The Fluid Framework
Fluid technology goals

• Build an architecture to support user interfaces that can be shared and adapted.

• Develop tools that support the inclusive design process.

• Give users tools to personalize their environment.
A software framework, in computer programming, is an abstraction in which common code providing generic functionality can be selectively overridden or specialized by user code providing specific functionality.

Frameworks are similar to software libraries in that they are reusable abstractions of code wrapped in a well-defined API. Unlike libraries, however, the overall program's flow of control is not dictated by the caller, but by the framework. This inversion of control is the distinguishing feature of software frameworks.

Huh?

Stuff that helps us write great user interfaces faster by not having to solve the same problem over and over again.
Imagining the framework
It’s like music...

Component

Configuration

Integration

Framework
What is our framework?

- Code tools to help us:
  - build flexible designs
  - avoid errors
  - shape our code
  - write less code
  - build richly accessible code
- Built with open web technologies
The framework gives us...

- A **life cycle** for components
- A way to **configure** & wire up components
- Separation of **presentation from logic**
- A way to **change markup** and appearance
You can’t bottle design

• Context is everything!
• Each new use case brings new design considerations
• We can’t get away with shipping one specific design and assume we’re done
Design for more design

- Our designs should invite new designs
- What are the range of choices and needs for an interaction?
- How can we support people in making the right choices for their particular context?
- The technology needs to help us...
Components

Pattern + Configuration = Solution

Component

Context

Implementation
Component families

**Inline Edit**

**Simple Text Inline Edit**

- Lecture Sections
  - Astronomy IAP, LEC
  - Meet: Monday, Tuesday, Wednesday

- Discussion Sections
  - Astronomy TA, 10:30-12:30 AM

**Dropdown Inline Edit**

- Groups
  - Choose one
  - Open: 07:02

**Rich Text Inline Edit**

- BioE 24 - Aspects of Engineering
- Sibley Auditorium, Wednesdays 4 - 5 PM
- Some additional details, including contact information and course objectives.
Value of the framework

- Positions us to write components faster
- Allows us to rework our designs for each new integration
- The framework is a design enabler
- Enables new developers to join our ranks and build their own solutions
Building the framework

• We didn’t build a framework because they’re fun to write. They’re not.

• We built a bunch of components, suffered, and then built a framework that addressed real challenges.
The wider context

- There are a lot of different JavaScript programming tools out there.
- Why did we build another one?
Measuring up

Foundational toolkits vs. application frameworks
Foundational toolkits

- Totally presentation focused
- DOM manipulation
- Event binding
- Ajax
- eg. jQuery
Application frameworks

- **Model notifications** “something changed here”
- **Views** to help keep your presentational code clean
- **Data binding** to sync the display with your model

- eg. Sproutcore; Dojo + Dojox
Where does Infusion fit?

- We recognize that we're not the only one in the browser: *we play nice with other toolkits.*
- We don’t want to force adopters down a one-way technology street
Where does Fluid fit?

Infusion is an application framework designed to provide unprecedented flexibility while preserving interoperability.
In summary

- Design for more design
- Offer a technology that isn’t all-or-nothing
- Grow it based on real experience
getting geekier
Tasty framework sandwich

Fluid Components

UI Options
Accessibility Plugins
Views
Renderer

Skinning System
jQuery
What’s the framework?

- jQuery
- keyboard-ally plugin
- that-ism
- Components & declarative options
- DOM Binder
- Views
- Events
- Subcomponents
Component goals

- Thoroughly accessible
- Easy to share and reuse
- Can be personalized
- Plays nice with other technologies
Addressing real pain

- Behaviour and presentation logic tended to glob together as a single component
- Too easy to write clever logic that prevented any changes to the markup
- Handling configuration required lots of repetitive code
# Goals and features

<table>
<thead>
<tr>
<th>Change markup without breaking code</th>
<th>DOM Binder</th>
</tr>
</thead>
<tbody>
<tr>
<td>Customize component</td>
<td>Declarative options</td>
</tr>
<tr>
<td>Inject custom behaviour into components</td>
<td>Events, subcomponents</td>
</tr>
<tr>
<td>Decouple presentation from model logic</td>
<td>Views</td>
</tr>
<tr>
<td>Easily testable</td>
<td>Events, views, subcomponents</td>
</tr>
<tr>
<td>Make accessibility easier</td>
<td>jquery.keyboard-a11y, ui.core</td>
</tr>
<tr>
<td>Stable and secure JavaScript objects</td>
<td>that-ism</td>
</tr>
</tbody>
</table>
Model View Controller

- Model is the application data and associated behaviour
- View presents the model and drives the interaction logic
- Controller is glue
Fluid MVC

