View Layer

At the the end of this session

You will be familiar with

- Creating custom taglibs
- Codecs
- Sitemesh
- Quirks with caching of resources

Custom Taglibs

A tag library is a "view helper" and helps with GSP rendering. Use a tagline when:

- When you've got some view code you'd like to reuse,
- or your GSP logic deserves unit/integration testing,
- or the view would be much easier to understand with simplified markup.

Don't forget that they are intended to be "view helpers!" When your tag is doing more than helping to render the view, then your controller should be doing a better/more-complete job generating the appropriate model.

Custom Taglibs

In Grails a tag library is a class

- with a name that ends in the convention "TagLib"
- and lives in the grails-app/taglib directory.

Use the create-tag-lib command create a tag library:

```
> grails create-tag-lib format
```

A Simple Example

```
import java.text.SimpleDateFormat

class FormatTagLib {
 def dateFormat = { attrs, body ->
   out << new SimpleDateFormat(attrs.format).format(attrs.date)
 }
}
```

In a GSP:

```
<g:dateFormat format="dd-MM-yyyy" date="${new Date()}" />
```

Writing HTML markup

Embedded:

```
def formatBook = { attrs, body ->
       out << "<div id="${attrs.book.id}">"
       out << "Title : ${attrs.book.title}"
       out << "</div>"
}
```

Using a template:

```
def formatBook = { attrs, body ->
       out << render(template: "bookTemplate",
                     model: [book: attrs.book])
}
```

Logical Tags

Body of the tag is only output once a set of conditions have been met.

```
def isAdmin = { attrs, body ->
        def user = attrs.user
        if (user && checkUserPrivs(user)) {
            out << body()
        }
}
```

The tag checks if the user is an administrator and only outputs the body content if he/she has the correct set of access privileges:

In a GSP:

```
<g:isAdmin user="${myUser}">
    // some restricted content
</g:isAdmin>
```

Iteration

```
def repeat = { attrs, body ->
        def var = attrs.var ?: "num"
        attrs.times?.toInteger()?.times { num ->
            out << body((var):num)
        }
}
```

In a GSP:

```
<g:repeat times="3" var="j">
    <p>Repeat this 3 times! Current repeat = ${j}</p>
</g:repeat>
```

Tag Namespaces

By default, tags are added to the default Grails namespace and are used with the g: prefix in GSP pages. However, you can specify a different namespace by adding a static property to your TagLib class:

```
class SimpleTagLib {
    static namespace = "my"

    def example = { attrs -> ... }
}
```

In a GSP:

```
<my:example ... />
```

Using JSP Tags

JSP tags can be used from GSP. Simply declare the JSP to use with the taglib directive:

```
<%@ taglib prefix="fmt"
           uri="http://java.sun.com/jsp/jstl/fmt" %>
```

Then you can use it like any other tag:

```
<fmt:formatNumber value="${10}" pattern=".00"/>
```

With the added bonus that you can invoke JSP tags like methods:

```
${fmt.formatNumber(value:10, pattern:".00")}
```

Tags as method calls

Tags can be invoked from within controllers and tag libraries. Simply call the tag as if it were a method. Like this:

```
def imageLocation =
    g.createLinkTo(dir:"images", file:"logo.jpg").toString()
```

Note that the default g: namespace tags can be invoked without the "g." prefix, but to avoid confusion always prefix the namespace.

Tags in custom namespaces use that prefix for the method call. For example

Workshop - Taglibs

Create a custom taglib - QuestionTagLib

Add 'summary' method that prints out the question title then the first 15 characters of the question text followed by "..."

Ex. "Thanksgiving: Why is Thanksgi..."

Call the QuestionTagLib in 'each' loop in question/index.gsp

Workshop - Example Solution

```
class QuestionTagLib {
  static namespace = "question"

  def summary = {attrs, body ->
      Question question = attrs.question

      String title = question.title
      int textEndIndex = Math.min(question.text?.length(),
15)
      String textSummary = question.text?.substring(0,
textEndIndex - 1) ?: ""

      out << "${title}: ${textSummary}..."
  }
}
```

Workshop - Example GSP

```
<g:each in="${questions}" var="question">
  <h3><g:link controller="question" action="show"
id="${question.id}">${question.title}</g:link> </h3>
  <div> <question:summary question="${question}" /> </div>
</g:each>
```

Codecs

- Grails supports the concept of dynamic encode/decode methods.
- A set of standard codecs are bundled with Grails.
- Grails also supports a simple mechanism for developers to contribute their own codecs that will be recognized at runtime.

Codecs

A Grails codec class has an encode closure, a decode closure or both.

When a Grails app starts up the framework dynamically loads codecs from grails-app/utils/

By convention a codec class ends with "Codec" (e.g. `HTMLCodec`, `MyCodec`, `YourCodec`, ...)

Grails creates dynamic encode/decode methods and adds them to the Object class with a name representing the codec that defined the encode/decode closures. For example, the HTMLCodec class defines an "encode" closure, so Grails attaches it with the name "`encodeAsHTML`". Similarly for "decode" closures.

For example

In a GSP:

```
${report.description.encodeAsHTML()}
```

Decoding is performed using `value.decodeHTML()` syntax.

Remember: the `decodeHTML` method was added to `Object`, so it is available everywhere!

