Extracted from:

Grails
A Quick-Start Guide

The Pragmatic Programmers

Grails

A Quick-Start Guide

Dave Klein

Edited by Colleen Toporek

Our Pragmatic courses, workshops, and other products can help you and your team create better software and have more fun. For more information, as well as the latest Pragmatic titles, please visit us at

http://www.pragprog.com

Printed in the United States of America.

ISBN-10: 1-934356-46-8

ISBN-13: 978-1-934356-46-3

Printed on acid-free paper.

B6.0 printing, July 28, 2009

Version: 2009-8-7

The Create Action

Download beyond/TekDays/grails-app/controllers/TekEventController.groovy

```
def create = {
    def tekEventInstance = new TekEvent()
    tekEventInstance.properties = params
    return ['tekEventInstance':tekEventInstance]
}
```

The create action creates a new TekEvent instance and then assigns the params to its properties property. (We'll see why this is done shortly.) Then it returns that instance in a Map with the key of tekEventInstance. Finally, it renders the create view.

The Save Action

Download beyond/TekDays/grails-app/controllers/TekEventController.groovy

```
    def save = {
        def tekEventInstance = new TekEvent(params)
        if(!tekEventInstance.hasErrors() && tekEventInstance.save()) {
            flash.message = "TekEvent ${tekEventInstance.id} created"
            redirect(action:show,id:tekEventInstance.id)
        }
        else {
            render(view:'create',model:[tekEventInstance:tekEventInstance])
        }
    }
}
```

The save action is called from the create view. It correlates to the update action and does pretty much the same thing, minus the concurrency check (which isn't an issue when creating new records). If all is well, the show view is rendered with the newly created instance. If there are problems, the user is redirected back to the create action. (This is why the params are assigned to the tekEventInstance.properties in the create action.)

So there's a tour of the generated actions of a Grails controller. We only looked at one of the six controllers that were generated by the generate-all script, but they all have the same code with different domain classes. Feel free to browse the rest of them. It should all look very familiar. Now we'll see what Grails gives us for views.

5.3 Grails Views with Groovy Server Pages

Grails uses Groovy Server Pages (GSP) for its view layer. If you've ever worked with Java Server Pages, well, you have my sympathy, but GSP

will seem familiar—only easier to work with. Grails also uses SiteMesh,[8] the page decoration framework from OpenSymphony, to assist in the page layout. SiteMesh will merge each of our .gsp files into a file called main.gsp. This is what gives a consistent look to all of our pages, as we saw with the dynamic scaffolding. We'll begin our tour of the generated views with main.gsp, followed by the four views generated for the TekEvent class. Then we'll look at a couple of the other views that take advantage of additional Grails features.

Exploring main.gsp

Download beyond/TekDays/grails-app/views/layouts/main.gsp

```
<html>
  <head>
    <title><g:layoutTitle default="Grails" /></title>
    <link rel="stylesheet"
          href="${resource(dir:'css',file:'main.css')}" />
    <link rel="shortcut icon"
          href="${resource(dir:'images',file:'favicon.ico')}"
          type="image/x-icon" />
    <g:layoutHead />
    <g:javascript library="application" />
  </head>
  <body>
    <div id="spinner" class="spinner" style="display:none;">
      <img src="${resource(dir:'images',file:'spinner.gif')}" alt="Spinner"/>
    </div>
    <div class="logo">
      <img src="${resource(dir:'images',file:'grails_logo.jpg')}" alt="Grails"/>
    </div>
    <g:layoutBody />
  </body>
</html>
```

The main.gsp starts out with a <*title*> in the <*head*> section. This tag contains a <*g:layoutTitle*> tag, which will substitute the <*title*> from the view that is being merged. Next, it links in a style sheet and favicon that will be used by all views. Then there is the <*g:layoutHead*> tag. This will merge in the contents of the target view's <*head*> section. The <*body*> section contains a spinner image, an application logo, and a <*g:layoutBody*> tag, which merges in the <*body*> contents of the target view.

