

Wicket Tutorial

Tue, 2007-02-06 19:55 in [Frameworks](#) [Justin Spradlin's blog](#) 47827 reads

It almost makes me shudder to think about how many people are going to be gunning for that little X in the top right corner of their browser window once they realize this post is about yet another web framework. BUT WAIT! This one is different, I swear. That's what they all say right? But it's the truth. Wicket is different, and if you spend a little time with me, I'll show you how.

What Makes Wicket So Different?

There are a few key distinctions that allow Wicket to really stand out from its peers:

Minimal XML Configuration

How many times have you had to trudge through bloated JSF faces-config.xml files to setup backing beans and URL navigation rules for your web application? What about those Spring Framework applicationContext.xml and <application-name>-servlet.xml files? Gross.

Wicket makes it simple. The only XML file you deal with is the web.xml file that is required by the servlet specification. Everything else is setup right in your Java code.

Extensive Support for Managing Server Side State

Wicket allows you to associate Plain Old Java Objects (POJOs) with its many components and will manage the state of these objects automatically. This saves a ton of development time because you don't have to worry about explicitly retrieving values from a form and setting them on some model object. This also assists greatly in the validation of user input.

Non-intrusive HTML Syntax

Navigation

[Recent posts](#)

Topic

[Business Logic](#)

[Databases](#)

[General](#)

[Java](#)

[Ant](#)

[Eclipse](#)

[Frameworks](#)

[Oracle ADF UIX](#)

[JBoss](#)

[JDK](#)

[Oracle ADF UIX](#)

[Oracle ADF UIX](#)

[Project Management](#)

[Security](#)

[Servers](#)

[Software Development](#)

[Technical Documentation](#)

[Web Technologies](#)

[XML](#)

User Login

Syndicate 

With Wicket, no special HTML markup is required. Wicket components are associated with HTML tags by using an XHTML standards compliant Wicket namespace. HTML tags are marked with a simple and unobtrusive “wicket:id” tag. The greatest benefit of this lack of special markup is that HTML can be rendered perfectly in HTML editing programs and browsers. This allows for a true separation of concerns between application developers and web designers.

An Example: The Platinum Address Book¹

Alright, enough chit chat, let’s look at some code. In the following example, I develop a simple address book application and highlight some of Wicket’s features. You can find the source for this example in the “Resources” section at the bottom of this page.

The Setup

You can download Wicket [here](#). All you have to do to get things up and running is place the wicket-1.2.4.jar file into the WEB-INF/lib directory of your web application.

The web.xml file is very simple and basically directs all requests to our “AddressBookApplication.”

Listing 1. web.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE web-app
    PUBLIC "-//Sun Microsystems, Inc.//DTD Web Application 2.3//EN"
    "http://java.sun.com/dtd/web-app_2_3.dtd">

<web-app>
    <display-name>Platinum Address Book</display-name>
    <servlet>
        <servlet-name>AddressBookApplication</servlet-name>
        <servlet-class>wicket.protocol.http.WicketServlet</servlet-class>

        <init-param>
            <param-name>applicationClassName</param-name>
            <param-value>
com.platinumsolutions.wicket.addressbook.AddressBookApplication
            </param-value>
        </init-param>

        <load-on-startup>1</load-on-startup>
    </servlet>
    <servlet-mapping>
        <servlet-name>AddressBookApplication</servlet-name>
        <url-pattern>/*</url-pattern>
    </servlet-mapping>
```

```
</web-app>
```

All Wicket web applications must have an implementation of the “WebApplication” class. This class is used to establish initial settings for our web application.

The Application

List 2. AddressBookApplication.java

