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

Navigating with the Arrow Keys

```
// 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());
});
```

Supporting Assistive Technology

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

Roles

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

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

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?

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

MyCamTools === JavaScript

Cambridge (Since January 2008)

Fluid

Goals MyCamTools

- Make (our local) Sakai more user friendly
- Make Sakai technologically more up-to-date
  + Introduce Web 2.0
    => JavaScript
- Lower Development Bar

Goals MyCamTools

- Better separation between front end and back end
  => Well balanced team
- Speed up development process
- => Make Sakai more dynamic environment!

Result

http://mycamtools.caret.cam.ac.uk

Soon to be in production

https://camtools.caret.cam.ac.uk

Fluid

MyCamTools Backend
=== SData

```
Sdata (Sakai Data) = cfr GoogleData
```

SData

Rest Based Service Architecture

=> Pure data services

=> No more front end in Java

GET --> Returns Sakai data in JSON format

Services

Sdata Project

- List of useful services (Content, MOTD, MGS, ME, ...)
- /sdata/me, ...
- http://bugs.sakaiproject.org/confluence/display/MYSAK/MyCamTools+Home

Entity Broker for your own project

- /direct
- http://bugs.sakaiproject.org/confluence/display/SAKDEV/Entity+Provider+and+Broker

=> Feeds out **JSON** data

Fluid

In between: JSON

http://json.org

Very easy data structure (associative array)

Easy to work with in JavaScript (eval)

