Mastering Grails: Grails and the mobile Web

M is the new WWW

Skill Level: Introductory

Scott Davis (scott@aboutgroovy.com)
Editor in Chief
AboutGroovy.com

17 Jun 2008

The number of cell phone users worldwide is at 3.3 billion and rising, and Internet access from mobile phones is on a rapidly upward trajectory. Developing for the mobile Web has its unique demands. In this *Mastering Grails* installment, Scott Davis shows you how to make your Grails applications mobile phone friendly.

These days, thankfully, it's rare to see a Web site include the disclaimer "Best viewed using [Browser X]." Modern Ajax libraries such as Prototype, Dojo, and YUI do a reasonable job of abstracting away the remaining differences among Firefox, Internet Explorer, and Safari. But people visiting your Web site from a Nokia, Motorola, or Apple cell phone might not enjoy the same degree of browser independence. Even the latest mobile browsers that claim "full HTML support" can benefit from a few simple tweaks to the Web content. This article shows you how to optimize your Grails applications for mobile browsers.

Mobile Web use on the rise

According to Internet World Stats, the Internet has 1.4 billion active users — roughly 20 percent of the world population (see Resources). In North America, three out of four people use the Internet.

Fully half of the 6.6 billion people on the planet have a mobile phone. Cell phones have roughly the same market penetration as the Internet in North America, but the same is not true elsewhere. Hong Kong has a 140 percent market penetration rate, and parts of the European Union (Lithuania, Italy, and Luxembourg) have rates topping 150 percent. Yes, in certain places there are more cell phones than people.

> Colin Crawford of IDG Communications says it best (see Resources): "Over the next few years, the mobile phone is on course to replace the PC as the primary device for getting online. Already 30% of worldwide Internet access is from mobile phones — but in certain countries such as Japan — it's as high as 70%."

If you're wondering why you should even bother with making your Web site mobile-friendly, the numbers in the Mobile Web use on the rise sidebar might convince you. As impressive as the global statistics are, my interest in a mobile-friendly Web is more personal. I bought an iPhone when they first came out in summer 2007. Since then, I have been actively seeking out Web sites that are usable on the device. Sure, I can visit any Web site (as long as it doesn't rely on Flash or Java™ applets, which the iPhone doesn't support). The problem is that content optimized for 800x600 resolution (or higher) doesn't look quite as nice on a 3.5-inch screen.

The Web sites I use regularly from my phone are the ones that meet me halfway with a UI that caters to the device's unique constraints. Substituting an *m* for the traditional *www* in popular Web sites' URLs is a good place to start. The pages at http://m.cnn.com, http://m.yahoo.com, and http://m.google.com all look good on my phone. Some Web sites, such as http://www.twitter.com, morph to the appropriate output magically: when I'm on a computer, I get all of the features; when I visit from my phone, the content is trimmed down and fits my screen like a glove. Same URL, optimized UI. I'll show how your site can do these things too.

Technologies for the mobile Web developer

As a Java developer, I've been spoiled by the promise of Write Once, Run Anywhere. Optimizing my Java application for a specific operating system or hardware model doesn't even enter into my mind. But if you are going to tackle mobile Web development, you should be familiar with the three primary technologies supported on different types of mobile devices:

- Wireless Markup Language (WML) 1.x
- WML 2.x or Mobile Profile (XHTML-MP)
- HTML tags targeted at the iPhone

As I'll show you, you can mix WML and XHTML-MP markup into the Groovy Server Pages (GSPs) you build with Grails to deliver mobile-friendly pages. And I'll show you how can tweak your Grails-generated HTML to make pages work better on the iPhone.

Using WML 1.x with Grails

WML is an HTML-like markup language, but it's not true HTML. (WML 1.0 was standardized in 1998. WML 1.3 is the latest version.) You can't view WML in a Web browser (at least not without the aid of an emulator), nor can you view HTML in a WML browser. Cell phone providers generally offer an HTML-to-WML gateway that translates one from the other behind the scenes.

About this series

Grails is a modern Web development framework that mixes familiar Java technologies like Spring and Hibernate with contemporary practices like convention over configuration. Written in Groovy, Grails give you seamless integration with your legacy Java code while adding the flexibility and dynamism of a scripting language. After you learn Grails, you'll never look at Web development the same way again.

WML is transported over Wireless Access Protocol (WAP), much as HTML is transported over HTTP. *WAP* and *WML* are generally interchangeable in casual conversation: it's common to see mobile phone specifications that tout either a WAP browser or WML 1.x support (see Resources for a link to the official WML and WAP specifications).

If you are targeting users of Research in Motion's BlackBerry, you should brush up on your WML. (BlackBerry has about 40 percent of the smartphone market, with iPhones and Windows® Mobile devices in second and third place, respectively.) BlackBerry smartphones come with a WAP browser, although many savvy users also download a true Web browser such as Opera Mini (see Resources).

Trip planning on the go

If you've been following along in the *Mastering Grails* series, you're ready to start making the now-familiar trip planner application mobile-phone friendly. Create a file named testwml.gsp in the trip planner application's web-app directory and type the code in Listing 1, which is some static WML:

Listing 1. Static WML

```
<% response.setContentType("text/vnd.wap.wml") %>
<?xml version="1.0"?>
<!DOCTYPE wml PUBLIC "-//PHONE.COM//DTD WML 1.1//EN"
   "http://www.phone.com/dtd/wml11.dtd" >

<wml>
  <card id="f1" title="Flight 1">
    <p mode="wrap">From: DEN</p>
    <p mode="wrap">To: ORD</p>
    <p mode="wrap">UAL 1234</p>
    <p mode="wrap">Jun 30, 10:30am</p>
    <p>
      <anchor>Next<go href="#f2"/></anchor>
    </p>
  </card>
```

```
  <card id="f2" title="Flight 2">
    <p mode="wrap">From: ORD</p>
    <p mode="wrap">To: DEN</p>
    <p mode="wrap">UAL 9876</p>
    <p mode="wrap">Jul 02, 1:15pm</p>
    <p>
      <anchor>Previous<go href="#f1"/></anchor>
    </p>
  </card>
</wml>
```

You can also see this page live (on your phone) at http://www.davisworld.org/testwml.gsp. You are used to seeing HTML in a GSP. In this case, you are using WML instead.

When you send out WML from a GSP, you must override the default MIME type from `text/html` to `text/vnd.wap.wml`, as in the first line of Listing 1. If you are serving up static WML, you can simply give the file a WML extension instead of a GSP extension. Most Web servers will then send back the correct MIME type without the need for the `response.setContentType` method call. In $TOMCAT_HOME/conf/web.xml, you'll find that the MIME mapping for WML files is already in place. If you are using Apache HTTPD, a similar default mapping for WML files is in the $APACHE_HOME/conf/mime.types file. Listing 2 shows Tomcat's MIME type mapping:

Listing 2. Setting the MIME type in Tomcat

```
<mime-mapping>
  <!-- WML Source -->
  <extension>wml</extension>
  <mime-type>text/vnd.wap.wml</mime-type>
</mime-mapping>
```

Looking back at Listing 1, the next thing you should notice is the `DOCTYPE`. Including the Document Type Definition (DTD) statement helps identify testwml.gsp as a WML document.

