Renderer API

This page provides a reference for the functions available for using the Renderer.

Primary Renderer Functions

selfRender

```
var templates = fluid.selfRender(node, tree, options);
```

Return: the original parsed template.

A simple driver for single node self-templating. Treats the markup for a node as a template, parses it into a template structure, renders it using the supplied component tree and options, then replaces the markup in the node with the rendered markup, and finally performs any required data binding. The parsed template is returned for use with a further call to `fluid.reRender()`.

Parameters

node

Either a raw DOM node in the current document, or else a JQuery object representing one. The markup in this node will be used as the template for rendering, and the node will be replaced with the rendered output.

tree

A Javascript Object representing a component tree. See Renderer Component Trees for more information.

options

A JavaScript object used to configure the details of the rendering process. Unlike options for most Fluid components, this structure sometimes has a bi-directional aspect - it can be used for returning information about the rendering process back to the client, as well as for receiving it. Here is a list of the supported fields and types within the Renderer options:

On This Page

See Also

- Renderer
- Renderer Component Trees
- Renderer Templates
- Renderer Data Binding
- Renderer Decorators

Still need help?

Join the infusion-users mailing list and ask your questions there.

Field	Description	Type	In/Out
`model`	Perhaps the most important parameter, contains the "data model" to which value bindings expressed within the tree will be expressed.	free `Object`	Mostly In, but the supplied model may be written to later as a result of servicing user actions, especially if the parameter `autoBind` is supplied.
`applier`	a ChangeApplier object associated with the `model`	`ChangeApplier`	In
`autoBind`	If set, any user modification of fields with `valuebindings` s set will immediately be reflected in the current state of the supplied `model`	`boolean`	In
`document`	If set, will cause the rendered ids to be uniquified against the supplied document, rather than the current one	`document`	In
`debugMode`	If set, mismatches between template and component tree will be highlighted in an unreasonable garish pink colour	`boolean`	In
`idMap`	This map operates in conjunction with the `identify` decorator which may be attached to nodes in the component tree. Whilst rendering takes place, this map will fill up with a lookup from the supplied nickname to the finally rendered id	free `Object`	In/Out
`messageLocator`	Configures the lookup from (I18N) messages referenced in the component tree, to some source of a message bundle	`function (key, args)->message`	In
`messageSource`	Will construct a `messageLocator` from a raw bundle specification via a call to `fluid.resolveMessageSource`	`MessageSource` structure	In
`renderRaw`	Will XMLEncode the rendered markup before insertion into the document. Can be useful for debugging	`boolean`	In
`cutpoints`	This is properly a directive to the parser, rather than the renderer, but the options structure is shared. This contains a list of pairs of `id, selector` which will be used to impute an `rsf:id` structure onto a document, by means of matching the paired selector	Array of `Cutpoint` object	In

reRender

```
var templates = fluid.reRender(templates, node, tree, options);
```

Return: the original parsed template.

`fluid.reRender()` is identical to `fluid.selfRender()`, except that it uses the supplied `templates` (the first parameter) as the template instead of the `node` parameter. This is useful for re-rendering a template that has been previously rendered.

The options for `fluid.reRender()` are identical to those of `fluid.selfRender()`.

Sneak Peek: render

New in v1.3: The new `fluid.render` function will eventually be the primary means of using the renderer, and will automatically be included in components created using the Sneak Peek `fluid.initRendererComponent()` component initialization function. It is available in Infusion version 1.3 as a sneak peek, to allow users to play with it and provide us with feedback.

```
var something = fluid.render(source, target, tree, options);
```

Return: the original parsed template.

A simple driver for single node self-templating. Treats the markup for a node as a template, parses it into a template structure, renders it using the supplied component tree and options, then replaces the markup in the node with the rendered markup, and finally performs any required data binding. The parsed template is returned for use with a further call to `fluid.reRender()`.

Parameters

source

Either a structure {`node: node, armouring: armourstyle`} or a string holding a literal template

target

The node to receive the rendered markup

tree

As for `fluid.selfRender`: A Javascript Object representing a component tree. See Renderer Component Trees for more information.

options

As for `fluid.selfRender`: A JavaScript object used to configure the details of the rendering process.

Renderer Utility Functions

explode

"Explodes" a section of data model to a flat section of component tree bound against the data. This is suitable only in very straight-forward cases of data binding, where each field in a model can be associated with a piece of markup with a corresponding `rsf:id` representing a `UIBound` component.

```
var tree = fluid.explode(hash, basepath)
```

Parameter	Type	Description
`hash`	`Object`	A section of data model representing a flat hash of keys to values
`basepath`	`String`	An optional prefix to be placed on the EL path supplied to the generated `UIBound` components

For example, the following hash:

```
var hash = {sku: "thing1", description: "explodes"}
```

could be exploded to

```
    [{ID: "sku",
     value: "thing1",
     valuebinding: "1.sku"},
    {ID: "description",
     value: "explodes",
     valuebinding: "1.description"}
     ]
```

if supplied with `fluid.explode(hash, "1")`.

explodeSelectionToInputs

A common markup representation is of a `UISelect` component as a series of named radio-buttons or checkboxes, in successive rows of a table (in comparison to its natural HTML representation as a <select> tag). This framework function automates the work of generating the additional component tree material to back this representation:

```
var tree = fluid.explodeSelectionToInputs(optionlist, opts)
```

Parameter	Type	Description
optionlist	Array of String	The raw list of options which are available for choice - in general, the value held at optionlist in the parent UISelect control
opts	Object	An options structure to guide the explosion process - described below
Return	Component Tree	A section of component tree which should be added to the same container as the parent UISelect control.