```
public class AddressBookApplication extends WebApplication {

    public Class getHomePage() {
        return HomePage.class;
    }

    /*
     * This init method tells wicket to allow all web pages in the same
     * package as "HomePage" to be accessible by a "friendly" URL.
     *
     * i.e. http://<server>/<application>/pages/HomePage
     */
    public void init(){
        mount("/pages",
            PackageName.forPackage(HomePage.class.getPackage()));
    }

    public ISessionFactory getSessionFactory() {
        return new ISessionFactory() {
            public Session newSession() {
                return new AddressBookApplicationSession(
                    AddressBookApplication.this);
            }
        };
    }
}
```

The `getHomePage` method will direct initial request to our application to the `HomePage` java object and HTML page.

By default URLs in Wicket are not pretty and creating links between pages is a bit cumbersome. By using the `init()` method we can mount our HTML pages and provide user friendly URLs for our application.

The `getSessionFactory()` method simply associates an `AddressBookApplicationSession` object with our application. This object is created

specifically for our application and will store our model. The AddressBookApplicationSession object extends Wicket “WebSession” class.

Listing 3. AddressBookApplicationSession.java

```
public class AddressBookApplicationSession extends WebSession {
    private static final long serialVersionUID = 1L;
    private ContactDAO contactDAO;

    public AddressBookApplicationSession(WebApplication application) {
        super(application);
        contactDAO = new ContactDAOImpl();
    }

    public ContactDAO getContactDAO() {
        return contactDAO;
    }
}
```

The Model

The model for this application is essentially just a list of “Contact” objects. Contact objects are simple Java Beans that contain attributes relating to a person's name, address, phone number, etc. This class also provides accessor and mutator (getter/setter) methods for each attribute.

Listing 4. Contact.java (partial)

```
public class Contact implements Serializable {

    private static final long serialVersionUID = 1L;

    private String id;
    private String name;
    private String address;
    private String city;
    private String state;
    private String zipCode;
    private String phoneNumber;
    private String relationship;

    ...
}
```

The Web Pages

All Wicket web pages must extend the “WebPage” class. I also wanted to share my data access object (DAO) among my web pages so I created an additional

base page called "BaseWebPage."

Listing 5. BaseWebPage.java

```
public class BaseWebPage extends WebPage {

    private static final long serialVersionUID = 1L;

    protected ContactDAO contactDAO = ((AddressBookApplicationSession)
        getSession()).getContactDAO();

}
```

Finally, it's time to see the true power of Wicket. On our application's home page, we want to display a list of all of the contacts in our address book. In order to accomplish this goal, we use Wicket's ListView container to hold our model (i.e. list of Contact objects) and define the method populateItem(). This method will essentially be used to iterate through our list of contacts and assign our individual Contact object's attributes to ListItem objects. In turn, these ListItem objects will store the individual components that will make up our webpage.

To add a component to a webpage, you simply need to use the add() method found in the "WebPage" base class.

Listing 6. HomePage.java

```
public class HomePage extends BaseWebPage {

    public HomePage() {

        /*
         * This List View component will iterate through all of the Contact
         * elements we are currently storing in the contactDAO. These values are
         * added to the HomePage and will be displayed to the user.
         */
        add(new ListView("addressList", contactDAO.findAll()) {
            public void populateItem(final ListItem listItem) {
                final Contact contact =
                    (Contact)listItem.getModelObject();
                listItem.add(new Label("name",
                    contact.getName()));
                listItem.add(new Label("address",
                    contact.getAddress()));
                listItem.add(new Label("city",
                    contact.getCity()));
                listItem.add(new Label("state",
                    contact.getState()));
                listItem.add(new Label("zipCode",
                    contact.getZipCode()));
                listItem.add(new Label("phoneNumber",
```

```

        contact.getPhoneNumber());
    listItem.add(new Label("relationship",
        contact.getRelationship()));
    listItem.add(new Link("updateLink"){
        public void onClick(){
            PageParameters pp = new PageParameters();
            pp.add("contactId", contact.getId());
            setResponsePage(AddContactPage.class, pp);
        }
    });
}
});
}
}
}

```

Notice that we assigned an “addressList” identifier to the ListView components from the previous example. Also, notice that we assigned “name”, “address”, etc. to the ListItem components that we populated. These are the identifiers that must match exactly with our HTML wicket:id values.

Listing 7. HomePage.html (partial)