Notice that the document is not wrapped in the familiar `<html>` tags. It starts and ends with `<wml>`. The next thing you probably notice in Listing 1 is the lack of `<head>` and `<body>` sections. Each WML page is a `card`, with a unique `id` attribute and user-friendly `title` attribute.

It's quite common to download multiple pages/cards in a single file. Older cell phones had especially narrow data pipes, so this was a concession to the limitations of the devices and their networks. The more you download at once, the less the phone needs to transfer data to and from the server. You can view only one card at a time, so the rest of the cards are effectively prefetched. At that point, navigation happens purely on the client side.

The <p> tags should be familiar to HTML developers. The WML <anchor> tag is similar in spirit to the HTML <a> tag, if not identical in syntax (see Resources to learn more about WML).

Here is a clever WML trick. Because you're dealing with dedicated content for mobile phones, you can create a hyperlink that dials the phone when the user selects the link. In the example in Listing 3, the phone number 303-555-1212 will be dialed:

Listing 3. A dialable link in WML

```
<do type="accept">
 <go href="wtai://wp/mc;3035551212"/>
</do>
```

Notice that the protocol of the link isn't the familiar `http://` — it's `wtai://`, short for Wireless Telephony Applications Interface.

A WML emulator

To render this page on your PC, you need a WAP emulator (see Resources for links to all the emulators introduced in this article). Visit the dotMobi emulator, which is implemented as a Java applet. Type in the URL `davisworld.org/testwml.gsp`. (Notice that the http:// prefix is already supplied, to the left of the input box.) You should see something similar to Figure 1:

Figure 1. Emulated WAP page

Notice that the dotMobi emulator has two different skins that represent not only the look and feel of the different devices, but also their capabilities. If you are interested in emulating a specific device, the hardware manufacturers generally offer a developer Web site from which you can download and install the emulator you need.

Dynamic WML from a GSP

The first WML example was static code. Listing 4 is an example using the familiar `<g:each>` and `<g:if>` tags:

Listing 4. Mixing GSP with WML

```
<% response.setContentType("text/vnd.wap.wml") %>
<%
def flightList = []
flightList << [iata1:"DEN", iata2:"ORD"]
flightList << [iata1:"ORD", iata2:"DEN"]
%>

<?xml version="1.0"?>
<!DOCTYPE wml PUBLIC "-//PHONE.COM//DTD WML 1.1//EN"
   "http://www.phone.com/dtd/wml11.dtd" >

<wml>
  <g:each in="${flightList}" var="${flight}" status="i">
    <card id="f${i}" title="Flight ${i}">
      <p mode="wrap">From: ${flight.iata1}</p>
      <p mode="wrap">To: ${flight.iata2}</p>
      <g:if test="${flightList.size() > i+1}">
        <p>
          <anchor>Next<go href="#f${i}"/></anchor>
        </p>
      </g:if>
    </card>
  </g:each>
</wml>
```

Notice that I just mock up some flight data in a HashMap rather than setting up the full MVC infrastructure. The important thing is to see that you can mix GSP tags in with WML, just as I did with JavaScript in "Changing the view with Groovy Server Pages." (You can see a live example of this page at http://davisworld.org/testwml2.gsp.)

WML 1.x: The end of an era

Despite frequent claims that WML is dead, mobile phones are still in circulation that support only WML. Make no mistake, though — WML 1.x is well on its way toward obsolescence. More and more modern cell phones are eschewing the "separate but equal" WML platform for a true Web browser. As the next sections demonstrate, creating a mobile-friendly Web site for WML 2.x devices or iPhones involves tweaking your existing HTML a bit instead of translating it into an entirely different markup language.

Using WML 2.X (or XHTML-MP) with Grails

When it comes to WML 2.x, *WML* is more like a brand name than the name of a truly separate markup language (which WML 1.x is). In reality, WML 2.x is simply an XHTML dialect: specifically, XHTML-MP.

The most stringent requirement of XHTML-MP is that you create well-formed XML. This means that you must close every container tag properly (`<p></p>`, `<li></li>`), use quotes around your attributes (`<a href="http://somewhere.com">`), and use only lowercase element names (`<h1>` instead of `<H1>`).

XHTML-MP is a superset of XHTML-Basic. You'll probably find that with just a little bit of adjustment, your Web site can conform to XHTML-Basic. It can't use nested tables or frames. The only image formats supported are gif and png. Other best practices (such as specifying image sizes and alternate text) are a requirement for XHTML-Basic. Many, if not most, of the familiar HTML tags are available to you. See Resources for links to lists of the tags allowed in XHTML-Basic and XHTML-MP.

It probably goes without saying that optimizing your Web site for a smaller screen should involve sending back less data with each request. Web pages (including the HTML, CSS, and images) should be less than 20KB. You should aggressively cache files using `Expires` or `Cache-Control` headers. You should break long Web pages into two or three pages when you serve up mobile content. http://m.cnn.com does a nice job of breaking articles up into three- to four-page portions but also provides an "entire article" link to override this if you don't mind the additional overhead.

As with WML 1.x, you should include the proper DTD at the start of the file. You should also modify the `<html>` tag to include the `xmlns` attribute, as shown in Listing 5:

Listing 5. Starting an XHTML-MP file

```
<!DOCTYPE html PUBLIC "-//WAPFORUM//DTD XHTML Mobile 1.0//EN"
"http://www.wapforum.org/DTD/xhtml-mobile10.dtd">
<html xmlns="http://www.w3.org/1999/xhtml" xml:lang="en">
...
</html>
```

