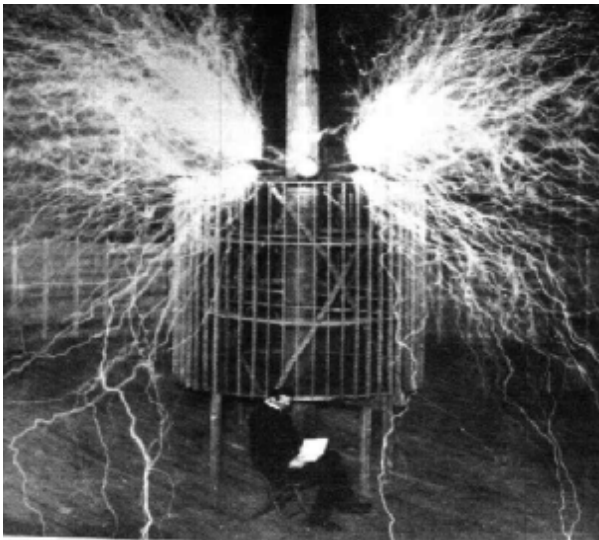


Struts

Fast Track

J2EE/JSP Framework

Practical Application with Database Access and Struts Extension



Vic Cekvenich
BETA EDITION

Major contributions by:

Vic Cekvenich

Published by:

BaseBeans Engineering
San Francisco CA 94133

Copyright © 2002 BaseBeans Engineering
All rights reserved.

This product and related documentation are protected by a registered copyright. Use, copying, and/or distribution is restricted under the license. No part of this product, framework, or related documentation may be reproduced in any form or by any means without prior written permission.

Permission has been granted to include other copyrighted materials in this manual and those copyrights are reserved by their respective authors and/or organizations.

The product described in this manual may be protected by one or more U.S. patents, foreign patents, or pending patents.

All product names mentioned herein are the trademarks of their respective owners.

ISBN 0-9716619-0-1

ADDENDUM & NOTES

The latest notes, support, and downloads will be available at basebeans.com, including newsgroups and mail lists.

If you have questions concerning topics covered in the book, you may post your questions on the mail list on the [BaseBeans forums page](#), [mvc-programmers](#). You may also download supporting files from the [BaseBeans web site](#) (basebeans.com).

Look for these improvements in upcoming releases of PostgreSQL.

Complete Sample Application Downloads

A complete sample application will soon be available for download at the BaseBeans web site (www.basebeans.com).

Struts Newsgroup

BaseBeans Engineering is now hosting newsgroups for the Struts and Orion mailing list. The news groups are automatically updated from the respective mailing list, effectively archiving critical knowledge that is usually lost due to the nature of mail lists. The news server address is news.basebeans.com.

MVC Programmer Mail List and Newsgroup

BaseBeans Engineering is also hosting a newsgroup for MVC programmers, as well as a MVC programmer mailing list. The news server address is news.basebeans.com. Please visit the BaseBeans web site (www.basebeans.com) to subscribe to the MVC programmer mailing list.

Public and Private Training Now Available

Public and Private training is now available to companies and groups. Please visit the BaseBeans web site (www.basebeans.com) or contact Vic Cekvenich (Vic@basebeans.com) for further information.

Professional Services Now Available

BaseBeans Engineering has professional consultants with extensive web infrastructure and development experience available for mentoring, development, code review, and strategic IT infrastructure engagements. Please visit the BaseBeans web site (www.basebeans.com) or contact Vic Cekvenich (Vic@basebeans.com) for further information.

TABLE OF CONTENTS

Chapter 1: Orientation.....	1
Chapter 2: Warm Up.....	11
Chapter 3: Requirements.....	17
Chapter 4: Framework Installation	23
Chapter 5: Support.....	33
Chapter 6: Application Architecture.....	37
Chapter 7: Searching	57
Chapter 8: Setup RDBMS	79
Chapter 9: Data Retrieval.....	103
Chapter 10: Object Orientation	113
Chapter 11: XML from JSP	127
Chapter 12: Drill Down	137
Chapter 13: Debugging.....	149
Chapter 14: Data Entry	155
Chapter 15: Master-Detail Processing	165
Chapter 16: Security	173

Chapter 17: Demo.....	185
Chapter 18: Data Validation	187
Chapter 19: Administration.....	195
Chapter 20: Portal Tiles	201
Chapter 21: Refactoring.....	211
Chapter 22: Menu	215
Chapter 23: Deployment	221
Chapter 24: Performance Assurance.....	237
Chapter 25: Audit Queue	241
Chapter 26: Content and Syndication.....	243
Chapter 27: Review	247
Appendix A: Downloads.....	249
Appendix B: Technology Architecture	269
Appendix C: Struts Tiles	273
Appendix D: Struts Validation	305
Appendix E: WebLogic Deployment.....	315
Appendix F: Why not use ... ?	317

This book is dedicated to my mom and dad, Veska and Zvonko Crkvencic, and my brother, Davor.

Chapter 1: Orientation

Nothing in this world can take the place of persistence. Talent will not; nothing is more common than unsuccessful people with talent. Genius will not; un-rewarded genius is almost a proverb. Education will not; the world is full of educated derelicts. Persistence and determination alone are omnipotent.

- Calvin Coolidge

Introduction

Programmers don't write books, programmers write programs. Having said that, I do consider myself a programmer. There are many people who asked me to teach them to develop web applications and with the slowdown in the software industry I decided to write this book.

So why do we need yet another book on J2EE technologies?

If you take a look at some of the web sites out there you'll find that many are slow, unusable, developed using older or limited technologies, and/or are just plain bad.

Maybe your last software development project you got locked into a non-standard, proprietary design that is now unsupported or too restrictive to be extensible.

An example of this trap is the proprietary portal framework. These closed portal products are like cars with their hoods welded shut, requiring specialized tools and personnel if and when you need something as simple as an oil change.

Also, when I had to work on these proprietary technologies, I sometimes didn't know if the bugs were somewhere in my code or buried in the bowels of the proprietary portal. And, of course, you couldn't actually view the code of the product that you were working on since the vast majority of these proprietary portal products are closed source.

So some vendors might try to lock you into their product suites, but you as an IT professional need the flexibility and freedom to expand your web applications to take advantage of an ever increasing pool of standard components and technologies.

There is little debate as to the need for dynamic application servers since the age of the static web page is behind us. Currently, the Apache web server has about 60% of market share of the web server market and anyone with DreamWeaver, or even a simple text editor, can churn out HTML pages.

Now, the need is for dynamic and secure content, simplified application management, and the ability to partition presentation styles from content, and application of those independent presentation styles to content dynamically.

So this is the time to start using dynamic portal type application servers. Maybe something that would allow for content syndication. It would be nice to have a standard way of developing those applications.

Currently, if we develop a relational database management system (RDBMS) to the structured query language (SQL) ANSI 92 standard, we are assured that there will be some support for a SQL database.

For example, if we developed a new database product that required its own query language (say in a conversational Latin

format), forsaking the SQL 92 standard, we shouldn't be surprised that not too many applications will be written to go to our database product. There is no question in this case that if you want to create a flexible, open application that has the chance to be widely adopted and easily extended, we would want to use the SQL standard and not some one-up, custom query language.

So then why do we not have the same reasoning when developing our own web applications?

If we did apply the same reasoning, we would develop our web applications to a standard with broad support, like Java 2 Enterprise Edition (J2EE).

Teaching Approach

This book is different in that it does not cover J2EE API. There are books out there that explain every method of the J2EE API's. After reviewing many different books, I now just use the JavaDoc documentation of the API.

This book will not cover all the possibilities of what you can do, it will just give you methods that I used in my examples. If you want a more complete handling of the J2EE API, consult the JavaDoc documentation.

Further, this book will use an open source framework to develop an application, as opposed to starting at the base levels of Servlets or JSP. The framework makes it easy for us to follow a solid application architecture, and allows us to build larger and more complex applications by encapsulating low-level functionality.

Also, when I have read examples in other books, I was frustrated at times at the trivial and unrealistic examples. This can make it difficult to see the practical applications of the presented concepts. So in this book we will create a more realistic application. This is not an application that will move mountains, but after you go through the steps of developing it, you will be able to apply this to your application development efforts.

This book is not one you can read in bed if you want to get a deeper understanding of the concepts presented. This is more of a workbook that you read while sitting in front of your computer and complete each of the labs.

There are many labs and some may take hours to complete. My aim is not to show you that I can do something, but to ensure, after you've gone through this book, that you can. I will not be "giving you fish", rather "I will teach you to fish".

The labs were created to this end. You may struggle to complete them, and learn as you work through many different issues.

If you only have a few years of development experience, then follow the sequence of labs and do your best to complete each one, including the demo labs.

If you are a seasoned developer, then feel free to modify and enhance the labs, trying different things if you have already mastered the primary concepts. For example, use a text editor instead of an IDE, or use a different servlet container, or a different pool manager, etc.

My ultimate goal is to teach programmers how to build a standards-based web application that allows for flexibility and functionality, including syndicated content, data entry, and reporting from scratch in a practical and realistic manner.

Most of the books out there are technology oriented, technology for technologist, and do not address the issue of applying those technologies to real-world applications.

This workbook is for programmers that should give a working understanding of and experience with Java, HTML and SQL. Otherwise, you will be stuck on basics, or worse, misunderstand something. You should be able to do the Warm-Up chapter with minimal help. That should give you an idea if you are ready for it.

Group Development

As you might know, software development is not an individual effort, but rather a group and social effort. Social interaction is a key part of the software development process. If you are a loner or lack minimal interpersonal skills, you will find yourself not growing your software development career.

An example would be building a bridge. How big of a bridge can you build by yourself? It takes many people many months to develop a useful application.

For this reason, I recommend you try the labs in this books as a group effort. This will allow you to experience the dynamics of software development and dealing with different skill sets, different technical levels, and different personalities over the life cycle of the project.

You may want to find someone to team up with to do the labs. If you have to go it alone, you will probably miss something. Consider forming a small group that can meet frequently for a few weeks to complete the labs and the book. Do you know

other web developers? Consider looking for someone at the JUG (Java User Group), or even three people, and meet nightly and on weekends. You might find a friend on-line by posting a message in one of the newsgroups or mailing lists.

It is also recommended that you have a dedicated server, a LAN, and it would be nice if you had broadband internet connectivity. There are several large downloads that you will need, such as latest Tomcat, Struts, PostgreSQL, etc.

Of course, group development is only a suggestion. It is definitely worth trying, but you can do all the labs by yourself if necessary.

You will need persistence to complete this book and you'll not find any secrets that will make you a master overnight. Only through understanding the core concepts and completing the labs will you gain a mastery over the material and know how to apply this technology to real-world development efforts.

Consider creating a text file of notes for your study that would be helpful to another reader to complete the labs and the book. Certainly, the technologies covered in this book are in a state of flux so some of the examples may and may not work with newer releases of the software packages and API's. Once you are done, please post the notes to a web site, a mail list, or a newsgroup. Please feel free to send any suggestions or comments, feedback definitely makes for an improved learning experience for those that follow you.

Software Components

The following are the software considerations and components necessary to build J2EE applications in this book:

- Operating System

- J2EE Application Server
- Relational Database Management System (RDBMS)
- IDE or text editor
- Struts

Operating System (OS)

You can use any OS of your choice, but I will use Windows for all my examples. I recommend Linux or FreeBSD if you are familiar with them, or have the time and patience to learn them in conjunction with the lessons in this book.

J2EE Application Server

The reference implementation for a J2EE servlet and JSP container is Apache-Tomcat. Although you wouldn't want to deploy eBay or Amazon on Tomcat, it should be sufficient for a majority of applications, even though Tomcat only contains a subset of the J2EE application server functionality (most notably lacking enterprise Java beans [EJB]).

Database

An open source SQL RDBMS called PostgreSQL will be used as the database for this book. Since PostgreSQL is an SQL-compliant database, you should have the possibility of being able to do the examples in this book on another J2EE standard application server and/or another SQL server of your choice. Tomcat and PostgreSQL are widely available and free for development and deployment.

Integrated Development Environment (IDE)

Programmers get religious over the IDE, so use any IDE or text editor of your choice. NetBeans (www.netbeans.org) is a free IDE and it runs on any OS, including Linux and Mac OS X.

You should not lock your development into a single vendor so that you can remain competitive and keep your options open.

Helper Libraries

There are also many helper JAR libraries I use in examples and you will need to download them, following instruction in the Appendix on Downloads.

Contact Information

The BaseBeans site (www.basebeans.com) is the official web site for this book. I also occasionally hold “Train the Trainer” sessions for instructors that want to purchase courseware and training materials, or teach this course as a public or private class. In the future, there may be an on-line course as well.

Review

The goal of this book is to teach you how to be an application programmer with the ability to solve business problems with appropriate web technologies and approaches.

The most important lab to do is the Warm Up Lab. The next lab in importance is the Data Load. Make sure that you have some data loaded, the more the better.

You should try to get and maintain momentum as you work your way through the book by allocating enough time to complete the Search labs as soon as possible.

A good place to start to make sure that you have the necessary software installed and configured is the Appendix on downloads.

Good luck.

Chapter 2: Warm Up

After this lab, you should have the basics in SQL, HTML, and Java necessary to complete the rest of the labs.

Requirements

You should have at least one (1) year of experience with SQL, HTML and Java.

Warm Up Lab

If you have the required experience, this lab should be done rather quickly, enabling you move to the next lab in the least amount of time while ensuring that you have the bare necessities to continue.

If you do not have the required experience, you may need to consult other reference material to complete this lab. This is the most important lab to complete, and if you do not complete it, there is no reason to do any other. If you are in a rush, make sure that you spend sufficient time understanding these basic topics before continuing, skipping later labs if necessary. Foundation is key!

Part 1 - SQL

Consider this design:

Let's say you want to keep track of names and take some notes each time you contact them so you have a data scheme with NAMES entity that contains a list of names (fields like first, last, phone, e-mail, etc).

There is a CONTACTS entity that contains the foreign-key reference to NAMES. The CONTACTS entity has details for NAMES and related fields such as the NAME contacted, the date of the contact, a description that would have some notes, and a planned date for next contact, to name a few.

In other words, we have a CONTACTS detail and NAMES master table. Of course, some of the NAMES do not have any CONTACTS, and some have one or many.

- Write down the SQL command that would SELECT all of the NAMES and join to the CONTACTS table based on the keys, all NAMES where 'last_name' is greater or equal to "CE".
- Make sure that NAMES that do not have CONTACTS are also selected. If you just use contacts.FK=names.PK then you won't retrieve all rows that satisfy the WHERE clause.

You should know a bit about OUTER and INNER JOINS, searchable arguments, and SQL query performance.

Part 2 – Java

- In the Tomcat "\webapps", create a "play" folder. In the "play" folder, create a "WEB-INF" folder. In the "WEB-INF" folder, create a "classes" folder. The resulting directory tree should look like "...\\webapps\\play\\WEB-

INF\classes”.

- In the “classes” folder, create a “myHelp” folder. In the “myHelp” folder, use a text editor or an IDE (ie NetBeans) to create “MyHelp.java” with a “getFunction” method that takes an Integer as a parameter and returns that Integer multiplied by 2. Compile it (using jikes or javac) and jar it.
- In the “classes” folder, create “MyApp.java” and add a main() method.

```
public static void main(java.lang.String[]  
args)
```

- Import the “MyHelp” class and make a call to the getFunction() method using the first command line argument as the getFunction() parameter. Return the result to the console.
- At a command prompt, type “java MyApp 2”. You should get 4 on console.

Part 3 – Tomcat and HTML

- Under the Tomcat directory “...\webapps”, create a “play” folder.
- In the “play” folder, create an “index.html” file that simply shows the text “Hello World”. You can use Mozilla’s Composer (it includes an HTML editor), a text editor, or any other HTML editor.
- In the “play” folder, create a “name.html” file that

contains an HTML <FORM..> that let's you enter first name, last name, phone and e-mail. Make sure it is an action form.

- In the “play\index.html” file, create a link to the “name.html” page. For the “name.html” page, use <TABLE...> tags to format the form elements.
- Start the Tomcat server from the “...\tomcat\bin” folder and point your browser to <http://localhost:8080/play>

The simple index page with the link to the “name.html” page..

- Click on “name” link and the “name.html” page should appear in the browser.

Part 4 – Web/Portal Applications

List the types of portal applications that are currently in production. Here are some examples:

- Product Sales Sites
- Dating Sites
- Auction Sites
- Magazine type sites (Community/Newspaper)
- Real Estate
- Travel Sites
- Financial Sites
- Software Development Project Sites
- University and School Sites
- Training Tutorial Sites

You can download the beginning files for this form www.basebeans.com/downloads You will need the files for your labs.

Optional Extra Credit

- Create a servlet that connects to a database, returns a greater or equal-to result set, and cancels the retrieval after the first 50 rows are returned.
- Make sure your SELECT is returning the results ordered by last name, without using ORDER BY. (HINT: by using a last name index, it won't need to be sorted at run time).

- Make a servlet take an argument of the last name used for searching.

Review

You will need knowledge of Java, SQL and HTML in order to use the application server and the frameworks to solve business problems. Hopefully you didn't have too much difficulty with the Warm-Up labs.

Chapter 3: Requirements

Every war is lost or won before it is ever fought.

- Sun Tsu

Goals

We should ensure that our project will be successful. Surprisingly, technology and coding play a small part in a projects success. The requirements are the critical key.

Requirements

Here is something that seasoned developers know and junior programmers might be surprised at:

The vast majority of software development projects fail.

So if the odds are against you, how can you turn the odds in your favor?

When large projects fail, significant amounts of resources, including money, time, and talent, are lost. Studies have been done to find the cause of these failures.

And the result:

The vast majority of software failures are due to flawed requirements.

We must be responsible, documenting that we have done due diligence in validating that we have proper requirements before we begin design and development.

Let's turn the odds in our favor by developing good requirements. Requirements must be based on how the system will be used and its functionality should be shown to have value within the organization.

For the requirements document, each report should be mocked-up. This report mock-up can be done in a spreadsheet or word processor since the aesthetics are not the point, hashing out the content is.

Here are some items to NOT include in requirements but will be handled in the Design and Analysis phase:

- Headers and Footers
- Menus
- Site Navigation
- Workflow
- Data Entry
- Look-and-feel
- Implementation Technologies

Requirements should only consist of the input and output of the system and how the new system will be used in the organization. The look-and-feel of the application should follow the "rule of least astonishment", which translates to providing a user interface that is similar to existing and popular web sites.

Example Mock up:

NAME:	PHONE:	E-MAIL:
Doe, John	(415) 555-1212	jdoe@nosspam.net

	CONTACTED:	NOTE:
	Jan 2, 2001	Left message about project
	Jan 3, 2001	Covered basic bill and set-up meeting for 2/5/01
Doe, Jane	(707) 555-1212	<u>janed@freenet.org</u>
	CONTACTED:	NOTE:
	Jun 13, 2001	Initial contact, send fruit basket...

Depending upon the application, the outputs of the application could include reports for new members, bill aging, membership source, application usage trends, advertising exposure, receivables, etc.

The goal of the requirements and the reports is to enable the design of data models and test cases. By looking at the report specifications, the relationships between entities become clear. Based upon these reports, we can tell what the possible SQL SELECT statements will look like. Good requirements, therefore, lead to optimal design, without stressing full joins that consume databaser server and other resources.

A return-on-investment (ROI) study is sometimes conducted to determine if the system is worth developing in the sense that the savings/income from having the system in place will offset the costs of development.

You should create a list of the outputs of the system. The list should include the output name, the type of user that would consume the output, how many users there might be in any particular timeframe, how often each user might generate or consume the output, and some kind of numeric scale of importance.

Output:	User Base:	Number of Users:	Usage Per Day:	Importance:
Names List	All Users	500,000	9	99
Xxx	Xxx	999	999	99

Based on this you can massage the scope of the first phase of the application system to match your budget.

Some good reasons for developing an application is if you are replacing an existing or a competing system, reducing the employee head count, or compliance to a government regulation.

Some methodologies advocate that when the requirements are done, you should be able to start writing user documentation. It does not matter how the program works. You should be able to document what the report is, how the fields are calculated, etc. This is the time to start writing user documentation. The programmers can refer to the growing body of documentation when in doubt about how something in the system should work.

Requirements Lab

- Create a requirements mock-up for a personal information manager (PIM) that you will construct throughout this book, from requirements to deployment. The sample application would let an organization (your company) list names of employees, vendors, and clients. For each name, we should be able to view a history of contacts (letter, phone call, meeting, e-mail), including the type of contact, the next step requested, and contact status. These are just suggestions, please be creative and specific about a PIM you want to build.
- We also want to track a list of issues, requests, bugs, or goals, in another table.

- Create at least three (3) report/output mock-ups. Keep the reports simple for the later labs since you will have to build it in a short period of time.
- Determine the benefits of and uses for this PIM application
- Create summary ROI points, such as cost savings from administrative personnel, lost sales due to bad customer contact management, etc.
- Create some brief user documentation for one (1) report, explaining the fields of the report and how the report will be consumed by the user base.

Review

Software development depends on requirements. And like most important lessons in life, you will forget this at times, and may have to re-learn this painful lesson.

Requirements should specify the usage, outputs, and benefits of the application.

Chapter 4: Framework Installation

Goals

We will use the Tomcat server, the NetBeans IDE, and the Struts/MVC framework.

Framework Installation

I assume you have installed and configured the NetBeans IDE and Tomcat, or another IDE and J2EE application server of your choice.

Struts

- Copy the “struts-blank.war”, “struts-documentation.war”, “struts-example.war”, and “struts-template.war” from the Struts installation directory tree to the “...\tomcat\webapps” directory.

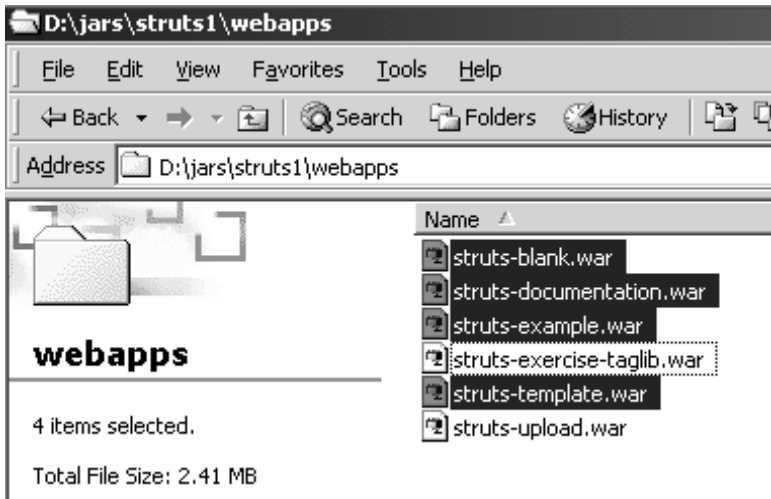


figure 4.1

- Restart Tomcat.
- Review the “struts-documentation” application in your browser (<http://localhost:8080/struts-documentation>).
- Review the “struts-example” application in your browser (<http://localhost:8080/struts-example>).

A Walking Tour of the Struts Example Application

This article is meant to introduce a new user to Struts by "walking through" the example application on the Struts

This article is based on the version 1.0 build of Struts. It is assumed that the reader has already installed a development environment on the client machine, and is ready to explore the example on their own development platform (e.g. J

This article also assumes the reader has a basic understanding of the Java language, JavaBeans, Web services, and various other development-related topics.

- [index.sdo](#)
 - [Web and JSP Application Resources in Struts](#)
 - [DatabaseStruts.java](#)
- [error.sdo](#)
 - [struts-test-hg.xml](#)
 - [LoginForm.java](#)
 - [LoginPage.java](#)
 - [struts-test-hg.xml.2](#)

figure 4.2

Tomcat

- It is a good idea during development to start the Tomcat server from the "...\tomcat\bin" folder using a command line:

WINDOWS: catalina run
LINUX: catalina.sh run &

- You must edit "catalina.bat" (or "catalina.sh" for

Linux/Unix) to include your environment's CLASSPATH.

```
WINDOWS (around line 69):  
set  
CP=%CATALINA_HOME%\bin\bootstrap.jar;%JAVA_HOME%\lib\tools.  
jar;%CLASSPATH%
```

```
LINUX (around line 89):  
CP$="$CATALINA_HOME/bootstrap.jar:$CLASSPATH"
```

NetBeans IDE

- Go to the NetBeans home page (<http://www.netbeans.org>).
- Download the XML Support module from the NetBeans site and expand it under "d:\jars". Copy the "XMLModuleSupport.jar" to "d:\netbeans\modules". We will need XML editing in the next lab. You can always edit any XML using a simple text editor.