```

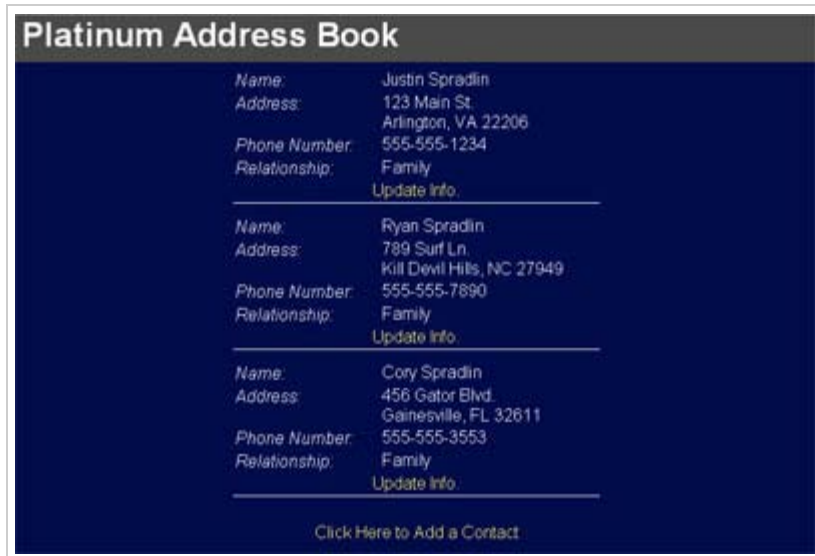
<table wicket:id="addressList">
    <tr>
        <td class="label">Name: </td><td wicket:id="name">[Name]</td>
    </tr>
    <tr>
        <td class="label">Address: </td>
        <td>
            <span wicket:id="address"></span><br />
            <span wicket:id="city"></span>, <span wicket:id="state">
            </span>&nbsp;<span wicket:id="zipCode"></span>
        </td>
    </tr>
    <tr>
        <td class="label">Phone Number: </td>
        <td wicket:id="phoneNumber">[Phone Number]</td>
    </tr>
    <tr>
        <td class="label">Relationship: </td>
        <td wicket:id="relationship">[Relationship]</td>
    </tr>
    <tr>
        <td colspan="2" class="center">
            <a wicket:id="updateLink" href="#">Update Info.</a></td>
        </tr>

```

</table>

So, what did we just accomplish? When combined, the previous HTML and Java object will produce a list of all the contacts we have stored in our contact list. You can think of this as being very similar to using a JSTL forEach loop.

Figure 1. HomePage Output.



Platinum Address Book

Name:	Justin Spradin
Address:	123 Main St. Arlington, VA 22206
Phone Number:	555-555-1234
Relationship:	Family
Update Info	
Name:	Ryan Spradin
Address:	789 Surf Ln. Kill Devil Hills, NC 27949
Phone Number:	555-555-7890
Relationship:	Family
Update Info	
Name:	Cory Spradin
Address:	456 Gator Blvd. Gainesville, FL 32611
Phone Number:	555-555-3553
Relationship:	Family
Update Info	

[Click Here to Add a Contact](#)

Now let's take a look at what it takes to add a contact to our contacts list. In order to gather input from the user we will need an HTML form. To create this form we must first define it in Java code.

Forms in Wicket extend the base "Form" class and must have a constructor that associates them with a particular form id (i.e. a String identifier that will match the wicket:id for the form tag in HTML). It is also very useful to associate a particular model object (Contact object in our case) to a form. If we do this, Wicket will take care of setting our model's attributes based on the form input entered by the user.

Individual form components (TextField, RadioChoice, etc.) must also be defined and added to our form. Each of these components will be assigned an id which must match up with its wicket:id counter part in our HTML file.

Listing 8. AddContactPage.java

```
public class AddContactPage extends BaseWebPage {

    String contactId;

    public AddContactPage(PageParameters params) {

        Contact lookupContact = null;

        CompoundPropertyModel contact =

            new CompoundPropertyModel(new Contact());

        FeedbackPanel feedback = new FeedbackPanel("feedback");
```

```

// These lists could come from a DAO, but I just hard coded
//them for this example.
List states = Arrays.asList(new String[] { "AL", "AK", "AZ",
    "AR", "CA", "CO", "CT", "DE", "DC", "FL", "GA", "HI",
    "ID", "IL", "IN", "IA", "KS", "KY", "LA", "ME", "MD",
    "MA", "MI", "MN", "MS", "MO", "MT", "NE", "NV", "NH",
    "NJ", "NM", "NY", "NC", "ND", "OH", "OK", "OR", "PA",
    "RI", "SC", "SD", "TN", "TX", "UT", "VT", "VA", "WA",
    "WV", "WI", "WY" });