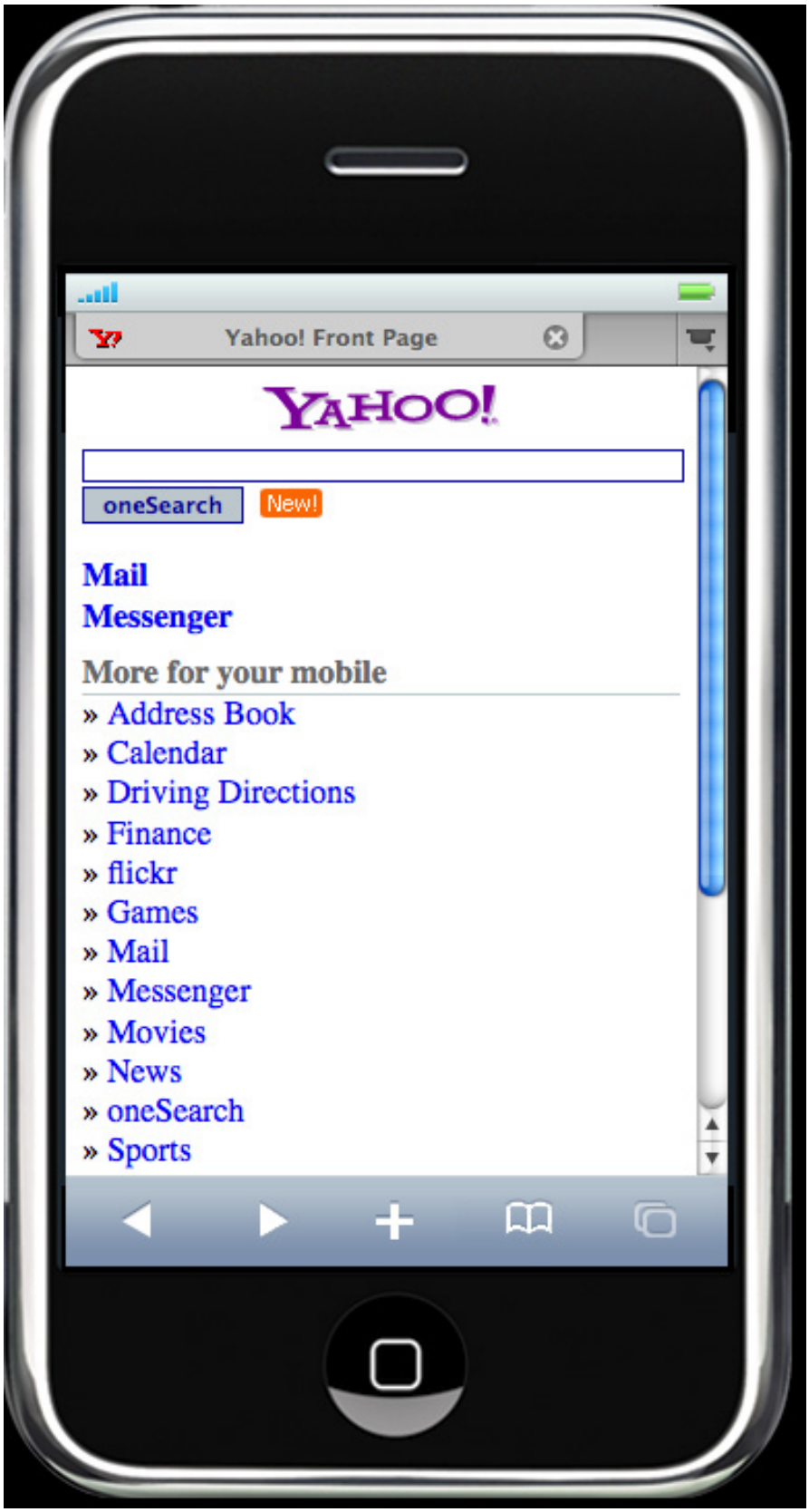
You should also send it with the proper MIME type of `application/vnd.wap.xhtml+xml`, although many mobile devices will also accept the more generic `application/xhtml+xml`. Using `application/xhtml+xml` also helps you debug the code in a standard desktop browser.

Viewing XHTML-MP

To view XHTML-MP, visit http://m.yahoo.com, for example. (It looks spartan in a Web browser, but it looks great on a mobile phone.) Select **View > Source**, and you'll see the XHTML-MP DTD at the top of the document.

To get a better idea of what the Web site looks like on an actual device, you'll need to find another emulator. For example, Sandip Chitale's blog offers a Firefox plug-in that looks an awful lot like an iPhone (see Resources). Be aware that this emulator is quite a bit larger than the actual device. It's really more of a pretty face than a true emulator, but it'll give you a rough approximation of how your Web site will look on the iPhone. (I'll point you toward some more rigorous validators in just a moment.) Figure 2 shows m.yahoo.com emulated on Chitale's emulator:

Figure 2. Viewing the Yahoo mobile Web site with an iPhone emulator



Validating XHTML-MP

Several online validators can help you ensure that you are sending out well-formed XHTML-MP. You can try either the W3C mobileOK Basic Checker or the ready.mobi testing tool (see Resources). Both are good, but the ready.mobi emulator gives quite a bit more information than the W3C's.

For example, Figure 3 shows what the W3C validator has to say about http://m.google.com:

Figure 3. W3C Validator report on Google's mobile Web site

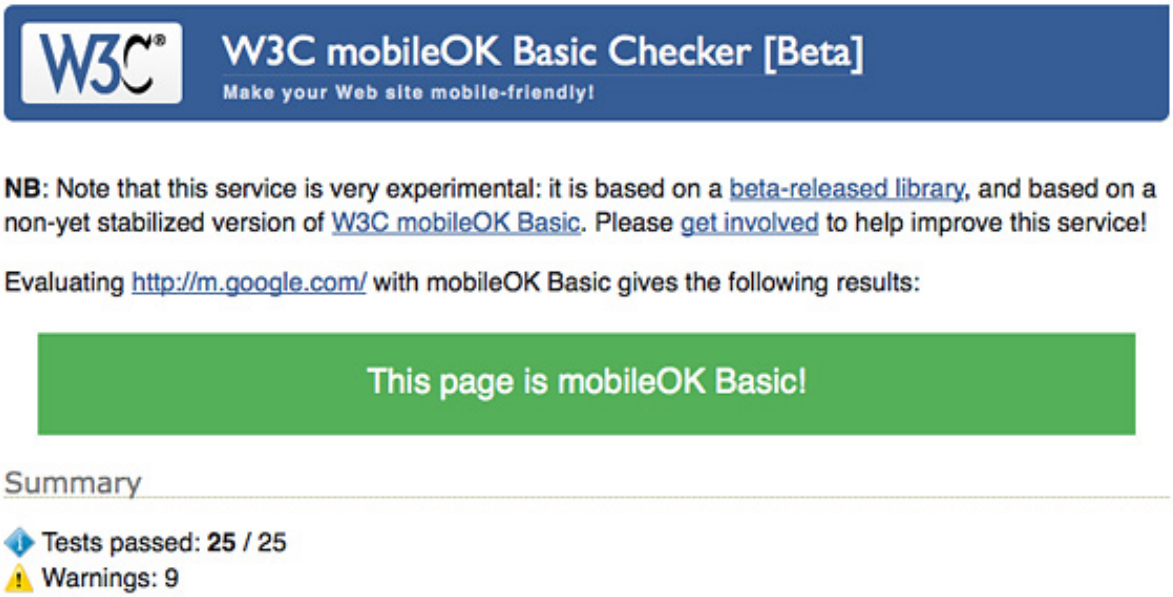
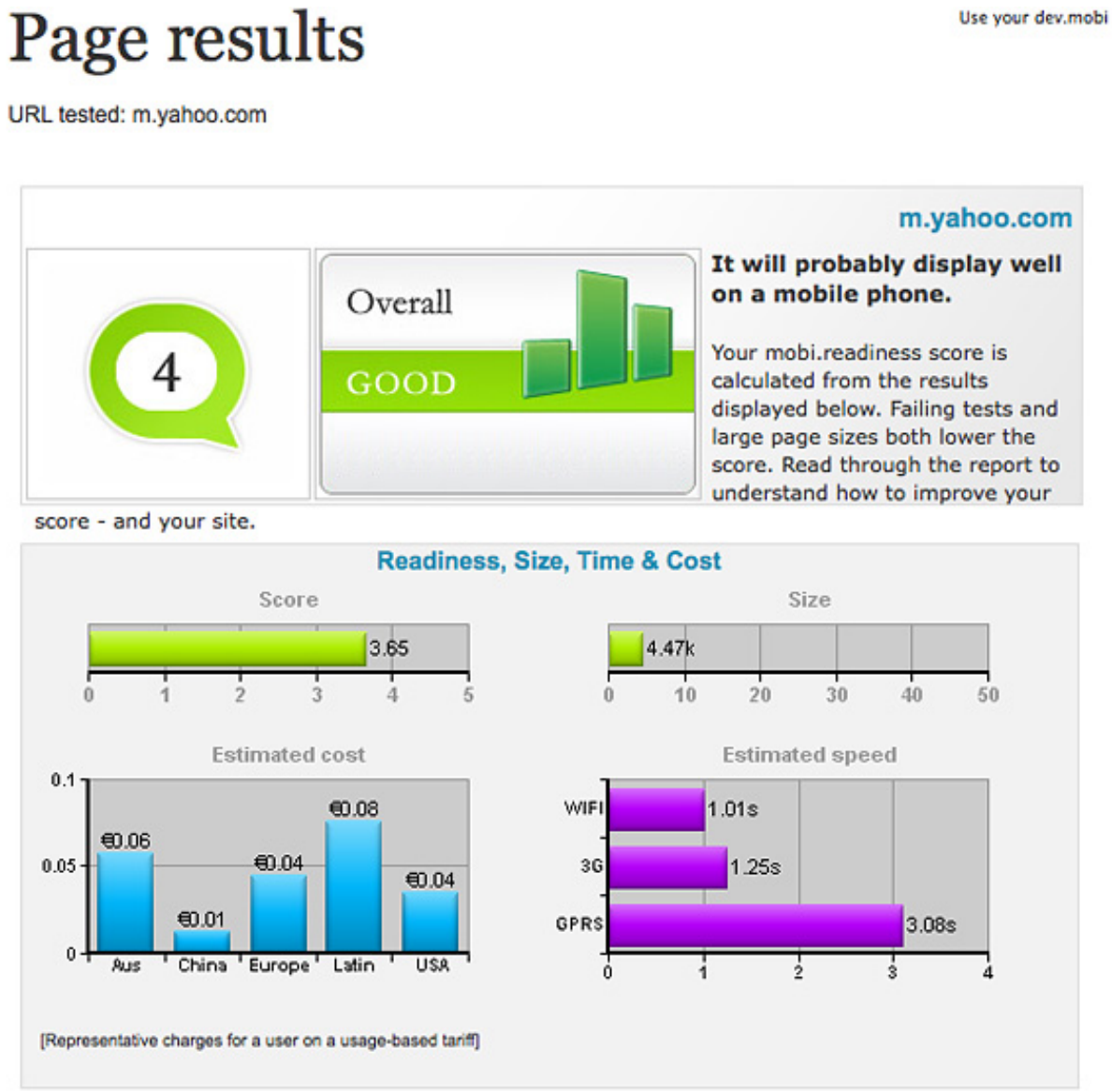