As you can see, this file gives us a convenient place to make some major improvements to our application. And that's just what we're going to do,

8. http://opensymphony.com/sitemesh

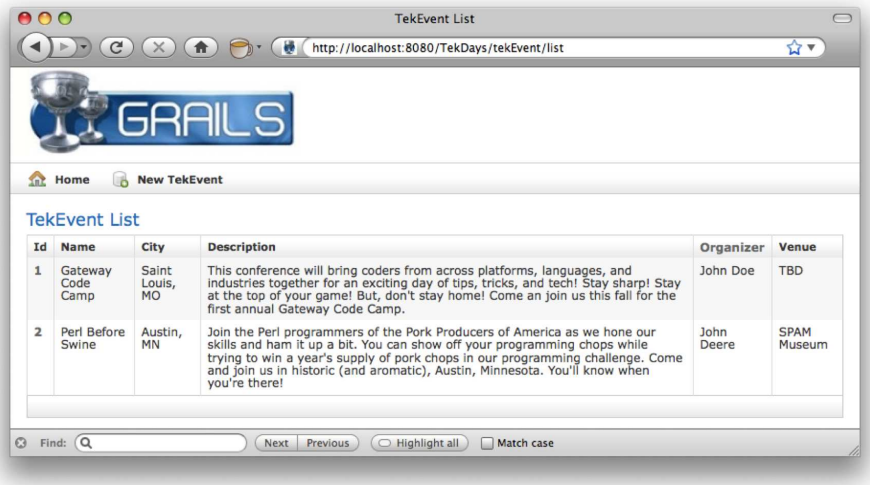

Figure 5.1: TekEvent List View

as soon as we finish our tour. As we discuss the four generated views, we will only be looking at portions of them, for the sake of space. I'll give you the name and path for each file so you can open the one on your system and follow along.

The List View

The TekEventlist view is shown in Figure 5.1. You can refer to that image as we look at the GSP code behind it. You'll find this code in TekDays/grails-app/views/tekEvent/list.gsp.

```
<span class="menuButton">
  <a class="home" href="${resource(dir:'')}">Home</a>
</span>
<span class="menuButton">
  <g:link class="create" action="create">New TekEvent</g:link>
</span>
```

This code creates the button bar just below the Grails logo. We can see two ways that Grails provides for creating links. The resource() method takes a relative path and creates a URL, which is assigned to the href attribute of an anchor tag. The *<g:link>* tag creates an anchor tag using the values of the controller, action, and id attributes (if they're provided).

If a controller is not provided, the current controller is assumed. In this case, a link to the create action of the TekEventController will be created.

```
<g:if test="${flash.message}">
  <div class="message">${flash.message}</div>
</g:if>
```

This code doesn't show up in Figure 5.1, on the previous page, but it is important to take note of. Recall that during our discussion of controllers, we often had code that would store text in the message element of flash. This is where that text will show up. The <*g:if*> tag checks for the existence of flash.message, and if found displays it.

```
<g:sortableColumn property="name" title="Name" />
<g:sortableColumn property="city" title="City" />
<g:sortableColumn property="description" title="Description" />
<th>Organizer</th>
<g:sortableColumn property="venue" title="Venue" />
```

The <*g:sortableColumn*> tag is what Grails uses to provide sorting on our list view. Note that, by default, this only works with regular properties, not object references or collections. That is why we see a <*th*> tag used for the Organizer property.

```
<g:each in="${tekEventInstanceList}" status="i"
                                     var="tekEventInstance">
  <tr class="${(i % 2) == 0 ? 'odd' : 'even'}">
    <td>
      <g:link action="show" id="${tekEventInstance.id}">
        ${fieldValue(bean:tekEventInstance, field:'id')}
      </g:link></td>
    <td>${fieldValue(bean:tekEventInstance, field:'name')}</td>
    <td>${fieldValue(bean:tekEventInstance, field:'city')}</td>
    <td>${fieldValue(bean:tekEventInstance, field:'description')}</td>
    <td>${fieldValue(bean:tekEventInstance, field:'organizer')}</td>
    <td>${fieldValue(bean:tekEventInstance, field:'venue')}</td>
  </tr>
</g:each>
```

This code is the heart of the list view. We start with the <*g:each*> tag, which iterates over the list that we passed in from the controller. Each item in the list is assigned to the tekEventInstance variable. The body of the <*g:each*> tag fills in the table row with the properties of the tekEventInstance. Notice that a Groovy expression is used to determine the CSS class of the <*tr*>—powerful stuff! Inside the <*td*> tags, the fieldValue() method is used to render the value of each TekEvent property. We'll learn more about the fieldValue() method when we look at the create view.