Less processing time

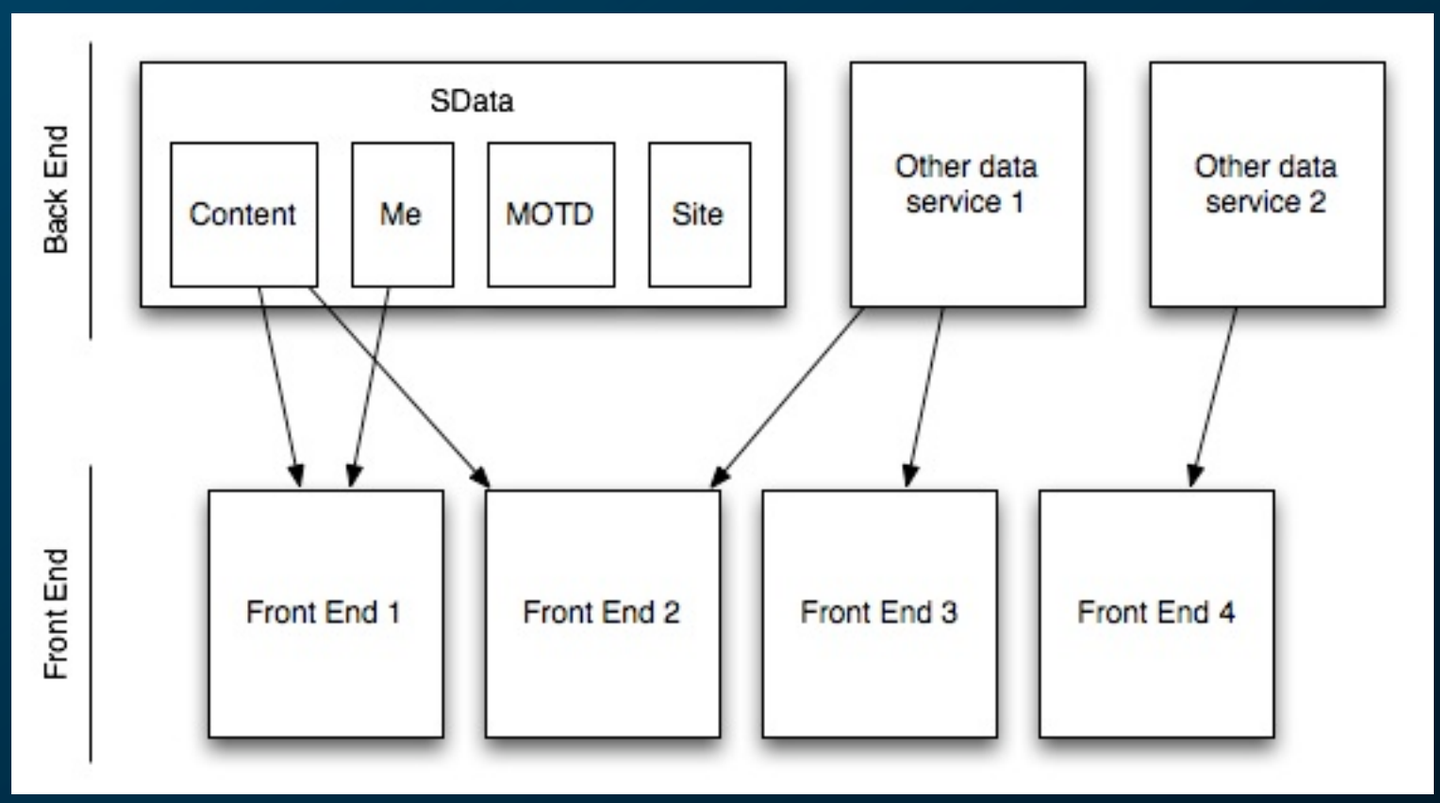
=== Successor of XML & Popularity ++

e.g.: MOTD


```
{"items":[{"motdBody":"<p>MOTD<\/
p>","motdUrl":"http://localhost:8080/access/
announcement/msg/!site/motd/29557475-19e1-4b29-
a391-092fc0d5c443"}]}
```


Fluid

Backend



- Able to use same data service in different front end applications (reusable)

  => Better opportunities for Mashup, Tool interoperability, ...

