

# A Short Tour of Infusion

## Overview

Here's a short tour of Infusion, based on presentations we've given about it. This tour is in point form, and designed to give you a few of the salient details about each major feature of Infusion, as well as linking you to more information.

## What is Infusion?

## What Motivated Infusion?

- Community source projects struggle to build successful user interfaces
- Our communities have a problem sharing scopes and technologies
- This is both a technical or a social problem
- Fluid is about stepping back, and trying to assure common some compatible approaches, at a deeper level

## Goals of Infusion

- 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

### On this Page

- [Overview](#)
- [What is Infusion?](#)
- [What Motivated Infusion?](#)
- [Goals of Infusion](#)
- [You Can't Bottle Design](#)
- [Infusion the Product](#)
- [Infusion At The High Level](#)
- [Low-level Technical Goals](#)
- [Components in Action](#)
- [Components](#)
- [Component Families](#)
- [Infusion's Framework](#)
- [Value of the Framework](#)
- [Where does Infusion Fit?](#)
- [Goals and Features](#)
- [that-ism](#)
- [Components](#)
- [Declarative Configuration](#)
- [DOM Binder](#)
- [Views](#)
- [Events](#)
- [Subcomponents](#)
- [The Renderer](#)
- [The ChangeApplier](#)
- [In Summary](#)

## You Can't Bottle Design

- Context is everything!
- Our designs should invite new designs
- We can't get away with shipping one specific design and assume we're done
- How can we support people in making the right choices for their particular context?
- The technology needs to help us...

## Infusion the Product

- [World, Meet Infusion 1.0](#)
- [Infusion](#)
- Includes:

- Great, reusable components
- A framework to help you build your own UIs
- Usability and accessibility baked in from the start
- Unprecedented level of customizability
- [Framework Concepts](#)

## Infusion At The High Level

- It's functional
- Declarative: less code
- No black boxes: open for extension
- Markup is free

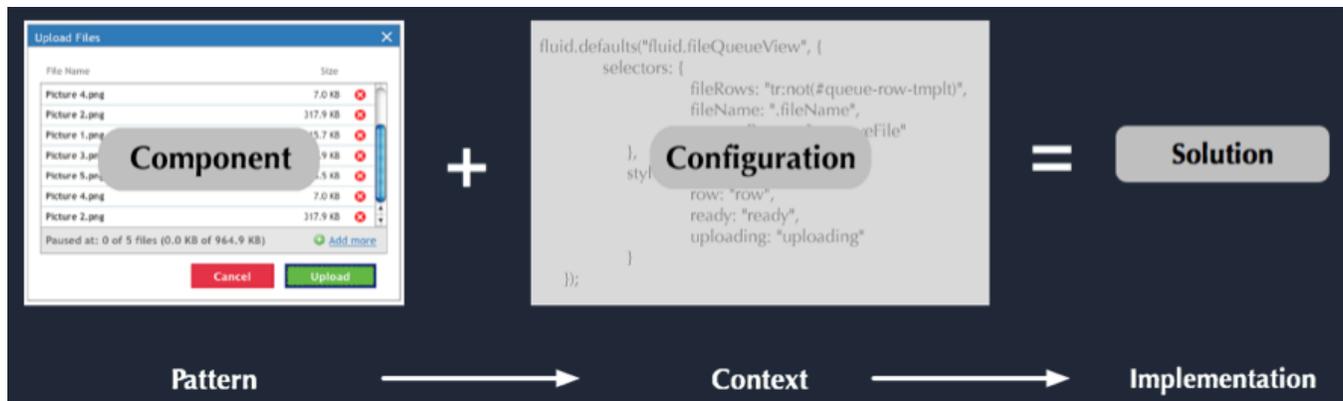
## Low-level Technical Goals

- Promote web standards (HTML, CSS, etc) as a commodity for building UIs
- DOM agnosticism
- Encapsulate better JavaScript practices
- Streamline use of productive techniques & minimise the effect of destructive techniques
- Promote the use of transparent models

## Components in Action

- [UI Options](#)
- [Uploader](#)

## Components



## Component Families

- Components provide many variations on an activity: [flavours of Inline Editing](#), [Reordering](#), etc.