- Controllers are the least interesting part of MVC
- Models are transparent
- Views can be easily swapped or altered
jquery.keyboard-a11y
Features

- tabindex normalization across browsers
- arrow key navigation
- activating elements
- migrating to jquery ui-core.js
- makes keyboard accessibility super easy
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.
Tabindex examples

Tabs.html

<!-- 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>
function keyNav(container, tabs, selectHandlers, activationHandlers) {
    // Make the tablist accessible with the Tab key.
    tabContainer.tabbable();

    // Make each tab accessible with the left and right arrow keys.
    tabs.selectable(tabContainer, selectionHandlers, {
        direction: jQuery.a11y.orientation.HORIZONTAL
    });

    // Make each tab activatable with Spacebar and Enter.
    tabs.activatable(activationHandlers);
}
Further customization

$.fn.selectable.defaults = {
    direction: $.a11y.orientation.VERTICAL,
    selectablesTabIndex: -1,
    autoSelectFirstItem: true,
    rememberSelectionState: true,
    selectableSelector: ".selectable",
    selectableElements: null,
    onSelect: null,
    onUnselect: null,
    onLeaveContainer: null
};
JavaScript pitfalls

- Lack of namespacing and privacy
- Confusing variability of `this`
- Security and stability issues: `prototype`
- No ability to link against multiple versions
Namespacing, privacy, and versioning

Fluid.js

var fluid_0_6 = fluid_0_6 || {};  
var fluid = fluid || fluid_0_6;

(function ($, fluid) {
    // Code goes here.

})(jQuery, fluid_0_6);
that

- Define objects within a function
- Provides privacy and a bound context
- Types can’t be maliciously altered
- Open for extension, not modification
- Douglas Crockford’s pattern, not ours.
UIOptions.js

fluid_0_6 = fluid_0_6 || {};
(function ($, fluid) {

    fluid.uiOptions = function (container, options) {

        var that = fluid.initView("fluid.uiOptions", container, options);

        that.save = function () {
            that.events.onSave.fire(that.model);
            fluid.applySkin(that.model);
        };

        that.refreshView = function () {
            pushModelToView(that);
        };

        setupUIOptions(that);

        return that;
    }
})(jQuery, fluid_0_6);
Components
What’s a component?

- Central hub for:
  - Events
  - Configuration
  - Public API
- A composition of Views and model logic
Component contract

Uploader.js

/**
 * Instantiates a new Uploader component.
 * 
 * @param {Object} container the DOM element containing the Uploader markup
 * @param {Object} options configuration options for the component.
 */
fluid.uploader = function (container, options) { ... }
Declarative Configuration
Tweaking components

• Transparent configuration
• Declarative: ask, don’t instruct
• Mini IoC
What can be configured?

- Modes and optional features
- Selectors
- Styles
- Subcomponents
- Events
- Language bundles
fluid.defaults("fluid.reorderer", {
  instructionMessageId: "message-bundle:\",
  styles: {
    defaultStyle: "orderable-default",
    selected: "orderable-selected",
    dragging: "orderable-dragging",
    mouseDrag: "orderable-dragging",
    hover: "orderable-hover",
    dropMarker: "orderable-drop-marker",
    avatar: "orderable-avatar"
  },
  selectors: {
    dropWarning: ".drop-warning",
    movables: ".movables",
    grabHandle: "",
    stylisticOffset: ""
  },
  avatarCreator: defaultAvatarCreator,
  keysets: fluid.reorderer.defaultKeysets,
  layoutHandler: "fluid.listLayoutHandler",
  events: {
    onShowKeyboardDropWarning: null,
    onSelect: null,
    onBeginMove: "preventable",
    onMove: null,
    afterMove: null,
    onHover: null
  },
  mergePolicy: {
    keysets: "replace",
    "selectors.selectables": "selectors.movables",
    "selectors.dropTargets": "selectors.movables"
  }
});
DOM Binder
Decoupling code from markup

• The most common component pitfall is hard-baking assumptions about markup.

• Use named selectors to separate the component implementation from the markup.