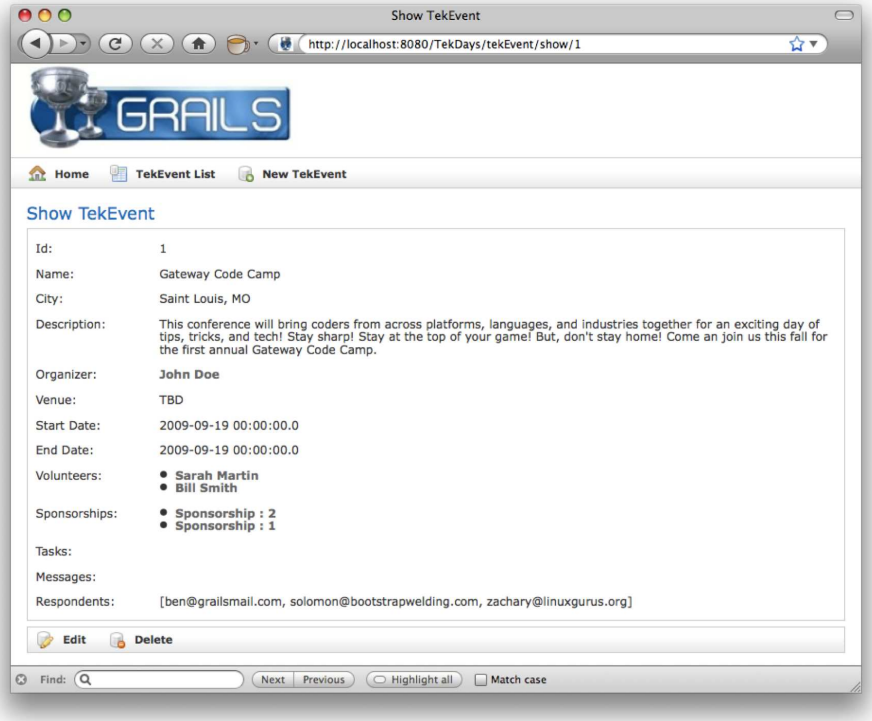

Figure 5.2: TekEvent Show View

```
<div class="paginateButtons">
  <g:paginate total="${tekEventInstanceTotal}" />
</div>
```

The final portion of the list.gsp we'll look at is another one that we can't see in Figure 5.1, on page 85. The *<g:paginate>* tag would cause pagination buttons to show up at the bottom of the list view if we had enough events displayed to warrant it. This tag uses the total that we passed in from the controller's list action.

The Show View

The show view, pictured in Figure 5.2, is in TekDays/grails-app/views/tekEvent/show.gsp. Open this file now as we look at a few interesting sections.

```
<tr class="prop">
  <td valign="top" class="name">Id:</td>
  <td valign="top" class="value">
```

```
    ${fieldValue(bean:tekEventInstance, field:'id')}
  </td>
</tr>
<tr class="prop">
  <td valign="top" class="name">Name:</td>
  <td valign="top" class="value">
    ${fieldValue(bean:tekEventInstance, field:'name')}
  </td>
</tr>
```

This code shows a couple of examples of how Grails displays text properties. Notice the CSS class hierarchy. The *<tr>* tag has a prop class and the *<td>* tags can have either a name or value class.

```
<tr class="prop">
  <td valign="top" class="name">Organizer:</td>
  <td valign="top" class="value">
    <g:link controller="tekUser" action="show"
            id="${tekEventInstance?.organizer?.id}">
      ${tekEventInstance?.organizer?.encodeAsHTML()}
    </g:link>
  </td>
</tr>
```

Here we have an example of the way Grails displays a related object. The Organizer property is rendered as a link to the TekUsershow view. The *<g:link>* tag has its controller and action attributes set accordingly. The id is set to a Groovy expression that reads the id property of the organizer property of the tekEventInstance that we passed in from the controller. Notice the ? after the tekEventInstance and organizer references: this is Groovy's *safe navigation* operator. When this expression is evaluated, if either of these items is null, the whole expression evaluates to null and no exception is thrown. This operator has saved the world from untold numbers of if blocks!

```
<tr class="prop">
  <td valign="top" class="name">Start Date:</td>
  <td valign="top" class="value">
    ${fieldValue(bean:tekEventInstance, field:'startDate')}
  </td>
</tr>
```

