



SpiderLogic
10000 Innovation Drive
Milwaukee, WI 53226

Phone: 414.290.8015
E-mail: info@spiderlogic.com

Where architects code and coders architect!



Spider Weaves BREWERFAN.NET By Quarterly Spin Staff

After a hard day of building top notch software applications, veteran Spider Brian Kapellusch goes home and thinks about baseball. More specifically, Brian is a huge Milwaukee Brewers fan and has combined his love for the Brewers with his web expertise to create BREWERFAN.NET

Brewerfan.net was founded in May 2001 (as brewerfan.com). Its goal since that time has been to provide an independent view of the Milwaukee Brewers baseball organization from top to bottom.

The site is community driven, drawing from the knowledge and enthusiasm of the die-hard fans of the club. Brewerfan.net is non-profit where any revenue derived is poured directly back into the site. Staff members work only out of the fondness for the club and in hope of converting casual fans into die-hard ones.

In case you're wondering, this is no "fly-by-night" site. Brian and his team have put together a comprehensive and informative site filled with content. The site is widely read in Brewer fan circles and has many adoring fans of its own.

Brian is particularly knowledgeable on the Brewer minor league system and is always eager to share his take on the latest major league prospect. In fact, you may hear Brian on one of the local sports talk shows or see him quoted in local sports blogs or newspapers.

Over the years, Brian has developed relationships with Brewers insiders and frequently taps his resources to get new scoops. As you might expect, brewerfan.net receives a lot of traffic during the MLB baseball draft period; a time when baseball geeks and fanatics revel in stats and rumors.

On a more technical note, Brian is in the process of converting the site from a classic Java/Struts based application to Grails.



Quarterly Spin



Volume 2, Issue 1
Spring 2009

Spinning Again in a New Office By Dave Buettner

After a long hiatus, we are pleased to bring back The Quarterly Spin Newsletter. We look forward to sharing with you some of our news and helpful technical insight.

After weathering a long construction period and a daily symphony of jack hammers and other construction equipment, the Milwaukee Spider team recently moved into a new office suite located in the new wing of the 10000 Innovation Drive Building. Our new office is located just down the hall from our old office.



This move accommodates our growing team with a bit more added room for expansion. In addition, we are now equipped with a very spacious conference room featuring our mega-whiteboard, sure to challenge even the most diligent and verbose presenter.

We have a very open environment in Milwaukee i.e. very few walls and no cubes. We like this set-up, as it facilitates and encourages discussion. Yet there are times when you just need some quiet. In order to accommodate this, we

added 2 "caves" (small quiet rooms) which are used by our team. Combine this with our new kitchen with real appliances and you have a great place to work. If you are in the Milwaukee area, please stop by and visit



Inside this issue:

<i>Spinning Again in a New Office</i>	1
<i>Refreshingly Efficient Meetings</i>	1
<i>Update table Data in Grails using Ajax Calls</i>	2
<i>Spider Weaves BREWERFAN.NET</i>	4

Refreshingly Efficient Meetings By Dan Tanner

Everybody dreads bad meetings. Meetings are often too long, too often, or don't accomplish anything. At some companies, this is what they are all like.

You can implement processes to help have better meetings (which is another meeting of course). Heck, I just attended a meeting last week titled *How to Structure Our Meetings to Have Less of Them*.

Like everything else in the thought-based industries, the effectiveness of a meeting is 10% process and 90% people.

The other day, I attended a meeting that was remarkably crisp and effective. It was a brainstorming session, so I wouldn't have expected it. The meeting started with a general leaning towards one solution to the problem, and 15 minutes later (15 minutes early), the meeting ended with a clear path towards the exact opposite style of solution.

I walked out of the meeting thinking, "Wow! That was fun!"

To which I quickly looked in the mirror wondering why the heck I would have just enjoyed being in a meeting. Developers have been

bred to hate meetings, so I obviously needed to do some serious soul-searching.

What seemed unique to this meeting vs. so many others were the following attributes:

1. An inherent sense of urgency by role: The key person in the room was a company officer with a ton of experience making decisions on a limited schedule. Sure, everybody's busy, but some people are really busy and yet have that ability to keep the people in the room at attention. Maybe it goes with the title, maybe it's decades of experience, maybe a little of both.

Continued on page 3





Update Table Data in Grails using Ajax Calls

By Geoff Lane

Using Ajax for simple forms can offer users a very clean, simple and fast way to input data. I came across a situation recently where I was looking into replacing a document based workflow with an application. The documents themselves contained a series of different kinds of transactions that could have occurred. There was not a set number of any of the types, and the user needed a simple way to enter many rows of data.

