

AJAX HACKS

Tips & Tools for Creating Responsive Web Sites



O'REILLY®

Bruce W. Perry

Ajax Hacks™

by Bruce Perry

Copyright © 2006 O'Reilly Media, Inc. All rights reserved.
Printed in the United States of America.

Published by O'Reilly Media, Inc., 1005 Gravenstein Highway North,
Sebastopol, CA 95472.

O'Reilly books may be purchased for educational, business, or sales promotional use. Online editions are also available for most titles (*safari.oreilly.com*). For more information, contact our corporate/institutional sales department: (800) 998-9938 or *corporate@oreilly.com*.

Editors: Simon St.Laurent

Production Editor: Mary Anne Weeks Mayo

Copyeditor: Rachel Wheeler

Indexer: John Bickelhaupt

Cover Designer: Hanna Dyer

Interior Designer: David Futato

Illustrators: Robert Romano, Jessamyn
Read, and Lesley Borash

Printing History:

March 2006: First Edition.

Nutshell Handbook, the Nutshell Handbook logo, and the O'Reilly logo are registered trademarks of O'Reilly Media, Inc. The *Hacks* series designations, *Baseball Hacks*, the image of an umpire's indicator, and related trade dress are trademarks of O'Reilly Media, Inc.

Many of the designations used by manufacturers and sellers to distinguish their products are claimed as trademarks. Where those designations appear in this book, and O'Reilly Media, Inc. was aware of a trademark claim, the designations have been printed in caps or initial caps.

While every precaution has been taken in the preparation of this book, the publisher and author assume no responsibility for errors or omissions, or for damages resulting from the use of the information contained herein.

Small print: The technologies discussed in this publication, the limitations on these technologies that technology and content owners seek to impose, and the laws actually limiting the use of these technologies are constantly changing. Thus, some of the hacks described in this publication may not work, may cause unintended harm to systems on which they are used, or may not be consistent with applicable user agreements. Your use of these hacks is at your own risk, and O'Reilly Media, Inc. disclaims responsibility for any damage or expense resulting from their use. In any event, you should take care that your use of these hacks does not violate any applicable laws, including copyright laws.



This book uses RepKover™, a durable and flexible lay-flat binding.

ISBN: 0-596-10169-4

[M]

[4/06]

This excerpt is protected by copyright law. It is your responsibility to obtain permissions necessary for any proposed use of this material. Please direct your inquiries to permissions@oreilly.com.

HACK
#43

Integrate DWR into Your Java Web Application

Design your Ajax application around a JavaScript framework bound to Java objects on the server.

The Direct Web Remoting code comes in the form of an archived or zipped Java Archive (JAR) file, *dwr.jar*. The download address is <http://www.getahead.ltd.uk/dwr/download.html>.



The top-level web page for this open source software is <http://www.getahead.ltd.uk/dwr/>. Check out the license details for more information while you are visiting this page.

To get started with DWR, you must first set it up in your server-side web application. Place the *dwr.jar* file in the */WEB-INF/lib* directory of your Java web application *on the server*, then restart or reload the application.



For those not familiar with Java web applications, they all have a top-level directory named *WEB-INF*. Inside *WEB-INF* are XML configuration files, the main one being *web.xml*. *WEB-INF* also contains a directory named *lib*, which encloses code libraries or JAR files that the application depends on, such as database drivers and helper classes. The *dwr.jar* file goes in this *lib* directory.

Configuring the Application

To get DWR going with your JavaScript, you have to declare in *web.xml* a Java servlet that DWR uses. Here is the chunk of code that you have to add to *web.xml*. If *web.xml* already includes registered servlets, nest this newly declared servlet in with the existing ones (the same goes for the servlet-mapping element):

```
<servlet>
  <servlet-name>dwr-invoker</servlet-name>
  <servlet-class>uk.ltd.getahead.dwr.DWRServlet</servlet-class>
  <init-param>
    <param-name>debug</param-name>
    <param-value>>true</param-value>
  </init-param>
</servlet>

<servlet-mapping>
  <servlet-name>dwr-invoker</servlet-name>
  <url-pattern>/dwr/*</url-pattern>
</servlet-mapping>
```



You may have to restart the Java web application for the servlet container to create a new instance of this DWR-related servlet.