The startDate is a Date type, and yet it is rendered the same way as a text property. Grails handles the conversion from Date to String for us.

```
<tr class="prop">
  <td valign="top" class="name">Volunteers:</td>
  <td  valign="top" style="text-align:left;" class="value">
    <ul>
      <g:each var="v" in="${tekEventInstance.volunteers}">
```

```
        <li><g:link controller="tekUser" action="show" id="${v.id}">
          ${v?.encodeAsHTML()}
        </g:link></li>
      </g:each>
    </ul>
  </td>
</tr>
```

The Grails scaffolding renders one-to-many relationships as an unordered list. Here we see the volunteers property being displayed using a <*g:each*> tag inside of a <*ul*> tag. Another thing to notice here is the use of the encodeAsHTML() method. This method is added to all String objects, and prevents any HTML code from being processed while the page is rendering. This is helpful in defending against cross-site scripting attacks.[9]

```
<tr class="prop">
  <td valign="top" class="name">Respondents:</td>
  <td valign="top" class="value">
    ${fieldValue(bean:tekEventInstance, field:'respondents')}
  </td>
</tr>
```

Rounding out the show view, we have the respondents collection. This property is a collection of String objects containing email addresses. This type of collection is rendered as if it were a single String field. Grails handles converting it to a comma-separated list, as we can see in Figure 5.2, on page 87. If we wanted to, we could use a <*g:each*> tag to show these as a list or in a table.

The Create View

We can see the create view in Figure 5.3, on the next page. The code for this view is in TekDays/grails-app/views/tekEvent/create.gsp. Open this file, and we'll see what new and exciting things it has in store for us.

```
<g:hasErrors bean="${tekEventInstance}">
  <div class="errors">
    <g:renderErrors bean="${tekEventInstance}" as="list" />
  </div>
</g:hasErrors>
```

In Section 5.3, *The List View*, on page 85, we saw how messages that we set in the controller are displayed in the view. Here we see another type of message block. When a domain instance fails to save, errors are stored in an errors property. The <*g:hasErrors*> tag is a conditional tag