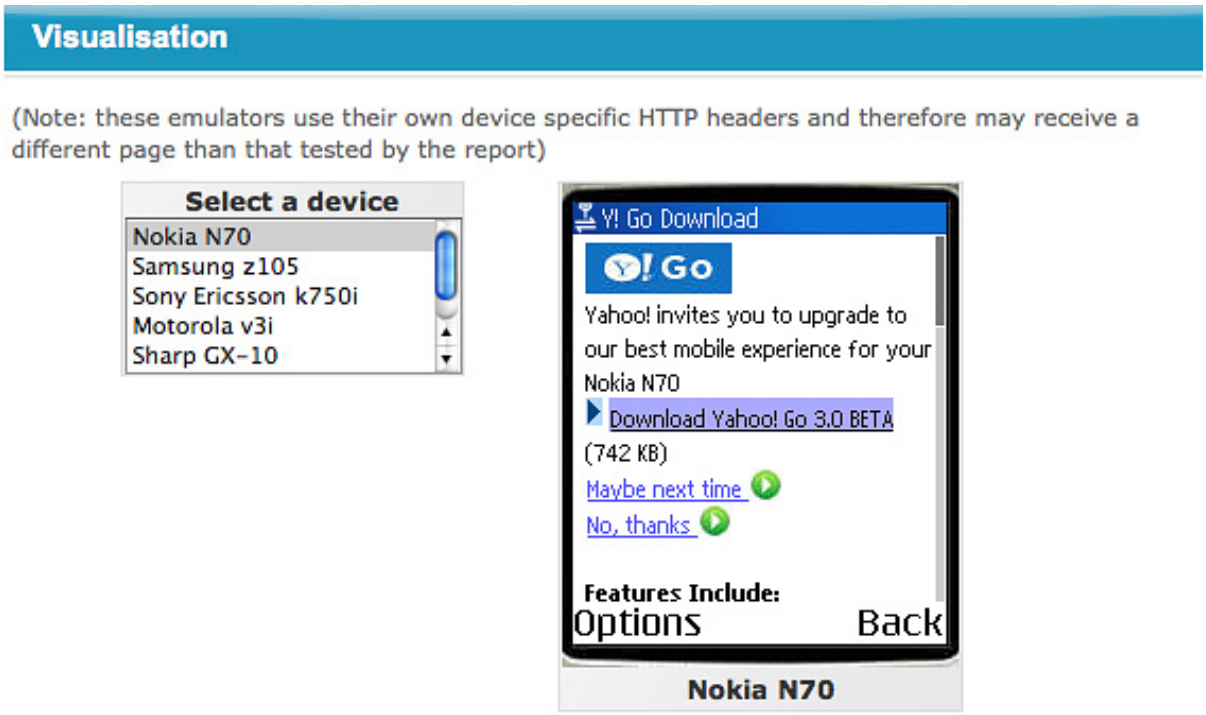
Figure 4 is the first part of the report from the ready.mobi tool for http://m.yahoo.com:

Figure 4. ready.mobi's report on Yahoo's mobile Web site



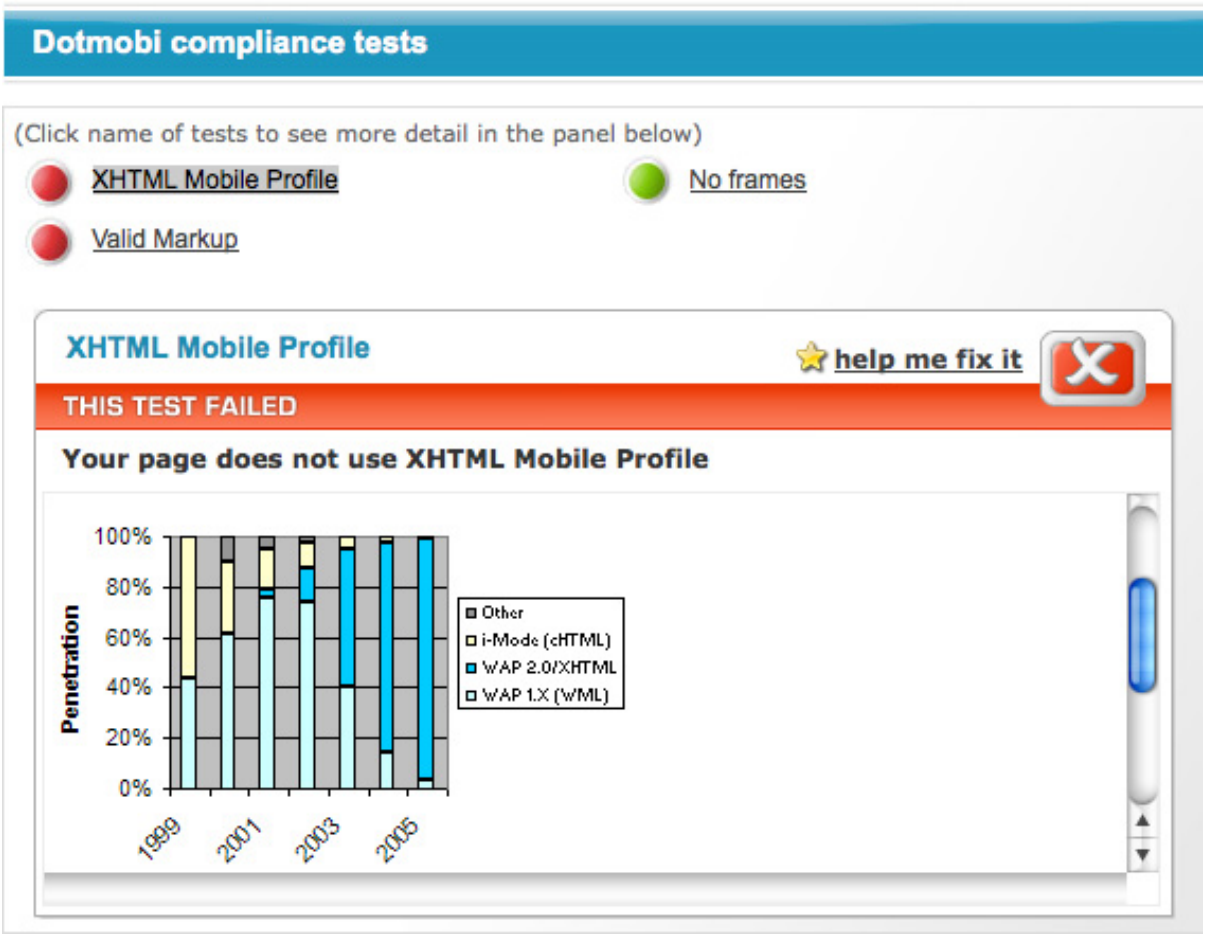
It gives Yahoo! a 4 out of 5. Scrolling down a bit, it offers a number of different visualizers so you can see what the page actually looks like. Figure 5 shows it looks like on the Nokia N70:

Figure 5. Viewing Yahoo's mobile Web site using an emulator



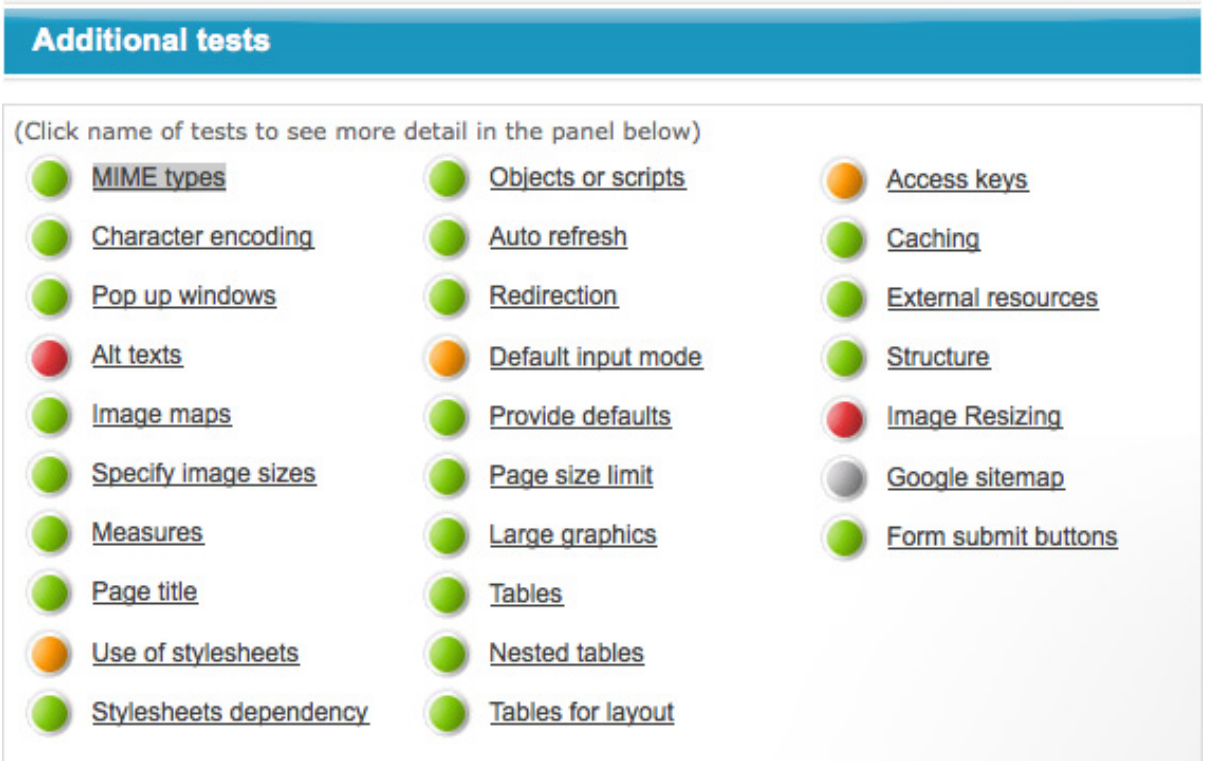
At the bottom of the page, the ready.mobi validator shows the results of a detailed set of tests, each marked pass (green), fail (red), or a warning (yellow). For example, even though http://m.yahoo.com seems to render well on the various devices, its XHTML isn't 100 percent compliant, as shown in Figure 6:

Figure 6. Viewing XHTML-MP compliance errors



Further down, as shown in Figure 7, you can see that Yahoo! missed some `alt` attributes on its images, and in some cases neglected to specify images sizes:

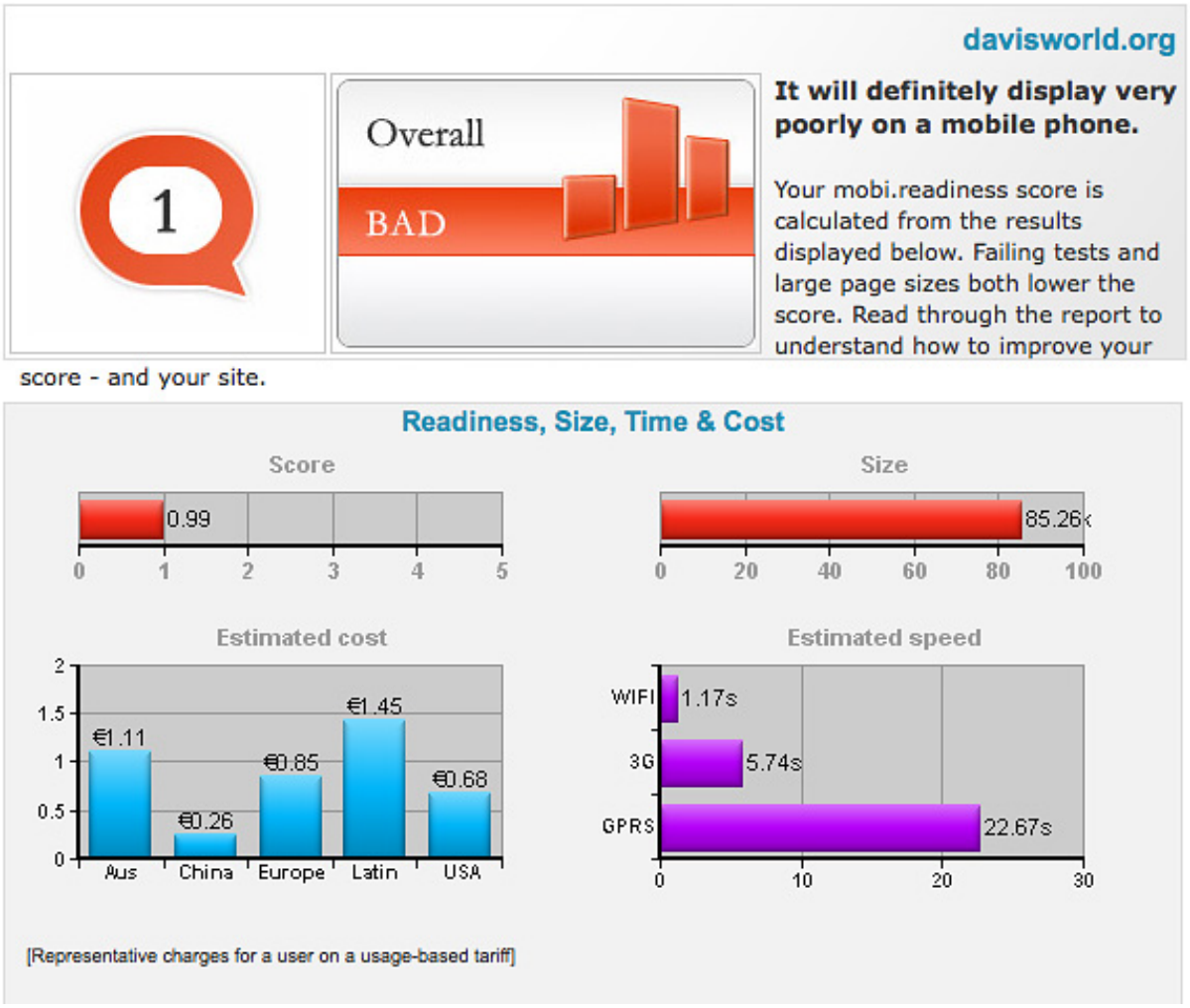
Figure 7. Viewing the exact errors



Grails and XHTML-MP

So, is Grails ready for the mobile Web out of the box? Figure 8 shows what the ready.mobi validator has to say about the trip planner application's original list page:

Figure 8. Grails is not XHTML-MP compliant out of the box



Okay, so some work needs to be done. To begin, create a `mlist` closure in grails-app/controllers/AirportController.groovy. It's not going to do anything different from the default list closure other than return 5 elements instead of 10. Creating a separate closure allows you to leave list.gsp intact for comparison, as shown in Listing 6:

Listing 6. A new closure in AirportController

```
def mlist = {
  if(!params.max) params.max = 5
  [airportList:Airport.list(params)]
}
```

Now copy grails-app/views/airport/list.gsp to mlist.gsp. (Later I'll give you some strategies for seamlessly redirecting mobile users to the correct content. This brute-force method will suffice for now.)

The validator complained about the Web site not returning XHTML-MP. Edit

mlist.gsp so that it includes the necessary DTD and `xmlns` attribute on the `<html>` tag. You should also disable the `meta` tag that sets the content type to `text/html`. One final step: copy the line that includes the CSS from grails-app/views/layout/main.gsp to this file. (SiteMesh — the templating library that Grails uses — is configured to decorate text/html files only by default.) Listing 7 shows mlist.gsp:

Listing 7. Changing a GSP page to XHTML-MP

```
<% response.setContentType("application/xhtml+xml")%>
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE html PUBLIC "-//WAPFORUM//DTD XHTML Mobile 1.0//EN"
        "http://www.wapforum.org/DTD/xhtml-mobile10.dtd">
<html xmlns="http://www.w3.org/1999/xhtml" xml:lang="en">
  <head>
    <!--meta http-equiv="Content-Type" content="text/html; charset=UTF-8"/-->
    <link rel="stylesheet" href="${createLinkTo(dir:'css',file:'main.css')}" />
    <meta name="layout" content="main" />
    <title>Airport List</title>
  </head>
  ...
</html>
```

While you're editing the file, you might want to simplify the table's layout as well. The validator complained about the `<thead>` and `<tbody>` tags, so you'll need to remove them. Because mobile phone screens have more space vertically than horizontally, the layout in Listing 8 will look better:

Listing 8. Simplifying the table

```
<table>
  <tr>
    <g:sortableColumn property="id" title="Id" />
    <g:sortableColumn property="name" title="Name" />
  </tr>
  <g:each in="${airportList}" status="i" var="airport">
  <tr class="${(i % 2) == 0 ? 'odd' : 'even'}">
    <td>
      <g:link action="show"
              id="${airport.id}">${airport.id?.encodeAsHTML()}
      </g:link>
    </td>
    <td>${airport.iata?.encodeAsHTML()}<br/>
        ${airport.name?.encodeAsHTML()}
    </td>
  </tr>
  </g:each>
</table>
```

Figure 9 shows what the new page looks like in the iPhone emulator:

Figure 9. List page customized for the iPhone

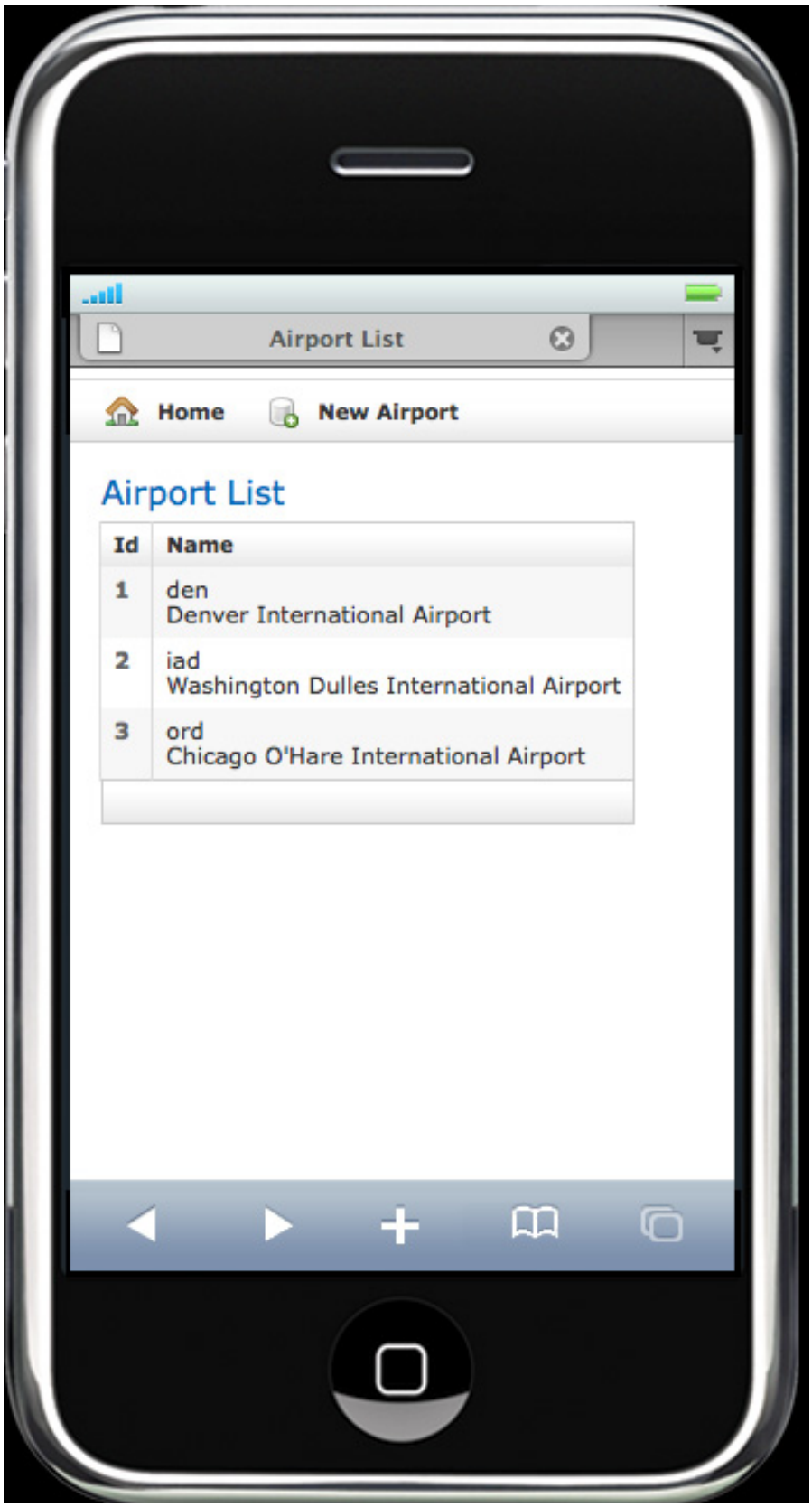
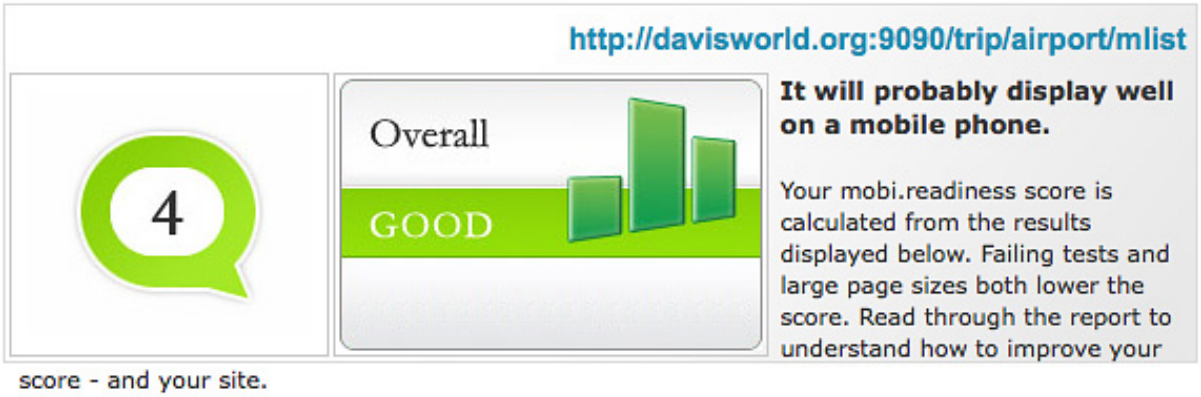


Figure 10 shows what the ready.mobi validator says now:

Figure 10. List page passing validation



Much better, isn't it? And the changes necessary to appease the validator were minimal. Recall from "Changing the view with Groovy Server Pages" that you can make these changes to the default templates by typing `grails install-templates`.

Developing pages for the iPhone

The iPhone is probably the easiest of the three types of devices to support. You don't need to do a single thing differently from normal Web page development. The Safari browser on the iPhone is based on the same codebase as the desktop browser, so what users see in one is essentially what they see in the other. You can, however, send down some rendering hints that affect the look and feel of the Web site only when it's viewed on an iPhone.

For example, an iPhone's screen dimensions are 320x480. Interestingly, the browser sets the default width of Web pages to 980 pixels. This makes the text barely readable when the phone is in landscape mode, and microscopic when it's in portrait mode. Never fear, though. With a simple `meta` tag that only the iPhone pays attention to, you can "right-size" the Web page: A `viewport` tag allows you to supply hints to the mobile Safari browser. The code in Listing 9 dramatically increases the readability of the Web page when it's viewed from an iPhone. (Unfortunately, the Firefox-based iPhone emulator doesn't pay attention to this `meta` tag. To see it in action, you need an actual iPhone.)

Listing 9. Setting the viewport for an iPhone

```
<meta name="viewport" content="initial-scale=1.0" />
```

The `inital-scale` ranges from 0 to 10 and supports fractional values. You can also pass in explicit `width` and `height` values (showin in Listing 10) that can range

up to 10,000 pixels:

Listing 10. Setting the width and height for the viewport

```
<meta name="viewport" content="width=600;height=400" />
```

Hyperlinks on the iPhone

The iPhone offers some specialized behavior when it comes to hyperlinks. If you use a `tel:` prefix instead of `http://`, as shown in Listing 11, clicking on the link dials a phone number:

Listing 11. Dialable links on the iPhone

```
<p>
telephone number:
<a href="tel:303-555-1212">303-555-1212</a>
</p>
```

If you use a traditional `mailto:` prefix, as shown in Listing 12, clicking on the link launches the mail application:

Listing 12. Mail links

```
<p>
mail:
<a href="mailto:scott@aboutgroovy.com">Scott Davis</a>
</p>
```

If you supply a link to Google Maps, as shown in Listing 13, clicking the link launches the native Google Maps application instead of being rendered in Safari:

Listing 13. Google Map links

```
<p>
local google maps:
<a href="http://maps.google.com/maps?q=denver+international+airport">DEN</a>
</p>
```

Passing in a start and end point, as shown in Listing 14, gives the user driving directions when they click the link:

Listing 14. Google Map driving directions

```
<p>
driving directions:
<a href="http://maps.google.com/maps?daddr=
```

```
    denver+airport&saddr=coors+field+denver,+co">Directions</a>
</p>
```

Strategies for the mobile Web

Now that you know what it takes to create content for the three basic types of devices, all you need to do is figure out how to serve it up on demand. Three basic strategies are at your disposal.