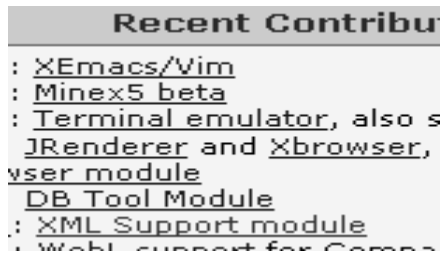


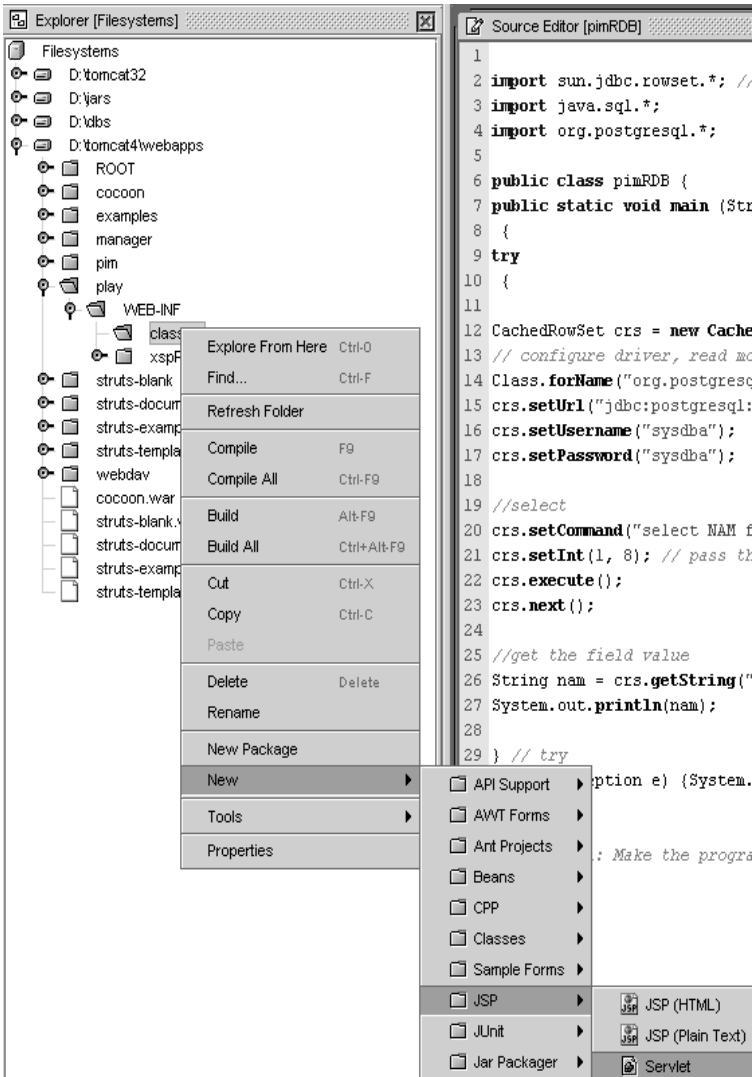
figure 4.3

- Now make sure we have a "play" application (done in the Warm-Up Lab) to test our development procedure.



figure 4.4

- If you don't have the "play" application, create a "play" folder in the "...\tomcat\webapps\" folder. Create a "WEB-INF" folder in the "play" folder. Create a "classes" folder in the "WEB-INF" directory. The resulting directory tree should be "...\tomcat\webapps\play\WEB-INF\classes".
- Create a new servlet using the IDE (or text editor).



- Create a simple servlet “FirstServlet.java” in the “...\play\WEB-INF\classes”:

Listing 4.1 - FirstServlet.java

```
import javax.servlet.*;
import javax.servlet.http.*;

import java.io.*;

public class FirstServlet extends HttpServlet {

protected void doGet(HttpServletRequest request,
HttpServletResponse response)
{
try{
response.setContentType("text/html");
PrintWriter out = response.getWriter();

out.println("<html>");
out.println("<head>");
out.println("<title>Servlet</title>");
out.println("</head>");

out.println("<body>");

out.println("Tomcat 4 Servlet");

out.println("</body>");
out.println("</html>");
out.close();

} // try
catch (Exception e) {System.out.println(e);}
} // doGet()
} // class
```

- Compile the servlet using javac (or jikes):

Now we have compiled the source code of the servlet, we need to configure Tomcat to know about this servlet.

- Create “web.xml” in “...\play\WEB-INF” folder.

Listing 4.2 - First web.xml

```
<?xml version="1.0" encoding="ISO-8859-1"?>

<!DOCTYPE web-app
  PUBLIC "-//Sun Microsystems, Inc.//DTD Web
  Application 2.2//EN"
  "http://java.sun.com/j2ee/dtds/web-
  app_2_2.dtd">

<web-app>

<servlet>
  <servlet-name>frst</servlet-name>
  <servlet-class>FirstServlet</servlet-class>
</servlet>

<servlet-mapping>
  <servlet-name>frst</servlet-name>
  <url-pattern>/FST</url-pattern>
</servlet-mapping>

</web-app>
```

- Create an “index.html” file in the “...\webapps\play” folder.

Listing 4.3 - Play index.html


```
<HTML>
<HEAD>
<TITLE>Hello TC </TITLE>

</HEAD>
<BODY>
<H1>TC</H1>
<HR>
  Before Dynamic Content Portals, we used a static
web server.
</BODY>
</HTML>
```

- Restart Tomcat.
- Browse to “<http://localhost:8080/play>”.
- Browse to “http://localhost:8080/play/FST”.

Development Environment

You should run the application server (ie Tomcat) on another machine and the database server (ie PostgreSQL) on third machine.

However, if you run PostgreSQL, Tomcat, NetBeans, WordPerfect, Mozilla, XTerminal, pgAdmin concurrently, it uses 300 megs of RAM on my Windows machine. Just FYI.

When setting up your development environment, source code should be on a central file server at all times. There is no reason for code on local PCs unless there is a “sandbox” configuration for each developer with a centralized source control server.

Also, if you keep the number of applications in the “webapps”

folder to a minimum, Tomcat will start more quickly.

Review

We have done some configuration on the Tomcat application server, added a module to the NetBeans IDE, created a simple servlet, and added a web configuration (web.xml) file to a J2EE application.

Chapter 5: Support

Goals

You should ensure that you have qualified resources to answer any technical development issues.

Support Lab

- Browse the Orion page (<http://www.orionsupport.com>) and subscribe to the mail list.
- Browse the Struts page (<http://jakarta.apache.org/struts>) and subscribe to the mail list.
- Review the PoolMan documentation.
- Browse the PostgreSQL home page (<http://www.postgresql.org>).

It sometimes takes a day for the mail list to start sending messages to your account. Do you know how to unsubscribe from a mail list?

You may want to create a mail folder in your email client to filter the incoming mailing list messages. The figure below shows that any messages sent from the Tomcat, Orion, NetBeans or PostgreSQL mailing lists will be moved to the appropriate folder. Be sure to read the mailing list messages regularly and save the better messages for future reference.

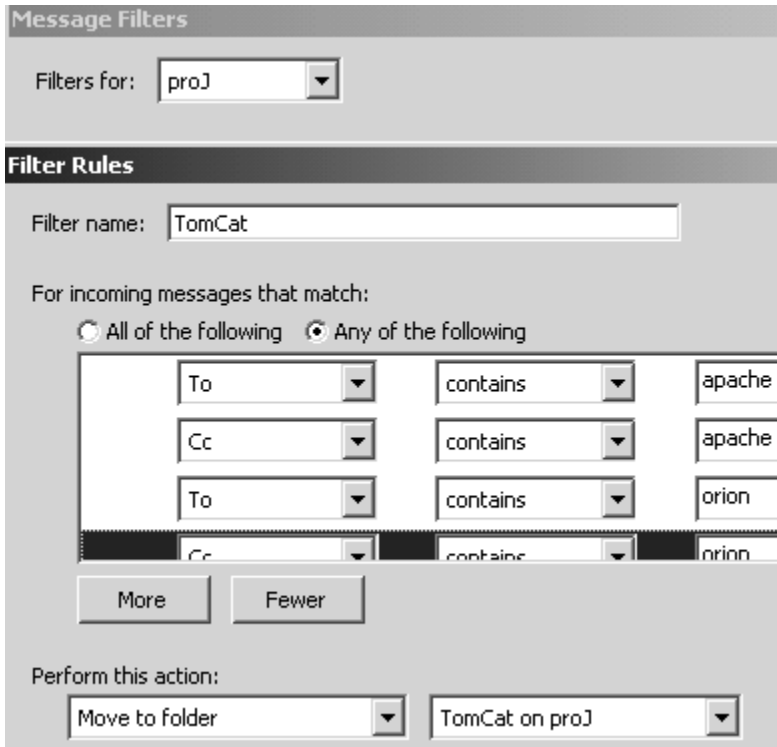


Figure 5.1

Consider the following support steps.

Your first level of support:

- <http://jakarta.apache.org/struts/userGuide/resources.htm>
|

- <http://www.us.postgresql.org/users-lounge>
- <http://www.netbeans.org>
- *Remember:* Small incremental steps during development. Keep going back to a simple example or your previous iteration.

Your second level of support:

- <http://www.google.com> for web site searches
- <http://groups.google.com> for newsgroup searches (very valuable)
- The /docs folder of any tool or package that we're using
- *Exercise:* Find the RowSet documentation and determine the procedure to save changes to a database.

Your third level of support:

- <http://www.mail-archive.com> - Find Struts.
- <http://archives.postgresql.org/pgsql-general> - PostgreSQL search.

Newsgroups and mailing lists are very important since chances are that you are not the first to encounter a particular issue. If you find that you determined a solution to your issue and that there weren't any solutions posted, please contribute what you've learned so that others may benefit.

- And last... look at the source code if you have to. The source code is available for everything in the framework.

Chapter 6: Application Architecture

Goals

The goals of this chapter are to install and configure the Struts framework.

Requirements

It is assumed you have installed and configured the NetBeans IDE (or an IDE of your choice) and Tomcat. If you are having problems installing these packages, please review the Appendix on Downloads.

Learning Levels

One of my college professors used to say that “in order to learn, you have to be able to say, ‘I do not know’”.

Unconsciously Incompetent

At this level, they do not even know that they truly do not know. Pretenders can act as if they know how to build a web application. They will use a constant stream of acronyms and definitions of bleeding edge designs culled from information weeklies. They tend to underestimate the duration and complexity of tasks. The term “buzz word bingo” comes to mind.

Consciously Incompetent

You realize that you do not know. This is the first step that leads to learning.

Consciously Competent

You think you know it now.

Unconsciously Competent

You realize that you've just climbed a single tree and that the tree is but one in a forest. And behind the forest is a hill, and behind that a mountain, and behind that other mountains. And that there are other planets beyond, other solar systems, other galaxies, and, quite possibly, other universes. A humble, soft-spoken, introspective monk. A mildly retarded lemming. Happy that the thing compiled and have no idea why a system this complex can even function.

In other words, you have some code in production that is generating revenue or a return on your development investment!

Application Architecture

The following is a rather basic design and a foundation of three-tier application architecture. Figure 6.1 is meant to illustrate a separation of the data access (or model) from the presentation (or view) from the application control logic (or controller). In software engineering parlance, this partitioning is called the model-view-controller (MVC), or Model 2, design.

Our PIM application will leverage an MVC framework as follows:

- **Model** – CachedRowSet (part of JDBC) via Java Form Beans
- **View** – Java Server Pages (JSP) generating HTML or XML to browser-side XSL
- **Controller** – Struts framework

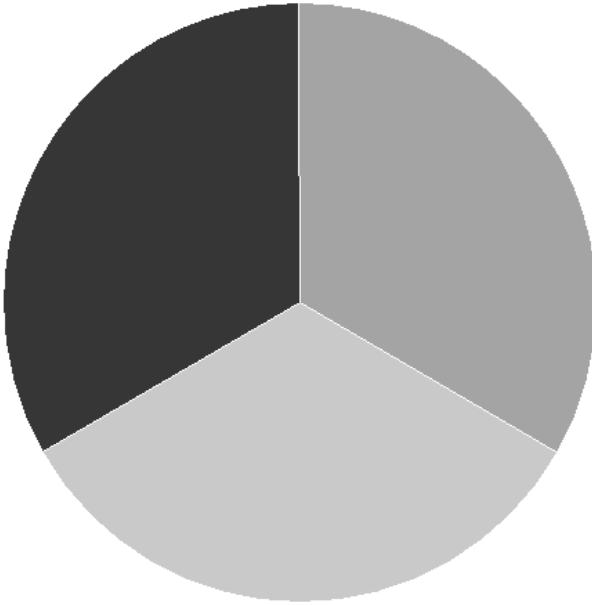


figure 6.1

The whole concept for MVC is to partition functionality into three (3) separate layers. Hence, data storage and retrieval (Model) functionality should not be contained by the view or controller, the user interfaces (View) should not be contained by the controller or model, workflow considerations (Controller) should not be contained by the view or the model layers.

However, application management may require hooks in any one of the MVC layers, thereby allowing application monitoring and usage analysis.

Back End

Most web applications have some sort of back-end batch components. Batch processing is the bread-and-butter of efficient data processing. In order for on-line processing to be fast, it must be very efficient and not do extra work. So in an off-line, asynchronous mode, we prepare the system for on-line processing. Many external interfaces may be asynchronous.

For example, we might create geo-coding in a batch process, so that when the on-line application has thousands of users looking for the nearest store, the selection of the store may occur with minimal delay and resources.

Another example of batch processing is data scrubbing and loading.

So our architecture looks like this:

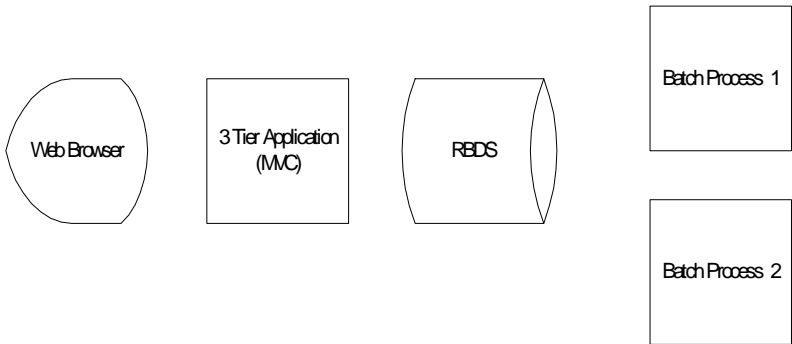


figure 6.2

Basic JSP Demo

Let's create a simple Java Server Page (JSP) using a standard tag to illustrate areas that need improvement. You will see how in the JSP the presentation logic (view) and the presentation style (view) is mixed in with presentation content (model).

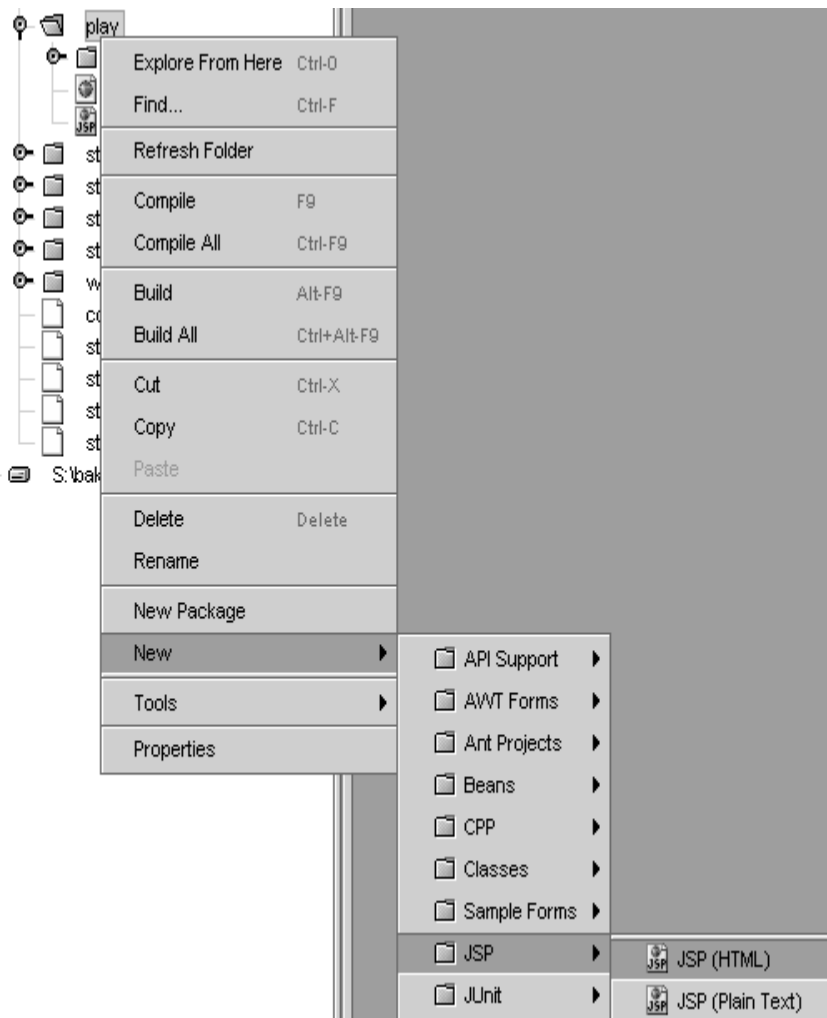


figure 6.3

In the “webapps\play” folder, create a “Hello World” JSP (unCleanView.jsp) that uses the standard tag library. The JSP tag library is a proposed standard tag library that doesn’t adhere to MVC.

If you followed the steps in the Download Appendix, then you have downloaded the standard tag library into the “d:\down” (or similar) directory. Now expand and put the resulting jar (jsptl.jar) into the “d:\jars” folder. You may need to add the expanded jar to your CLASSPATH.

You may configure the tag library for use in Tomcat by the following steps:

1. Copy the tag library jar to the “...\play\WEB-INF\lib” folder.
2. Since we are setting this up, always copy the “struts jar” to “WEB-INF\lib” every time you create an application directory tree in Tomcat.
3. Copy the tag library “.TLD” files to the “...\play\WEB-INF” folder. (Also copy the Struts related “.TLD” files in future apps.) The .TLD file is an XML file that describes the tag library.
4. Now we need to tell the “web.xml” file about the new capabilities described by the TLD. Recall that the “web.xml” file is located in “...\play\WEB-INF”. (After you write a few Tomcat applications, you will become familiar with the “web.xml” file and the “WEB-INF” folders). The entry in “web.xml” for taglib is <taglib>.

So let’s demo setting up the standard tag library and running the simple JSP tags page we’ve just created.

- Here are the modifications in **BOLD** for step #4 above to the “web.xml” file to tell the web server about our new

tag capabilities:

Listing 6.1 - Modified web.xml for standard tag library

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<!DOCTYPE web-app
  PUBLIC "-//Sun Microsystems, Inc.//DTD Web
  Application 2.2//EN"
  "http://java.sun.com/j2ee/dtds/web-app_2_2.dtd">
<web-app>
<servlet>
  <servlet-name>frst</servlet-name>
  <servlet-class>FirstServlet</servlet-class>
</servlet>
<servlet-mapping>
  <servlet-name>frst</servlet-name>
  <url-pattern>/FST</url-pattern>
</servlet-mapping>
<taglib>
  <taglib-
  uri>http://java.sun.com/jsp1/jr</taglib-uri>
  <taglib-location>/WEB-INF/jsp1-
  jr.tld</taglib-location>
</taglib>
</web-app>
```

You can take a peek at the Struts sample “web.xml” (...\\tomcat\\webapps\\struts-sample\\WEB-INF) file to see some other tags we will use to configure our web application.

- Restart Tomcat!

Here is the “unCleanView.jsp” source:

Listing 6.2 - unCleanView.jsp

```
<!-- uri should match uri in web.ini -->
<%@ taglib prefix="jr"
uri="http://java.sun.com/jsptl/jr" %>
<%@ taglib prefix="jx"
uri="http://java.sun.com/jsptl/jx" %>

<html>
<head>
  <title>All mixed up</title>
</head>

<body bgcolor="#FFFFFF">
<!-- Note color style above. We could also put tr and
td in here to make it layout all nice -->

<h4>Mixed up presentation with count from 1 to
10</h4>

<jx:expressionLanguage
evaluator="org.apache.taglibs.jsptl.lang.spel.Evaluat
or"
<jx:forEach var="i" begin="1" end="10">
  <jx:expr value="$i"/>
</jx:forEach>
</jx:expressionLanguage>

</body>
</html>

<!--
So we have content, style and code logic presentation
components here.
Bad design.
```

-->

XSL

The days of HTML are numbered. All of the latest browser versions support XML. Cascading style sheets (CSS) were nice, but extensible style language (XSL) is much more powerful.

For our application, we will generate XML from JSP's to our client browsers, allowing the browsers to commit XSL transformations (XSLT) to HTML to off-load some of the work from the application server to the client. If we take this XSLT approach, then we can make "skinable" user interfaces in the browsers. The XSL may be stored in a centralized location which will allow for centralized control over look-and-feel.

Having extolled the virtues of XML, it will be a while before XML completely supplants HTML. Actually, HTML will eventually be replaced by XHTML, a DTD that models the HTML tags in a valid XML document.

Keeping with the MVC approach, no presentation information (fonts, colors, etc.) should be anywhere but contained within the XSL (view).

XSL Lab

- Create the "playXML" folder in the "...\play\WEB-INF" folder.
- Create "source1.xml" in the "...\WEB-INF\playXML" folder.

Listing 6.3 - source1.xml

```
<?xml version="1.0"?>
<?xml-stylesheet href="transform1.xsl"
type="text/xsl" ?>

<page>
  <title>Hey, Vinny!</title>
  <content>
    <paragraph>Here we have a paragraph. Anyone can
type in content</paragraph>
  </content>
</page>
```

- Create “transform1.xsl” in the “..\WEB-INF\playXML” folder.

Listing 6.4 - transform1.xml

```
<?xml version="1.0" ?>
<xsl:stylesheet version="1.0"
xmlns:xsl="http://www.w3.org/1999/XSL/Transform">

  <xsl:template match="title">
    <h1 align="center">
      <xsl:apply-templates/>
    </h1>
  </xsl:template>

  <xsl:template match="paragraph">
    <p align="center">
      <i><xsl:apply-templates/></i>
    </p>
  </xsl:template>

</xsl:stylesheet>
```

- Create “pipe.bat” (Windows) or “pipe.sh” (Linux) in the “..\WEB-INF\playXML” folder.

Listing 6.5 - pipe.bat

```
REM Add Xalan to our path so we can transform (Also
when done).
```

```
SET CLASSPATH =
d:\jars\xalan22\bin\xalan.jar;%CLASSPATH%
ECHO %CLASSPATH%
```

```
REM Run an XSLT process on 1st argument, using 2nd
argument as style sheet and
REM redirect to create a file
```

```
java org.apache.xalan.xslt.Process -in %1 -xsl %2 >
%3
```

```
type %3
```

- Transform the “source1.xml” document using “transform1.xsl”.

```
pipe source1.xml transform1.xsl serialize1.html
```

This will transform the XML input (source1.xml) to HTML (serialize1.html) using the XSL input (transform1.xsl). Note that we can use any XML source file and XSL transform file and generate an HTML output file. By using different XSL transformer files, we can generate different output files (i.e. HTML, WML, RTF, etc).

XML Lab

- Open the XML file created in the lab (source1.xml) in Mozilla (or IE). You may view the XML file using

different XSL files to transform the XML to HTML.

Data Model Layer Java Beans using CachedRowSet

Any serious web application implements database connection pooling. Instead of connecting to the database every time a data request is made, connections to the database are established and then “pooled” for later use.

Since initial connections to databases are time intensive, we will use the connection pooler “PoolMan” (PoolMan jar). If you followed the steps in the Download Appendix, you should have PoolMan installed in the “d:\dbs\Poolman2” (or similar) folder.

Do you remember the JDBC parameters to connect to PostgreSQL from the RDBMS chapter?

Let’s configure PoolMan by creating the “Poolman.xml” file in the “\dbs\Poolman2\lib” folder. (This folder should be in your CLASSPATH, so that the application server can find it.)

Connection Pool Lab

- Familiarize yourself with PoolMan by browsing the PoolMan “doc” folder and possibly the PoolMan web site (<http://www.codestudio.com>).

The CachedRowSet is used in place of the older ResultSet for handling result sets. We have a pool of connections, and SELECTS are executed against connections within that pool to reduce the overhead of the RDBMS calls. Calls to the RDBMS are usually the slowest part of the application architecture.

Once we issue a SELECT call, the result set is returned as a CachedRowSet. The CachedRowSet is used to get and set individual columns (or fields) in the returned row(s).

- The source for “..\WEB-INF\poolman.xml”:

Listing 6.6 - First poolman.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<poolman>

  <management-mode>local</management-mode>

  <datasource>

    <dbname>pmiDB</dbname>
    <jndiName>jndiPMIdb</jndiName>

    <driver>org.postgresql.Driver</driver>
    <url>jdbc:postgresql://localhost:5432/pimDB</url>
    <!--next 2 should be in code for production for
security -->
    <username>sysdba</username>
    <password>sysdba</password>

    <minimumSize>50</minimumSize>
    <maximumSize>200</maximumSize>
    <connectionTimeout>600</connectionTimeout>
    <shrinkBy>5</shrinkBy>

    <logFile>d:\logs\poolman.log</logFile>
    <debugging>true</debugging>

  </datasource>
</poolman>
```

You will find testing after each incremental development step the fastest way to code since it will reduce guesswork during troubleshooting.