9. http://en.wikipedia.org/wiki/Cross-site_scripting

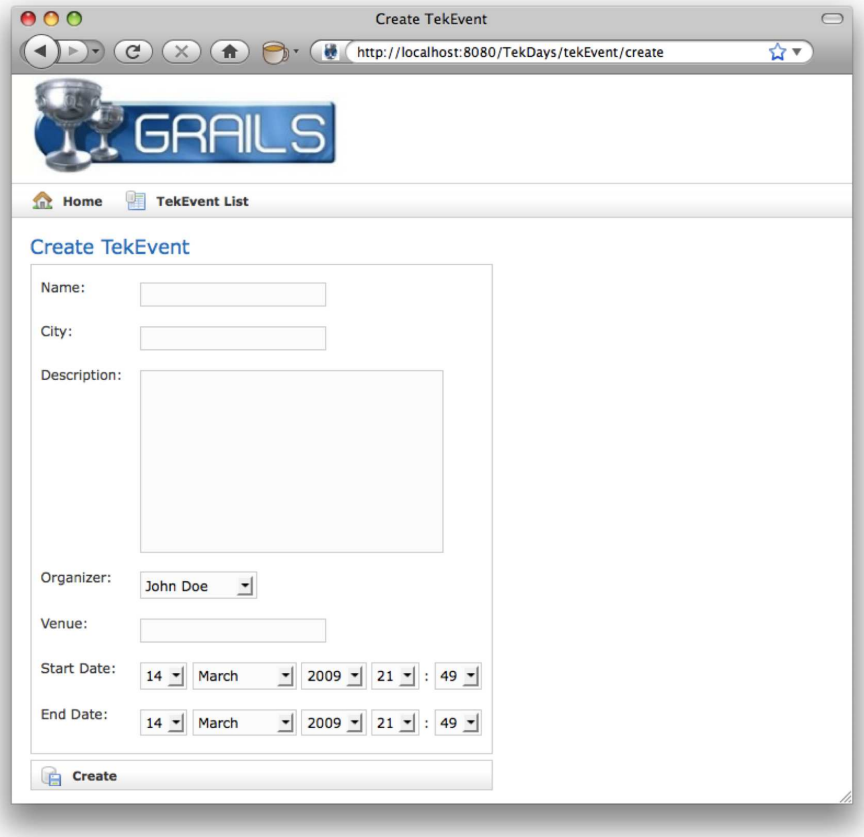

Figure 5.3: TekEvent Create View

that examines the domain instance assigned to its bean attribute and renders its body if errors are found. In the body of the tag, we find the *<g:renderErrors>* tag, which will display the errors in a list at the top of the page. (See Figure 5.4, on the next page.)

```
<g:form action="save" method="post" >
```

The *<g:form>* tag sets up an HTML form. This tag has controller, action, and id attributes, which will result in the URL to submit the form to. In this case, we're only using the action attribute.

```
<tr class="prop">
  <td valign="top" class="name">
    <label for="name">Name:</label>
```

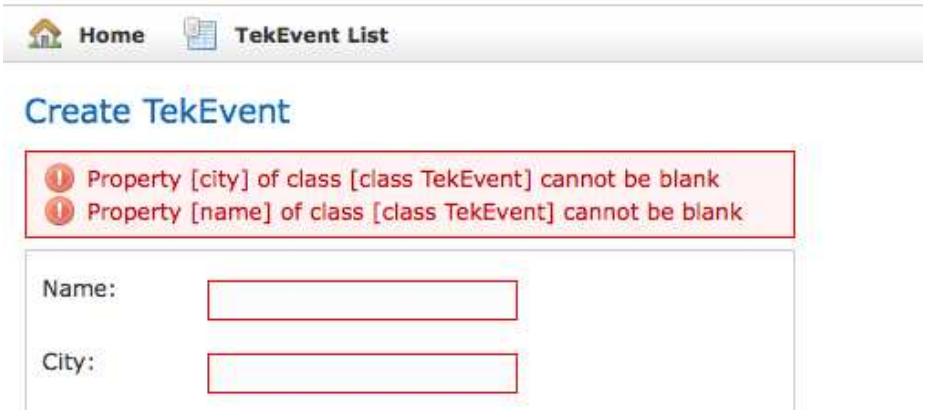

Figure 5.4: Built-in Error Handling

```
  </td>
  <td valign="top" class="value
      ${hasErrors(bean:tekEventInstance,field:'name','errors')}">
    <input type="text"
           id="name"
           name="name"
           value="${fieldValue(bean:tekEventInstance,field:'name')}"/>
  </td>
</tr>
```

The create view uses the same two-column table layout as the show view. The difference is that here the second column contains HTML input elements. Notice how the *<g:hasErrors>* tag is used in a Groovy expression to determine the CSS class to use. It doesn't look like a tag, does it? All GSP tags can also be called as methods. How's that for versatile?

Next, the value attribute of the input element is set to another Groovy expression using the fieldValue() method. This is where this method really shines.

In Section 5.2, *The Save Action*, on page 83, we saw that if validation fails, we redirect the user back to the create view. In this case, we don't want to show the actual values of the tekEventInstance. We want to re-display the values that the user has entered. These values are stored as part of the errors collection, and fieldValue() knows how to get them. If there are no errors, then the tekEventInstance properties are

displayed. This method also calls encodeAsHTML() for us, since that is almost always what we want.

```
<tr class="prop">
  <td valign="top" class="name">
    <label for="description">Description:</label>
  </td>
  <td valign="top" class="value
    ${hasErrors(bean:tekEventInstance,field:'description','errors')}">
    <textarea rows="5" cols="40" name="description">
      ${fieldValue(bean:tekEventInstance, field:'description')}
    </textarea>
  </td>
</tr>
```

For the description property, Grails is using a <*textarea*> element.

```
<tr class="prop">
  <td valign="top" class="name">
    <label for="startDate">Start Date:</label>
  </td>
  <td valign="top" class="value $
      {hasErrors(bean:tekEventInstance,field:'startDate','errors')}">
    <g:datePicker name="startDate"
                  value="${tekEventInstance?.startDate}" >
    </g:datePicker>
  </td>
</tr>
```

