fluid.model.transformWithRules

This functionality is Sneak Peek status. This means that the **APIs may change**. We welcome your feedback, ideas, and code, but please use caution if you use this new functionality.

This page is still a rough draft.

fluid.model.transformWithRules(model, rules)

Transforms a model based on a specified expansion rules objects.

```
fluid.model.transformWithRules(model, rules);
```

File name: `ModelTransformations.js`

Parameters

`model`	(Object) the model to transform
`rules`	(Object) a rules object containing instructions on how to transform the model (see below for more information)
`options`	(Object) a set of rules governing the transformations. At present this may contain the values: `isomorphic: true` indicating that the output model is to be governed by the same schema found in the input model, or `flatSchema` holding a flat schema object which consists of a hash of EL path specifications with wildcards, to the values "array"/"object" defining the schema to be used to construct missing trunk values.

Return Value

Object	the transformed model

See Also

- Model Transformation

Rules

Rules objects take the form of:

```
{
    "target.path": "value.el.path"
}
```

or

```
{
    "target.path": {
        transform: {
            type: "transform.function.path",
             ...
        }
    }
}
```

Transformation Expanders

The Framework currently provides the following expanders that can be used as part of a model transformation.

- `fluid.transforms.value`
- `fluid.transforms.arrayValue`
- `fluid.transforms.firstValue`
- `fluid.transforms.merge`

Each of these is described below.

`fluid.transforms.value`

This extracts and/or the value of a given path, and can be used for the following purposes:

To rename a property:

Start:

```
var source = {
    cat: "meow",
    ...
}
```

>>
Rule to rename "cat" to "feline":

```
var rules = {
    feline: {
        transform: {
            type: "fluid.transforms.value",
            // specify only the path to transform
            inputPath: "cat"
        }
    },
    ....
}
```

>>
Result:

```
{
    feline: "meow",
    ...
}
```

To set a default value:

Start:

```
var source = {
    gerbil: undefined,
    // or if "gerbil" doesn't exist
    ...
}
```

>>
Rule to set default value of "gerbil":

```
var rules = {
    gerbil: {
        transform: {
            type: "fluid.transforms.value",
            // specify path and default value
            inputPath "gerbil",
            value: "squeek"
        }
    },
    ....
}
```

>>
Result:

```
{
    gerbil: "squeek",
    ...
}
```

Note that if "gerbil" has a value initially, it will be unaffected.

To specify a literal value:

Start:

```
var source = {
    // no mention of kangaroos
    ...
}
```

>>
Rule to set a value for "kangaroo":

```
var rules = {
    kangaroo: {
        transform: {
            type: "fluid.transforms.value",
            // specify only a value
            value: "boingg"
        }
    },
    ....
}
```

>>
Result:

```
{
    kangaroo: "boingg",
    ...
}
```

To change the structure/nesting:

Start:

```
var source = {
    goat: false,
    sheep: [
        "baaa",
        "wooooool"
    ],
    ...
}
```

>>
Rule to change the nesting:

```
var rules = {
    "farm.goat": {
        transform: {
            type: "fluid.transforms.value",
            inputPath "goat"
        }
    },
    "farm.sheep": {
        transform: {
            type: "fluid.transforms.value",
            inputPath "sheep"
        }
    }
    ....
}
```

>>
Result:

```
{
    farm: {
        goat: false,
        sheep: [
            "baaa",
            "wooooool"
        ]
    },
    ...
}
```

`fluid.transforms.arrayValue`

The `arrayValue` expander will transform a value into an aray. If the value is already an array, the expander will have no effect.

For example:

Start:

```
var source = {
    cat: "meow",
    sheep: [
        "baaa",
        "wooooool"
    ],
    ...
}
```

>>
Rule to convert to arrays:

```
var rules = {
    cat: {
        transform: {
            type: "fluid.transforms.arrayValue",
            inputPath "cat"
        }
    },
    sheep: {
        transform: {
            type: "fluid.transforms.arrayValue",
            inputPath "sheep"
        }
    }
    ....
}
```

>>
Result:

```
{
    cat: ["meow"],
    sheep: [
        "baaa",
        "wooooool"
    ],
    ...
}
```

Note that the value of `cat` is now an array, but the value of `sheep` is unaffected.

`fluid.transforms.firstValue`

`fluid.transforms.merge`

Example

```

```

In this example, description here...