Create a separate, dedicated Web site for mobile content

As you saw earlier, the *m* strategy is the one many Web sites use. Google, Yahoo!, and CNN all have set up an *m* domain that's separate from the main site and serves up mobile content. If you don't want to mess around with the Domain Name System (DNS), you can create a URL like http://mysite.org/mobile that is a leaping-off point. You can also register a .mobi domain that is dedicated to mobile content.

Sniff the user agent

Every Web browser identifies itself to the server when it makes its request for data. You can use this information to serve up content that's customized for the device. (This is the technique that http://twitter.com uses.)

Visit http://davisworld.org/echo.gsp. Using a simple loop, the page in Listing 15 echoes back the interesting bits of the HTTP request:

Listing 15. Displaying the Request Headers

```
<h2>Request Headers</h2>
<table border="1">
  <tr>
    <th>Header</th>
    <th>Value</th>
  </tr>

  <g:each in="${request.headerNames}" var="${name}">
    <tr>
      <td>${name}</td>
      <td>${request.getHeader(name)}</td>
    </tr>
  </g:each>
</table>
```

As you can see in Figure 11, my Firefox browser gives plenty of hints about itself when I open http://davisworld.org/echo.gsp:

Figure 11. Viewing HTTP headers

Request Headers

Header	Value
host	davisworld.org
user-agent	Mozilla/5.0 (Macintosh; U; Intel Mac OS X 10.5; en-US; rv:1.9) Gecko/2008051202 Firefox/3.0
accept	text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8
accept-language	en-us,en;q=0.5
accept-encoding	gzip,deflate
accept-charset	ISO-8859-1,utf-8;q=0.7,*;q=0.7
Keep-Alive	300
connection	keep-alive
Max-Forwards	10

Based on the `user-agent` string shown in Figure 11, you can tell that the requester is running a Mac with an Intel CPU. You know the version of the OS (10.5), of the HTML rendering engine (Gecko), and the actual browser (Firefox). Listing 17 shows some other common user-agent strings:

Listing 17. Common user-agent strings

```
BlackBerry7520/4.0.0 Profile/MIDP-2.0 Configuration/CLDC-1.1
    UP.Browser/5.0.3.3 UP.Link/5.1.2.12

Mozilla/5.0 (iPhone; U; CPU like Mac OS X; en)
    AppleWebKit/420+ (KHTML, like Gecko) Version/3.0 Mobile/1A543a Safari/419.3

Mozilla/4.0 (compatible; MSIE 6.0; Windows NT 5.1; .NET CLR 1.1.4322)
```

By capturing the value of `request.getHeader("user-agent")`, you can serve up the appropriate content to the browser in question.

Send back what the browser accepts

The third strategy is to give the browser what it actually wants. Each request contains an `accept` value as well as a `user-agent` value. Firefox sends up this `accept` value:

```
text/html,application/xhtml+xml,application/xml;
```

This tells the server that Firefox prefers `text/html`. If the server has no `text/html` data, then it can send down `application/xhtml+xml`. If it doesn't have either of the two, the server walks the list sequentially until it finds a MIME type that it can return.

A WAP 1.x browser will ask for `text/vnd.wap.wml`, and a more modern cell phone will ask for `application/vnd.wap.xhtml+xml`. By paying attention, a

clever developer can send back the appropriate data.

Of course, these three strategies are not mutually exclusive. Use all of them to ensure that your Web site is ready for the 3.3 billion cell phone users worldwide.

Conclusion

Making your Grails application ready for cell phone use can range from nearly zero effort (for the iPhone), to minor tweaks (for XHTML-MP devices), to a full rewrite (for WML 1.x devices). With the collection of emulators and validators discussed in this article, you should be well on your way toward supporting the mobile Web.

In the next article, you will learn how to deal with legacy databases in Grails. You'll learn about the Mapping DSL, as well as using Hibernate annotations and HBM files. With these techniques, you'll coax Grails into using your existing table and field names, even if they don't conform to the Grails standard naming conventions. Until then, have fun mastering Grails.

Resources

Learn

- *Mastering Grails*: Read more in this series to gain a further understanding of Grails and all you can do with it.
- Grails: Visit the Grails Web site.
- Grails Framework Reference Documentation: The Grails bible.
- *Groovy Recipes: Greasing the Wheels of Java*: Learn more about Groovy and Grails in Scott Davis' latest book.
- *Practically Groovy*: This developerWorks series is dedicated to exploring the practical uses of Groovy and teaching you when and how to apply them successfully.
- Groovy: Learn more about Groovy at the project Web site.
- AboutGroovy.com: Keep up with the latest Groovy news and article links.
- Internet World Stats: Get the big picture on Internet use trends.
- "The dawn of a new mass media" (Colin Crawford, IDG Knowledge Hub, April 2008): Crawford's blog argues that "[w]e're moving from the PC World to Personal Communications World."
- WML and WAP: Read Wikipedia's entries on these mobile technologies.
- OMA technical section: The Open Media Alliance houses the official WML and WAP specifications.
- "Developing wireless Web applications" (Carol Jones, developerWorks, June 2000): Get a handle on WAP and WML basics.
- WAP/WML Tutorial: Learn more about WAP and WML.
- XHTML-Basic 1.1: The W3C's XHTML-Basic specification. Find a list of allowable tags here.
- XHTML Mobile Profile: Additional tags allowed by XHTML-MP.
- Mobile phone emulators:
  - dotMobi emulator
  - IPhone emulator for Firefox
  - W3C mobileOK Basic Checker
  - ready.mobi testing tool

- Apple Web Apps DevCenter: A detailed guide for designing content for the iPhone.
- Technology bookstore: Browse for books on these and other technical topics.
- developerWorks Java technology zone: Find hundreds of articles about every aspect of Java programming.

Get products and technologies

- Grails: Download the latest Grails release.
- Opera Mini: A freely available mobile browser.

Discuss

- Check out developerWorks blogs and get involved in the developerWorks community.

About the author

Scott Davis
Scott Davis is an internationally recognized author, speaker, and software developer. His books include *Groovy Recipes: Greasing the Wheels of Java*, *GIS for Web Developers: Adding Where to Your Application*, *The Google Maps API*, and *JBoss At Work*.

Trademarks

Java and all Java-based trademarks are trademarks of Sun Microsystems, Inc. in the United States, other countries, or both.
Microsoft, Windows, Windows NT, and the Windows logo are trademarks of Microsoft Corporation in the United States, other countries, or both.