The <*g:datePicker*> tag renders that series of select elements that we see in Figure 5.3, on page 90. This tag can be configured to be much more useful by using the precision and noSelection attributes.[10]

The Edit View

The last of the scaffolded views is the edit view. See Figure 5.5, on the following page. You'll find the code in TekDays/grails-app/views/tekEvent/edit.gsp. By now, we've seen most areas of interest covered in the preceding views, but open this one up and follow along as we see what nuggets might be awaiting discovery.

```
<tr class="prop">
  <td valign="top" class="name">
    <label for="organizer">Organizer:</label>
  </td>
  <td valign="top" class="value
      ${hasErrors(bean:tekEventInstance,field:'organizer','errors')}">
    <g:select optionKey="id" from="${TekUser.list()}"
```

10. See http://www.grails.org/doc/1.1/ref/Tags/datePicker.html.

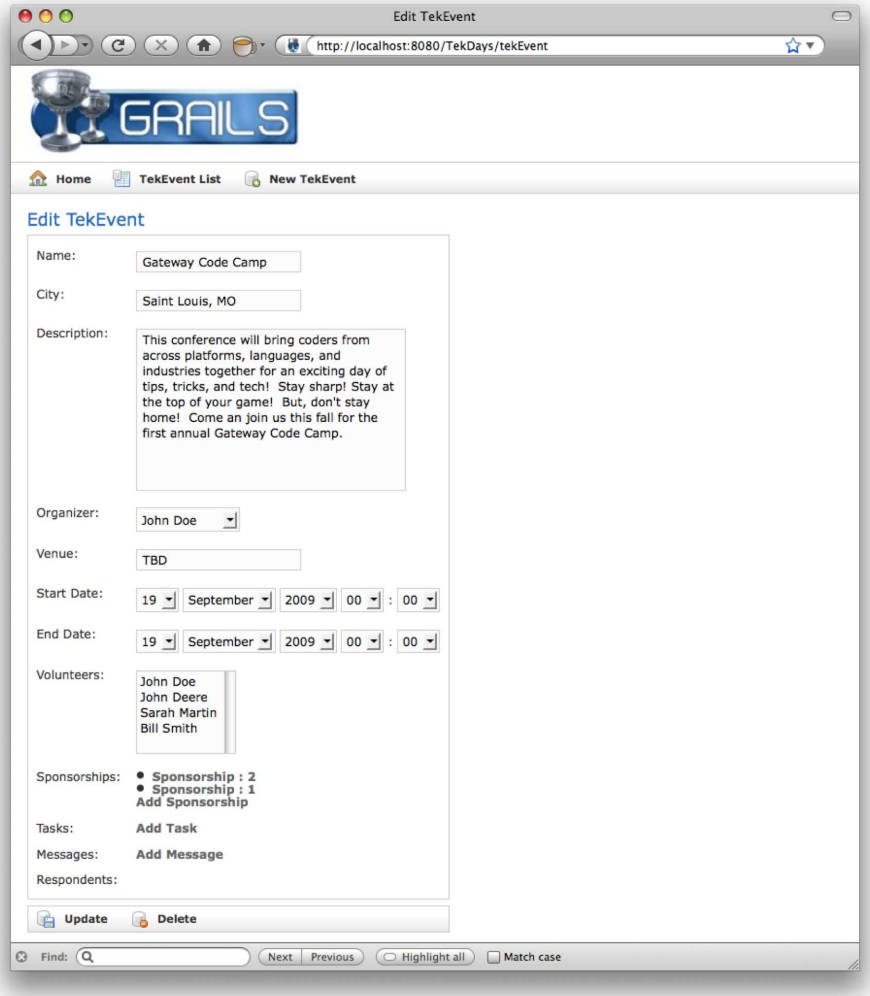

Figure 5.5: TekEvent Edit View

```
            name="organizer.id"
            value="${tekEventInstance?.organizer?.id}" >
    </g:select>
  </td>
</tr>
```

Here's something new. For properties that are references to another domain class, Grails uses a *<g:select>* tag, which will render a *<select>* element loaded with all the available choices for that class. In this case, we end up with a list of TekUser instances that can be assigned to the organizer property.

```
<tr class="prop">
  <td valign="top" class="name">
    <label for="volunteers">Volunteers:</label>
  </td>
  <td valign="top" class="value
      ${hasErrors(bean:tekEventInstance,field:'volunteers','errors')}">
    <g:select name="volunteers" from="${TekUser.list()}"
            size="5" multiple="yes" optionKey="id"
            value="${tekEventInstance?.volunteers}" />

  </td>
</tr>
```