- Front end and back end are disconnected

Front End

=== Widgets

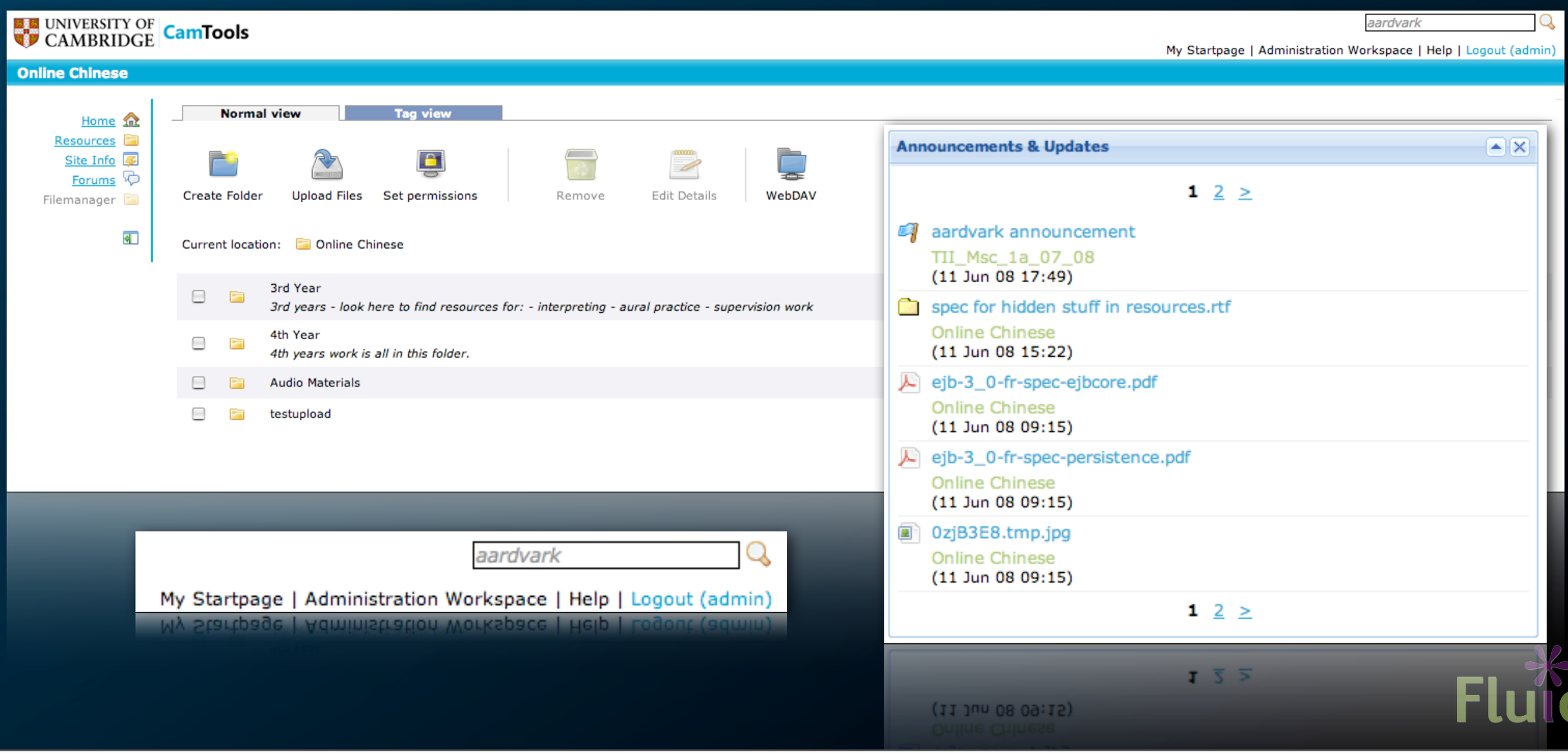
HTML/CSS/JavaScript file(s) on disk

- No more maven rebuilds
- Just hit refresh
- Lower development bar

GWT *(Experimental)*

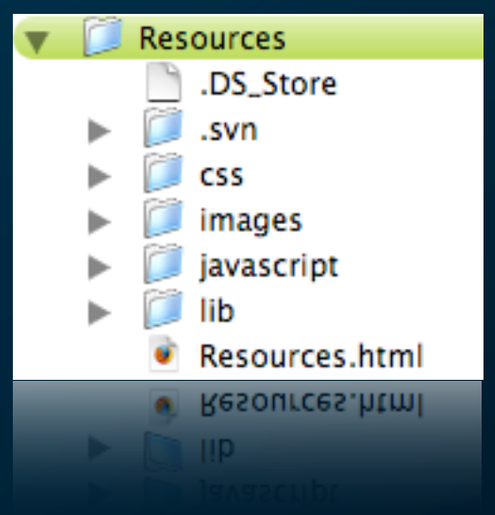
Widgets

Can be brought up anywhere
(Portal, Site template, In different widget, ...)



Widget Structure

- HTML
- CSS
- JavaScript
  - Widget specific
    - Namespacing
    - In-line: no iframes !!! (Can be)
  - Libraries == Widget Development Kit
- Stand-alone “website” uses Sakai data services



Widget Structure Example

MOTD Widget:

```
<link href="/widgets/MessageOfTheDay/css/
MessageOfTheDay.css" rel="stylesheet" type="text/
css"></link>

<script src="lib/sdata.js" language="JavaScript"
type="text/javascript"></script>

<div id="motd_container"></div>

<script src="/widgets/MessageOfTheDay/javascript/
MessageOfTheDay.js" language="JavaScript"
type="text/javascript"></script>
```

Fluid

JavaScript Libraries

jQuery + jQuery UI: JavaScript Toolkit + Toolkit for Dragging, Dropping, Reordering, ...

SWFUpload : Multi file upload

http://swfupload.org

TrimPath : Client Side Templating, takes template and JSON object and returns HTML

http://code.google.com/p/trimpath/wiki/JavaScriptTemplates

Really Simple History: Makes the back button work in an Ajax application !!!

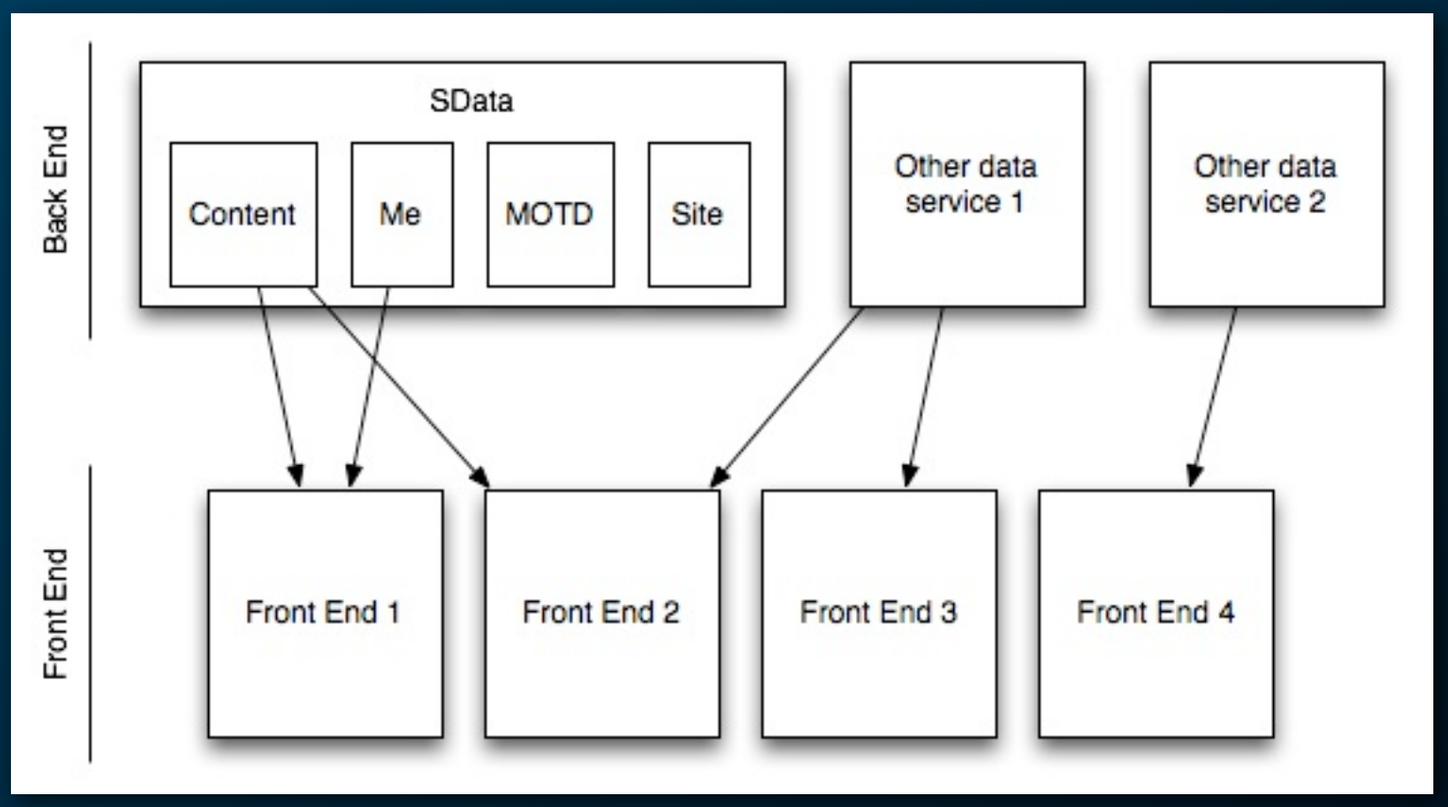
http://code.google.com/p/reallysimplehistory/