List relationships = Arrays.asList(new String[] { "Family",
    "Friend", "Coworker" });

//Retrieve the contactId from the request parameters.
contactId = params.getString("contactId");

if(contactId != null){
    lookupContact = contactDAO.findById(contactId);

    if(lookupContact != null){
        contact = new CompoundPropertyModel(
            new Contact(lookupContact));
    }
}

//This is where we add the form elements to the page.
AddContactForm form = new AddContactForm("contactAddForm",
    contact);

TextField name = new TextField("name");
TextField address = new TextField("address");
TextField city = new TextField("city");
DropDownChoice state = new DropDownChoice("state", states);
TextField zipCode = new TextField("zipCode");
TextField phoneNumber = new TextField("phoneNumber");
RadioChoice relationship = new RadioChoice("relationship",
    relationships);

//This ensures that all of our form elements must be populated
//before they are submit.
name.setRequired(true);
address.setRequired(true);
city.setRequired(true);
state.setRequired(true);
zipCode.setRequired(true);

```



```

        phoneNumber.setRequired(true);
        relationship.setRequired(true);

        form.add(name);
        form.add(address);
        form.add(city);
        form.add(state);
        form.add(zipCode);
        form.add(phoneNumber);
        form.add(relationship);

        add(feedback);
        add(form);
    }

```

```

class AddContactForm extends Form {

```

```

    public AddContactForm(String id, IModel model) {
        super(id, model);
    }

```

```

    public void onSubmit() {
        if (isPhoneNumberValid()) {

            if(contactId != null){
                contactDAO.update(contactId,
                    (Contact) getModelObject());
            } else {
                Contact newContact = (Contact) getModelObject();
                newContact.setId(""+contactDAO.findAll().size());
                contactDAO.add(newContact);
            }

            setResponsePage(HomePage.class);
        }
    }

```

```

/**

```

```

 * Determines if the special case phone number is in the correct
 * format. The user will receive an error message if the phone
 * number is not in the correct format.

```

```

 *

```

```

 * @return

```

```

 */

```

```

private boolean isPhoneNumberValid() {
    Contact contact = (Contact) getModelObject();
    String phoneNumber = contact.getPhoneNumber();

    Pattern pattern = Pattern.compile("\\d{3}-\\d{3}-\\d{4}");

    if (!pattern.matcher((String) phoneNumber).matches()) {
        error("Phone Number must be in the form XXX-XXX-XXXX");
        return false;
    }

    return true;
}
}
}

```

Notice that for each of the form elements we define in the AddContactPage object we also call the method `setRequired(true)`. This will automatically require that each element be filled out in the form before it can be submitted. This greatly reduces the code we need to write for validation. However, if there is custom validation that needs to be addressed we can simply pull the model from the session using the `getModelObject()` method and test for specific validation requirements (Note the `isPhoneNumberValid()` method).

Listing 9. AddContactPage.html (partial)

```

<table class="addContact">
    <tr>
        <td class="label">Name: </td>
        <td><input wicket:id="name" type="text" /></td>
    </tr>
    <tr>
        <td class="label">Address: </td>
        <td><input wicket:id="address" type="text" /></td>
    </tr>
    <tr>
        <td class="label">City: </td>
        <td><input wicket:id="city" type="text" /></td>
    </tr>
    <tr>
        <td class="label">State: </td>
        <td><select wicket:id="state" class="selectMenu"></select></td>
    </tr>
    <tr>
        <td class="label">ZIP Code: </td>

```