- Go to a command prompt in the PoolMan samples folder:

```
java PoolManSample "select NAM from NAM where  
PK=8" "pmiDB"
```

Connection Pool Test Lab

- Create a simple servlet (...WEB-INF/classes/PoolManTest.java) to test the connection pool. Make sure that you can get this to run before continuing to the next lab.

Listing 6.7 - PoolMan Test Servlet (PoolManTest.java)

```
// this is why Data Access is in another layer,  
  
import javax.servlet.*;  
import javax.servlet.http.*;  
import java.io.*;  
  
import java.util.*;  
// to talk to our connection pool  
import com.codestudio.util.*;  
import com.codestudio.sql.PoolMan;  
import java.sql.*;  
// The JDBC 2.0 Extension  
import javax.sql.*;  
// The JNDI package  
import javax.naming.*;
```

```

// we know this already, right?
import sun.jdbc.rowset.*;

public class PoolManTest extends HttpServlet {
// how long will we have to extend from a simplistic
HttpServlet?

protected void doGet(HttpServletRequest request,
HttpServletResponse response)
{
// outside the try
DataSource ds;
Connection con=null;
CachedRowSet crs=null;

try{

response.setContentType("text/html");
PrintWriter out = response.getWriter();

// from out poolman XML, this is fast!
ds = PoolMan.findDataSource("jndiPMIdb");
//we should in production give it a password in code
not in file
con=ds.getConnection();

// nothing new down
crs=new CachedRowSet();
crs.setCommand("select NAM from NAM where PK = ?");
crs.setInt(1, 8);
crs.execute(con);
crs.next();

out.println("<html>");
out.println("<head>");
out.println("<title>Servlet</title>");
out.println("</head>");
out.println("<body>");
out.println("Tomcat 4 Servlet:");
//get the field value
String nam = crs.getString("NAM");

```

```

    out.println(nam);
out.println("</body>");
out.println("</html>");
out.close();
// quick, not MVC, but a small steps!
} // try
catch (Exception e) {System.out.println(e);}
// Oh:
finally {try{if (con!=null) con.close();}
        catch (Exception e) {System.out.println(e);}
        }//finally
} // doGet()
} // class

```

- Change the “web.xml” file so it knows about this new servlet.

Listing 6.8 - Modified web.xml for PoolManTest Servlet

```

<?xml version="1.0" encoding="ISO-8859-1"?>
<!DOCTYPE web-app
  PUBLIC "-//Sun Microsystems, Inc.//DTD Web
  Application 2.2//EN"
  "http://java.sun.com/j2ee/dtds/web-app_2_2.dtd">
<web-app>
<servlet>
  <servlet-name>frst</servlet-name>
  <servlet-class>FirstServlet</servlet-class>
</servlet>
<servlet>
  <servlet-name>PMTTest</servlet-name>
  <servlet-class>PoolManTest</servlet-class>
</servlet>

```

```

<servlet-mapping>
  <servlet-name>frst</servlet-name>
  <url-pattern>/FST</url-pattern>
</servlet-mapping>

<servlet-mapping>
  <servlet-name>PMTest</servlet-name>
  <url-pattern>/PMTest</url-pattern>
</servlet-mapping>

<taglib>
  <taglib-
uri>http://java.sun.com/jsp/j2ee/jstl/jstl-jr</taglib-uri>
  <taglib-location>/WEB-INF/jstl-
jr.tld</taglib-location>
</taglib>

</web-app>

```

- Restart Tomcat.
- Browse to <http://localhost:8080/<app>/PMTest> to test PoolMan servlet.

Did you get a connection out of the pool?

- Make sure you changed the “catalina.bat” (or “catalina.sh”) classpath to include the %CLASSPATH% (or \$CLASSPATH).

Java Form Beans for Model Access

We are not doing raw programming where we have to hand-code each line. We are trying to leverage a framework that can

handle lower level functionality for us. We will use JavaBeans for data access, which is just a class that has mutators (set<attribute>()) and accessors (get<attribute>()) methods for each field.

JavaBean Example

Listing 6.9 - Person JavaBean example

```
public class Person {  
  
    // Attribute (private to enforce  
encapsulation)  
    private String name;  
  
    // Constructor  
    public Person() {}  
  
    // Accessor Method (getter's)  
    public String getName() {  
        return name;  
    }  
  
    // Mutator Methods (setter's)  
    public void setName(String name) {  
        this.name = name;  
    }  
  
} // Person JavaBean
```

We will want to leverage Struts features, and Struts requires our Data Access JavaBeans extend the FormBean (a Struts class).

Short Cut

You can download a sample web application from BaseBeans (www.basebeans.com) . You can also get a learning start-up war that contains all the required files.

Chapter 7: Searching

Goals

Create our first and second page using the Struts framework and understand the “struts-config” file.

Requirements

You should have a basic understanding of the HTML FORM processing.

For example, the <FORM> tag for posting is:

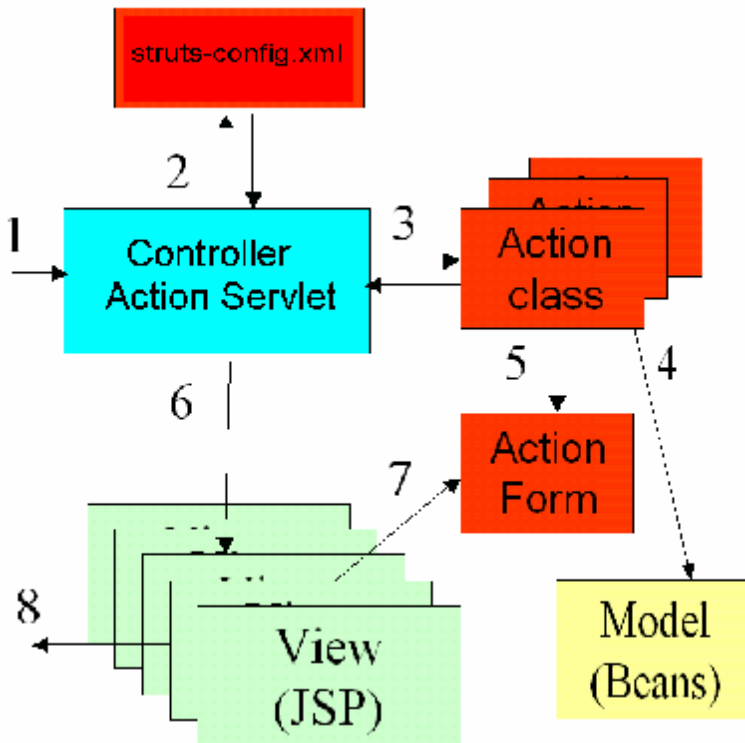
```
<FORM METHOD="post"  
ACTION="http://www.mysite.com/here">
```

This might be a good time to review the HTML <FORM> tags from your favorite HTML book before continuing.

Struts Overview

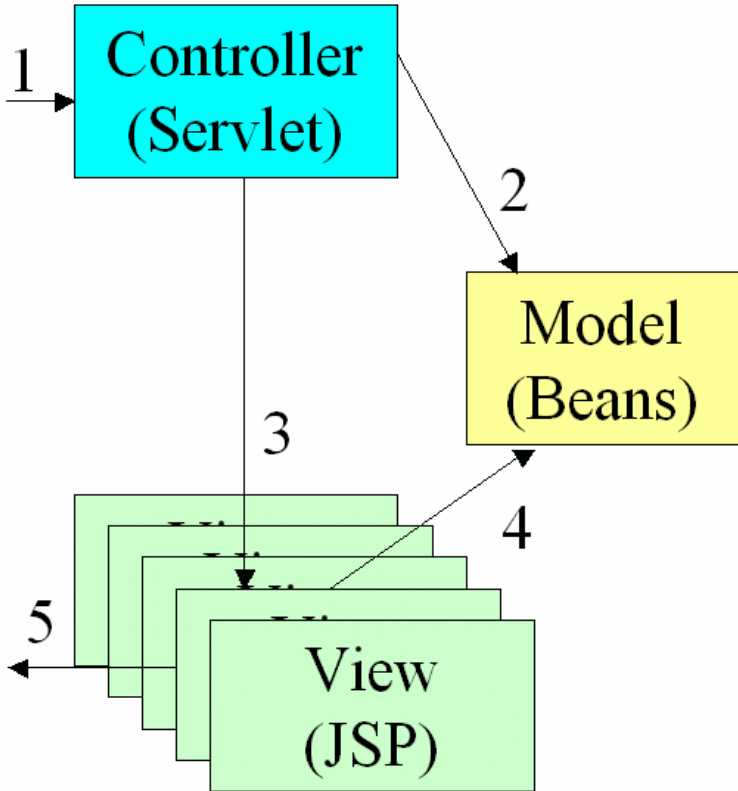
The most important part of MVC is NEVER to call the JSP, to call the action. The action calls the JSP. When you call the JSP, you are not doing MVC.

The following diagrams illustrate how Struts works.



1. Request sent by client (usually a browser).
2. Look-up in "struts-config.xml" which Action maps to the URL.
3. Run selected Action class, access the Model logic, and decide which JSP will display the requested results.
4. Possible access of the Model bean from the Action class for data.
5. Struts stores input values in ActionForm which can be accessed by the Action for processing.
6. The controller servlet forwards to the correct View (JSP).

7. The JSP retrieves request parameters from the ActionForm.
8. The response is sent back to the client.



1. Client request accepted by the controller servlet.
2. Data validated against the Model and business logic is executed.
3. Control is passed to the View (JSP).
4. JSP retrieves the relevant data from the Model state.

5. A fully formed response is sent back to the client.

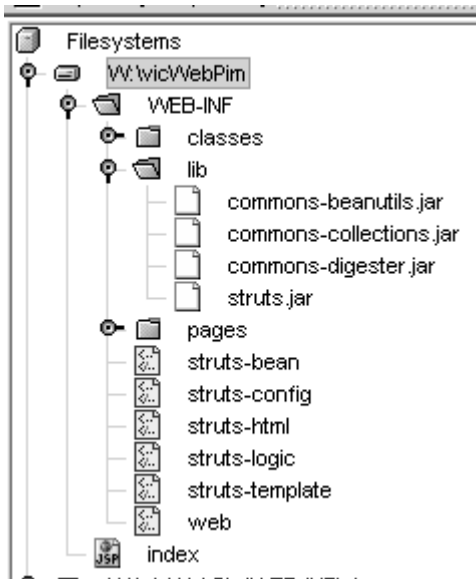
Searching

Struts-Blank

Let's look at the Struts framework.

- Copy the “struts-blank.war” to the “...\tomcat\webapps” folder. If you start Tomcat, Tomcat will automatically expand the web archive file. However, you may manually expand the war file to examine the Struts start-up skeleton.
- Now is a good time to name your software application. I like naming it after a person and not an acronym so think of the name for your PIM.
- Under “...\tomcat\webapps”, right click the “struts-blank” and rename it (I called my <app> “VicWebPim” but you may use any other name).

It should look something like this:



Note that skeleton “web.xml”, jars, and TLD files for the Struts tag library are already copied, expanded from “struts-blank.war”.

So let’s use a Struts tag.

- Create “index.jsp” in the “...\\vicWebPim” folder as our first framework “Hello World” application.

```
<%@ page language="java" %>
<%@ taglib uri="/WEB-INF/struts-html.tld"
prefix="html" %>
<head><title>Welcome</title></head>

<body>
Hello World,<p>
<html:link page="/do/searchAct"> Search </html:link>
</body>
```

Note that we are using the Struts HTML taglib to do our link to a page we will create later.

- Browse the “index.jsp”
(<http://localhost:8080/VicWebPim/index.jsp>).

Struts-Config Lab

- Let’s look at the “...\WEB-INF\web.xml” file. It is rather straightforward.

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<!DOCTYPE web-app
  PUBLIC "-//Sun Microsystems, Inc.//DTD Web
  Application 2.2//EN"
  "http://java.sun.com/j2ee/dtds/web-app_2_2.dtd">
<web-app>
  <!-- Action Servlet Configuration -->
  <servlet>
    <servlet-name>action</servlet-name>
    <servlet-
class>org.apache.struts.action.ActionServlet</servlet
-class>
    <init-param>
      <param-name>application</param-name>
      <param-value>ApplicationResources</param-value>
    </init-param>

    <init-param>
      <param-name>config</param-name>
      <param-value>/WEB-INF/struts-config.xml</param-
value>
    </init-param>
    <init-param>
```



```

        <param-name>debug</param-name>
        <param-value>2</param-value>
    </init-param>
    <init-param>
        <param-name>detail</param-name>
        <param-value>2</param-value>
    </init-param>
    <init-param>
        <param-name>validate</param-name>
        <param-value>true</param-value>
    </init-param>
    <load-on-startup>2</load-on-startup>
</servlet>

<!-- Action Servlet Mapping -->
<servlet-mapping>
    <servlet-name>action</servlet-name>
    <url-pattern>/do/*</url-pattern>
</servlet-mapping>

<!-- The Welcome File List -->
<welcome-file-list>
    <welcome-file>index.jsp</welcome-file>
</welcome-file-list>

<!-- Struts Tag Library Descriptors -->
<taglib>
    <taglib-uri>/WEB-INF/struts-bean.tld</taglib-uri>
    <taglib-location>/WEB-INF/struts-
bean.tld</taglib-location>
</taglib>
<taglib>
    <taglib-uri>/WEB-INF/struts-logic.tld</taglib-
uri>
    <taglib-location>/WEB-INF/struts-
logic.tld</taglib-location>
</taglib>
<taglib>

```

```
        <taglib-uri>/WEB-INF/struts-html.tld</taglib-  
uri>  
        <taglib-location>/WEB-INF/struts-  
html.tld</taglib-location>  
    </taglib>  
    <taglib>  
        <taglib-uri>/WEB-INF/struts-  
template.tld</taglib-uri>  
        <taglib-location>/WEB-INF/struts-  
template.tld</taglib-location>  
    </taglib>  
</web-app>
```

NOTE: *Noticed that I modified the <url-pattern> for the “action” <servlet-name> to “/do/”. Struts normally has the “/*.do” pattern for the action servlet but I’ve had difficulties with firewalls using the default Struts pattern so we’ll use the “/do/” pattern throughout this book as a work-around.*

We have a single servlet defined that will run things for us, a mapping to forward, and tag libraries that will help us develop our application.

Anything we send to “/do/” will be handled by Struts. Struts uses another configuration file, “struts-config.xml”. Once you become comfortable configuring this file, you will be able to leverage the Struts framework.

- In order to make it easier to understand the Struts MVC, let’s create folders “data” and “app” in the “...\WEB-INF\classes” directory.

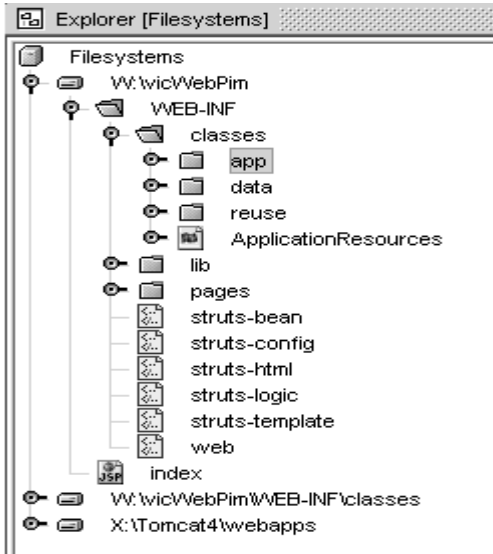
In the “data” directory, we will store the classes that will act as our model layer, JavaBeans that extend the Struts FormBean.

In the “app” directory, we will store the classes that will act as

our controller layer, the application code that extends Struts Action and has the perform() method.

- Create a “pages” folder in the “...WEB-INF” directory to hold the view (pages) layer which will consist of JSP’s.

It should look like this:



- We have the “...WEB-INF\classes\data” (model), the “...WEB-INF/pages” (view), and the “...WEB-INF\classes\app” (controller) folders.

Do you see how this relates directly to MVC?

- Now we can read the “struts-config.xml” file easier:

```
<?xml version="1.0" encoding="ISO-8859-1" ?>
```

```

<!DOCTYPE struts-config PUBLIC
    "-//Apache Software Foundation//DTD Struts
    Configuration 1.1//EN"

    "http://jakarta.apache.org/struts/dtds/struts-
    config_1_1.dtd">
<struts-config>

<!-- ===== Global Names Forward Definitions,
=== -->
<global-forwards>
    <!-- <forward name="search"
    path="/do/searchAct"/> -->

</global-forwards>

    <!-- ===== Form Data Bean Definitions
    ===== -->
<form-beans>
    <form-bean        name="SearchFrm"
                      type="data.SearchFrm"/>

    <form-bean        name="NameLstFrm"
                      type="data.NameLstFrm"/>

</form-beans>

<!-- ===App Controller Action Mapping Definitions,
main part of config, do forward= -->
<action-mappings>
<!-- Should map to our action in page, and the bean
to use, and where to go next -->
    <action path= "/searchAct"
            type= "app.SearchAct"
            name= "SearchFrm"
            scope="request"
            input="/WEB-INF/pages/SearchPg.jsp">
        <forward name="searchPg" path="/WEB-
        INF/pages/SearchPg.jsp"/>
        <forward name="nameLstAct"
        path="/do/nameLstAct"/>
    </action>

```

```

<action path= "/nameLstAct"
        type= "app.NameLstAct"
        scope="request"
        input="/WEB-INF/pages/NameLstPg.jsp">
    <forward name="nameLstPg" path="/WEB-
INF/pages/NameLstPg.jsp"/>
    <forward name="nameZoomAct"
path="/do/nameZoomAct"/>
</action>

</action-mappings>
</struts-config>

```

So these are the settings concerning our FormBean. We will have a FormBean for each page.

There are also settings for actions. We will have an action for each page.

Note that “web.xml” will forward “/do” to our action servlet. The action servlet will read “struts-config.xml” to determine which class will handle the action.

- Our “index.jsp” above asks for a “/do/searchAct”.
- The request is forwarded to the Struts action which determines the handler class. In this case, the class is “app.SearchAct”.
- So when we click on the ‘Search’ link in “index.jsp”, we will ask “app.SearchAct” to handle that request via its perform() method. Most of the time, perform() simply forwards the request to a particular page. Reading “struts-config.xml”, “app.SearchAct” can forward to a “searchPg” under “/pages” or ask for another action “/do” of ‘nameLstAct’.

Looking at this, you can conclude that the “app” folder, which hosts our controller classes, will decide whom to forward to, based on logic contained within the controller classes themselves.

Again, most of the time the logic is simple and it just forwards. Even when it is very simple, we call the pages by the controller. The controller could do some validation of data or find out what was entered and, based upon that information, decide what the appropriate page is.

- Create a simple action class “..\WEB-INF\classes\app\SearchAct.java”:

```
package app;

import java.io.*;
import java.util.*;
import javax.servlet.*;
import javax.servlet.http.*;
import org.apache.struts.action.*;

import data.*;

public class SearchAct extends Action {

public ActionForward perform( ActionMapping mapping,
ActionForm form,
                HttpServletRequest request,
                HttpServletResponse response)
                throws IOException,
ServletException
{
    return(mapping.findForward("searchPg"));
}

} // perform Act
} // class
```

Note that the only method we have is “perform()” and that it has only one line, forwarding to “searchPg”.

So “struts-config.xml” tells us to go to “\pages” to display this request and the page name to display.

- Create “SearchPg.jsp” in “...\WEB-INF\pages” folder.

```
<%@ page language="java" %>
<%@ taglib uri="/WEB-INF/struts-html.tld"
prefix="html" %>

<head>
<title> Search </title>
<html:base/>
</head>

<body>
<html:form action="/searchAct">
<p>First Name:
<html:text property="fstName"/><p>
<p>Last Name:
<html:text property="lstName"/><p>

<html:submit property="submit" value="Submit"/>

</html:form>
</body>
```

This is a simple use of the <html:> tags to allow us to enter a property. We are planning to use this page to enter the name we want to search our database for and it will be stored in a JavaBean.

- Review the “struts-config.xml” again to see which FormBean to use. Form is under action, and class is under FormBean.

- Create a JavaBean to handle the data (“...\WEB-INF\classes\data\SearchFrm”).

```
package data;

import org.apache.struts.action.*;

public class SearchFrm extends ActionForm {
    // this data layer does not go to RDBMS yet

    // we just need to store the search parms
    private String fstName;
    private String lstName;
    // beans have getters and setters so

    public String getFstName() {

        return (this.fstName);
    } // get

    public String getLstName() {
        return (this.lstName);
    } // get

    public void setFstName(String aFstName) {
        this.fstName=aFstName;
    } // get

    public void setLstName(String aLstName) {
        this.lstName=aLstName;
    } // get
} //class
```

For this simple bean, we have two (2) properties with accessor and mutator methods. Note how they map to the view.

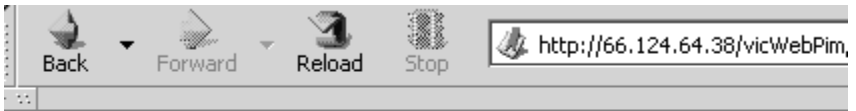
So to review, “SearchPg.jsp” asks for the name to search in our application. The controller forwards to a page which binds data

to a bean. This is the foundation of the Struts framework.

Also, look at the “SearchPg.jsp” and note that it returns control to ‘SearchAct’, our application, when the user clicks on the Submit button. So it is like a little loop. Note that we have removed data access and application control from the page.

From now on, there should be no need for Java code in pages. We can put that code in our action, or maybe we can write our own application custom tags.

So we can enter data now.



First Name:

Last Name:

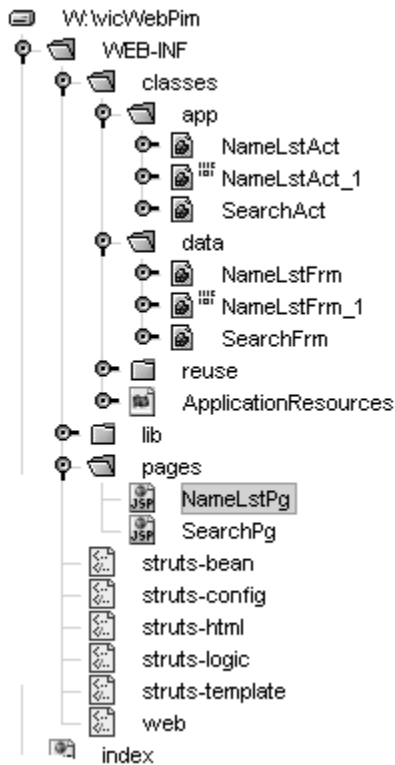
Make sure you take a coffee break here after you get this to work.

Messages Lab

So we have data in our bean and our controller can now access

it. We need to be able to pass this data to the next page that will eventually display our retrieval.

This is what our classes will look like at the end of this lab:



Let's update our controller to take a peek at the data and look at the "struts-config.xml" again. It says that our "SearchAct" controller can forward to two (2) different places, so the controller should decide where to go.

- Here is the modified “SearchAct.java” with if / then logic:

```
package app;

import java.io.*;
import java.util.*;
import javax.servlet.*;
import javax.servlet.http.*;
import org.apache.struts.action.*;

import data.*;

public class SearchAct extends Action {

public ActionForward perform( ActionMapping mapping,
ActionForm form,
                        HttpServletRequest request,
                        HttpServletResponse response)
                        throws IOException,
ServletException
{

boolean data=false;
HttpSession session = request.getSession();

// let's decode the bean
String fn= ((SearchFrm)form).getFstName();
String ln= ((SearchFrm)form).getLstName();
// do we have data yet?
if ((fn!=null) | (ln!=null)) {
    session.setAttribute("SearchFrm", form);
    data = true;
    System.out.println("Search"+fn+ln);
} //if

// control were to go!
if (data)
    return (mapping.findForward("nameLstAct"));
else
```

```
        return (mapping.findForward("searchPg"));
    } // perform Act
} // class
```

In the first case above, since we had no data, it sent it to a page for us to enter data, and then it got back the control.

Now, we have data, and it is captured in our bean.

So our controller can now take a peek at that data, and forward to another controller.

- You can see that one of the arguments passed to `perform()` is the form bean. This is done by Struts. We can now access the properties of that bean to see if there is anything there.
- So if there is none, we have our first case and we go to the page to enter it.
- If we have data, we should go on to our next page, the page that will display the retrieval data later.

Recall that we should ask the controller to display our page. Sometimes one controller controls many pages.

In the second case, we go to the 'nameLstAct' controller. According to "struts-config.xml", the class is stored in the "app" folder as 'NameLstAct' and we already have data in our other bean. However, the bean for the "NameLstPg.jsp" page will have different fields. This means it will need a different Form bean. Each page should have its own form bean and its own controller.

So we have data from one bean that needs to go to another bean. Our Action `perform()` should re-map it, and then display the data we will be searching for, just to make sure that we can

pass data from one page to another page.

- Here is the code for our second controller (“...\\WEB-INF\\classes\\app\\NameLstAct.java”).