The opts structure has the following keys, all of which must be supplied:

Member	Type	Description
rowID	String	The rsf:id to be given to each successive row component (UIContainer) - this id must contain a colon
inputID	String	The rsf:id to be given to the UIBound control rendered in each row - this will generally peer with a checkbox or radio button
labelID	String	The rsf:id to be given to the UIOutput label control rendered in each row - this will generall peer with an HTML <label> element
selectID	String	The rsf:id which has already been given to the parent UISelect control from which the rendered tree should be backed. This control is expected to be in the same container where the rendered subtree is placed.

Following rendering, the returned subtree needs to be added to the same container as the parent UISelect control by the user - for example, here is a compact call showing compound copying, explosion and addition, where selection_tree directly holds the UISelect control:

```
var tree = {children: [fluid.copy(selection_tree)].concat(
    fluid.explodeSelectionToInputs(selection_tree.optionlist, explode_options("radio")))};
```

The explode_options function sourcing the options could have been implemented as:

```
var explode_options = function(type) {
    return {
    selectID: "select",
    rowID: type + "-row:",
    inputID: type,
    labelID: "label"
    };
};
```

Other Important Renderer Functions

fetchResources

New in v1.3: In Infusion v1.3, the API of fluid.fetchResources has changed. Please see our new documentation for more information: fluid.fetchResources.

For v1.2 and earlier:

```
fluid.fetchResources(resourceSpecs, callback)
```

fluid.fetchResources automates the likely tedious work of performing multiple back-to-back AJAX requests for resource (be they markup, JSON or otherwise) and collecting the responses. It operates on a basic resourcesSpecs structure which is a hash of objects with a particular layout. The keys in resourceSpecs represent a unique key to represent each resource specification. On conclusion of all of the fetches, the function callback will be called with the now filled-in resourceSpecs structure as argument. The fields in a resourcesSpec entry are as follows:

Field	Direction	Purpose
href	In	A full resolvable URL holding a template or template fragment. This will be fetched via AJAX and used to populate the resourceText entry. NOTE that if this value is provided, the nodeId will be ignored.
nodeId	In	The id of a node within the current document holding a template, for which innerHTML is to be treated as a template. NOTE that if href is provided, this value will be ignored.
baseURL	Out	Computed from href, in order to rebase local URLs in the template.
resourceText	Out	The full text of the template fragment, as a string

fetchError	Out	Filled in if the AJAX request to fetch the resource failed. It will be populated with a structure `status:` response status, `textStatus:` textual version of status, and `errorThrown:` holding details of an exception.
queued	Out	A reserved flag by the system to track the progress of AJAX calls

The filled-in `resourceSpecs` structure is suitable for passing directly as the argument to the low-level function `fluid.parseTemplates`.

parseJavaProperties

```
var messageSource = fluid.parseJavaProperties(text)
```

The Java Properties File Format is a well-specified and common but rather baroque format for storing messages used to internationalise an application. A good tool chain exists for this structure although its main deficiency is that it is not directly Unicode-ready. An application without a legacy background might want to consider encoding the same information directly in a JSON structure. This call converts a textual representation of a `.properties` file in this format into the equivalent JSON structure, with a key for each property key, and a value consisting of the properly unescaped message format. The resultant structure is a suitable argument for the `fluid.messageLocator` function to create an active message locator.

Parameter	Type	Description
`text`	`String`	String form of the contents of a Java-formatted properties file
Return	`MessageSource`	JSON form of parsed file

fluid.parseTemplates

The driver for parsing a multiple template set:

```
var templates = fluid.parseTemplates(resourceSpec, templateList, opts)
```

Parses a set of textual representations of templates into a complex binary form, suitable for rapid rendering against a component tree. This is part of the low-level functionality which is packaged by high-level calls such as `fluid.selfRender`.

Parameter	Type	Description
`resourceSpec`	`ResourceSpecs` structure	A hash of `ResourceSpec` objects, as returned from `fetchResources`
`templateList`	`Array of String`	A list of the keys within `resourceSpec` which should be be parsed into a template structure for this call. The first entry in the list represents the "root template" at which rendering will start.
`opts`	`Object`	Options to guide the parsing process - see options
Return	`Templates` structure	A complex structure holding the parsed templates. In general this should not be accessed by users, but passed around to rendering functions as an immutable object.

renderTemplates

The driver for rendering a template set against a component tree:

```
var markup = fluid.renderTemplates(templates, tree, options, fossilsIn);
```

Parameter	Type	Description
`templates`	`Templates` structure	A parsed template structure as returned from `fluid.parseTemplates` or `fluid.selfRender`
`tree`	`Component Tree`	A Renderer Component Tree
`options`	`Object`	Options to guide the rendering process - see options
`fossilsIn`	`Object`	A hash of component "submitting name" to a `Fossil` structure consisting of `name, EL, oldvalue`. In general not of direct interest to users, but can be ferried around as a mutable structure indicating renderer UI state.
Return	`String`	A string holding the markup rendered by the renderer for this pass