## Infusion's Framework

- A life cycle for components
- A way to configure & wire up components
- Separation of presentation from logic
- A way to change markup and appearance

## Value of the Framework

- Helps you write UIs faster and more flexibly
- Allows you to rework designs for each new context
- The framework is a design enabler

## 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

## Goals and Features

Change markup without breaking code	<a href="#">DOM Binder</a>
Customize components	<a href="#">Declarative options</a>
Inject custom behaviour into components	<a href="#">Events, Subcomponents</a>
Decouple presentation from model logic	<a href="#">Views</a>
Easily testable	<a href="#">Events, Views, Subcomponents</a>
Make accessibility easier	<a href="#">jquery.keyboard-a11y</a> , <a href="#">ui.core</a>
Stable and secure JavaScript objects	<a href="#">that-ism</a>
Open, transparent, extensible architecture	<a href="#">ChangeApplier</a> , <a href="#">Events</a>

## that-ism

### 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

- Some of the most crucial issues for a portal deployment
- Here's how we solve it:

```
var fluid_0_6 = fluid_0_6 || {};  
var fluid = fluid || fluid_0_6;  
  
(function ($, fluid) {  
    // Code goes here.  
})(jQuery, fluid_0_6);
```

## that

- Define your 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  
[How to Define a Unit](#)

## Putting It All Together

```

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

- Simple creator function with two primary arguments:
  - a container for scoping DOM queries
  - options, a bundle of declarative configuration

```

/**
 * 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

### Customizing Components

- Transparent configuration
- Declarative: ask, don't instruct
- Mini IoC
  - [More IoC coming in Infusion 1.2](#)

### What Can Be Configured?

- Modes and optional features
- Selectors
- Styles
- Subcomponents
- Events
- Language bundles

## Example: Reorderer

```
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
  - raw Elements
  - Arrays of elements
  - jQuery objects
  - Functions

### Declaring interesting things

```
selectors: {
  fileQueue: ".fluid-uploader-queue",
  browseButton: ".fluid-uploader-browse",
  uploadButton: ".fluid-uploader-upload",
  resumeButton: ".fluid-uploader-resume",
  pauseButton: ".fluid-uploader-pause",
  totalFileProgressBar: ".fluid-scrolltable-foot",
  stateDisplay: "div:first"
}
```

## locate()

```
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);
});
```

## Views

### Managing the presentation layer

- Views are DOM-anchored objects
- They encapsulate the presentational behaviour of a component
- They show a view on model-sourced data

### View Contract

- Views...
  - Are automatically DOM-bound
  - Have a container
  - May be shared with their parent component
  - May have options
  - May use events
  - Should implement refreshView()

### Becoming A View

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

## Events

### About the events system

- Pure model-based events
- Designed for sending messages between Javascript objects
- Totally free argument signature
- Not encumbered by the DOM or presentational concerns
- Not for the same purpose as jQuery or DOM events

### Declaring events

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

## Types of event

- null "hey everyone, something is happening"
- preventable "should I do this?"
- unicast "our little secret"

## Listening For Events

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

## Subcomponents

### Subcomponents Express Dependencies Between Components

- Provides loose coupling between parts (IoC)
- Look up dependencies by name, and the framework will instantiate them for you
- Users can implement their own version, or swap out alternatives
- Unlike top-level views, not necessarily DOM-anchored
- A Subcomponent is not necessarily a View, although it might still be

### Configuring a Subcomponent

```
var myUploader = fluid.uploader(".fluid-uploader", {
  uploadManager: {
    type: "fluid.gearsUploadManager",
    options: {
      uploadUrl: "../uploads",
      fileTypes: ["img/jpg", "img/gif", "img/png"]
    }
  }
});
```

### Instantiating Subcomponents