```
package app;

import java.io.*;
import java.util.*;
import javax.servlet.*;
import javax.servlet.http.*;
import org.apache.struts.action.*;

import data.*;

public class NameLstAct extends Action {

public ActionForward perform( ActionMapping mapping,
ActionForm form,
                                HttpServletRequest
request, HttpServletResponse response)
                                throws IOException,
ServletException
{
    HttpSession session = request.getSession();
    boolean data=false;
    // let's decode the bean
    NameLstFrm nl = new NameLstFrm();
    SearchFrm sf = new SearchFrm();
    String fn=null;
    String ln=null;
sf=(SearchFrm)session.getAttribute("SearchFrm");
    // do we have data?
    if (sf!=null){
        // remap beans
fn= sf.getFstName();
ln= sf.getLstName();
nl.setSearchFstName(fn);
nl.setSearchLstName(ln);
        session.setAttribute("NameLstFrm", nl);
    }
}
```

```

        data = true;
    }//if
System.out.println("NameLst"+fn+ln+nl.getSearchFstName()+nl.getSearchLstName());

// control were to go! Nowhere for now
if (data)
    return(mapping.findForward("nameLstPg"));
else
    return(mapping.findForward("nameLstPg"));

} // perform Act
} // class

```

So we will get our bean out of the session, create a new bean, and then put it in the session. We are doing our application, step-by-step, so the two (2) beans are similar, but only for now. In the next chapter, we will go to the database to perform the search for the requested names.

- Here is our first cut at our second form bean (“...WEB-INF\classes\data\NameLstFrm.java”).

```

package data;

import org.apache.struts.action.*;

public class NameLstFrm extends ActionForm {
    // this data layer does not go to RDBMS
    // so it's a dummy bean

    // we just need to display the search parms, so we
    know what we are searching for
    private String searchFstName;
    private String searchLstName;
    // beans have getters and setters so

    public String getSearchFstName() {

```

```

        return (this.searchFstName);
    } // get

    public String getSearchLstName() {

        return (this.searchLstName);
    } // get

    public void setSearchFstName(String aFstName) {

        this.searchFstName=aFstName;

    } // get

    public void setSearchLstName(String aLstName) {

        this.searchLstName=aLstName;

    } // get

} //class

```

- And our second JSP page (“...WEB-INF\pages\NameLstPg.jsp”):

```

<%@ page language="java" %>
<%@ taglib uri="/WEB-INF/struts-html.tld"
prefix="html" %>
<%@ taglib uri="/WEB-INF/struts-bean.tld"
prefix="bean" %>
<head>
<title> Names Lst </title>
<html:base/>
</head>

<body>
<!-- we have beans in the session, set by perform() -
->
<p>

```

Search for:<p>

```
<bean:write name="NameLstFrm"  
property="searchFstName" /><p>
```

and ..

```
<bean:write name="NameLstFrm"  
property="searchLstName" /><p>
```

Results:

```
<!-- I wonder if we can get some DB data here later ?  
-->
```

Note that we are using a <bean:write> tag and not the <html:text> tag this time, since we are just displaying data.

Review

We have created our first Struts pages and we now know how to create and read the “struts-config.xml” file. Our Struts folders (“data”, “pages”, and “app”) are consistent with our MVC approach.

Chapter 8: Setup RDBMS

Goals

Install a relational database management system (RDBMS), create a data model, and load sufficient data so that our development is realistic.

RDBMS

There are many good SQL servers out there. Some are expensive to operate year after year if you have several large RDBMS servers. PostgreSQL is fast under large load. Let me define a large RDBMS as something with thousands of concurrent users and gigabytes of storage. An RDBMS where the database can fit on a laptop is not large. PostgreSQL is also free for commercial use for both development and deployment.

If you have a large load and require multiple RDBMS servers in production and have a separate QA and development environments, deploying an open source RDBMS could add up to significant savings, especially if you compute the costs over several years.

You may use another RDBMS server if you like. I have found that MS Access, MySQL and mSQL are very slow and hard to work with at times. There are dozens of other viable RDBMS packages, including Sybase, Oracle, MSSQL, DB2, and Informix.

Let's set up PostgreSQL on Windows. New developers are

more comfortable on Windows, but development and operations on Linux and Unix is much more efficient, so if you are comfortable with one of those, please use Linux or Unix.

PostgreSQL comes with most distributions of Linux. You can also run PostgreSQL using Cygwin under Windows. Also, Intel-compatible machines can be faster and cheaper than other platforms, such as SPARC and HP, according to the reports from www.tpc.org, an independent testing group. The RDBMS server should be a rack mounted machine with many caching SCSI controllers to optimize performance.

You should always run your RDBMS on a separate machine, even in development. However, if you only have a single machine, you can still learn and develop applications, but keeping in mind that the behavior of your system is not indicative of a real production environment.

If you have not done so at this time, please download and install PostgreSQL. Consult the Download Appendix if you have problems.



figure 8.1

- Start the Cygwin bash shell by following the image in figure 8.1. The following commands may be slightly different depending upon your configuration.
- Create the “pimDB” database.

WINDOWS :

```
cd /cygdrive/d                * get's to D drive
cd dbs/postgreSQL/bin        * and the right folder
createdb pimDB                * and create the db space
called pimDB
```

LINUX:

```
cd /usr/local/pgsql/bin
createdb pimDB
```

- Start the PostgreSQL server.

WINDOWS: postmaster -I -S -D ../Data (in /d folder)

LINUX: /usr/local/pgsql/bin/postmaster -i (as non-root)

- [WINDOWS] Start the Zde Database Explorer (Windows; figure 8.1). Connect to “pimDB” using “sysdba” as both user and password. These are the default administration passwords. You should create another user to be the “owner” of the “pimDB” and change the admin password. In general, avoid using the “admin” user for any database development.
- [WINDOWS] Create an ODBC connection. (Optional create ODBC short cut; figure 8.2). In the System folder, click Create New, scroll down to PostgreSQL, and fill out the form, or the IP address (figure 8.3).

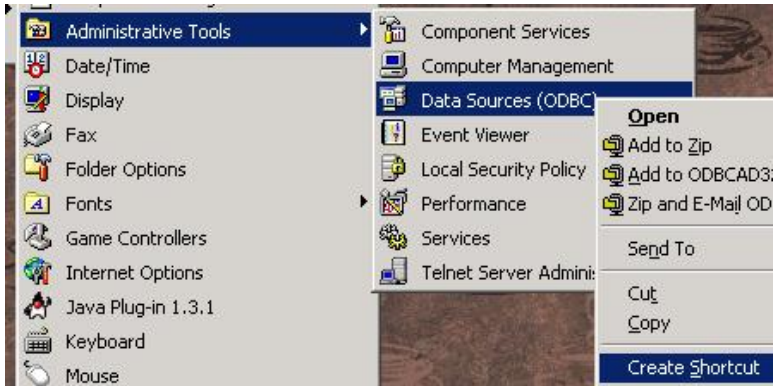


figure 8.2

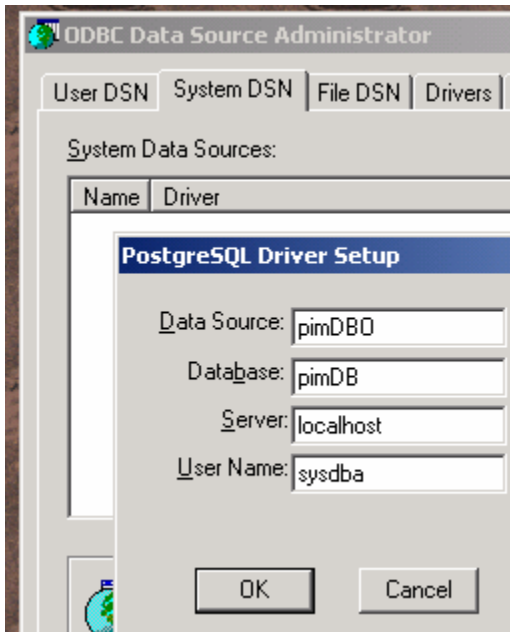


figure 8.3

Note that I used “localhost” for the Server, but you should have an IP address or alias for your development database server (you should not use the “admin” user).

If you want to setup your database server on the same machine as your development environment for convenience as you study, feel free to do so. However, if you are developing a real application, you should partition out the database server onto a separate server.

This is just an example to guide people who are not familiar with ODBC. Later we will configure a Java Database Connectivity (JDBC) driver, but first, let’s see if we can get the ODBC-based administration tools to work.

- Change advanced driver settings:

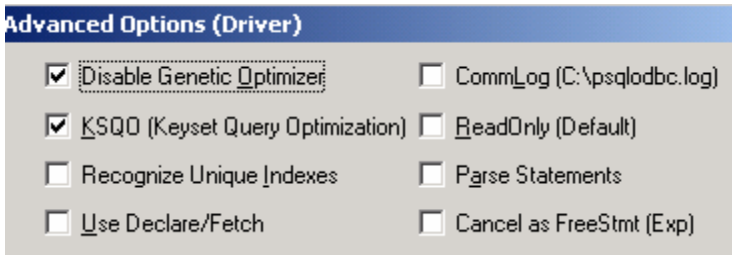


figure 8.4

and

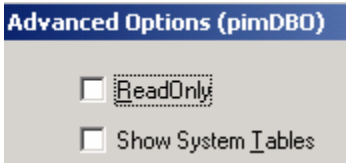


figure 8.5

so that “ReadOnly” is unchecked.

Optional: Using a third party interactive SQL tool, test the ODBC connection to PostgreSQL. Feel free to use any other ODBC-based tool, such as ERWin, Visual Basic, Visual Foxpro, Delphi, etc.

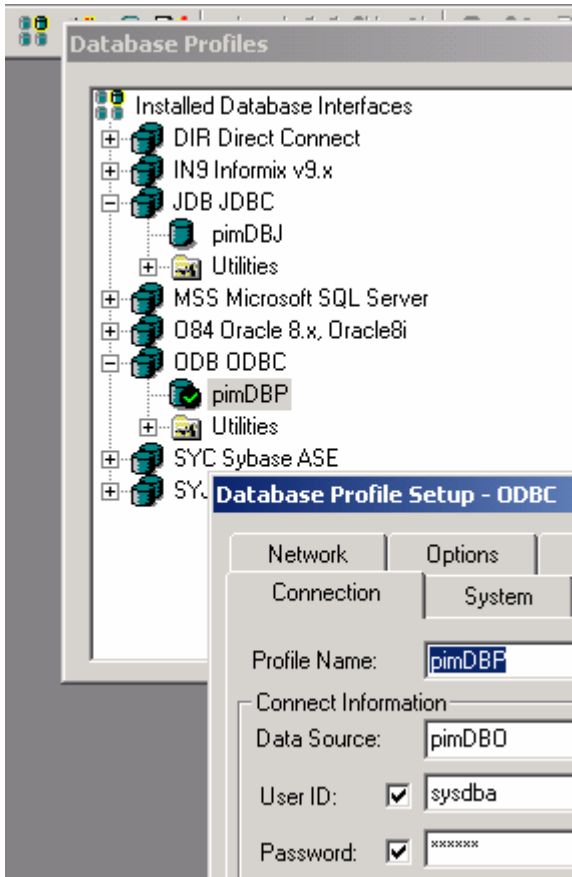


figure 8.6

It is important that you establish ODBC connectivity to the database so that we can create our tables, load data, and administer our RDBMS.

Also, I do not use Java for any batch database processing or interfaces to external systems or loads. When I clean, format, or validate data (a.k.a. data scrubbing), I do it in a flat file format such as .DBF (dBase) or a delimited text file. RDBMS, via SQL, can't handle large bulk loads or large scale bulk operations, such as nightly loads of millions record with field validations, such as zip code.

- Install “pgAdmin”. Make sure you download an additional ODBC driver (MAC) from <http://www.microsoft.data>. See Appendix on download.



figure 8.7

The “pgAdmin” tool should help you, in addition to “psql.exe” (ISQL) and ZDE Database Explorer to administer the RDBMS.

You should be able to connect to the database from other Windows applications. If not, you will have a really hard time

connecting with JDBC.

Query Performance Engineering

Do you know how to use a word processor? Good. But you realize that knowing how to use a word processor does not necessarily make you a novelist, right?

Let me ask you another question: If you read a book on surgery, would you tell people you were a surgeon?

My point is that just by reading a few books, you don't necessarily make you an expert, or even proficient, in a particular field. This is true in software as well.

Application designs should be done by software engineers with extensive database query performance experience. Knowing how to use a data modeling tool does not make you a designer. Simply because your data model looks nice in the data modeling tool does not mean that it will perform in a production environment.

Learning how to design data models is not straightforward. What works in development sometimes does not work in production. A rowboat in harbor handles differently than an oil tanker on the open sea.

I say this because web application architecture is limited in performance by the database design. A web application can have more than 1 million concurrent users at peaks, with a terabyte of storage, and your goal is to have a high traffic web site with sub-second response time. What happens if one million users do a query with an ORDER BY on a table with 50 million records at about the same time? Don't worry, our design will handle these conditions.

Please note that you will find that number of rows, indexes, type of hardware configuration, or number of concurrent users impact performance somewhat, but the major impact to performance are JOINS and therefore the data model design. So to avoid having to use EXPLAIN to determine how the query is being handled by the database engine, you should design with performance in mind.

Engineering Design Lab

- We have our reports and the outputs of our application. Spend some time thinking about a simple entity design. Simple designs are the most flexible and optimal for performance under load.
- Design a data model that will let us report on CONTACT names, a contact history for a particular name, and a list of issues for a NAME. (HINT: NAME is master, CONTACT is detail)
- Implement that data model in your “pimDB” database created above, using an ODBC-based tool of your choice. Do not create the tables using the ADMIN ID, use another user ID in the RDBMS.
- Do you have a primary key (PK) for each table? IMPORTANT: Your PK should always be a single numeric column and it should be an RDBMS-generated autoincrementing number.
- Do you have a foreign key (FK) from CONTACT to NAME?

- Create a few records in each table.
- Are the PK sequence numbers working?
- Make sure the CONTACTS table is working.
- Write some simple SELECTs using ZDE Database Explorer without using the GUI SELECT Designer. Try both INNER and OUTER joins.

Engineering Design Lab Solution

Your tables should look better than those presented below with a few more fields. Do not use the same field names as below, this is your PIM application so use your design. Note that you should be able to join the CONTACTS table to the NAMES table. You wrote the requirements, so match them to your requirements for the PIM. I do not add any constraints or relationships so that batch loads are easier and to keep the performance high.

```
create table ISSUES
(
    PK SERIAL not
null,
    ISSUE VARCHAR(80) null,
    WHO VARCHAR(27) null,
    ISSUE_TYPE_CDE CHAR(12) null,
    DUE_DAT DATE null,
    STATUS_CDE CHAR(12) null,
    APRVD_FLF CHAR(3) null
);
```

```
create table NAM
(
    PK SERIAL not null,
```

```

        NAM          CHAR(12)          null,
        LS_NAM       CHAR(15)          null,
        SRC_CDE      CHAR(12)          null,
        PHONE        CHAR(20)          null,
        EMAIL        CHAR(15)          null,
        DUPE_HASH    CHAR(20)          not
null
    );

create table CONT
(
    PK          SERIAL          not null,
    NAM_KEY     INT4            null,
    DT          DATE            null,
    QASKED_CDE  VARCHAR(60)     null,
    NXT_RESP    VARCHAR(240)     null,
    NOTES      TEXT            null
);

```

Data

We need much more data in order to write a real system. We have added a few records, but RDBMS might appear fast with even thousands of records (that fit in the RAM cache) during development and we won't see issues until the system is in production.

In production, there might be 100 million records and a query that runs one way with thousands of records executes differently with millions of records. It is not uncommon to have a terabyte of data in a web RDBMS. Since hard disk storage is relatively inexpensive, more and more data is being stored and accessed.

So we need to load our development RDBMS with records. For your web application, you need to figure out how much data will

be stored after the system is in production for a few years and use those numbers. You will get very little benefit if you do not load the RDBMS with production-sized data.

For this PIM project, we will load the table that will hold the list of names with at least 100,000 records. This is a relatively small database, but it should work as a bare minimum for training purposes.

Let's say we want to locate a person named "Jones" or cache resulting rows in our application.

What if there are 5,000 people named Jones?

For this lab, you may have to go to a computer store and spend some money. I apologize for this but it is the best way to get a large database that you may use to upload sufficient data to test design and application architecture under load. Purchase a CD software that has a directory of names, or a directory of business names. Any large list of names will do. (If someone knows of a free large list of realistic "fake" names on the web to download, please let me know so I can post it on the web site). First name and last name are required, the other fields can be blank.

Data Lab

- Load the table that will hold a list of names with at least 100,000 records.
- Using an ISQL tool, write a manual INSERT command that will add one row to database manually.
- Export the list to a .DBF format from your CD of names.
NOTE: In a real production system, at this time we

would scrub and validate the list since flat files may be processed quickly in batch mode. We typically validate the zip code and address and remove duplications, or similar work.

- Create a temporary work table matching the DBF layout and field types. We never import directly to a real table, so that we can audit the bulk load data.
- Import the DBF names list using software of your choice to the work table in “pimDB”.
- SELECT the work table wrk_nam INTO real nam table. I also hardcode a SRC_CODE (source code) column which, in my model, tracks the batch that loaded the names. This is a source code of the name, since sometimes the day after the load into a production database we have to roll back a bad input list. An example would be a SRC_CODE of “B08272001” (Batch on Aug 27th 2001). I also create a HASH code so I can remove any duplications from the flat file before production load.
- Did the PK sequence number work? How many records did you load?
- Shutdown RBDMS and restart it.

It might take over 30 minutes for each load step. Later, when you are in production, this is what the restore might look like, in case of an RDBMS crash.

Data Lab Solution

Please note that this is only one version of a solution and that

there are many. If you can, come up with another way to batch load data into a work table in “pimDB”.

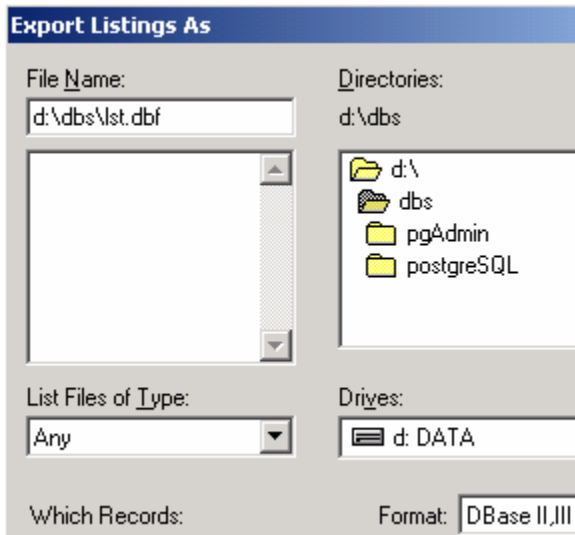


figure 8.8

- Generate a large DBF list to “d:\dbs” (or whatever temporary storage directory) and then create a new ODBC driver pointing to that directory:

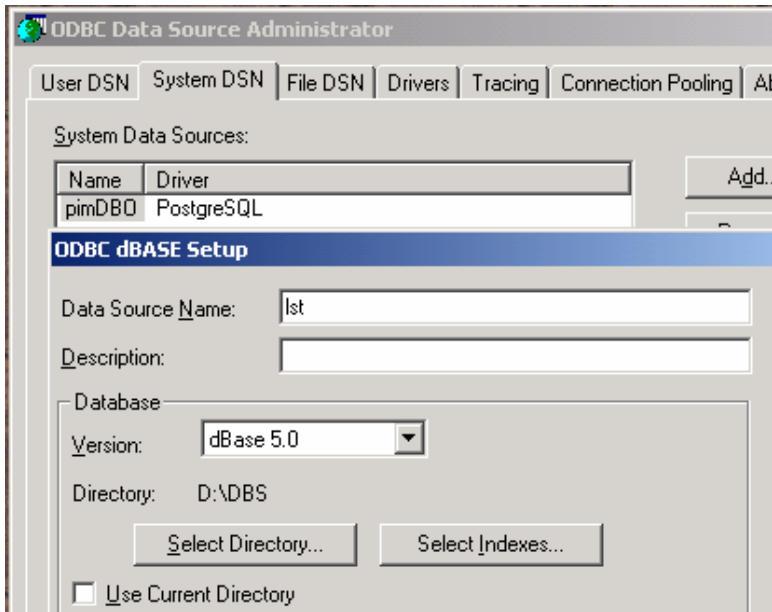


figure 8.9

- The “pgAdmin” program is one way to load the data from DBF to SQL using the migrate tool. In figure 8.10, we create a new table that will receive the records.

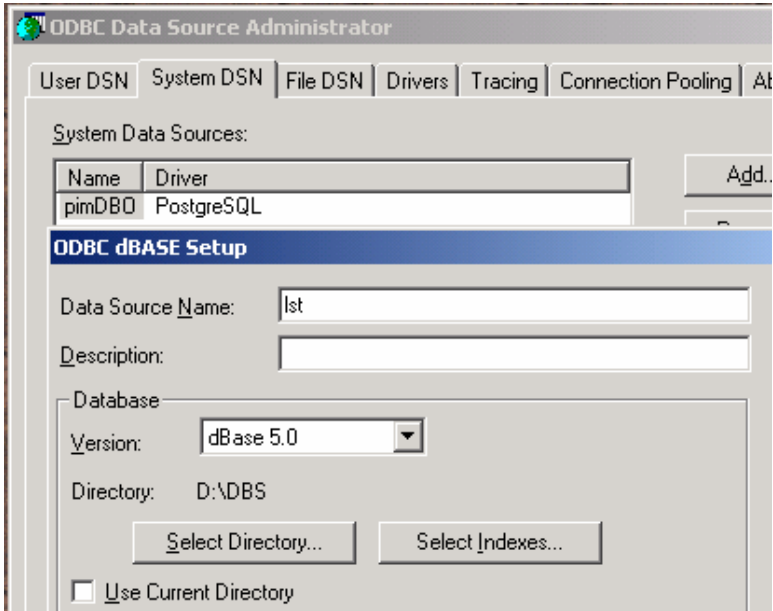


figure 8.10

- So now we have a work table “lst” in the RDBMS. Enter the SRC_CDE if you would like. Using my fields, the INSERT may look like this:

```
insert into NAM ( NAM, LS_NAM, SRC_CDE, DUPE_HASH)
select lastname, firstname, 'B082801', lastname from
LST;
```

- You can now drop the table “lst”, and delete the temporary DBF file.
- Query “SELECT count of records FROM NAM”. You should have more than 100,000 records.

And we have a database! Do not go ahead without a large RDBMS.

JDBC Connect

A list of drivers that supports RowSet in JDBC 3.0 are listed at <http://industry.java.sun.com/products/jdbc/drivers>.

Other JDBC drivers may or may not support JDBC 3.0, but most support only ResultSet and not the RowSet interface which is needed for the framework used in this book.

Since JDBC 3.0 is a new standard, some of the JDBC drivers claim to be compliant, but may be only partially so. For instance, there are drivers that claim to be compliant but still do not support the RowSet interface so do not take the word of the marketing on the driver's website, make sure that the driver works before committing.

Now let's connect using Java to our database. We will need JDBC drivers. You can get them from the Cygwin install in `d:\cygwin\use\share\postgresql\java` or <http://jdbc.postgresql.org/download>.

- Copy the JDBC driver downloads to the “d:\dbs” folder for now and make sure that you have added the driver jar(s) to your CLASSPATH in your OS and in the IDE (if necessary).
- Later you should copy the JDBC driver jar to the “...\webapps\WEB-INF\lib” of your application.

Of course, if you are using another database, and not PostgreSQL, you will need the appropriate JDBC-compliant drivers for that database.

For example, Sybase ASA uses jConnect.

The “D:\dbs” folder should look something like this.

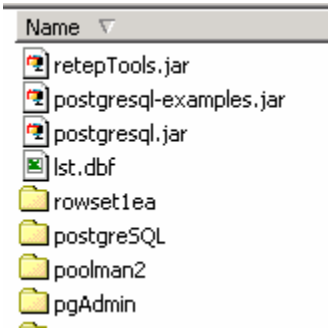


figure 8.11

Now let's do a client-side Java connect to the “pimDBS” server. We will do a server-side connection a bit later.

- Start NetBeans or the IDE of your choice. NetBeans is a free IDE and runs on any OS, since it is Java-based.
- In NetBeans, right click on the Explorer so you can mount some directories and access files. Mount the “tomcat” directory.

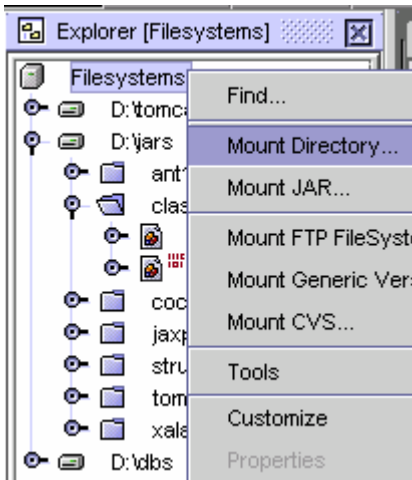


figure 8.12

- Create a stand-alone application “TestPIMDB” “d:\jars\classes” and add it to our CLASSPATH.

Listing 8.1 - TestPIMDB stand-alone source