Grails offers some very easy to use simple Ajax controls. The `formRemote` tag has an `update` parameter that can be used to specify a DOM element to update after the form is submitted. This is Ajax in it's simplest form. Submit some data and replace the contents of a DOM element with a partial-page update. This works well if you want to update a div for example. But what if you want to add a row to a table? In the standard update method, you would have to re-render the entire table contents. In many cases that will be fine, but I figured I could do better than that and just render the new row and add it to the table.

Defining Domain Classes

For this example we'll start with some simple classes. An *Account* can have many *Checks* posted against it. There is no set number, so we define a *has-Many* relationship between the *Account* and the *Checks*.

```
class Account {
    static transients = ['checksTotal']
    static hasMany = [checks:Check]

    Date date = new Date()

    Double getChecksTotal() {
        if (! checks)
            return 0
        return checks.inject(0) { current, check-> current + check.amount }
    }

    static mapping = {
        checks joinTable: false
    }
}

class Check {
    String name
    Double amount
    String checkNumber
    String reason
}
```

Creating a Controller

Next we need to create a Controller to mediate between our Domain and our Views. The Controller is set up to generate the default scaffold. We'll end up overriding the `show` view to add a form on the page to easily post Checks to that Account. The action is the `postCheck` action. This is our Ajax action that will post the Check to the Account.

```
class AccountController {
    def scaffold = true

    def postCheck = {
        def account = Account.get(params.id)
        if(account) {
            def check = new Check(params)
            account.addToChecks(check)
            if(! account.hasErrors() && account.save()) {
                render template:'check', bean:check, var:'check'
            }
        }
    }
}
```

The `postCheck` method calls the `render` method to render a template and return it as the Ajax response. This template contains the markup for a single Check row.

You can see the check template here:

```
<tr>
  <td>${check?.name}</td>
  <td>${check?.checkNumber}</td>
  <td><g:formatNumber number="${check?.amount}" format="\$###,##0" /></td>
  <td>${check?.reason}</td>
</tr>
```

Continued on page 3



Check out Geoff's blog at Zorched.net



Refreshingly Efficient Meetings

continued from pg. 1

2. Flexibility in thought (an open mind): There weren't any fixed or hidden agendas, which allowed the discussion to flow quickly through the options without getting hung up.

3. Real-time calculations: Part of the discussion involved ROI. We could've marked that data as follow-up items, which would've left the discussion muddy and forced another meeting, but it wasn't necessary. With the business having a good handle on its critical numbers, and the technology team having a good rough estimate for the project size, we were able to do the math in our heads and flip through the choices in minutes.

The simplest written analogy to this that I've seen is the chapter titled "Rattle Yer Dags" in the book [Adrenaline Junkies and Template Zombies](#), by the legendary Tom DeMarco and Atlantic Systems Guild. (Highly recommended.)

And you can follow the guidelines given when you Google "how to run an effective meeting". That's all good stuff...the unfortunate reality for most is that meetings like this don't happen often enough.

So, cheers to refreshingly efficient meetings!

Gotta go...I'm late for another meeting.



Check out Dan's blog at Otherthanthink.blogspot.com

Update Table Data in Grails ...

continued from pg. 2

Creating the Ajax Form

Now we need to add the form to our `account/show.gsp` so that we can call the `AccountController.postCheck` action. The example form is below. The thing to notice is that we're not using the `update` parameter of the `formRemote`, but rather the `onSuccess` parameter. The update method tells the Ajax callback what element to replace but we can't do that with a table. `onSuccess`, on the other hand, allows us to handle the returned values in our own way. Here, we're going to call the `appendTableRow` JavaScript function to handle the response.

In this case I definitely don't want to replace the entire node, because the first row of my table is actually the form itself. This gives it a very nice look where the data is entered under the appropriate header and then immediately shows up as the following row when the user hits enter.

```
<h3>Checks</h3>
<g:formRemote name="postCheck" url="[action: 'postCheck', params: [id:account?.id]]"
  onSuccess="appendTableRow('checks', e)">
  <table id="checks">
    <thead>
      <tr>
        <th>Name</th>
        <th>Check #</th>
        <th>Amount</th>
        <th>Reason</th>
      </tr>
    </thead>
    <tbody>
      <tr>
        <td><g:textField id="name" name="name" /></td>
        <td><g:textField id="checkNumber" name="checkNumber" /></td>
        <td><g:textField id="amount" name="amount" /></td>
        <td><g:textField id="reason" name="reason" /></td>
      </tr>
    </tbody>
  </table>
</g:formRemote>
```

Luckily the `appendTableRow` function is made pretty easy by using the `prototype` APIs. The event returned to us is an HTML snippet, so in this case we just want to insert that snippet directly into the DOM which is accomplished with the `insert` prototype function.

```
function appendTableRow(tableId, e) {
  $(tableId).down('tbody').down('tr').insert({after: e.responseText});
}
```

In the end you have a data table that allows for easy, fast updates with almost no hassle.