```

        <td><input wicket:id="zipCode" type="text" /></td>
    </tr>

    <tr>
        <td class="label">Phone Number: </td>
        <td><input wicket:id="phoneNumber" type="text" /></td>
    </tr>
    <tr>
        <td class="label">Relationship: </td>
        <td><span wicket:id="relationship"></span></td>
    </tr>
    <tr>
        <td colspan="2" class="center">
            <br />
            <input type="submit" value="Add/Update" />
            <input type="button" value="Cancel"
onClick="location.href='https://gate.platinumsolutions.com/addressbook/pages/HomePage' "
/>
        </td>
    </tr>
</table>

```

Figure 2. Add Contact Form with Validation Errors

Platinum Address Book

Phone Number must be in the form XXX-XXX-XXXX

Name: Thomas
Address: Spradlin
City: Centonment
State: FL
ZIP Code: 32632
Phone Number: 123-234-dddd
Relationship: ☒ Family ☐ Friend ☐ Coworker

Add/Update Cancel

Once our form passes validation it is submitted. We are then returned to the HomePage which now displays our updated list.

¹I apologize about the strange formatting of the Java code. I had to do that so it would fit in the browser window. The example code included with this tutorial is formatted correctly.

Thoughts

Pros

- Since applications are built entirely out of Java objects they are easy to test using mock objects and a framework such as JUnit.
- An additional benefit of having applications built entirely of Java objects is that they are very IDE friendly. Eclipse and other IDEs are built to work very well with Java files, but the same does not necessarily hold true for XML configuration files.
- Wicket is easy to use. If you know Java, figuring out Wicket will not be much of an issue. I think this framework will find a lot of popularity among developers with strong Java skills and limited web application experience.
- There is no need to learn JPS, JSTL, or any other front end language. This could have a great impact on development time and maintenance efforts.

Cons

- For every web page that you create you must create a corresponding java object. These files must have the same name and must reside in the same location. That's right, you will have HTML files living side by side with you java classes in their respective packages. There is probably a work around for this, but this is the default behavior and I find it a bit quirky.
- While Wicket is very easy to use, it's not necessarily easy to learn. Strong documentation and solid examples are a bit hard to come by.
- Wicket relies very heavily on a web application's session. This could potentially tax the servers to a great extent. The Wicket team argues that development and maintenance cost are expensive while server resource costs are cheap in comparison. While this is probably true, it still gives me a bad feeling in my stomach.

Resources

The Wicket Website: <http://wicket.sourceforge.net/index.html>

Gurumurthy, Karthik. Pro Wicket. New York: Apress, 2006.

Installation Instructions

[Platinum Address Book.zip](#)

- Download the zip file to your computer and extract it.
- Use ant to run the build file. It will create a war file: <extract-directory>/build/distribution/addressbook.war
- Put the war file in the web application deploy directory of your favorite app server and start it up.
- Access the application at:
<http://<servername>:<port>/addressbook>

Comments

kush (not verified)

Wed, 1969-12-31 19:00

[reply](#)

This seems to quite a bit of work for the programmer. that's good I must say. Other frameworks are just reducing the programmer to a stenographer well that he'll always be but a dignified one. nywaz I just wanted to find out how will you connect wicket to hibernate can I have some text to read about it

sharief (not verified)

Wed, 1969-12-31 19:00

[reply](#)

Can you please modify your example for 1.4.2

soft (not verified)

Wed, 1969-12-31 19:00

[reply](#)

Thanks for the article. One note is that WebApplication already implements ISessionFactory, so just overriding newSession in your application

Trishul (not verified)

Wed, 1969-12-31 19:00

[reply](#)

Thanks for the great sample application. I tried it today with the Wicket 1.3.6 release, and I had to make the following changes for the sample to work:

web.xml

Wicket Address Book

configuration development

WebApplication

org.apache.wicket.protocol.http.WicketFilter

applicationClassName com.xxx.yyy.addressbook.AddressBookApplication

WebApplication

/*

AddressBookApplication.java

Replace the inner class for handling the custom session with the following:

```
public final Session newSession(Request request,
Response response){

return new AddressBookApplicationSession(AddressBookApplication.this,
request);
}
```

AddressBookApplicationSession.java

The call to super(application) is no longer available. Replace with super(request)
- that is the reason why the 'request' object had to be passed from
AddressBookApplication 's newSession() method

Regards,
TP

Sarah (not verified)

Wed, 1969-12-31 19:00 [reply](#)

I recently came across your blog and have been reading along. I thought I would leave my first comment. I don't know what to say except that I have enjoyed reading. Nice blog. I will keep visiting this blog very often.

Sarah

lives2sail (not verified)

Wed, 1969-12-31 19:00 [reply](#)

Nice tutorial thanks for creating it. I did have a problem with the "Click Here to Add Contact" link on the home page. I had to change the href in the HomePage.html from "/addressbook/pages/AddContactPage" to "pages/AddContactPage" to make it work.

Bob (not verified)

Wed, 1969-12-31 19:00 [reply](#)

Is anyone having the problem with the tutorial where if you add enough entries to the address book that it causes the list to be longer than the page, but you get no scrollbars to see the ones at the bottom?

I've tried this in Safari and Firefox, and it does the same in both. Could this be an effect of Wicket processing?

Other than that, nice tutorial - easy to follow.

collector (not verified)

Wed, 1969-12-31 19:00 reply

Easy to understand. Thanks for making it available online.

Rajesh (not verified)

Wed, 1969-12-31 19:00 reply

Hi friends, I am new to wicket and i want to learn about wicket, actually i have R&D topic "wicket framework and comparison with other component framework". But i am not getting enough data on net regarding this. Can u please help me in this matter. Do u have any books or other material on "Wicket framework"

Rajesh

Golfman (not verified)

Wed, 1969-12-31 19:00 reply

I've used Eelco's suggestion of providing your own
wicket.examples.customresourceloading resource loader to allow me to store my HTML in a separate directory to my class files and it works fine.

Virtually every time I've come across a "Oh damn, I wish I could do XYZ in wicket" it turns out that I actually can! The mailing group is very helpful.

devdanke (not verified)

Wed, 1969-12-31 19:00 reply

i was just adding login/logout to my wicket app and i didn't know how to piggy back on wicket's session to hold my app specific info. then i googled and saw how you did it. thanks for your nice tutorial!

wicket is an elegant web framework. i like it.

i'm not sure i understood what eelco said. does he mean that your session override could be done with the following code:

```
public Session newSession(){ return new AddressBookApplicationSession(this); }
```

instead of your:

```
public ISessionFactory getSessionFactory()
```

```
{
```

```
return new ISessionFactory()
```

```
{
```

```
public Session newSession()
```

```
{
```

```
return new AddressBookApplicationSession(AddressBookApplication.this);
```

```
}  
};  
}
```

Anonymous (not verified)

Wed, 1969-12-31 19:00 [reply](#)

```
//This is where we add the form elements to the page. AddContactForm form =  
new AddContactForm("contactAddForm", contact); TextField name = new  
TextField("name"); TextField address = new TextField("address"); TextField city  
= new TextField("city"); DropDownChoice state = new DropDownChoice("state",  
states); TextField zipCode = new TextField("zipCode"); TextField phoneNumber =  
new TextField("phoneNumber"); RadioChoice relationship = new  
RadioChoice("relationship", relationships);
```

```
//This ensures that all of our form elements must be populated //before they are  
submit.
```

```
name.setRequired(true); address.setRequired(true); city.setRequired(true);  
state.setRequired(true); zipCode.setRequired(true);  
phoneNumber.setRequired(true); relationship.setRequired(true);  
form.add(name); form.add(address); form.add(city); form.add(state);  
form.add(zipCode); form.add(phoneNumber); form.add(relationship);  
add(feedback); add(form); } class AddContactForm extends Form { public  
AddContactForm(String id, IModel model) { super(id, model); } public void  
onSubmit() { if (isPhoneNumberValid()) { if(contactId != null){  
contactDAO.update(contactId, (Contact) getModelObject()); } else { Contact  
newContact = (Contact) getModelObject();  
newContact.setId(""+contactDAO.findAll().size()); contactDAO.add(newContact);  
} setResponsePage(HomePage.class); } } /** * Determines if the special case  
phone number is in the correct * format. The user will receive and error message  
if the phone * number is not in the correct format. * * @return */ private  
boolean isPhoneNumberValid() { Contact contact = (Contact) getModelObject();  
String phoneNumber = contact.getPhoneNumber(); Pattern pattern =  
Pattern.compile("\\d{3}-\\d{3}-\\d{4}"); if (!pattern.matcher((String)  
phoneNumber).matches()) { error("Phone Number must be in the form XXX-  
XXX-XXXX"); return false; } return true; }
```

=====

The form code can be reduced:

```
super.createNewForm(id,contact);  
TextField name = super.createTextField("name"); TextField address =  
super.createTextField("address"); TextField city = super.createTextField("city");  
DropDownChoice state = super.createDropDownChoice("state", states); zipCode  
= super.createTextField("zipCode"); TextField phoneNumber =  
super.createTextField("phoneNumber"); RadioChoice relationship =  
super.createRadioChoice("relationship", relationships);
```



```
ValidationHookList list = super.createDefaultValidationHooks();
super.OverrideValidations("name",new NewValidationUnit()); //NewValidationUnit
extends ValidationUnit
super.OverrideValidations("password","repeatpassword",NewConfirmPasswdUnit());

addValidationLogic(list); //list is ValidationHookList
```

svenmeier (not verified)

Wed, 1969-12-31 19:00 [reply](#)

Thanks for this interesting post.

Some remarks from me:

>>These files must have the same name and must reside in the same location
... there is probably a work around for this

You can indeed easily tweak this to your liking. Whether the default behavior is 'quirky' is debatable:

<http://cwiki.apache.org/WICKET/control-where-html-files-are-loaded-from.html>

>>Wicket relies very heavily on a web application's session

This is just the same as every Struts application with a session scoped form bean. So no need for medicine to cure your bellyache ;).

One tip gratis:

If you override one method in your ListView:

```
protected IModel getListItemModel(IModel model, int index) {
return new CompoundPropertyModel(super.getListItemModel(model, index));
}
```

... you can create your labels without a model (as you're doing it on the AddContact page) e.g.:

```
listItem.add(new Label("state"));
```

Sven

Eelco Hillenius (not verified)

Wed, 1969-12-31 19:00 [reply](#)

Hi,

Thanks for the article. One note is that WebApplication already implements ISessionFactory, so just overriding newSession in your application will suffice.

I agree that having to keep the Java with markup in sync can be a bit of a pain. You can't have it all I guess, as the alternative would be configuration. If you

would really be unhappy with it, you could always implement your own lookup strategy. Check out `wicket.examples.customresourceloading` for examples of that.

The default of keeping markup and Java in the same package never bothered me though. It's easy to find like that, but more importantly, as those files are packaged in the jars you'll have reusable components with hardly any effort. Maybe this is not that important for Pages, but certainly for panels and components that use markup like trees etc, I think this is pretty cool. Put it in your classpath and it's all there for you without any extra effort.

About Wicket's reliance on the web app's session... Well, I hear what you are saying, and sometimes it even bothers me :) But our approach always have been to reverse what most other frameworks are doing by first providing the programming model we want, and then optimizing it. And the latter is where major effort is going on right now. You may have read about stateless pages (and deferred session creation). It is totally possible to write Wicket applications that don't use any server state now. Though you'll pay with the programming model - it is intended to enable users to optimize parts of their application.

Also, while it is the default that Wicket uses the `HttpSession` (and in 1.3 it is the default to use that only for the current page and overflow older pages to a temp dir or database for backbutton support), this is not a rule. In fact, by implementing a custom `ISessionStore`, you can choose any storage of your liking. The only thing we don't support is client state saving. We might have that sometime, and we have talked about different strategies to get there, but no-one of the team feels it should have priority right now (though patches would be welcome).

And as a final remark, we're now working together with Terracotta (<http://terracotta.org/>) to get a configuration that works really well for Wicket. That will provide a great fail over mechanism and can potentially even serve as a kind of virtual memory solution. Watch that space :)

Dan Howard (not verified)

Wed, 1969-12-31 19:00 reply

Nice tutorial. Thanks!

Post new comment

Your name:

E-mail:

The content of this field is kept private and will not be shown publicly.

Homepage:

Subject:

Comment: *

- Lines and paragraphs break automatically.
[More information about formatting options](#)

yakking

else'

Type the two words:

