Component Model Interactions and API

This is a planning document laying out a proposal for how a component API might work.

External requirements for components

In order to enable harmonious and orderly interactions where multiple components are required to interact, some "external" requirements should be placed on them. The largest head for these interactions rests on interaction with what could be called a "model" (in the classic MVC terms).

As a principle of good organisation, each Fluid component should be associated with (either directly, or "on behalf" of another component which it is wrapping/interacting with) a collection of Javascript objects which constitute the "model", the "pure data" which it is operating on.

On this Page

Model requirements

Models should consist of pure data, (in terminology not fully established on the client, "POJOs" - Plain Old Javascript Objects) - that is, they should consist of Javascript Objects containing only other Objects and primitives, without the presence of any Functions. This is to aid the ability to version, manipulate, serialize, ship to the server &c these objects without the possibility of breakage and minimum requirements on understanding model semantics by the manipulating code.

Model access requirements

Each suitable Fluid component should make its model "easily available" - either as a return available from its top-level "that", or else stored within its DOM instance using jQuery.data under a well-known value. For example, the "model" structure stored using under `rsf-binding-root` by the Fluid Renderer - Background is an example of this kind of advertisement.

Model interaction requirements

Incoming

Each model-bearing component should expose a top-level operation, a function named `render` can be invoked without arguments, which will cause a complete "re-rendering" into the DOM of the component reflecting any updates that have occured to its model. It may in addition expose a more fine-grained event or update API suitable for allowing more localised and efficient updates to the DOM and its structure.

Outgoing

On receiving user-directed modifications to its model, a model-bearing component is responsible for maintaining a list of listeners who may subscribe to model-directed events representing the update.

How to represent/hold this listener list? Either we can use jQuery "custom events" using jQuery.bind and jQuery.trigger applied to the "container" DOM element, OR we can expect event-handling functionality to be mixed-in to every outgoing "that" element on construction