```
import sun.jdbc.rowset.*;
// get it from Java Developer's Connection
// or look at Appendix Download
import java.sql.*;
import org.postgresql.*;

public class TestPIMDB {

public static void main (String args[])
{
try
{
```

```

CachedRowSet crs = new CachedRowSet();
// row set is better than ResultSet,
// configure driver, read more in book JDBC
Database Access :
Class.forName("org.postgresql.Driver");
crs.setUrl("jdbc:postgresql://localhost:5432/p
imDB");
// yours should not be localhost, but
//an IP address of DBS server. The 5432 is the
IP port
// you should never use DBA as owner/creator
of table
// and another user id for
access!
crs.setUsername("sysdba");
crs.setPassword("sysdba");

//select
crs.setCommand("select NAM from NAM where PK =
?");
// use your field names
crs.setInt(1, 8);
// pass the first argument to the select
command to
// retrieve PK id of Name #8, the 8th name
entered
crs.execute();
crs.next();

//get the field value
String nam = crs.getString("NAM");
System.out.println(nam);

} // try
catch (Exception e) {
    System.out.println(e);
} //catch
} // main
} // class
//Optional: Make the program take an argument

```

```
//of the PK for the record it should retrieve
```

I assume most of you have had some experience with JDBC in the past. If not, study the above code and make sure you have an understanding of how the code works.

- Compile “TestPIMDB.java”.
- Run “TestPIMDB”. (`java TestPIMDB`)

Looks like our JDBC driver to our “pimDB” works! In a later chapter, we will create a connection pool and beans. Remember the JDBC URL above.

Select Lab

- Execute:

```
select * from nam where nam >= 'A' and ls_nam >= 'A'  
limit 40
```

Note how the query is stopping after 40 rows, otherwise we would get the entire list of 100,000 or millions of rows. Look up help on “limit” in PostgreSQL.

Let’s create a sorted index on `ls_nam` and `nam`, so we can search on those 2 columns without having to do an `ORDER BY`. An `ORDER BY` sorts the results before returning, but if 1,000 users issue a `SELECT` with an `ORDER BY`, it will slow down the database server.

If we have a sorted index that contains all the columns in `select`, it will not need to sort, but it will return the result in sorted order. This is because the optimizer never leaves the index to get the data.

- Create an index on ls_nam, nam. (Last Name, First Name).

Table to Index: nam

Index name: nam_idx

Columns to Index:

- dupe_hash
- email
- ls_nam
- nam
- phone

Unique Index?

Index Type: btree

Create Index

figure 8.13

Note that if we create a unique index on some hash code, no duplicate records will be allowed. I use a hash code of last name, plus postal code, plus e-mail.

The DBA should not be the owner of the tables, and another ID should connect to the database server to access the tables. Maintaining backups to DBF (or some other stable storage format) is recommended so that you can upgrade or recover.

Extra Credit - Super Type

You will learn with experience that the fewer tables you have in your design, the faster your application will be. If you want to test this, try many joins with large tables and multiple of users.

Here is another tip. You might have “issue_code” type field in your table to track whether the issue is a bug, task, goal, etc. So you might design a table to list available codes (bug, task, goal) and call those issue codes. But there might be another list you need for “status_code” (open, assigned, approved, etc.).

- Is that 2 tables?
- How many codes would you have in a data model?
- Do you need a table for each?

You could have a single type table having three (3) fields: TYPE_TYPE, CODE, DESC. CODE will list available codes, and DESC any description. The TYPE_TYPE field will list STATUS if the row is for STATUS, ISSUE if the row is for ISSUES, with the PK being (TYPE_TYPE, CODE). No matter how many codes you have, you still have only one (1) table.

Review

If your application will require processing of large data sets, make sure that you have ample data to put the application under load during development and testing.

Chapter 9: Data Retrieval

Goals

We have a search form, using MVC, but we are not talking to the database yet. Retrieve data from RDBMS using the connection pool.

Data Bean Lab

We have previously done simple applications and servlet database retrievals. Now it is time to plug that into our Struts framework.

Our page already receives retrieval arguments, so we will need to add code to get retrieval data from our connection pool to the 'NamedLstFrm' class.

Note that the example code I am using refers to my data model for the NAMES table. Your fields and field names will be different.

It will look long and complex, but it is all the code you have already seen and done.

- Here is the modified "NameLstFrm.java".

```
package data;
import org.apache.struts.action.*;

// does this mean we'll be getting data?
```

```

// to talk to our connection pool,
//note no info on JDBC driver, that's in the pool
import com.codestudio.util.*;
import com.codestudio.sql.PoolMan;
import java.sql.*;
import javax.sql.*;
import javax.naming.*; // for look up
import sun.jdbc.rowset.*;
import java.util.*;

public class NameLstFrm extends ActionForm {

private String searchFstName;
private String searchLstName;

private String fstName;
private String lstName;

private Connection con=null;
private DataSource ds;
// we'll use crs a lot
public CachedRowSet _crs;

public void dbCon() { // connect to db
    try{ ds = PoolMan.findDataSource("jndiPMIdb");
        //we should in production give it a password in
code not in file pool.xml
        con=ds.getConnection();
        System.out.println("XXX We Coned"); } // try
    catch (Exception e) {
        System.out.println(e);
        e.printStackTrace();} // catch
} // con

public void dbDisCon()
{
try{if (con!=null) con.close();
    ds=null;
    System.out.println("XXX We DisCo"); // isn't
there a better way to debug?
}
}

```

```

    catch (Exception e) {System.out.println(e);
} //catch
} //disCon

public int retrieve(String aFst, String aLst, int
arowCount) {
try{
    _crs = new CachedRowSet(); // easier then
resultset
    // you must test this sql string manually in a
SQL IDE
    crs.setCommand("select NAM, LSNAM from NAM where
nam >= ? and ls_nam >=? limit ?");
// note how we are retrieving columns in index and
not *
// add at most 2 more columns for your lab.

    _crs.setString(1,aFst);
    _crs.setString(2,aLst);
// to limit the number of rows coming back, we
only need a page at a time
    _crs.setInt(3,arowCount);
    _crs.execute(con);
    _crs.next();
    System.out.println("XXX We got data, it is: ");
    System.out.println(_crs.getString("LS_NAM"));
} // try
catch (Exception e)
{ System.out.println(e);
  e.printStackTrace();} // catch
return 0;
} // retrieve

// this is a bean after all
public String getFstName() { //get it from _crs
RowSet
    try{ this.fstName = _crs.getString("NAM");
} // try
    catch (Exception e)
{System.out.println(e);} // catch
    return (this.fstName); } // get

```

```

public String getLstName() { //get it from _crs
RowSet
    try{ this.lstName = _crs.getString("LS_NAM");
} // try
    catch (Exception e)
{System.out.println(e);} // catch
    return (this.lstName);
} // get

public Hashtable rowCount()
{ //_crs.getRowCount() to do iterations later
return new Hashtable();
}
// old code for the search args
public String getSearchFstName() {
    return (this.searchFstName); } // get
public String getSearchLstName() {
    return (this.searchLstName); } // get
public void setSearchFstName(String aFstName) {
    this.searchFstName=aFstName; } // get
public void setSearchLstName(String aLstName) {
    this.searchLstName=aLstName; } // get
} //class

```

The major new code blocks are **BOLDED**. This is our new “NameLstFrm” class that will retrieve data from the RDBMS.

Before we start displaying the data in a page, let’s take smaller steps. We will create a simple servlet that will test the data bean only, outside of the framework.

- Create a servlet “BeanTest.java” in the “...WEB-INF/classes” folder.

```

import javax.servlet.*;
import javax.servlet.http.*;

```

```

import java.io.*;
import data.*;

public class BeanTest extends HttpServlet {

protected void doGet(HttpServletRequest request,
HttpServletResponse response)
    throws ServletException, java.io.IOException {

// if the data bean does not work in a simple
enviroment ....
NameLstFrm nl = new NameLstFrm();

nl.dbCon();
// test the args manually first with the SQL string in
a SQL IDE
//select * from nam where nam >= 'A' and ls_nam >=
'A' limit 10
nl.retreive("A","A",40); //get's names starting with
'A',40 rows please
nl.dbDisCon();

// servlet stuff
response.setContentType("text/html");
PrintWriter out = response.getWriter();

out.println("<html>"); out.println("<head>");
out.println("<title>beanTest</title>");
out.println("</head>"); out.println("<body>");
out.println("Does this bean work?");

//get the field value
String nam = nl.getFstName();
out.println(nam);

out.println("</body>");
out.println("</html>");

} // doGet
} // class

```

- You also recall that when we do not work with pages, but work with a servlet, we have to tell our application server about the servlet in “web.xml”.

```
<?xml version="1.0" encoding="ISO-8859-1"?>

<!DOCTYPE web-app
  PUBLIC "-//Sun Microsystems, Inc.//DTD Web
  Application 2.2//EN"
  "http://java.sun.com/j2ee/dtds/web-app_2_2.dtd">

<web-app>

<!-- . . . -->

<servlet>
  <servlet-name>beanTest</servlet-name>
  <servlet-class>BeanTest</servlet-class>
</servlet>

<!-- . . . -->

<servlet-mapping>
  <servlet-name>beanTest</servlet-name>
  <url-pattern>/beanTest</url-pattern>
</servlet-mapping>

<!-- . . . -->

</web-app>
```

- Test out the SQL command in your ISQL first.
- Browse to the BeanTest servlet URL
“http://localhost:8080/<app>/beanTest”.

Page Data

Let's get the page to display the bean. Our action should ask the Form Bean to retrieve.

- Modify the action bean ("NameLstAct") to look like this new code in **bold**, added to the old "NameLstAct":

```
sf=(SearchFrm)session.getAttribute("SearchFrm");
// do we have data?
if (sf!=null){
    // remap beans
    fn= sf.getFstName();
    ln= sf.getLstName();
    nl.setSearchFstName(fn);
    nl.setSearchLstName(ln);
    session.setAttribute("NameLstFrm", nl);
    data = true;
    // new code
    nl.dbCon();
    nl.retrieve(fn,ln,10);
    nl.dbDisCon();
} //if
```

- And now, write out the answer we got from the form bean to the page. Add this to NameLstPg.jsp.

Results:<p>

```
First Name - Last Name - Phone - City - Postal Code -
eMail<p>
<bean:write name="NameLstFrm" property="fstName"/> -
<bean:write name="NameLstFrm" property="lstName"/>
<!-- there are is only first and last name here? -->
```

Since the form bean has the additional data now, we can display it.

Part II Lab

I am only getting the first name and last name. I would like for you to display some additional fields from your NAMES table, such as phone, email, or postal code. Also, maybe you should display the results in an HTML table so that they look more aligned.

Iterate

We are only displaying a single result and we should be able to display the entire result set. So instead of one name, display a list of names that match the search.

Struts has an iterate tag. Let's iterate through our result set.

- First, teach our form bean about iteration, by adding these methods:

```
public int getRowNext() { // get the next row
    try{
        _crs.next();
    } catch (Exception e) {
        System.out.println(e);
        e.printStackTrace();
    } // catch
    return 0;
} // rowNext()

public Hashtable getIterateMap()
{ //is there a better way?
    int rc=_crs.size();
    Hashtable iterateMap = new Hashtable();
```



```

        // count out the number of rows with dummy map
        for (int i=1; i <= rc; i++) { iterateMap.put(new
Integer(i),new Integer(i));}
return iterateMap;
}

```

Second, note that I named the methods get<X>, this way I can access them from a page. The getRowNext() method goes to the next row, so that my getNameXX() methods will do the same as well. The getIterateMap() method will return a dummy HashMap the same size as the ResultSet. In the page, we will iterate the Map, and go to the next row for each element in the Map.

- I also have some Java code in the page, showing the row, but you can omit the <%> code.

```

Results:<p>
<% int i=0; %>
Num - First Name - Last Name - Phone - City - Postal
Code - eMail<p>
<table>
<logic:iterate id="iterateMap" name="NameLstFrm"
property="iterateMap">
<tr>
<td><% i++; %> <%= i %> </td>
<td><bean:write name="NameLstFrm"
property="fstName" /> </td>
<td><bean:write name="NameLstFrm"
property="lstName" /></td>
<td><logic:equal name="NameLstFrm"
property="rowNext" value="" /></td>
</tr>
</logic:iterate>
</table>
</body>

```

See how you can use the bean methods?

The <logic:iterate> tag just repeats for each element. And the

rowNext() method is a dummy call to getRowNext(), which moves to the next row in “_crs” so that next time getFstName() and getLstName() get executed, it gets the next row.

Review

We have added real database access to our MVC application via a JNDI connection pool. Note that our connection was cached, but PoolMan can also caches a lot more. Also, our database has an index on our result set, so we do not need to sort the large database with an ORDER BY. If we had many users forcing the database to sort for each request, we would have scalability issues. And we are also limiting the maximum number or rows returned.

Chapter 10: Object Orientation

Goals

Leverage object-oriented (OO) and word-based programming for development productivity.

Object-Disoriented

What makes a good programmer?

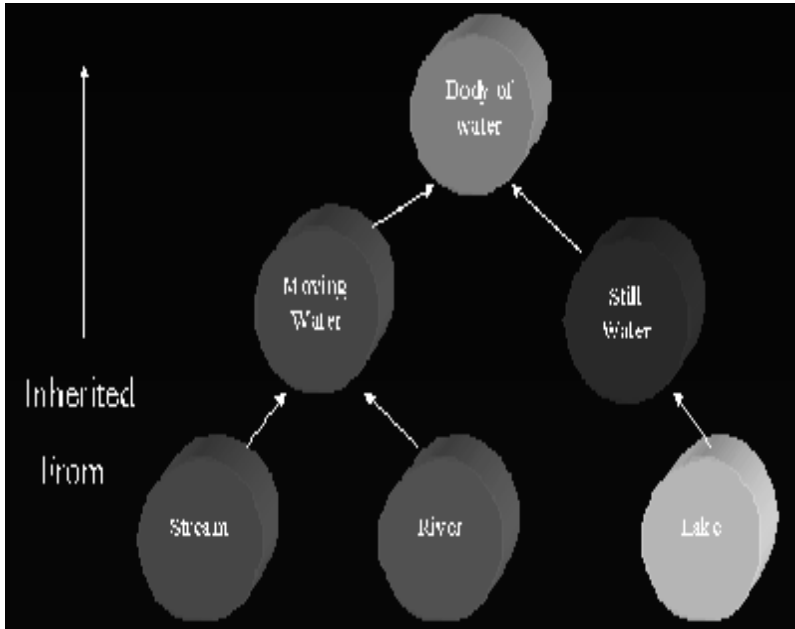
The same thing that makes someone a good employee, someone who does a lot of high quality work in very little time.

Does that mean that good programmers are fast typists, in order to crank out lots of code?

No, good programmers leverage their work, by implementing reusability, via object oriented programming.

Basically OO does not look at classes as apples and oranges, but rather looks at what is similar, and then creates a base class that the 2 classes (apples and oranges) can share.

The more code and functionality you put in the base class, the better.



Some people are familiar with skeletons or template code, basically cut-and-paste programming. The reason object-orientation is better is that if you have a base class, you may modify behaviour AFTER you have coded the concrete classes.

For example, let's imagine that we are nearly done with a project with 80 action classes and then new requirements are added to solve business problems.

If you do not have a base action class of your own, you would be forced to modify code in 80 places and each of the objects has slightly different code. This is object-disoriented programming.

Nothing in the Java language will prevent you from doing object-disoriented programming. Java is an object-oriented language but that doesn't mean that it makes every programmer an object-oriented developer.

Developers need formal training and possibly some mentoring to start thinking in object-oriented terms. If you were trained in top-down, or bottom-up, functional decomposition, it is like quitting smoking. You still think in terms of a flow chart. But one day, you will have an epiphany and begin thinking in objects.

At least that is how I learned it. At first, I only understood objects academically. I could define and develop polymorphic methods but didn't have a grasp on the whole picture.

Until, on a large project, we were able to refactor and our productivity was easily increased ten-fold or more. So, a single OO programmer can out-produce some ten fast-typing, object-disoriented coders, no matter what IDE the typers are using.

If in the above example you create a base class, which the 80 objects extend, you can just change the code in the base class, and all the 80 descendant objects will change.

An experienced developer leverages object-orientation into her designs because she knows that you cannot predict what will need modification later in the development lifecycle. A layer of indirection can solve almost any design problem, or, at the least, lessen the pain of correction.

Of course object-orientation does not make a project more successful, it just allows for greater flexibility and productivity, especially in larger development efforts. On a small project, leveraging objects may have little or no impact. The same may be said concerning frameworks in that the ramp-up time for smaller projects may be counter-productive in delivering an

application with minimal functionality.

If you are crossing a creek, you do not need a suspension bridge, but knowledge of bridge building will help you no matter the size of the span.

Object-Orientation Lab

The goal of this lab is to create our base action and our base form class in the “reuse” folder, and refactor the four (4) concrete objects.

- Create a new folder, called “reuse” under “...\WEB-INF\classes” directory. In the newly created “reuse” directory, create a new class (“SpaBaseAct.java”) that will be used as our base for all of our action classes.

```
package reuse;

import java.io.*;
import java.util.*;
import javax.servlet.*;
import javax.servlet.http.*;
import org.apache.struts.action.*;

import data.*;

// base call
public class SpaBaseAct extends Action {
    public ActionForward perform( ActionMapping mapping,
        ActionForm form,
                                HttpServletRequest request,
        HttpServletResponse response)
                                throws IOException,
        ServletException
    {
```

```

    // here we could add code before call
    // just rethrow the method:
        return spaPerform( mapping, form, request,
response);
    //here we could code after call
} //perform

//base classes will override this method and NOT
perform() anymore....
public ActionForward spaPerform( ActionMapping
mapping, ActionForm form,
                                HttpServletRequest
request, HttpServletResponse response)
                                throws IOException,
ServletException
{ // a dummy method here
    return new ActionForward();
} //spaPerform
} // class

```

Note that we have created a new method so later we can more easily modify the perform() behaviour for the entire application. Also note that spaPerform() has the same signature as the perform() method.

- Modify the two (2) action classes (SearchFrm; NameLstFrm) to extend the new action base class (SpaBaseAct), calling the spaPerform() method instead of the perform() method.

```

...
import org.apache.struts.action.*;

import data.*;
import reuse.*;

public class SearchAct extends SpaBaseAct {

```

```

public ActionForward spaPerform(ActionMapping
mapping, ActionForm form,
        HttpServletRequest request,
        HttpServletResponse response)
        throws IOException,
        ServletException
{
...

```

The rest of the code is the same. Make sure your application still runs.

- Examine the base class again and see how we have a code placeholder in the `spaPerform()` method should we need it.
- Let's now create a base for Form Beans (“...\\WEB-INF\\classes\\reuse\\SpaBaseBean.java”).

```

package reuse;
import org.apache.struts.action.*;

// does this mean we'll be getting data?
// to talk to our connection pool,
//note no info on JDBC driver, that's in the pool
import com.codestudio.util.*;
import com.codestudio.sql.PoolMan;
import java.sql.*;
import javax.sql.*;
import javax.naming.*; // for look up
import sun.jdbc.rowset.*;
import java.util.*;
import java.math.*;

public class SpaBaseBean extends ActionForm {

private Connection con=null;
private DataSource ds;

```



```

// we'll use crs a lot
private CachedRowSet _crs;

private String sqlString;
private BigDecimal PK;

public Connection getCon() {return con;}
public CachedRowSet getCrS() {return this._crs;}
public void setCrS(CachedRowSet aCRS)
{this._crs=aCRS;}

public SpaBaseBean () { // cons
    super();
    try{
        _crs = new CachedRowSet();
    } // try
    catch (Exception e) {
        System.out.println(e);
        e.printStackTrace();}
} //cons

public int exec(CachedRowSet pCR)
{ int c=0;
try{
    pCR.execute(getCon());
    pCR.next();
    c= pCR.size();
    System.out.println(c+"rowCount");
    setCrS(pCR);
} // try
    catch (Exception e) {System.out.println(e);}
// catch
    return c;
}

public void dbCon() { // connect to db
    try{ ds = PoolMan.findDataSource("jndiPMIdb");
        //we should in production give it a password
in code not in file pool.xml
        con=ds.getConnection();
        System.out.println("XXX We Coned"); } // try

```

```

        catch (Exception e) {
            System.out.println(e);
            e.printStackTrace();} // catch
    } // con

public void dbDisCon()
{
    try{if (con!=null) con.close();
        ds=null;
        System.out.println("XXX We DisCo"); // isn't
        there a better way to debug?
    }
    catch (Exception e) {System.out.println(e);
    } //catch
} //disCon

public int getRowNext() { // get the next row
    try{ _crs.next(); }
    catch (Exception e) { System.out.println(e);
    e.printStackTrace();} // catch
    return 0; } // rowNext()

public Hashtable getIterateMap()
{ //is there a better way?
    CachedRowSet cr = getCrS();
    int rc=cr.size();
    System.out.println(rc+"XX");
    Hashtable iterateMap = new Hashtable();
    // count out the number of rows with dummy map
    for (int i=1; i < rc; i++) { iterateMap.put(new
    Integer(i),new Integer(i));}

return iterateMap;
}

public void setSqlString(String aSQL)
{
    sqlString = aSQL;
}

```

```

// it is possible to write a more generic retrieve
later, but for now, each concrete class will just
have a retrieve
// public int retrieve(String aFst, String aLst, int
arowCount) {

} //class

```

Note that I have **bolded** some methods that might be generic for all beans. All beans connect and disconnect and might need next row or to iterate. But there is no need to code this behaviour in each and every bean since the base class code handles the base functionality.

- Now our new “NameLstFrm” is a bit simpler with only specific code here and the generic code is in the base class. Again, if there are only a few beans, the base class functionality would be of no significance, but if there are many beans, the benefits are readily apparent.

```

package data;

import sun.jdbc.rowset.*;
import java.util.*;

import reuse.*;

public class NameLstFrm extends SpaBaseBean {

private String searchFstName;
private String searchLstName;

private String fstName;
private String lstName;

public int retrieve(String aFst, String aLst, int
arowCount) {
try{

```

```

        CachedRowSet cr = getCrs();
        cr = new CachedRowSet(); // easier than resultSet
        // you must test this sql string manually for a
SQL IDE
        cr.setCommand("select * from NAM where nam >= ?
and ls_nam >=? limit ?");
        cr.setString(1,aFst);
        cr.setString(2,aLst);
        // to limit the number of rows coming back, we
only need a page at a time
        cr.setInt(3,arowCount);
        exec(cr);
    } // try
    catch (Exception e)
    { System.out.println(e);
      e.printStackTrace(); } // catch
    return 0;
} // retrieve

public String getFstName() { //get it from _crs
RowSet
    CachedRowSet cr = getCrs();
    try{ this.fstName = cr.getString("NAM"); }
// try
    catch (Exception e)
{System.out.println(e);} // catch
    return (this.fstName); } // get
public String getLstName() { //get it from _crs
RowSet
    CachedRowSet cr = getCrs();
    try{ this.lstName = cr.getString("LS_NAM");
} // try
    catch (Exception e)
{System.out.println(e);} // catch
    return (this.lstName);
} // get

public String getSearchFstName() {
    return (this.searchFstName); } // get
public String getSearchLstName() {

```

```
        return (this.searchLstName);    } // get
public void setSearchFstName(String aFstName) {
    this.searchFstName=aFstName;    } // get
public void setSearchLstName(String aLstName) {
    this.searchLstName=aLstName;    } // get
} //class
```

- And to finish up, change the other Form Bean to use our new base class (even if it does not need RDBMS).

Does everything still work?

Word-based Programming

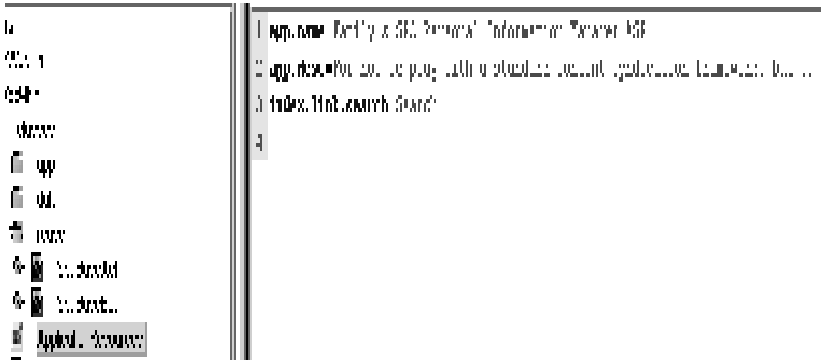
We can also re-use words in pages, making them easier to remap and edit. Instead of saying “First Name”, it is customary to code:

```
<bean:message key="field.name.FirstName" />
```

This way, the text can be easily changed, without having to edit a working JSP. This enables you to easily internationalize (aka i18n) your web applications but just changing the ResourceBundle as explained in Struts documentation.

In “web.xml” we pointed to the “ApplicationResources.properties” file, which is in the “classes” folder. It got placed there when we expanded “struts-blank.war”.

- Let’s edit it to look more like this for now.



- And remove all the text from index.jsp:

```
<%@ page language="java" %>
<%@ taglib uri="/WEB-INF/struts-html.tld"
prefix="html" %>
<%@ taglib uri="/WEB-INF/struts-bean.tld"
prefix="bean" %>

<html:html locale="true">
<!-- it looks more cryptic now, so more comments are
needed
This is a entry point to the application for non
members that shows up
by a search engine link
-->
<head><title><bean:message key="app.name"/></title>
<html:base/></head>

<body>
<p><bean:message key="app.name"/>
<p><bean:message key="app.desc"/>
<!-- <p> bean:message key="index.desc"/> -->
<!-- we should ask the application what to do, and
allow it to set up -->
```

```
<p><html:link page="/do/searchAct"> <bean:message  
key="index.link.search"/> </html:link>  
</bead>  
</html:html>
```

- In Tomcat, there is a “work” folder that is used for caching, so delete files in the “work” directory so the JSP gets recompiled and the “ApplicationResources.properties” file gets re-read, and then restart Tomcat.

Does the application work?

Word-based Lab

- Replace all the field names and any text (including button) with a reference to the application resources properties list.
- You should never have any text that displays in your JSP. All text should be in resource properties, with presentation style in XSL.

Review

If we leverage object-orientation, we can improve flexibility and productivity. We should also leverage the application resources property to display text instead of embedding text in JSP’s.

Chapter 11: XML from JSP

Goals

Practice skinable interface. Be able to print a real report for users with pagination.

HTML/CSS vs. XML/XSLT

Browsers have for years been able to accept XML. We have in the past sent CSS with HTML, but the future of web applications is XML and XSLT. In the near term, however, HTML will remain the medium for the mainstream.

Text Format

We have talked about using XML in browsers and not HTML.

This is a partial view of the HTML source.