Fluid

Sdata.js

- Ajax
- Persistent Data Storage
- WidgetLoader
- Logging
- Templating

Front End



- Feed Reusability
- Clear separation

Examples

- Portal

- File manager

Modifying a widget

Just HTML/CSS/JavaScript on disk

90 % of the time: just change HTML or CSS (low bar)

Fluid

Playing Nice With Others

Portals: Sharing JavaScript

- Lots of different JavaScript code running
- High chance of collisions
- Can't expect control of the document
- Namespacing is essential

Everything is a portal now:

- third party libraries, widgets, gadgets, applications all share the same space

Writing Collision-Free 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 || {}
```

Use Closures for Privacy

```
(function() {
    // Private stuff.
    function myPrivateFunction () {
    }

    // Add public stuff to your namespace.
    fluid.tabs = function () {
      // Public creator function.
    };
})();
```

Keep Common Aliases Private

Pass important dependencies in as an argument to the closure:

```
jQuery.noConflict(); // Tell jQuery to surrender $

(function ($) {
    // $ is now only visible in our private space.
    //$ === jQuery;
}) (jQuery);

//$ === undefined;
```

More Than One On a Page

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

Fluid

Don't Wildly Scan by Class

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

not

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

Fluid

Instance-Based Styling

```
function tabs(parentContainerId, tabSelector, styles) {
      ...
      // Mix together defaults with per-instance style overrides.
      that.classNames = fluid.mixin(defaults.styles, styles);
      return that;
};
```

```
// Styling two different Tab components on the same page.
var portalTabs = tabs("portalTabs", "li", {
   selected: "portalTabs-selected"
});

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

Fluid Components

We use all these techniques and a few more:

- Unobtrusiveness
- DOM Agnosticism
- Highly configurable

Fluid Design Goals

- Components should be customizable
- Skinnable with style sheets
- Easy to change the HTML
- Inject custom handlers and logic
- Accessible from the start
- Plays nice with other people's code

Fluid

Unobtrusiveness

Separation of code and content

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

NOT

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

DOM Agnostic

- Don't make assumptions that will prevent customization...
  - Containment hierarchy
  - Types of elements
  - Class names
- jQuery selectors are very helpful here!