You also have to create a simple XML file declaring the Java classes that you want to use from your client-side JavaScript code. Don't worry, I'll show you how to use the JavaScript objects that are bound to Java classes shortly! The file is named *dwr.xml*. Place this XML file in */WEB-INF/*:

```
<dwr>
  <allow>
    <create creator="new" javascript="JsDate">
      <param name="class" value="java.util.Date"/>
    </create>
    <create creator="new" javascript="JsBikeBean">
      <param name="class" value="com.parkerriver.BikeBean"/>
    </create>
  </allow>
</dwr>
```

This XML states that the client-side JavaScript can use two Java classes remotely. The JavaScript objects that bind the client-side code remotely to the Java classes are named *JsDate* and *JsBikeBean*. As part of the server-side preparations, you must have already developed the Java class *com.parkerriver.BikeBean* and installed it in your application. *java.util.Date* is part of the Java software development kit; it's not your own custom class. *Date* is already available as part of the Java virtual machine your server component is using.



The *BikeBean* class file is typically stored in */WEB-INF/classes*, as in */WEB-INF/classes/com/parkerriver/BikeBean.class*.

This XML file binds the two JavaScript names to the *Date* and *BikeBean* objects, so that these objects are available to use in your client-side JavaScript. This means that JavaScript code can call all the public methods of these Java objects. But how is the JavaScript in the local web page connected to the remote Java instances running on the server? Figure 5-1 shows in general terms the path a JavaScript method call takes in DWR's form of web remoting.

The web page that will use DWR contains these script tags, which connect the JavaScript code via the DWR servlet to the server code:

```
<script type="text/javascript" src=
  "[name of web app]/dwr/interface/JsBean.js">
</script>
<script type="text/javascript" src=
  "[name of web app]/dwr/interface/JsDate.js">
</script>
```

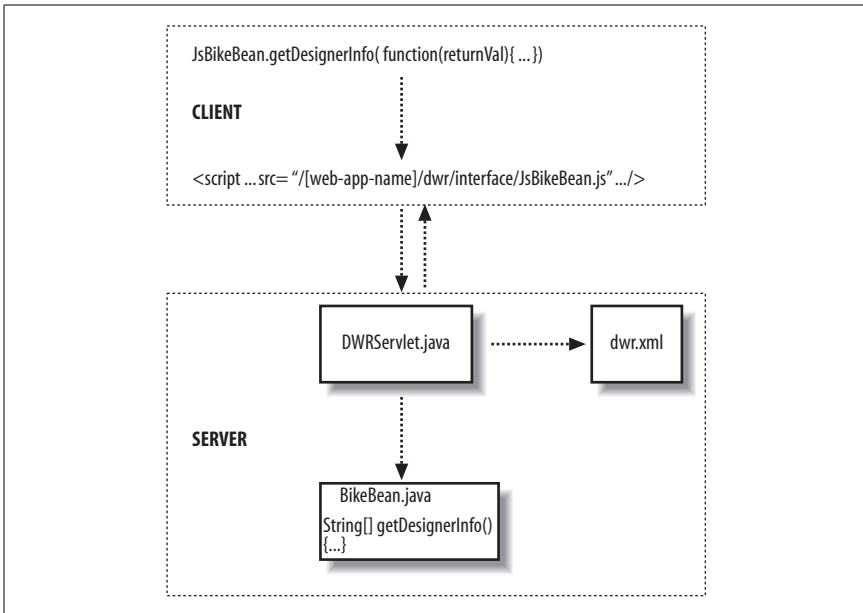


Figure 5-1. Calling a Java method remotely

```
<script type="text/javascript" src=
  "/[name of web app]/dwr/engine.js"></script>
<script type="text/javascript" src=
  "/[name of web app]/dwr/util.js"></script>
```

Think back to the simple XML file that we just added to the web application. The first two `script` tags reference the JavaScript names we bound to the Java classes that we want to remote: `JsBikeBean` and `JsDate`. The XML file configured certain Java classes to be used with these names in JavaScript code. Remember the `dwr.jar` file that we installed in the web application? It contains two JavaScript libraries, `engine.js` and `util.js`. The first of these files is required to use DWR; the second is optional and contains a bunch of DWR functions that the client-side code can use.

The URL that the `script` tag uses, such as `/parkerriver/dwr/interface/JsBean.js`, connects to the special DWR servlet that we enabled. The servlet in turn makes available to our code the public methods of the Java classes that we configured in XML. The next few hacks will use these classes and functions.