```
<head>
<title> Names Lst </title>
<base href="http://66.124.64.38/vicWebPim/WEB-
INF/pages/NameLstPg.jsp">
</head>
<body>
<!-- we have beans in the session, set by perform() -
->
<p>
```

```
Search for:<p>
A -
A<p>
Results:<p>
Num - First Name - Last Name - Phone - City - Postal
Code - eMail<p>
<table>
<tr>
<td> 1 </td>
  <td>A </td>
  <td>A </td>
  <td></td>
</tr>
```

HTML is rather limited. It doesn't have pagination (the ability to control the pages, including headers and footers) and is not equipped to handle professional reporting needs.

We have done a simple style change. We will practice a radical style change here.

Printing

Printing requires that one knows about the print driver, printer capabilities, margins, fonts, etc., etc.; things that a word processor does.

RTF Lab

The Biblioscape web site (http://www.bblioscape.com/rtf15_spec.htm) has the RTF specification. Note that it has table and column support, headers, footers, so we can wrap.

- Create the “play.rtf” file using a text editor with the following content:

```
{\rtf1\ansi\defformat\deff0{\fonttbl{\f0\modern\prq
l Courier New;}}
\lndscpsxn\pgwsxn15840\pghsxn12240\f0\fs14
\par
\trowd\trgaph80\trkeep
\cellx5000
\cellx10000
\pard\intbl Row 1, Cell 1\cell
\pard\intbl Row 1, Cell 2\cell
\row
\trowd\trgaph80\trkeep
\cellx5000
\cellx10000
\pard\intbl Row 2, Cell 1\cell
\pard\intbl Row 2, Cell 2\cell
\row
\pard
\par
}
```

Rich text format (RTF) is an ANSI-standard mark-up language. RTF proponents prefer it to PDF since all platforms and most browsers have RTF reader capability built-in without having to install some third-party viewer.

The “play.rtf” document above contains the following meta information:

- I am an RTF document.
- I have a font 0 called Courier (fixed pitch).
- I want the page to be landscape.
- Use font 0 to font size of 14 measures.
- Open “play.rtf” in a word processor of your chose and print

it.

Can we set a MIME type to RTF and view in browser?



```
Source: [url]
<?xml version="1.0" encoding="UTF-8" standalone="yes" type="application/xml" ?>
<page contentType="application/xml" ?>
  <metaName>Hey, Vinny!</metaName>
  <field>Here we have a field.</field>
</page>
```

We may use XML and XSLT to transform the document to HTML for browsing and RTF for printing.

You can see that it is a good approach to generate XML from JSP's. By specifying which XSL to use, we can create different types of documents. Also, by adjusting XSL, we can change the look-and-feel of our entire application from a centralized location.

XML Lab

- Create the JSP below in preparation for converting a "live data feed" to XML.

```
<%@page contentType="text/xml"%>
<?xml version="1.0"?>
<?xmlstylesheet
href="http://66.124.64.38/vicWebPim/browse.xsl"
type="text/xsl" ?>
<page>
<metaName>Hey, Vinny!</metaName>
<field>Here we have a field.</field>
```

</page>

- Create the XSL file (browse.xsl) listed below:

```
<?xml version="1.0" ?>
<xsl:stylesheet version="1.0"
xmlns:xsl="http://www.w3.org/1999/XSL/Transform">

  <xsl:template match="metaNam">
    <h1 align="center">
      <xsl:apply-templates/>
    </h1>
  </xsl:template>

<xsl:template match="metaDesc">
  <h2 >
    <xsl:apply-templates/>
  </h2>
</xsl:template>

<xsl:template match="table">
  <table>
    <xsl:apply-templates/>
  </table>
</xsl:template>

<xsl:template match="tr">
  <tr>
    <xsl:apply-templates/>
  </tr>
</xsl:template>

<xsl:template match="td">
  <td>
    <xsl:apply-templates/>
  </td>
</xsl:template>

</xsl:stylesheet>
```

This is a style sheet. We can add FOP-based tags for font and style information, sort of like a sophisticated CSS.

FOP is an application distributed by the Apache-Jakarta project that reads an XSL formatting object tree and generates a PDF.

In your JSP, you must set up FOP tags that you will generate. XSL can then match them and add appropriate style. Make sure that the simple JSP works with browser-side XSL, otherwise when we do a complex table iteration in Struts, you will be stuck.

- Now, let's change our "NameLstPg.jsp" to use XML/XSL, something like this:

```
<%@page contentType="text/xml"%>
<%@ taglib uri="/WEB-INF/struts-html.tld"
prefix="html" %>
<%@ taglib uri="/WEB-INF/struts-bean.tld"
prefix="bean" %>
<%@ taglib uri="/WEB-INF/struts-logic.tld"
prefix="logic" %>
<?xml version="1.0"?>
<?xml-stylesheet
href="http://66.124.64.38/vicWebPim/browse.xsl"
type="text/xsl" ?>
```

```
<page>
<metaNam>Use Word Oriented plz</metaNam>
Search for:
<bean:write name="NameLstFrm"
property="searchFstName"/> -
<bean:write name="NameLstFrm"
property="searchLstName"/>
Results:
<% int i=0; %>
```

```
Num - First Name - Last Name - Phone - City - Postal
Code - eMail
```

```

<table>
<logic:iterate id="iterateMap" name="NameLstFrm"
property="iterateMap">
<tr>
<td><% i++; %> <%= i %> </td>
<td><bean:write name="NameLstFrm"
property="fstName" /> </td>
<td><bean:write name="NameLstFrm"
property="lstName" /></td>
<td><logic:equal name="NameLstFrm"
property="rowNext" value=" " /></td>
</tr>
</logic:iterate>
</table>

</page>

```

- Test it out in a browser to see that we are getting the same result.

From now on, send XML to the browser. HTML is not completely out of the picture, but the future is definitely heading towards XML. In the transition period, you may still need to generate HTML for your web applications.

XSL Demo

Now let's get the same XML output from "NameLstPg.jsp" to be transformed using a different XSL. And instead of rendering <TR> and <TD> tags, let's render the RTF escape strings.

- Here is our "new", XML-enabled version of the "NameLstPg.jsp".

```

<?xml version="1.0"?>
<?xml-stylesheet
href="http://66.124.64.38/vicWebPim/rtf.xsl"
type="text/xsl" ?>

<page>
<metaNam>Use Word Oriented plz</metaNam>
Search for:
<bean:write name="NameLstFrm"
property="searchFstName"/> -
<bean:write name="NameLstFrm"
property="searchLstName"/>
  Results:
<% int i=0; %>

Num - First Name - Last Name - Phone - City - Postal
Code - eMail
<table>
<logic:iterate id="iterateMap" name="NameLstFrm"
property="iterateMap">
<tr>
<td><% i++; %> <%= i %> </td>
  <td><bean:write name="NameLstFrm"
property="fstName"/> </td>
  <td><bean:write name="NameLstFrm"
property="lstName"/></td>
  <td><logic:equal name="NameLstFrm"
property="rowNext" value=""/></td>
</tr>
</logic:iterate>
</table>

</page>

```


- And here is a simple “rtf.xml”.

```

<?xml version="1.0" ?>
<xsl:stylesheet version="1.0"
xmlns:xsl="http://www.w3.org/1999/XSL/Transform">

  <xsl:template match="/">

    {\rtf1\ansi\defformat\deff0{\fonttbl{\f0\fmmodern\frq
1 Courier New;}}
      \ndscpsxn\pgwsxn15840\pghsxn12240\f0\fs14
      \par
      <xsl:apply-templates/>
    }
  </xsl:template>

  <xsl:template match="tr">
    \trowd\trgaph80\trkeep
    \cellx5000
    \cellx10000
    \cellx10000
    <xsl:apply-templates/>
    \row
    \pard
  </xsl:template>

  <xsl:template match="td">
    \pard\intbl
    <xsl:apply-templates/>
    \cell
  </xsl:template>
</xsl:stylesheet>

```

You can see how you can create a central place where you can change the look-and-feel of the site.

Some of the older browser versions do not support XML. For compatibility, you can write JavaScript functions that will do the

XSLT on the client-side.

I sometimes do line counts in XSL so I can issue an RTF page break. RTF can group, subgroup, and XSL can even subtotal. Cool!

Review

It is a good idea to generate XML from your JSP's and send XML and XSL to the browser, if possible.

Chapter 12: Drill Down

Goals

So far we can search and retrieve. Let's click on a link and display the details for a record.

Link Lab

- Let's add this code to our "SpaBaseBean.java" in the "reuse" folder.

```
public BigDecimal getPK() { //get it from _crs
RowSet
    try{ this.PK = _crs.getBigDecimal("PK");    }
// try
    catch (Exception e)
{System.out.println(e);} // catch
    return (this.PK);
} // get
```

This says that most of the time when we make a bean, it will have a primary key (PK). In my tables I call all the primary keys PK and they are always database-generated sequence numbers.

- And let's make sure our 'NameLstPg.jsp' page has a link tag.

```
<?xml version="1.0"?>
<page>
```

```

Num - First Name - Last Name - Phone - City - Postal
Code - eMail
<table>
<logic:iterate id="iterateMap" name="NameLstFrm"
property="iterateMap">
<tr>
    <td><bean:write name="NameLstFrm" property="PK"/>
</td>
<td><html:link page="/do/nameZoomAct" paramId="PK"
paramName="NameLstFrm"
    paramProperty="PK">
    <bean:write name="NameLstFrm" property="firstName"/>
</html:link> </td>
    <td><bean:write name="NameLstFrm"
property="lastName"/></td>
    <td><logic:equal name="NameLstFrm"
property="rowNext" value=""/></td>
</tr>
</logic:iterate>
</table>
</page>

```

Note in the link we are passing the link page, just like in the “index.jsp” page. Then we are specifying the parameter ID, how to retrieve the parameter value, and finally what will be displayed.

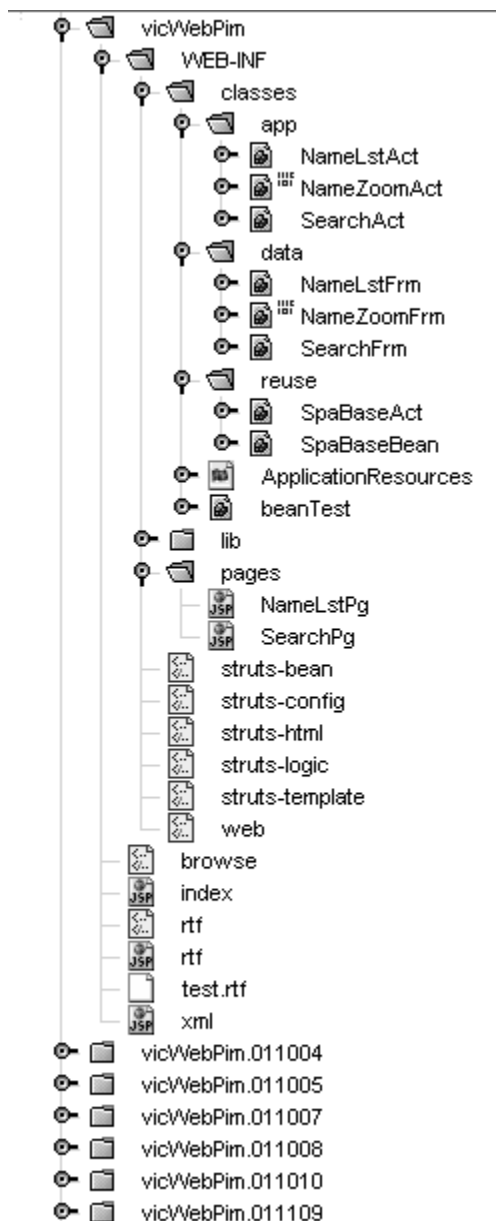
So if the user clicks on First Name in our example, we will pass control to a nameZoomAct() action for further processing. Spend some time to understand it, because it is one very long command line.

- Test the page and hover on few of the links. Do you see the link?

Next, we will need to create the “NameZoomAct” action bean, “NameZoomFrm” bean, and the “NameZoomPg” JSP.

Action / Data / Page Lab

- Take a look at how I have working code folder below.



- Let's create another MVC page. We need to update the "struts-config.xml" so that the application knows about these new capabilities.

```

<struts-config>
<data-sources>
<!-- we use Poolman2 -->
</data-sources>
<!-- ===== Form Data Bean Definitions
===== -->
<form-beans>
    <form-bean        name="SearchFrm"
                      type="data.SearchFrm"/>

    <form-bean        name="NameLstFrm"
                      type="data.NameLstFrm"/>

    <form-bean        name="NameZoomFrm"
                      type="data.NameZoomFrm"/>

</form-beans>

<!-- ===== Global Names Forward Definitions,
=== -->
<global-forwards>
    <!-- <forward    name="search"
path="/do/searchAct"/> -->

</global-forwards>

<!-- ===App Controler Action Mapping Definitions,
main part of config, do forward= -->
<action-mappings>
<!-- Should map to our action in page, and the bean
to use, and where to go next -->
    <action    path= "/searchAct"
                type= "app.SearchAct"
                name= "SearchFrm"
                scope="request"

```



```

        input="/WEB-INF/pages/SearchPg.jsp">
<forward name="searchPg"          path="/WEB-
INF/pages/SearchPg.jsp"/>
    <forward name="nameLstAct"
path="/do/nameLstAct"/>
    </action>

    <action path="/nameLstAct"
        type="app.NameLstAct"
        scope="request"
        input="/WEB-INF/pages/NameLst.jsp">
<forward name="nameLstPg"          path="/WEB-
INF/pages/NameLstPg.jsp"/>
    <forward name="nameZoomAct"
path="/do/nameZoomAct"/>
    </action>

    <action path="/nameZoomAct"
        type="app.NameZoomAct"
        name="nameZoomFrm"
        scope="request"
        input="/WEB-INF/pages/NameZoom.jsp">
    <forward name="nameZoomPg" path="/WEB-
INF/pages/NameZoomPg.jsp"/>

    </action>

</action-mappings>
</struts-config>

```

- Then we need a form bean (“...WEB-INF\classes\data\NameZoomFrm.java”).

```

package data;

import sun.jdbc.rowset.*;
import java.util.*;
import java.math.*;

import reuse.*;

```

```

public class NameZoomFrm extends SpaBaseBean {

private String fstName;
private String lstName;

public int retrieve(BigDecimal aBD) {
try{
    CachedRowSet cr = getCrs();
    cr = new CachedRowSet(); // easier then resultset
    // you must test this sql string manually for a
SQL IDE
    cr.setCommand("select * from NAM where PK = ?");
    // to limit the number of rows coming back, we
only need a page at a time
    cr.setBigDecimal(1,aBD);
    exec(cr);
} // try
catch (Exception e)
{ System.out.println(e);
  e.printStackTrace();} // catch
return 0;
} // retrieve

public String getFstName() { //get it from _crs
RowSet
    CachedRowSet cr = getCrs();
    try{ this.fstName = cr.getString("NAM"); }
// try
    catch (Exception e)
{System.out.println(e);} // catch
    return (this.fstName); } // get
public String getLstName() { //get it from _crs
RowSet
    CachedRowSet cr = getCrs();
    try{ this.lstName = cr.getString("LS_NAM");
} // try
    catch (Exception e)
{System.out.println(e);} // catch
    return (this.lstName);
} // get

```

```

public void setFstName(String aFstName) { //get it
from _crs RowSet
    CachedRowSet cr = getCrS();
    try{ cr.updateString("NAM",aFstName); } //
try
    catch (Exception e) {System.out.println(e);}
// catch
    setCrS(cr);
} //setFst

public void setLstName(String aLstName) { //get it
from _crs RowSet
    CachedRowSet cr = getCrS();
    try{ cr.updateString("LS_NAM",aLstName); } //
try
    catch (Exception e) {System.out.println(e);}
// catch
    setCrS(cr);
} //setFst

} //class

```

- Now, the “NameZoomAct.java” action class:

```

package app;

import java.io.*;
import java.util.*;
import javax.servlet.*;
import javax.servlet.http.*;
import org.apache.struts.action.*;

import data.*;
import reuse.*;
import java.math.*;

public class NameZoomAct extends SpaBaseAct {

```

```

public ActionForward spaPerform( ActionMapping
mapping, ActionForm form,
                                HttpServletRequest
request, HttpServletResponse response)
                                throws IOException,
ServletException
{
String PK = request.getParameter("PK");
System.out.println(PK+"XXX");
BigDecimal pkBD = new BigDecimal (PK);

NameZoomFrm frm = new NameZoomFrm();
frm.dbCon();
frm.retreive(pkBD);
frm.dbDisCon();
System.out.println(pkBD + frm.getFstName());

HttpSession session = request.getSession();
session.setAttribute("NameZoomFrm", frm);

return(mapping.findForward("nameZoomPg"));

} // perform Act
} // class

```

- And a page (NameZoomPg.jsp) to display the detail.

```

<%@ taglib uri="/WEB-INF/struts-html.tld"
prefix="html" %>
<%@ taglib uri="/WEB-INF/struts-bean.tld"
prefix="bean" %>
<?xml version="1.0"?>

<page>
Here is a name...
<p>
<bean:message key="name.First"/>: <html:text
name="NameZoomFrm" property="fstName" /> <p>
<bean:message key="name.Last" /> : <html:text
name="NameZoomFrm" property="lstName" />

```

</page>

As always, I only have a few fields, you should have more. In the zoomed detail of a single name you should have all the fields from the table.

Review

We have drilled-down from a list of names to zoom the detail of a single name.

Chapter 13: Debugging

Goals

To be able to debug the code and disable the debugging when we do not need it.

Container settings

- Let's change the container settings in "...\\tomcat\\config\\server.xml".

```
<!-- Tomcat Root Context -->
<Context path="" docBase="VicWebPim" debug="0"/>
<!-- Tomcat Examples Context
<Context path="/examples" docBase="examples" debug="0"
    reloadable="true">
    <Logger className="org.apache.catalina.logger.FileLogger"
        prefix="localhost_examples_log." suffix=".txt"
        timestamp="true"/>
    <Ejb name="ejb/EmplRecord" type="Entity"
        home="com.wombat.empl.EmployeeRecordHome"
        remote="com.wombat.empl.EmployeeRecord"/>
```

PersistentManager: Uncomment the section below to test Persistent Sessions.

...<PersistenceManager>: If true, all active sessions will be saved

- Now delete all folders and WARs other the “...\webapps\VicWebPim” (after making a copy of the webapps folder first).

The modifications to the “server.xml” configuration file will set our PIM as the default application on that IP address. We now only have to use the URL <http://<servername>:8080> to get the “index.htm” of the PIM application without having to use the “/VicWebPim” alias.

- Also, in “struts-config.xml”, let’s turn debugging off:

```
<param-name>debug</param-name>  
  <param-value>0</param-value>
```

- Now we can restart Tomcat and not see any error messages on the console anymore, which is what the production environment will be like. But during development, we have bugs and need to debug.

The traditional debug approach is to use `System.out.println()`. The problem with this is that sometimes a few of these commands are left in the production application. Any `System.out`’s in a production application is problematic since `System.out` is a resource intensive operation. When our web application is loaded, this resource drain may cause many problems.

I have seen a web application that kept crashing intermittently and the fix was to remove ALL the `System.out.println()` statements. If you do use `System.out.println()` statements, please make sure that they will be removed once your code module is released to testing.

We will write a debug method. Since all our objects come from SpaBaseBean or SpaBaseAction class, we can place the debug methods there. From then on, we can just call it as needed. We will build the debug method so that we can turn it on and off globally.

```

11 public int exec(CachedRowSet pCR) {
12     int c=0;
13     try{
14         pCR.execute(getCon());
15         pCR.next();
16         c= pCR.size();
17         // debug(this,c+"rowCount"+pCR.getCommand());
18         setCrS(pCR);
19 } // try
20     catch (Exception e)    {procE(this,e);
21     } // catch
22     return c;
23 } // exec
24 public void debug (Object o, String aMsg)    {
25     // if (getServlet().getDebug() >= 3)
26     //     getServlet().log("Debug: " + aMsg );
27     // if (getServlet().getDebug() >= 4)
28         System.out.println("FBX-Debug- "+aMsg+ " ; " +o.getClass().getName());
29     } // debug()
30 public void procE(Object o, Exception e)
31 { //if (getServlet().getDebug() >= 1)
32     //     getServlet().log("Exception: " + e );
33     // if (getServlet().getDebug() >= 2)
34         System.out.println("FBX Exception:" + e );    } // procExc()
35 public Hashtable getIterateMap()
36 { //is there a better way? like use crs.getCollection?
37     CachedRowSet cr = getCrS();
38     int rc=cr.size();
39
40     Hashtable iterateMap = new Hashtable();
41     // count out the number of rows with dummy map
42     for (int i=1; i < rc; i++) { iterateMap.put(new Integer(i),new Integer(i));}
43 return iterateMap;}

```

Place the debug() and procE() methods in the “SpaBaseBean”

and “SpaBaseAct” classes, removing all other references to System.out.println().

The procE() method is used when Exceptions are thrown during processing. Exceptions are not necessarily errors, at least not in the fatal sense. Processing may continue after Exceptions have been handled and if System.out.println() statements are placed in the catch blocks, you may be creating another bottleneck.

You now have a centralized debug framework for all application classes.

Another useful utility is JWhich. JWhich let’s you know which of class the JVM is using if there are conflicts (say, two (2) classes of the same package and name, but of different versions).

Extra Credit Labs

- You can write a snoop servlet in Action.
- You can also turn on Tomcat debugging.

Extra Credit – NetBeans IDE Debugging

- Add all the libraries to the project.
- Make “org.apache.catalina.startup.Bootstrap” the main class
- Configure your IDE so that you can Execute project or Debug project from the main menu and debug your Struts application.

```
java -Xdebug -  
Xrunjdpw:transport=dt_shmem,server=y,suspend=y  
-jar orion.jar
```

- Make sure you have "jpda.jar" in your CLASSPATH.

Error page

- Set up your web application so that errors will be taken to an error page.

Debugging Servlets

You can look in the “work” folder to see the JSP servlet.

Review

Avoid `System.out.println()` calls as much as possible. Create a debugging method of some sort that can monitor certain users and certain classes via properties in a file.

Chapter 14: Data Entry

Goals

So far we have selected from the database, but we have not inserted or updated. Our goal after this lab is to be able to Create, Retrieve, Update, and Delete (CRUD).

Update Lab

We will now get the “NameZoomFrm” to update to the database.

- We need to change the “SpaBaseBean”. It knows about retrieve, but not update yet.