• Let users specify alternative selectors.
We’ll take anything

- The DOM Binder supports:
  - jQuery selectors
  - Elements
  - Arrays of elements
  - jQuery objects
  - Functions
Declaring interesting things

Uploader.js

**selectors**:

```javascript
selectors: {
    fileQueue: ".fluid-uploader-queue",
    browseButton: ".fluid-uploader-browse",
    uploadButton: ".fluid-uploader-upload",
    resumeButton: ".fluid-uploader-resume",
    pauseButton: ".fluid-uploader-pause",
    totalFileProgressBar: ".fluid-scroller-table-foot",
    stateDisplay: "div:first"
}
```
that.events.onFileSuccess.addListener(function (file) {
    var row = rowForFile(that, file);
    that.locate("removeButton", row).unbind("click");
    that.locate("removeButton", row).tabindex(-1);
    changeRowState(row, that.options.styles.uploaded);
});
fastLocate()

Reorderer.js

function firstSelectable(that)
{
    var selectables = that.dom.fastLocate("selectables");
    if (selectables.length <= 0) {
        return null;
    }
    return selectables[0];
}
refresh()

Reorderer.js

thatReorderer.refresh = function () {
    thatReorderer.dom.refresh("movables");
    thatReorderer.dom.refresh("selectables");
    thatReorderer.dom.refresh("grabHandle",
        thatReorderer.dom.fastLocate("movables");
    thatReorderer.dom.refresh("stylisticOffset",
        thatReorderer.dom.fastLocate("movables");
    thatReorderer.dom.refresh("dropTargets");
    initItems();
    thatReorderer.selectableContext.selectables = thatReorderer.dom.fastLocate("selectables");
    thatReorderer.selectableContext.selectablesUpdated(thatReorderer.activeItem);
};
Managing the presentation

• Views are DOM-oriented objects
• They encapsulate the presentational behaviour of a component
• They show a view on model-sourced data
• They often represent only a portion of the overall component’s screen real estate
View Contract

- Views:
  - Are automatically DOM-bound
  - Have a container
  - May share with their parent component
  - May have options
  - May use events
  - Should implement `refreshView()`
Becoming a View

FileQueueView.js

    fluid.fileQueueView = function (container, events, parentContainer, uploadManager, options) {
        var that = fluid.initView("fluid.fileQueueView", container, options);
    }
refreshView()
About the events system

- Pure model-based events
- Designed for sending messages between JavaScript objects
- Not encumbered by the DOM or presentational concerns
- Analogous to jQuery events, but intentionally a bit different
Declaring Events

Reorderer.js

events: {
    onShowKeyboardDropWarning: null,
    onSelect: null,
    onBeginMove: "preventable",
    onMove: null,
    afterMove: null,
    onHover: null
}
Types of events

null  “hey everyone, something is happening”
preventable  “should I do this?”
unicast  “our little secret”
listening for events

section-info-inner.html

listeners: {
  afterFinishEdit: function (newValue, oldValue) {
    // Save the data to the server.
  },
  modelChanged: function (newValue, oldValue, that) {
    // Update state.
  }
}


Using events in code

FileQueueView.js

Firing events:

```javascript
var finishUploading = function (that) {
    that.events.afterUploadComplete.fire(that.queue.currentBatch.files);
    that.queue.clearCurrentBatch();
};
```

Listening for events programmatically:

```javascript
that.events.afterFileQueued.addListener(function (file) {
    that.queue.addFile(file);
});
```
Subcomponents

- Provides very loose coupling between parts
- Look up dependencies by name, and the framework will instantiate them for you
- Share portions of overall configuration
- Users can implement their own version or configure alternatives
Instantiating subcomponents

Uploader.js

```javascript
var setupUploader = function (that) {
    // Instantiate the upload manager and file queue view,
    // passing them smaller chunks of the overall options for the uploader.
    that.uploadManager = fluid.initSubcomponent(that,
        "uploadManager",
        [that.events, fluid.COMPONENT_OPTIONS]);

    that.fileQueueView = fluid.initSubcomponent(that,
        "fileQueueView",
        [that.locate("fileQueue"),
        that.events,
        that.container,
        that.uploadManager,
        fluid.COMPONENT_OPTIONS]);
```
Configuring a subcomponent

Uploader2.html

var myUploader = fluid.uploader(".fluid-uploader", {
    uploadManager: "fluid.demoUploadManager"
});
Overriding subcomponent options

```javascript
var myUploader = fluid.uploader("#simple_uploader", {
    fileQueueView: {
        type: "fluid.fileQueueView",
        options: {
            selectors: {
                fileRows: ".row",
                fileName: ".fileName",
                fileSize: ".fileSize",
                removeButton: ".removeFile"
            }
        }
    }
});
```
Where are we going?
UI Options

User Interface Options

Customize the template by setting preferences to suit your needs. Changes will be reflected in the Preview Window.

Color
- Color Palette 1
- Color Palette 2
- Color Palette 3
- Color Palette 4

Layout
- Default
- Simple

Graphics
- Default
- Simple

Table of Contents
- No
- Yes

Text
- Font: Default, Arial, Verdana, Courier, Times
- Size: -2, -1, 0, +1, +2, +3, +4, +5, +6
- Spacing: Default, Wide, Narrow

Links
- Highlight links (on hover)
- No
- Yes

Select these preferences and continue or Cancel.
Skinning system
Renderer