Fluid

Provide Useful Callbacks

```
var myTabs = tabs(myContainerId);

// Components will call you at interesting moments.
myTabs.delegate = {
  shouldSelect: function (tab) {
    return (!tab.is(".disabled"));
  },
  willDisplayPanel: function (activeTab) {
    getContentFromServerForTab(activeTab);
  }
};
```

Fluid

Putting It All Together

```
var fluid = fluid || {}; // Unique namespace
(function () { // Closure for privacy.
    // Private functions.
    var showPanelForTab = function (tabContainer, tabs, tab, panels) { ... };

    // Good defaults.
    fluid.defaults("tabs", { ... });

    // Creator function.
    fluid.tabs = function(componentContainerId, selectors, options) {
      var that = {}; // Stable pointer to the current instance.
      // Public methods.
      that.select = function(tabToSelect) { ... };

      return that;
    };
})();
```

Hands on example

- Create a widget from scratch
- Includes :
  - Using Sakai datafeeds
  - Using Fluid component: Inline Edit
  - Implementing accessibility

Installing MyCamTools

1. Have a running instance of Sakai (f.e. a programmers cafe build)

   http://bugs.sakaiproject.org/confluence/display/BOOT/Development+Environment+Setup+Walkthrough

Installing MyCamTools

2. Install JCR

   Svn co https://source.sakaiproject.org/svn/jcr/trunk/

3. Add to Sakai properties:

   jcr.experimental = true

Installing MyCamTools

4. Install Search

Svn co https://source.sakaiproject.org/svn/search/trunk/

5. Add to Sakai properties:

search.enable = true

Installing MyCamTools

6. Install SData

Svn co https://source.caret.cam.ac.uk/camtools/trunk/camtools/sdata

Installing MyCamTools

7. Checkout User Interface into /tomcat/webapps/ROOT

   Svn co https://source.caret.cam.ac.uk/camtools/trunk/camtools/flat/src/main/webapp/ ROOT

8. Checkout Widget Library

   Svn co https://source.caret.cam.ac.uk/camtools/trunk/camtools/widgets/src/main/webapp/

Fluid

Installing MyCamTools

9. Go to localhost:8080

10. Make changes and hit refresh without rebuilding

Where to go next?

Resources

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

http://fluidproject.org/

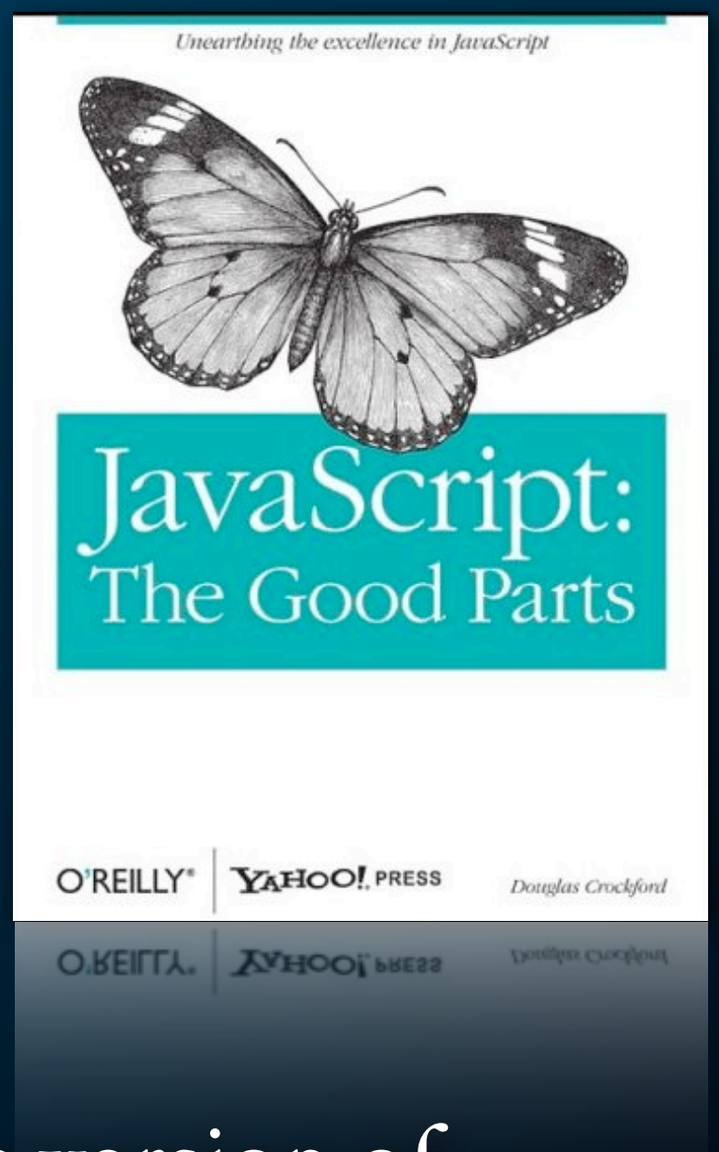
Fluid

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