```
public long update(){
    try{
        // update should be using a different con pool
to same db
        dbCon();
        _crs.updateRow();
        _crs.acceptChanges(getCon());
        dbDisCon();
    } catch (Exception e) { procE(this,e); } //
catch
    return 0; } // update() should return new PK
```

- And we need to modify the “NameZoomFrm” data bean:

```
public void setFstName(String aFstName) { //get it
from _crs RowSet
    CachedRowSet cr = getCrS();
```

```

        try{ cr.updateString("NAM",aFstName);
        this.fstName = aFstName;
        } // try
        catch (Exception e)    {System.out.println(e);}
// catch
        setCrs(cr);
} //setFst
public void setLstName(String aLstName) {    //get it
from _crs RowSet
    CachedRowSet cr = getCrs();
    try{ cr.updateString("LS_NAM",aLstName);
    this.lstName=aLstName;
    debug(this,"XXFrm fst SET "+this.lstName);
    } // try
    catch (Exception e)    {System.out.println(e);}
// catch
    setCrs(cr);
} //setFst

```

- Now we can fix up the “NameZoomPg.jsp”.

Here is a name edit...<p>

```

<html:form action="/nameZoomAct">

    <bean:message key="name.First"/>:
        <html:text    property="fstName"/> <p>
    <bean:message key="name.Last"/> :
        <html:text    property="lstName"/>

<html:submit property="submit" value="Submit"/>

</html:form>

```

- And now the perform() for “NameZoomAct”:

```

{
NameZoomFrm frm = (NameZoomFrm) form;
String PK = request.getParameter("PK");

```

```

if (PK != null) { // do we have an argument PK
passed?
    BigDecimal pkBD = new BigDecimal (PK);
    frm.dbCon();
    frm.retrieve(pkBD);
    frm.dbDisCon();
    //debug(this,"DB fst nam is "+pkBD +
frm.getFstName());
} else { // we are editing then!
    String fn= ((NameZoomFrm)form).getFstName();
    frm.update(); // save the cached row to DB
    debug(this,"form Edited fst nam is " +
frm.getFstName());

} // else
return(mapping.findForward("nameZoomPg"));
// go to search pg.
} // perform Act
} // class

```

- You should be able to change the “struts-config.xml” on your own at this point.
- Now pull up a name, edit it, and then save it in to the database.
- Do this to a few names.
- Now go to a SQL tool and see if you can find the changed names. Please add a few more fields to your example, I only have First and Last Name.

Insert Parameter

- Now, on the Search page (SearchPg.jsp) let’s add a link to an insert action for “NameZoomAct”.

```

<p><html:link page="/do/nameZoomAct/New"> Add a new
name </html:link>

```

- Create a duplicate action in the config for “nameZoomAct/New” in “struts-config.xml” with an extra parameter. We will use this second action mapping with a parameter to determine if we are inserting in our action.

```

<action path= "/nameZoomAct"
. . .
</action>
<action path= "/nameZoomAct/New"
parameter = "new"
. . .
</action>

```

- Add these CRUD methods to the “SpaBaseBean”. Notice how we have factored out some of the complex commands from the code and broken them into simpler functions. These are the new “SpaBaseBean” methods:

```

public void newRow()
{ try {
    dbCon();
    BigDecimal tmp = new BigDecimal(1);
    retrieve(tmp); // descemdat SQL Row signature
    CachedRowSet cr = getCrS();
    cr.moveToInsertRow();
    blank();
    cr.insertRow();
    cr.moveToCurrentRow();
    cr.next();
    debug(this, " we HAVE set up an insert");
} catch (Exception e) {
    debug(this, "oh no new row, check driver");
    e.printStackTrace();
    procE(this,e);} // catch
} // newRow()

public void blank() { } // stub

```



```

public int retrieve(BigDecimal aBD) {
    System.out.println("should never be called");
    return 0;
} // stub

```

- We need to simplify the code in our “NameZoomAct” bean. We can put all CRUD stubs in the “SpaBaseAct” bean.

```

public int update(SpaBaseBean aFrm) {
    //generic update
    aFrm.update(); // save the cached row to DB
    return 1;
}
public int display(SpaBaseBean aFrm, BigDecimal aBD)
{
    aFrm.dbCon();
    aFrm.retrieve(aBD);
    aFrm.dbDisCon();
    return 1;
}

```

- And this in “NameZoomFrm” bean to tell the driver we are going to enter data.

```

public void blank()
{ try { CachedRowSet cr = getCrS();
    cr.updateNull("NAM");
    cr.updateNull("LS_NAM");
    cr.updateNull("PK");
    } catch (Exception e) { procE(this,e);} //
catch
} // blank

```

- Now our “NameZoomAct” bean is determining if we are creating, reading, updating, or deleting.

```

{
NameZoomFrm frm = (NameZoomFrm) form;

```

```

if (frm == null) {frm = new NameZoomFrm();}
String PK = request.getParameter("PK");
String parameter = mapping.getParameter();

if (PK != null) { // do we have an argument PK
passed? Display!
    BigDecimal pkBD = new BigDecimal (PK);
    frm.dbCon();
    frm.retreive(pkBD);
    frm.dbDisCon();
} else
if (parameter!=null) // must be new
    { frm.newRow();
    }
else { // Save
    frm.update(); // save the cached row to DB
    return(mapping.findForward("searchNam"));
} // else

return(mapping.findForward("nameZoomPg"));
// go to search pg.
} // perform Act

```

- You should be able to add new names and update existing names.
- Search for the new names to make sure they are there.
- Add additional fields.

Delete Archive

We have connected to a SQL database, searched a large result set, retrieved, used XSL, printed a report, drilled down, edited, saved and inserted. Now we need to be able to delete.

Absolute deletion of an entity in an application should be seldom allowed without having a mechanism for archiving the deleted

entity. Typically, archives are done on the monthly basis.

So we will just mark the records as a deleted by adding a field to all master tables to track if the user has requested that the record be deleted. Our display SELECTS will not return those entities that have been “flagged” for removal at the next archive cycle. Users of our application will think the records are deleted immediately since they will not be able to see them.

The archive cycle is simply when an operator, or automated process, archives the flagged entities rows, and their related detail rows to an off-line (or on-line) storage media and then issues a DELETE SQL command for all rows that are flagged for removal.

Delete Lab

- Let’s add a link on the “NameZoomPg.jsp” to delete the displayed record:

```
<p>
<html:link page="/do/nameZoomAct/Del" paramId="PK"
paramName="NameZoomFrm" paramProperty="PK">
  Delete </html:link>
```

- Now create a new Action Mapping on for the “/Del” action in the “NameZoomAct” class.
- Alter the Names and Issues table to add a “DEL_FLAG” CHAR(1) column. Do not add it to the Contact table since those are detail records and will be archived when the master record is archived.
- We need now to create a “delete” method in

“SpaBaseBean” and call the delete in our action. The “SpaBaseBean” change allows all the beans to know how to delete:

```
public void setDelete() { // mark this row
deleted
    CachedRowSet cr = getCrS();
    try{
        cr.updateString("DEL_FLG","X");
        // send to DB
        update();
    } // try
    catch (Exception e) {System.out.println(e);}
// catch
} //del
```

- And the action for “NameZoomAct”.

```
if (parameter == null) parameter = "X";
debug(this,parameter+" this is the parm");
```

```
if (parameter.equals("del"))
{   BigDecimal pkBD = new BigDecimal (PK);
    frm.setDelete();
    return(mapping.findForward("searchNam"));
} else
```

- So far we have marked the record as deleted. In order not to display it, let’s change the “NameLstFrm” WHERE clause:

```
cr.setCommand("select * from NAM where nam >= ? and
ls_nam >= ? and DEL_FLG is null");
```

We are done with CRUD.

Review

We can now Create, Read, Update, and Delete (CRUD) using object-orientation and MVC.

Chapter 15: Master-Detail Processing

Goals

Practice constructing master-detail data processing.

Master-Detail

We have done CRUD with the Names master record. The data model design and requirements call for us to allow for entering detailed transactions related to names.

Depending on requirements, we might track every time we contact a name, and every time the name contacts us as a separate transaction. This way we have a history on that name.

It is very important that a business application framework allow master-detail processing. The use of SQL, JDBC, OO, MVC and XSL is technology-oriented, we need to be able to apply these technologies to real world applications.

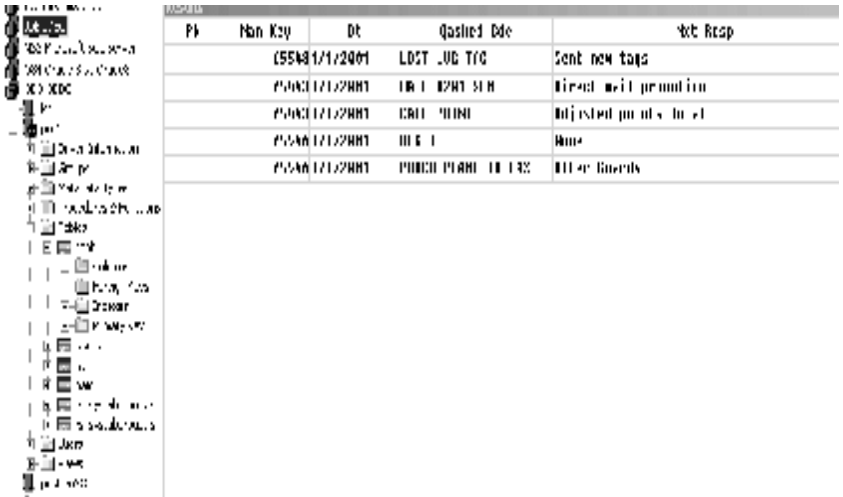
So for each name we should allow a list of contacts to be displayed, based on a foreign key relationship between the master table (Names) and the detail table (Contacts). The primary key of the Names table is a sequence number.

Master-Detail Lab

- Before we do the processing, let's pick two (2) master

names to work with.

- Go to a SQL tool and add about five (5) contact detail records for the two (2) names.
- We will not be able to select the list of contacts, if there are no contacts since we did not write the insert yet.
- Make sure that the FK of detail joins to a PK of master.
- Always have data in the database before beginning development.



The screenshot shows a database tool interface with a table view. The table has five columns: 'Pk', 'Name Key', 'Dt', 'Qualified Cde', and 'Text Resp'. The data rows are as follows:

Pk	Name Key	Dt	Qualified Cde	Text Resp
65543	1712001	LOST JOB TRC		Sent new tags
65402	1712001	DIRT BIRTH SIGN		Direct mail promotion
65403	1712001	CRUI TRIP		Adjusted points to 21
65404	1712001	HOME		None
65405	1712001	PURCH PRIME TRICE		Direct Goods

- Now that we have some contact data for a few names, let's add a link on the "NameZoomPg" to drill-down to a list of contacts for a name.

```
<p>  
<html:link page="/do/ContLstAct" paramId="PK"  
paramName="NameZoomFrm" paramProperty="PK">  
  Details </html:link>
```


- Modify the necessary “struts-config.xml” entries.
- We will now need the MVC objects for Contact List: “ContLstPg”, “ContLstAct”, and the “ContLstFrm” classes. I will copy the three (3) NameLst classes as templates, rename them, and then edit.
- The Contact List Form SQL (...data\ContLstFrm) with accessors (getters):

```

package data;
import sun.jdbc.rowset.*;
import java.util.*;
import reuse.*;
import java.math.*;
public class ContLstFrm extends SpaBaseBean {
private String fstName;
private String lstName;

private BigDecimal namKey;
private String qaskedCde;
private String nxtResp;

public int retrieve(BigDecimal abdFK) {
try{
    CachedRowSet cr = getCrS();
    cr = new CachedRowSet(); // easier then resultset
    // you must test this sql string manually from a
SQL IDE
    cr.setCommand("select NAM.NAM, NAM.LS_NAM,
CONT.PK, CONT.QASKED_CDE, CONT.NXT_RESP      from
CONT, NAME where CONT.nam_key = ? and NAME.pk=
CONT.nam_key");
    // the join
    cr.setBigDecimal(3,abdFK);
    exec(cr);
} // try
catch (Exception e)
{ debug(this,"did not retrieve");

```

```

        procE(this,e);} // catch
return 0;
} // retrieve
// PK getter is in base
public BigDecimal getNamKey() { //get it from _crs
RowSet
    CachedRowSet cr = getCrS();
    try{ this.namKey =
cr.getBigDecimal("NAM_KEY"); } // try
    catch (Exception e)
{System.out.println(e);} // catch
    return this.namKey; } // get

public String getQaskedCde() { //get it from _crs
RowSet
    CachedRowSet cr = getCrS();
    try{ this.qaskedCde =
cr.getString("QASKED_CDE"); } // try
    catch (Exception e)
{System.out.println(e);} // catch
    return this.qaskedCde; } // get

public String getNextResp() { //get it from _crs
RowSet
    CachedRowSet cr = getCrS();
    try{ this.nextResp =
cr.getString("NXT_RESP"); } // try
    catch (Exception e)
{System.out.println(e);} // catch
    return this.nextResp; } // get

public String getFstName() { //get it from _crs
RowSet
    CachedRowSet cr = getCrS();
    try{ this.fstName = cr.getString("NAM"); }
// try
    catch (Exception e)
{System.out.println(e);} // catch
    return this.fstName; } // get
public String getLstName() { //get it from _crs
RowSet

```

```

        CachedRowSet cr = getCrS();
        try{ this.lstName = cr.getString("LS_NAM");
    } // try
        catch (Exception e)
    {System.out.println(e);} // catch
        return this.lstName;
    }// get

} //class

```

- Create the Contact List Page (ContLstPg.jsp):

```

<%@ taglib uri="/WEB-INF/struts-html.tld"
prefix="html" %>
<%@ taglib uri="/WEB-INF/struts-bean.tld"
prefix="bean" %>
<%@ taglib uri="/WEB-INF/struts-logic.tld"
prefix="logic" %>

```

```

List of contacts for<p>
<bean:write name="ContLstFrm" property="FstName"/> -
<bean:write name="ContLstFrm"
property="LstName"/></p>
<p>Results:</p>

```

```

<table>
<logic:iterate id="iterateMap" name="ContLstFrm"
property="iterateMap">
<tr>
    <td><html:link page="/do/contZoomAct" paramId="PK"
paramName="ContLstFrm" paramProperty="PK">
        <bean:write name="ContLstFrm"
property="qaskedCde"/> </html:link> </td>

    <td> - <bean:write name="ContLstFrm"
property="nxtResp"/></td>
    <td><logic:equal name="ContLstFrm"
property="rowNext" value=""/></td>
</tr>

```

```
</logic:iterate>
</table>
```

Trick question: Above when we say PK, is that the PK for Name?
Answer: It is a PK for Contact, PK from name is a Foreign key for Contact.

- And, finally, the action bean (ContLstAct.java):

```
ContLstFrm frm = (ContLstFrm) form;
if (frm == null) frm = new ContLstFrm();
String PK = request.getParameter("PK");
String parameter = mapping.getParameter();
BigDecimal pkBD = new BigDecimal(PK);

frm.dbCon();
frm.retrieve(pkBD);
frm.dbDisCon();

return(mapping.findForward("contLstPg"));
```

Do you see how we went from master record to detail records using a PK to a foreign key?

Contact Report review

- Can we print this as RTF?
- We need to make it XML, and track the parameter passed. If getParam() is HTML then we use one XSL, if getParam() is RTF then we use another XSL.

Every page we do should be in XML. I have done shorter examples here and cheat sometimes, but you should have your pages generating XML and the browser (or server) transforming using XSL.

Contact Zoom review

- We have listed details.
- Do you know how to Zoom a detail?
- Create MVC classes for Contact Zoom.
- Do you know how to update?
- Add a submit form button.
- Do you know how to insert?
 - Add a link on “ContLstPg.jsp” to “ContZoomAct/New”
- Do you know how to delete?
 - Add a delete link on “ContZoomPg.jsp”

Review

You should be able to print, create XSL, and develop master-detail CRUD.

Chapter 16: Security

Goals

Encrypt the transmission of data, authenticate the user, enforce user roles, and apply row-based security on retrieval.

Encryption

Each application server has clearly documented steps to install HTTPS. HTTPS encrypts data in between the web browser and the application server so that it is extremely difficult to intercept and view the stream of text. The stream of text could contain data from our database or a password or any sensitive data. Most database web applications transmit securely.

Our application server needs to answer on HTTPS like this:

<https://localhost:8443/index.jsp>

For Tomcat, the documentation says to download JSSE (which is built into JDK1.4) and then use the keytool which ships with the JDK.

```
keytool -genkey -alias Tomcat -keyalg RSA
```

will generate an encryption key.

- Modify the “.../tomcat/conf/server.xml” file.

```

    <Connector
className="org.apache.catalina.connector.http.HttpCon
nector"
        port="8443" minProcessors="5"
maxProcessors="75"
        enableLookups="true"
        acceptCount="10" debug="0"
scheme="https" secure="true">
    <Factory
className="org.apache.catalina.net.SSLServerSocketFac
tory"
        clientAuth="false" protocol="TLS"
keystoreFile="/keystore" />
    </Connector>

```

Now you can see if the server can also use encryption channel to talk to browser.

- Change the “index.jsp” to this instead of the simple page link:

```

<p><html:link
href="https://localhost:8443/vicWebPim/do/searchAct">
<bean:message key="index.link.search"/> </html:link>

```

Now the data is encrypted before being sent. Other than the first page, you should always use HTTPS for any application that contains sensitive or secured information.

You should also note that encryption does make your application slower since everything is encrypted before being sent.

JDBC User Role Realms

We will need our users to login to use most parts of our

application. The list of the users and the password will be stored in the database. Each user will have a role that will identify their security level.

We will leverage some of the standard guidelines and then build on them.

- Let's create a table to store authentication and roles. Typically, the user name is an e-mail address. I have added an extra field, organization (orgID), to be used later.

```
create table users (  
  user_name      char(15)      not null primary  
key,  
  user_pass      char(15)      not null,  
  orgID          char(15)      not null  
);
```

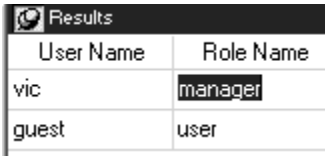
```
create table user_roles (  
  user_name      char(15)      not null,  
  role_name      char(15)      not null,  
  primary key (user_name, role_name)  
);
```

- And now we need to configure ".../conf/server.xml" to leverage this security.

```
<Realm  
className="org.apache.catalina.realm.JDBCRealm"  
debug="99"  
  
driverName="org.postgresql.Driver"  
connectionURL="jdbc:postgresql://66.124.64.38:5432/pi  
mDB?user=pim;password=sec"  
  
  userTable="users" userNameCol="user_name"  
  userCredCol="user_pass"
```

```
userRoleTable="user_roles"  
roleNameCol="role_name" />
```

- Copy your JDBC Drivers to "...tomcat\server\lib".
- Enter a few roles in users through your ISQL application.
Enter a few rows in user_roles like this:



User Name	Role Name
vic	manager
guest	user

- Add security to "web.xml".

```

73 </taglib>
74 |
75 <security-constraint>
76 <web-resource-collection>
77     <web-resource-name>Secure</web-resource-name>
78     <url-pattern>/do/*</url-pattern>
79 </web-resource-collection>
80 <auth-constraint>
81     <role-name>manager</role-name>
82     <role-name>user</role-name>
83 </auth-constraint>
84 </security-constraint>
85
86 <login-config>
87     <auth-method>BASIC</auth-method>
88     <realm-name>default</realm-name>
89 </login-config>
90
91 </web-app>

```

- Restart Tomcat.

Now, when you try to use the “/do” action, the application server will require that you to login based on the data that it has in the JDBC realm.

Login Form Lab

Let’s create a login form and functionality so that the user does not get the basic popup form.

- Create login page (login.jsp) like this:

```

<%@page contentType="text/html"%>
<html>
<head><title>WebPim</title></head>
<body>
Login form<p>
<form method ="POST" action="j_security_check">
Enter your e-mail user id:
    <input type="text" name="j_username"><p>
Enter your password:
    <input type="password" name="j_password"><p>
<p>
<input type="submit" value="Login">
<p>
</form>
</body>
</html>

```

NOTE: You must use `j_security_check`, `j_username` and `j_password`.

- Add “loginBad.jsp”

```

<%@ taglib uri="/WEB-INF/struts-html.tld"
prefix="html" %>
<html>
<head><title>Login</title></head>
<body>
Bad user ID or password!
<p><html:link page="index.jsp"> Main Page
</html:link>
</body>
</html>

```

- Modify “web.xml”.

```

<login-config>
    <auth-method>FORM</auth-method>
    <form-login-config>
        <form-login-page>/login.jsp</form-login-page>

```

```
<form-error-page>/loginBad.jsp</form-error-  
page>  
  </form-login-config>  
</login-config>
```

- Test it out. So now we have a database of users and roles.

Optional row based security demo

In addition to J2EE and Struts, we will add some extensions to provide common functionality. Typically, all users can access most of the pages and most of the tables. However, different users should or should not see certain rows within tables. So let's set that up.

You might have seen tables that always have “lastModifiedDate” or “lastModifiedUserID”. People use it to know when the record was changed. We are not going to do that here due to time considerations, but you are free to do it on your own.

Instead, we will add three (3) extra fields to the master tables, such as NAM. The fields will track who can view the row and what roles can modify the row. We will alter the table to have “roleViewID” to track which role can view and “roleModID” to track which role can modify the row.

For example, in NAM table, most rows will have “roleViewID” as `null`, which means no security and that anyone can view it. Some rows will have “roleViewID” as “user” which means that only users in role “user” can retrieve these rows, others should not see them.

- Add the third column as a different type of security. Imagine that you are writing an application service

provider (ASP) application so your application could be used by different organizations.

Let's say your web application has three (3) clients, organizations, or departments (Client A, Client B, and Client C) and that there is some kind of charge-back for application usage.

So any one of the three (3) clients' users could enter records into your tables. However, when users from Client B retrieve records, they should not see other clients' (such as Client A and C) data.

One approach is to have three (3) different databases with three (3) different connection pools. But what if you go from three (3) clients to over a hundred? This approach is not scalable and a maintenance nightmare.

What we need is a way to track what row belongs to what client. So we will add a column to the master tables to track what client organization they are associated with. Let's use org field for that.

- Alter the master tables, like NAM, to have these fields added:

```
roleViewID    char(15),  
roleModID     char(15),  
orgID         char(15)
```

So that is our row-based security design. Let's now construct it.

In our form beans, when we retrieve rows, we will need to modify our WHERE clause to account for these three (3) new fields if we want row-based security. We will check the getRemoteUser() and based on that we will be able find out what Org and Role they are in.

Row based security for NAM will take three (3) lines of code.

- First, in “NameLstAct” action, we send in the UserID like this:

```
String user = request.getRemoteUser();
nl.retrieve(fn,ln,user,30);
```

- So now we just have to change our SQL command in “NameLstFrm” form, with a longer SQL command:

```
select NAM, LS_NAM from NAM where ( nam >= 'B' and
ls_nam >= 'B' and DEL_FLG is null) and (roleViewID is
null and roleModID is null and orgID is null) LIMIT ?
```

This retrieves rows that match our retrieval AND do not have any security setting.

- Now let’s add row-based security. Go to your ISQL application and select two (2) rows in Names that we added and give them security by adding ‘A’ for the OrgID and “Manager” as the Role.

User Name	User Pass	Orgid
vic	vic	A
guest	guest	B

Results									
Pk	Nam	Ls Nam	Src Cde	Phone	Email	Del Flg	Roleviewid	Rolemodid	Orgid
65545	Vic	Cekvenich					Manager		A

- Before trying to do this security in Java, try it in a SQL select.

Our Java will need to use this SQL now:

```

select NAM, LS_NAM from NAM N, users U, user_roles R
where
( N.nam >= 'B' and N.ls_nam >= 'B' and N.DEL_FLG is
null
  and U.user_name='vic' and U.user_name=R.user_name
  and N.orgID=U.orgID)
and
(
  (N.roleViewID=R.role_name or
N.roleModID=R.role_name)
)

```

So this says “Give me all the rows I am searching; where the users role joins to roleID; and the row is in the users organization”.

- And our new “NameLstFrm” form bean code:

```

public int retrieve(String aFst, String aLst, String
aUser, int arowCount) {

try{ CachedRowSet cr = new CachedRowSet();
  String SQLcmd= "" +
"select PK, NAM, LS_NAM from NAM N, users U,
user_roles R where " +
"( N.nam >= ? and N.ls_nam >= ? and N.DEL_FLG is null
" +
"  and U.user_name=? and U.user_name=R.user_name " +
"  and N.orgID=U.orgID) and " +
" ( (N.roleViewID=R.role_name or
N.roleModID=R.role_name) )";

cr.setCommand(SQLcmd);

cr.setString(1,aFst);
cr.setString(2,aLst);
cr.setString(3,aUser);
cr.setInt(4,arowCount);
exec(cr); } // try

```


Did you get row-level security? Cool!

- Create a logout page that basically has a session invalidate action.

Review

We have encrypted our transmission and authenticated the user. Also, I have shown you how you can implement security down to row-level.

Chapter 17: Demo

Goals

To be able to demonstrate skill developing data-based, MVC web applications.

Midterm Test

Go back to the issues requirements. Recall that we have created an Issues table. This is where we can track bugs, goals, things to do, due dates, whatever your requirements say.

Our goal is to create the MVC classes to CRUD the Issues table. Our page should generate XML so the browser may transform the XML via XSL. We have already practiced doing this with Names and Contacts, so this is the third time for you.

The challenge? To do it quickly and smoothly. Pretend you are doing an *ad hoc* demo to your Java User Group or that you are at a job interview and are asked to do this assignment. Your resume said you know Struts, right?

- Create the data bean that extends our SpaBaseBean that will search and list issues.
- Create the page with Struts tags that will iterate and create a table. Generate XML from JSP.
- Create the action to control the page.
- Zoom a single issue.

Go to it, and track how long it takes you.

You will need to have Issues searching, listing, reporting and zooming, just like names, for later in the application.

Note: One day I will track down or write a JSP tag that does client-side XSL w/ JavaScript that works with older browsers.

Review

If you had an application with a dozen pages, how long would it take you to develop that application?

Now, take the time it took you to complete the lab and multiply it by twelve (12). With practice, you'll get to the point that you won't have to consider how to implement the pages and you'll be able to concentrate on the business requirements.

Chapter 18: Data Validation

Goals

To be able to validate data entry.

Simple Validation Set up

In the Download Appendix we downloaded a Struts validator.

- Place “struts-validator.jar” in “...\WEB-INF\lib” folder.
- Extract the sample “validation.xml” and place it in WEB-INF.
- We will practice doing validation for “NamesZoomFrm”. Let’s make sure we have at least an e-mail field displayed.

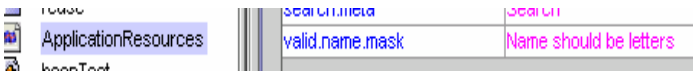
```
public void setEmail(String aEmail) { //get it from
_crs RowSet
    CachedRowSet cr = getCrs();
    try{ cr.updateString("EMAIL",aEmail);
        this.email=aEmail;
    } // try
    catch (Exception e) {System.out.println(e);}
// catch
    setCrs(cr);
} //setFst
public String getEmail() { //get it from _crs RowSet
    CachedRowSet cr = getCrs();
```