Standard Codecs

HTMLCodec -- HTML escaping and unescaping, so that values can be rendered safely in an HTML page without creating any HTML tags or damaging the page layout. For example

```
"Don't you know that 2 > 1?".encodeAsHTML() -->
        "Don't you know that 2 &gt; 1?"
```

Standard Codecs

URLCodec -- used when creating URLs in links or form actions, or when data is used to create a URL. It prevents illegal characters from getting into the URL and changing its meaning, for example "Apple & Blackberry" is not going to work well as a parameter in a GET request as the ampersand will break parameter parsing.

Base64Codec -- Performs Base64 encode/decode functions.

JavaScriptCodec -- Escapes Strings so they can be used as valid JavaScript strings.

HexCodec -- Encodes byte arrays or lists of integers to lowercase hex strings, and decodes hex strings into byte arrays.

Standard Codecs

MD5Codec -- Uses MD5 to digest byte arrays, lists of integers, or the bytes of a string, as a lowercase hexadecimal string.

MD5BytesCodec -- like MD5Codec, but this one encodes as a byte array.

SHA1Codec, SHA1BytesCodec -- like MD5 codex, but uses the SHA algorithm

SHA256Codec, SHA256BytesCodec -- like MD5 codex, but uses the SHA256 algorithm

Custom Codecs

When an app defines their own codecs grails loads them along with the standard codecs. A custom codec class must be defined in the grails-app/utils/ directory and the class name must end with Codec.

A custom codec must have a static encode closure, a static decode closure or both. The closures must accept a single argument which will be the object that the dynamic method was invoked on. For Example:

```
class PigLatinCodec {
  static encode = { str ->
    // convert the string to pig latin and return the result
  }
}
```

With the above codec in place an application could do something like this:

```
${lastName.encodeAsPigLatin()}
```

Testing Codecs

The GrailsUnitTestMixin provides a mockCodec method for mocking custom

codecs which may be invoked while a unit test is running.

```
mockCodec(MyCustomCodec)
```

Failing to mock a codec which is invoked while a unit test is running may result in a MissingMethodException.

Sitemesh

Deeper dive into the Sitemesh templating library

Sitemesh is...

It is a HTML templating framework based on the "Decoration" model.

It is a web-page layout and decoration framework and web application integration framework to aid in creating large sites consisting of many pages for which a consistent look/feel, navigation and layout scheme is required.

Orignally Developed in 1999 by Joe Walnes, now part of the OpenSymphony Project http://www.opensymphony.com/sitemesh/

Layout Tags

layoutTitle, layoutHead, and layoutBody

```
<html>
<head>
  <title><g:layoutTitle default="my page" /></title>
  <g:layoutHead />
</head>
<body>
  <div class="menu"><!-- common menu here--></div>
  <div class="body">
    <g:layoutBody />
  </div>
</body>
</html>
```

Triggering Layouts

- meta.layout
- static 'layout' property on the controller
- controller/action conventions
- configured grails.sitemesh.default.layout
- default application layout

Triggering layouts by meta tag

```
<html>
<head>
  <meta name="layout" content="main"/>
  ...
</head>
<body>...</body>
</html>
```

This triggers grails-app/views/layouts/main.gsp

Triggering layouts by static controller property

```
class BookController {
    static layout = 'customLayout'

    def list = { ... }
}
```

This will trigger grails-app/views/layouts/customLayout.gsp for all of the controller actions in the BookController.

Unless a meta.layout tag is present!

Triggering layouts by naming convention

```
class BookController {
    def list = { ... }
}
```

Looks first for

```
grails-app/views/layouts/book/list.gsp
```

If not found then looks for

```
grails-app/views/layouts/book.gsp
```

In Order of Precedence

- meta.layout

- static 'layout' property on the controller

- controller/action conventions:

```
grails-app/views/layouts/${controller}/${action}.gsp
grails-app/views/layouts/${controller}.gsp
```

- configured grails.sitemesh.default.layout

```
// grails-app/conf/Config.groovy
grails.sitemesh.default.layout='myLayoutName'
```

- finally

```
grails-app/views/layouts/application.gsp
```

Nested layouts with applyLayout

```
<g:applyLayout name="fieldsetWrapper">
  This goes into the layoutBody of
  the 'fieldsetWrapper' layout.
</g:applyLayout>
```

Sitemesh Page Properties

In the GSP:

```
<html>
<head>
  <meta name="layout" content="myLayout" />
</head>
<body onload="alert('hello');">
    Page to be decorated
</body>
</html>
```

In the layout (myLayout.gsp):

```
<html>
<head><g:layoutHead /></head>
<body onload="${pageProperty(name:'body.onload')}">
  <g:layoutBody />
</body>
</html>
```

Page Properties with content tags

In the GSP:

```
<content tag="context">buyer</content>
```

In the layout:

```
<g:set value="${pageProperty(name:'page.context')}" var="
content" />
```

and in the layout:

```
<g:ifPageProperty name="showTheContent">
  ... <!-- This content is only displayed if the page
property is present -->
</g:ifPageProperty>
```

Page Properties with parameter tags

In the GSP:

```
<parameter name="myParameter" value="foo">
```

In the layout:

```
${pageProperty(name: 'page.myParameter')}
```

Page Properties with meta tags

All <meta> tags get added as a pageProperty meta.propertyName'

In the GSP:

```
<meta name="myProp" content="myContent"/>
```

In the layout:

```
<g:pageProperty name="meta.myProp" />
```

Workshops

layouts

- split the navbar in main.gsp into a template
- add header/footer templates