```

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]);
}

```

## The meaning of Subcomponents

- Note that like event signatures, subcomponent instantiation signatures are completely free
- A subcomponent can mean what you want
- It's not a *thing*, it's a *relationship*

## The Renderer

### The Fluid Renderer

- The ultimate in markup-agnosticism
- Use markup as its own template
- No funky `#{}` or `<%` nonsense
- Specify all data-orientation and binding in a separate, pure JSON "component tree"

### A Fluid Template

```

<div id="testDataRoot">
  <div id="parseTest1">
    <table>
      <tr id="table-header"><th>Count</th><th>Name</th>
        <th class="column-header">1</th><th>Median ave</th></tr>
    </table>
  </div>
</div>

```

- A perfectly normal block of HTML

### A Component Tree

```

var tree = {
  "header:" : [1, 2, 3, 4, 5]
}

var templates = fluid.selfRender($("#table-header"), tree,
  { outpoints: [{selector: "th.column-header", id: "header:"}]
});

```

- A perfectly normal block of JSON
- A map of standard selectors

### No Black Boxes

- Virtually every popular library (YUI, Ext, jQuery UI, etc.) bakes its component HTML into Javascript
- The Fluid renderer liberates it
- Create free component "libraries" rather than simple widgets

## The ChangeApplier

## Familiar GUI Pain Point

- We build your UIs as several independent Views
- Problem: *how do we notify them when important changes happen in the data?*
- Solution: transparent models + change notification

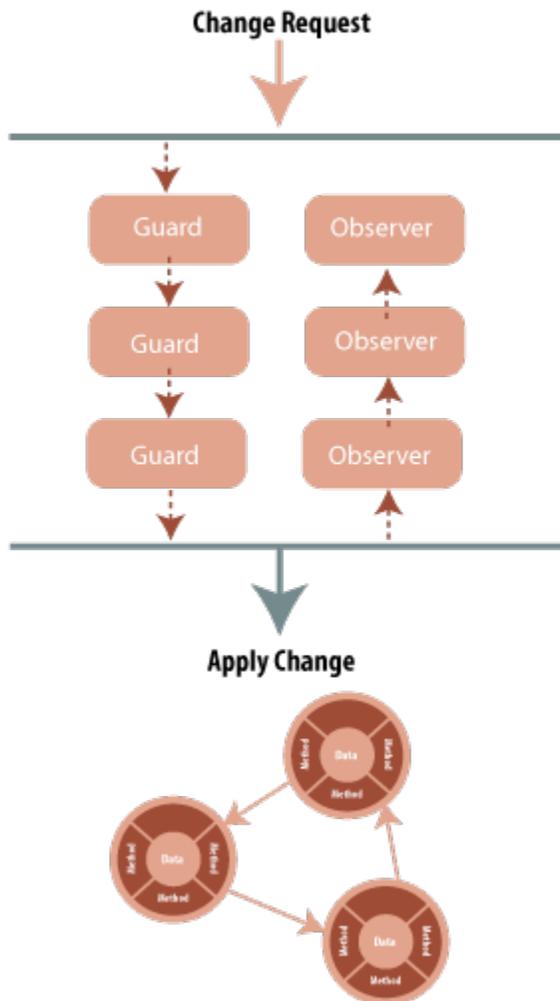
## ChangeApplier Features

- No inheritance hierarchy or constraints placed on the model itself.
- Guards and observers are expressed elegantly through simple event listeners
- Unifies the process of validation and observation (and soon transactions)

## ChangeApplier

- Users to request that changes be made to the model
- Provides hooks for event listeners ("guards") to accept or reject the requested change
- Notifies interested parties about changes that have been accepted in the model

## ChangeApplier Illustrated



In Summary

- Fluid stuff is ineffably cool
  - Infusion 0.8 is built into uP 3.1
  - Tools to make JS development in the portal easier:
    - Views
    - DOM Binder
    - Renderer
    - Accessibility
    - and more
  - Share your thoughts and help us make it better
- </div>