Example of using Template in Prototype

Posted At : February 19, 2009 2:49 PM | Posted By : Jon Hartmann

Related Categories: Templating, Software, Javascript

In response to a blog post by Ray Camden about jQuery and duplicating form fields, I put together an example of using the Template class in Prototype. As an added bonus, I designed this example to also handle form resets and to feature a controller based design. You can click here to view a demo of the example in action. Read more to get the break down of the code.

To start, we need a basic framework for the form containing the template markup:

```
<form>
    <input id="current-count" type="hidden" name="count" value="0" />
    <div id="form-content">
        <label>
            Name #{number}: <input type="text" name="name_#{number}" value="" /><br />
        </label>
    </div>
    <button id="add-field" type="button">Add Field</button><br />
    <button id="reset-button" type="reset">Reset</button><br />
    <button type="submit">Submit</button>
</form>
```

The area inside form-content will be the area to hold new new field information, and the text inside will become the template for the records.

```
var Controller = function () {
    var config = {
        IDs: {
            contentarea:    'form-content',
            addbutton:    'add-field',
            countfield:     'current-count',
            reset:        'reset-button'
        }
    };

    return {
        init: function () {
        }
    };
}();

document.observe('dom:loaded', Controller.init);
```

This is a self calling function to create the Controller object to run the page, a config variable that holds all of the required ID values from the form, and an empty init called when the DOM is loaded. Now we add some content to the init method:

```
var Controller = function () {
    var config = {
        IDs: {
            contentarea:    'form-content',
            addbutton:    'add-field',
            countfield:     'current-count',
            reset:        'reset-button'
        }
    };

    var contentarea,
```

```
        template,
        addbutton,
        countfield,
        reset;

    return {
        init: function () {
            addbutton = $(config.IDs.addbutton);
            contentarea = $(config.IDs.contentarea);
            countfield = $(config.IDs.countfield);
            reset = $(config.IDs.reset);

            template = new Template(contentarea.innerHTML);
            contentarea.innerHTML = '';

            addbutton.observe('click', Controller.addField);
            reset.observe('click', Controller.resetFields);

            Controller.addField();
        },
        addField: function () {
        },
        resetFields: function () {
        }
    };
}();

document.observe('dom:loaded', Controller.init);
```

As you can see, we now have a basic init that grabs pointers to the important DOM elements and stores them. Due to the glory of closures, those references will be available later in our addField() and resetFields() methods. We also grab the content of form-content and convert it into a template to be used later, empty the area, and then call addField() to add our first real value.

```
addField: function () {
    var count = ++countfield.value;

    contentarea.insert(template.evaluate({
        number: count
    }));

    countfield.value = count;
}
```

addField() gets the value of countfield and increments it, calls the template and inserts the content into contentarea, and then saves the incremented count value.

To handle resets, we have the following snippet:

```
resetFields: function () {
    countfield.value = 0;
    contentarea.innerHTML = '';
    Controller.addField();
}
```

That resets the countfield value, clears the HTML, and then adds a fresh new field.