Grails also uses a *<g:select>* tag for uni-directional one-to-many relationships. In this case, the multiple attribute is set to yes and the value attribute is set to the volunteers collection property. This will render a multi-select list box loaded with TekUser instances. When submitted, all of the selected instances are automagically added to the volunteers property.

```
<tr class="prop">
  <td valign="top" class="name">
    <label for="sponsorships">Sponsorships:</label>
  </td>
  <td valign="top" class="value
      ${hasErrors(bean:tekEventInstance,field:'sponsorships','errors')}">
    <ul>
      <g:each var="s" in="${tekEventInstance?.sponsorships?}">
        <li>
          <g:link controller="sponsorship" action="show"
                  id="${s.id}">${s?.encodeAsHTML()}</g:link>
        </li>
      </g:each>
    </ul>
    <g:link controller="sponsorship"
          params="['tekEvent.id':tekEventInstance?.id]"
          action="create">
      Add Sponsorship
```

```
    </g:link>
  </td>
</tr>
```

In this block, we can see how the sponsorship collection property is rendered as an unordered list of links. We already saw this in Section 5.3, *The Show View*, on page 87. What's interesting here is that immediately after the <*ul*> tag is closed, there is a <*g:link*> tag that will render a link to the create action of the SponsorshipController. The value in the params attribute will cause this TekEvent instance to be assigned to the tekEvent property of the newly created Sponsorship.

And this concludes our tour of the code behind the scaffolded views. Now that this code is available to us and we've got a working understanding of what it is doing, we can see how we could make a few changes to make our application a little better looking and easier to use. We'll do that beginning in the next chapter, but first, let's see how we can hook-up to a real database so we no longer lose our data changes every time we restart the application.

5.4 Configuring a Database

The in-memory database that comes with Grails is handy and we have been making good use of it, but a time comes in the life of any application when you need to have your data stored in a real database. (Hopefully this happens before you go to production.) As with most things, Grails makes this a snap to do.

"Configuration?" You may be wondering what happened to "Convention over Configuration". Well, keep in mind that it's *over*, not *instead of*, and besides, no matter how hard Larry Ellison tries, there's still no convention for which database to use.[11] Also, Grails takes much of the pain out of the word *configuration* by allowing us to write all of our configuration code in Groovy instead of XML. The information about our database is in TekDays/grails-app/conf/DataSource.groovy. By default, it looks like this:

```
dataSource {
  pooled = true
  driverClassName = "org.hsqldb.jdbcDriver"
  username = "sa"
```

11. Larry Ellison is the co-founder and CEO of Oracle, maker of the leading enterprise database. See http://en.wikipedia.org/wiki/Larry_Ellison.

The Pragmatic Bookshelf

The Pragmatic Bookshelf features books written by developers for developers. The titles continue the well-known Pragmatic Programmer style and continue to garner awards and rave reviews. As development gets more and more difficult, the Pragmatic Programmers will be there with more titles and products to help you stay on top of your game.

Visit Us Online

Grails Quick Start's Home Page
http://pragprog.com/dkgrails
Source code from this book, errata, and other resources. Come give us feedback, too!

Register for Updates
http://pragprog.com/updates
Be notified when updates and new books become available.

Join the Community
http://pragprog.com/community
Read our weblogs, join our online discussions, participate in our mailing list, interact with our wiki, and benefit from the experience of other Pragmatic Programmers.

New and Noteworthy
http://pragprog.com/news
Check out the latest pragmatic developments, new titles and other offerings.

Buy the Book

If you liked this eBook, perhaps you'd like to have a paper copy of the book. It's available for purchase at our store: pragprog.com/dkgrails.

Contact Us

Online Orders:	www.pragprog.com/catalog
Customer Service:	support@pragprog.com
Non-English Versions:	translations@pragprog.com
Pragmatic Teaching:	academic@pragprog.com
Author Proposals:	proposals@pragprog.com
Contact us:	1-800-699-PROG (+1 919 847 3884)