```

        try{   this.email = cr.getString("EMAIL");   }
// try
        catch (Exception e)
{System.out.println(e);} // catch
        return (this.email);
    }// get

```

- Add the e-mail property to nameZoom.
- See if you can enter a bad e-mail and save it to RDMBS.
- Now let's place some validating error messages in the "ApplicationResource.properties" text file.



So when we have an error on a Name field, we will ask it to display the valid mask above.

Also, there are standard validation error messages that are shown by validation.

- Copy those from validations into your own ApplicationResource.properties.
- The "ApplicationResource.properties" should have at least these lines so you have some standard messages to use:

```

app.name=Family's SPA Personal Information Manager
ASP
valid.name.mask=Name should be letters
name.Email=E-Mail
search.meta=Search
name.Last=Last Name
name.First=First Name

```

```

app.desc=For now we play with a standard content
syndication framework, but ..
index.link.search=Search

button.cancel=Cancel
button.confirm=Confirm
button.reset=Reset
button.save=Save
# Errors
errors.footer=
errors.header=<h3><font color="red">Validation
Error</font></h3>You must correct the following
error(s) before proceeding:
errors.ioException=I/O exception rendering error
messages: {0}
error.database.missing=<li>User database is missing,
cannot validate logon credentials</li>
errors.required={0} is required.
errors.invalid={0} is invalid.
errors.byte={0} must be an byte.
errors.short={0} must be an short.
errors.integer={0} must be an integer.
errors.long={0} must be an long.
errors.float={0} must be an float.
errors.double={0} must be an double.
errors.date={0} is not a date.
errors.range={0} is not in the range {1} through {2}.
errors.creditcard={0} is not a valid credit card
number.
errors.email={0} is an invalid e-mail address.
# -- sql --
sql.access.denied=Access denied.
sql.access.error=Unable to access database. Please
try again later.
sql.access.empty=Requested records were not found.
sql.record.updated=Record updated.
sql.record.inserted=Record inserted.
# -- simple --
simple.integer.displayname=Integer
simple.date.displayname=Date
simple.phone.displayname=Phone

```

```
simple.zip.displayname=ZIP
simple.email.displayname=Email
simple.text.displayname=Text
```

Now we need to edit the “validation.xml” file. Note that there it is a long file and that there are two (2) tags there, <global> and <formset>.

The <global> tag is needed for generic validation and the <formset> tag is custom for your application.

- Delete the <formset> tag and keep the <global> tag.
- We will now define our <formset> tags like this. It will help if you review regular expressions so that you can write your own complex masks:

```
</global>
<formset>
    <form name="NameZoomFrm">
        <field property="fstName"
depends="required,mask">
            <arg0 key="name.Fist"/>
            <var>
                <var-name>mask</var-name>
                <var-value>^\w+$</var-value>
            </var>
        </field>
        <field property="lstName"
depends="required,mask">
            <msg name="mask" key="name.Last"/>
            <arg0 key="name.Last"/>
            <var>
                <var-name>mask</var-name>
                <var-value>^[a-zA-Z]*$</var-
value>
            </var>
        </field>
```



```

        <field    property="email"
depends="required,mask">
            <arg0 key="name.Email"/>
            <var>
                <var-name>mask</var-name>
            <var-
value>^\w+@\w+.\w{3}$</var-value>
                </var>
        </field>
    </form>

</formset>
</form-validation>

```

So far we have some fields to validate, some messages, and a mapping into “validation.xml”.

- Validation is run via a servlet so we have to edit “web.xml” and that servlet to activate the validation servlet that Struts will call internally.

```

<!-- Validator Initialization Servlet Configuration
-->
<servlet>
    <servlet-name>validator</servlet-name>
    <servlet-
class>com.wintecinc.struts.action.ValidatorServlet</s
ervlet-class>
    <init-param>
        <param-name>config</param-name>
        <param-value>/WEB-INF/validation.xml</param-
value>
    </init-param>
    <init-param>
        <param-name>debug</param-name>
        <param-value>1</param-value>
    </init-param>
    <load-on-startup>2</load-on-startup>
</servlet>

```

Now in struts-config, we need to set a flag that it validates for NameZoom.

- Insert `validate="false"` for the two (2) action mappings where we edit or insert NameZoom in order to have manual control of validation.
- In our NameZoom action, we want to call `validate` method, before saving now, like this:

```
else { // Save
    ActionErrors errors =
frm.validate(mapping,request);
    if (!errors.empty()) {
        saveErrors(request, errors);
        return (new ActionForward(mapping.getInput()));}
    frm.update(); // save the cached row to DB
    return(mapping.findForward("searchNam"));
} // else
```

And we have to teach our Form beans about validation.

- In “SpaBaseBean”, we need to change it to say:

```
import com.wintecinc.struts.action.ValidatorForm;
public class SpaBaseBean extends ValidatorForm {
```

- Add a tag to display any errors in the page.

```
<%@ taglib uri="/WEB-INF/struts-validator.tld"
prefix="validator" %>
```

```
<validator:errorsExist>
  <bean:message key="errors.header"/>
  <ul>
    <validator:errors id="error">
      <li><bean:write name="error"/></li>
```

```
</validator:errors>
</ul><hr>
</validator:errorsExist>
```

- Modify “web.xml” to set up the new tag:

```
<!-- Validator Tag Library Descriptor -->
<taglib>
  <taglib-uri>/WEB-INF/struts-
validator.tld</taglib-uri>
  <taglib-location>/WEB-INF/struts-
validator.tld</taglib-location>
</taglib>
```

This might seem like a lot of work for one page, but what it does is set up a framework that you can use for the entire application. Also, validation is a separate set of functionality from the rest of the application. Validation of data is done in the Form Beans, and the validation is called by the action.

Some of the validation is not simple and is data-driven. For example, for the validation case “You can’t issue a PO because you have exceeded your credit line of unpaid orders “, you would have to write a validate method in the form bean that also calls the base bean.

Review

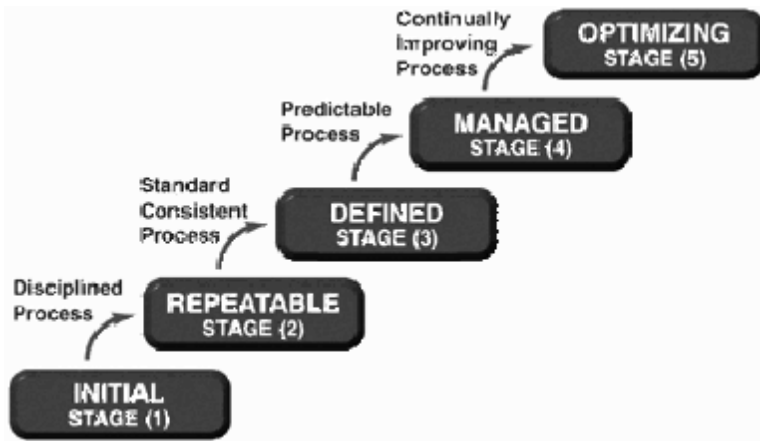
We had enabled validation in the Struts framework.

Chapter 19: Administration

Goals

Discuss aspects of the capability maturity model (CMM).

Project Monitoring



Good projects are run by qualified project managers. A project can be on any of the levels of the CMM (Capability Maturity Model).

Being a manager does not make you a project manager. If you read a book on surgery, you would not say you are a surgeon. Project management certification is available at the PMI.org website.

Using a project management software to print a GANTT chart is not enough. The key is to continuously monitor the progress of the project.

Mobilizing

When staffing a project, a key consideration is that at some point in the future, the project will eventually go into production and operations.

When you are in operations, after deployment, you will need the following roles:

DB loaders

They will load the DBF files (or other mass storage) into the RDBMS. In case of an RDBMS server failure, they will build a new database and load it with archived data. They might need to work on replication of multiple RDBMS, depending on the volume. Their secondary role is archiving and backups. Most of the hours are weekend and nights, with no essential need during regular working hours.

Unix Security

Each application server will be housed in a rack, possibly outside a firewall. This means that it must be secured. The RDBMS is

behind the firewall. You will need a Unix expert that will keep up with the latest security protocols and patches. Most of the hours are nights and some weekends with no essential need during regular working hours.

User Phone Support

No matter the application, users will need support. If they do not know how to place the order, or there is a problem with the order, etc., they must be able to call user support and have it resolved. All of the hours are during regular business hours.

Report Writer

You will always need more reports and someone to write them.

Business Development

If your application is commercial, you will need some business development professionals to push the product in the marketplace.

User Support

Before construction, this is who documents the requirements and test cases, based on direction from the chief engineer. During construction they write user documentation. After construction they execute the test cases. Before operation, they conduct user training if necessary.

You should staff these roles in the construction phase so that the transition from construction to operation is as smooth as possible.

Additional construction phase roles

Chief Software Engineer

You need someone responsible and accountable for R&D. The more experience the chief engineer has in developing quality web applications, the better.

Mentor

This role could be taken on by the chief software engineer, but you may want to have a mentor that can spend sufficient time with mid- and junior-level developers. This role could also provide project refactoring and keep the development effort on track.

Database Performance Engineer

If you are on a budget, this is not where you want to skimp. Save money elsewhere since I consider this the most critical resource and most expensive resource. This should only be one person, no matter how large the team, and this person should be held accountable for the database performance.

Programmers

Junior or mid level developers, with mostly JSP development. Foot soldiers that take the architecture and leverage it in code.

Object Administrator

Senior Java engineer responsible for developing base objects, system programming, custom tags, etc. This should only be one (1) person, no matter how large the team, and this person should be held accountable for the development productivity.

Project Management Admin

Administrative support for the development team is important in that logistical issues (such as timecards, office supplies, expense reports, etc.) may drive the technical team to distraction, reducing productivity and disrupting creative rhythm.

Most of these roles might phase out on success and do not have life after operations, unless there are additional phases after initial rollout.

Depending on the scope of your project, these roles might be assumed by two (2) people wearing many hats, or you could have many more resources in support and documentation roles.

Review

User support, documentation, and test cases are key roles. The most important resources are the Database Performance Engineer and the Object Administrator.

Chapter 20: Portal Tiles

Everything should be as simple as possible, but not simpler.
- A.E.

Goals

Set up the tiles around our pages and portal content.

Graphical User Interface

Take a look at some popular web sites, such as ESPN.com. The goal of the user interface is that “if they know how to use major popular sites, they should know how to use this application”.

This approach is known as coding for the *least amount of astonishment*.



Note how we have regions on most pages. In the example above, the body would be our pages we constructed on names, contacts, issues, searching, login, etc. The header is typically the branding of the site. The footer can be legal or sponsor messages. Also, there is sometimes a “east” region that shows syndicated content or sponsoring partners.

We will be doing a lab with those five (5) regions. We will need to create an HTML-type table to be used as a template. You will need to create a small logo for the header. Java or SQL programmers are not necessarily good at creating eye candy, so I hope you have access to a good graphic artist.

Simple Tiles Lab

- Add the “tiles.tld” to the “...\WEB-INF” folder, and make sure “web.xml” has the “tiles.tld” reference.

Now we will create a layout, much like in the figure above.

- Create a folder called “portal” under “...\WEB-INF”.
- Create “SimpleLay.jsp” in the “portal” folder.

```
<%@ taglib uri="/WEB-INF/tiles.tld" prefix="tiles" %>
<html>
<head>
  <title><tiles:getAsString name="title"/></title>
</head>
<body>

<TABLE width="100%">
  <TR>
    <TD colspan="2"><tiles:insert attribute="header"
/></TD></TR>
  <TR>
    <TD width="120"><tiles:insert attribute="navBar"
/></TD>
    <TD><tiles:insert attribute="body" /></TD></TR>
  <TR>
    <TD colspan="2"><tiles:insert attribute="footer"
/></TD>
  </TR>
</TABLE>
</body>
</html>
```

Note that this layout expects some arguments for header, footer, etc.

- In the “...\WEB-INF\portal” folder, create simple versions of header, footer, etc. You can go back and edit them later.

header.jsp:

```
<%@ taglib uri="/WEB-INF/struts-html.tld"
prefix="html" %>
<html:img page="/images/logo.gif"/><p>
World's greatest standard PIM portal application<p>
<p><p>
```

- Create a folder “images” in the “...\webapps\VicWebPim” folder and place a logo image so that you can check to see if the header works.
- Make “navBar.jsp” in the “portal” folder. You will put live links to the Name search, and Issues search.

NavBar.jsp:

```
<p>
<p>
Links<p>
Menu:<p>
<p>
Names<p>
Issues<p>
Print<p>
Logout<p>
```

- Now we need a simple “footer.jsp”:

```
Copyright 2002 MyOrg Inc.
```

```
<! - - OK, so I am lazy - - >
```

So we have a layout template, and top, left and bottom page. We now need the main body page.

- Copy “index.jsp” to “indexTile.jsp”. This will be our body.
- Now edit “index.jsp” to look like this:

```
<%@ taglib uri="/WEB-INF/tiles.tld" prefix="tiles" %>
<tiles:insert page="/WEB-INF/portal/simpleLay.jsp" >
  <tiles:put name="title" value="Family WebPim" />
  <tiles:put name="header" value="/WEB-
INF/portal/header.jsp" />
  <tiles:put name="footer" value="/WEB-
INF/portal/footer.jsp" />
  <tiles:put name="navBar" value="/WEB-
INF/portal/navBar.jsp" />
  <tiles:put name="body" value="/indexTile.jsp" />
</tiles:insert>
```

This assembles the tiles. So what this says is when Struts opens this JSP, it will look at the “simpleLay.jsp” for a layout definition. Then it will pass, via the <tiles:put> tag the JSP pages to use.

- Test it out.

Portal Definitions

Tiles can be made more dynamic via definitions and can be used to create a portal. A definition is a tile associated with a parameter. Parameters are needed to dynamically assign the main body JSP. Otherwise we would have to create a duplicate of page assembly information, like we did for “index.jsp”.

Parameters are also used for tiles based on security roles. You

can have guests, registered members, paid members, and site managers using your web application.

- We will need to change “web.xml” to tell it that Struts will now be processed a bit differently and more dynamically.

```
<!-- web.xml fragment -->
<!-- Action Servlet Configuration -->
  <servlet>
    <servlet-name>action</servlet-name>
    <servlet-
class>org.apache.struts.tiles.ActionComponentServlet<
/servlet-class>
      <init-param>
        <param-name>definitions-config</param-name>
        <param-value>/WEB-INF/tilesPortalDef.xml<
param-value>
      </init-param>
      <init-param>
        <param-name>definitions-debug</param-name>
        <param-value>1</param-value>
      </init-param>
    </servlet-name>action</servlet-name>
  </servlet-
class>org.apache.struts.action.ActionServlet</servlet
-class> -->
```

- Modify the “...\WEB-INF\tilesPortalDef.xml” file.

tilesPortalDef.xml:

```
<?xml version="1.0" encoding="ISO-8859-1"?>
  <!DOCTYPE tiles-definitions PUBLIC
    "-//Apache Software Foundation//DTD
    Tiles Configuration//EN"

    "http://jakarta.apache.org/struts/dtds/tiles-
    config.dtd">
```



```

<!-- html definition Mappings -->
<tiles-definitions>

  <!-- Instance description -->
  <definition name="baseDef" template="/WEB-
  INF/portal/simpleLay.jsp">
    <put name='title'    content='Blank Tile
  Title Def' />
    <put name='header'  content='/WEB-
  INF/portal/header.jsp' />
    <put name='footer'  content='/WEB-
  INF/portal/footer.jsp' />
    <put name='navBar'  content='/WEB-
  INF/portal/navBar.jsp' />
    <put name='body'
  content='/indexTile.jsp' />
  </definition>

</tiles-definitions>

```

So we have one definition, almost a clone of our “index.jsp”.

- Now our “index.jsp” can be this:

Index.jsp:

```

<%@ taglib uri="/WEB-INF/tiles.tld" prefix="tiles" %>

<tiles:insert definition="baseDef"/>

```

This is the only code needed since it will use the definition and we do not need to change anything. Layout is used for the positional table info and definition places JSP’s in those locations. We can have many layouts and definitions.

Note that there might be some confusion with XSL and tiles. XSL is used for style and formatting. We do not put that information in our JSPs.

- You now need to create a definition for each on of the pages. You should have at least 8 pages under “\pages”, here is a sample.

tilesPortalDef.xml:

```
<definition name="name.search" extends="baseDef">
    <put name='title' content='Name' />
    <put name='body' content='/WEB-INF/pages/SearchPg.jsp' />
</definition>
<definition name="name.list" extends="baseDef">
    <put name='title' content='Name' />
    <put name='body' content='/WEB-INF/pages/NameLstPg.jsp' />
</definition>
```

- And now, we need to use the new page “names” in struts-config. **Remove all references to *.jsp** in “struts-config.xml” to use the page name definitions. This will activate the portal tiles.

struts-config.xml:

```
input="name.zoom">
    <forward name="nameZoomPg"
path="name.zoom" />
    <forward name="searchNam"
path="name.search" />
```

- Test it out. You should have footers and headers everywhere.

In the definition, you can also nest layouts within each other.

- Change the “logon.jsp” and “logonBad.jsp” to use the

simple tiles example above, just like “index.jsp”.

```
1 <%@ taglib uri="/WEB-INF/
2 <tiles:insert definition=
3     <tiles
4 </tiles:insert> JSP completion
5     <tiles:insert>
        <tiles:definition>
            <tiles:put>
                <tiles:putList>
                    <tiles:add>
                        <tiles:get/>
                            <tiles:getAsString/>
                                <tiles:useAttribute/>
```

Notice how the NetBeans IDE helps us with the tags.

Login.jsp:

```
<%@ taglib uri="/WEB-INF/tiles.tld" prefix="tiles" %>
<tiles:insert definition="baseDef">
    <tiles:put name='title' content='Login' />
    <tiles:put name='body'
content='/loginTile.jsp' />
</tiles:insert>
```

Link Lab

- On the “navBar.jsp”, get the links to work.
- Make sure you can print the list of contacts and list of issues searched to RTF in a pop-up browser window.
- We will need working links for a later menu lab.

Review

Our web application is now more portal-like due to the use of tiles.

Chapter 21: Refactoring

Goals

Learn how to improve the design.

Throw away development

“Do it once, do it right” is one of the more inefficient development project mantras.

Doing iterative development saves time, money, and other resources. If you do the “do it once, do it right” approach, you spend a lot of time planning and analyzing. By the time you are done with all the analysis of the business processes and the workflows, the business environment may have changed, sometimes rendering requirements useless.

The most efficient way to develop is to fire then aim, then repeat, as opposed to aim and then fire. Do not even hardcode the business process, it should be data driven anyway. So iterative development is more efficient than doing it once.

Ironically, people with small budgets waste it sometimes on a “we must be careful to get it right the first time” instead of doing iterative development, which would have completed faster with less resources.

When you do iterative development, you throw way ALL the code

at the end of the iteration.

Re-design

At the end of the iteration, you re-design. The improvements in the design get you closer to the requirements and specification and save you money. When you re-design, you make your system more maintainable.

Over eighty percent (80%) of software development cost is operations, upgrades, and fixes that happen AFTER the application goes to production.

Have you every seen spaghetti code all over the place of a production system that is almost impossible to fix?

You must build a maintainable, elegant, and terse design. When you iterate a design, you build an order of magnitude better design that is easier to maintain and closer to the requirements.

So what do you design? You do not design a framework, you just follow the Struts guidelines. It is hard to have a bad design with a framework that has, more or less, been done for you.

You do not design a flashy user interface, you must follow users expectations that it works like other sites.

Your design is in the *drum roll*.... data model. This is your proprietary design that should be legally protected. If I have your data model design, I could quickly build your same application in another tool, like Delphi or PowerBuilder.

So you improve the data model design at each iteration. When you change the data model, you must create new beans that

access it and new pages that use the new beans.

(Another less desirable alternative is to change the beans, and change the pages, it is much faster to create new beans and new pages instead).

Also, at the end of each iteration, you improve your object reuse package. At the first iteration, you might have 5% re-use of code. By the second iteration, it could be 15%, and so forth. (A 10% improvement over 60 pages is a large improvement.)

Example iteration

When the first iteration starts, your SQL Engineer and Object Admin stop working on the first iteration. While coders are doing the first iteration, these two work on the next iteration in another environment. The fact is that the data model changes during development. If these data changes are done in development environment then it directly impacts the productivity of the developers. There should be minimal changes to the data model during the actual iteration.

If you decided to go down the path of “do it once, do it right”, the data model changes day-to-day, forcing programmers to adapt instead of creating. In iterative development, changes are expected and are part of the whole iterative process.

When the Data Modeler is done with the second design and Object Admin has built new base objects or other reusable components, all of the code from the first iteration may be abandoned.

Review

You have an idea now how to do iterative development and refactoring.

Chapter 22: Menu

Goals

Create a nicer looking menu for our link.

Menu Lab

- We need to create a new Struts controller “BaseStrutMnu.jsp” in folder “reuse”:

```
package reuse;

import org.apache.struts.tiles.*;
import org.apache.struts.action.*;
import com.fgm.web.resources.*;
import com.fgm.web.menu.*;
import javax.servlet.http.*;
import javax.servlet.*;

//contributed by sskyles
// this is systems programin, not application
//programing
public class BaseStrutMnu extends
ActionComponentServlet {
// we are extending the tiles controler here

public void loadMenu()
{
    MenuRepository reps = new MenuRepository();
    reps.setLoadParam("/WEB-INF/menu-config.xml");
    reps.setServlet(this);
}
```

```

try {
    reps.load();

getServletConfig().getServletContext().setAttribute(M
enuRepository.MENU_REPOSITORY_KEY, reps);
    } catch (Exception e) {System.out.println(e);
System.out.println("Menus might not work.. ");}
} // loadMenu

public void init() throws ServletException {
    super.init(); // call the struts and tiles
    loadMenu(); // our menu loader
} // init

} // class

```

- And we have to change “web.xml” to put this new class in charge of Struts, instead of the tiles.

“web.xml” fragment:

```

<servlet>
  <servlet-name>struts</servlet-name>
  <servlet-class>reuse.BaseStrutMnu</servlet-class>
  . . .
<servlet-mapping>
  <servlet-name>struts</servlet-name>
  <url-pattern>/do/*</url-pattern>

```

- Copy “struts-menu.jar” to “...\WEB-INF\lib” and “struts-menu.tld” to “...\WEB-INF\tlds”.
- Change “web.xml” to know about they new tags.
- Copy the sample “menu-config.xml” to “...\WEB-INF”.
- We now have to change “navBar.jsp” in “...\WEB-INF\portal”.

navBar.jsp:

```
<%@ taglib uri="/WEB-INF/struts-menu.tld"
prefix="menu"%>
<p>
<p>
<menu:useMenuDisplayer name="Simple">
  <table cellpadding=1 cellspacing=1>
    <tr><td>
      <menu:displayMenu name="PimMenu"/>
    </td></tr>
  </table>
</menu:useMenuDisplayer>
```

- And write a menu to display. Delete the rest. The “menu-config.xml” is in “...\WEB-INF”:

```
<?xml version="1.0" encoding="UTF-8" ?>
<MenuConfig>
  <Displayers>
    <Displayer name="DropDown"
type="com.fgm.web.menu.displayer.DropDownMenuDisplaye
r"/>
    <Displayer name="Simple"
type="com.fgm.web.menu.displayer.SimpleMenuDisplayer"
/>
  </Displayers>
  <!-- ===== Case View Menus
  ===== -->
  <Menus>
  <Menu name="PimMenu" title="Menu" description="Pim
Menu">

    <Item name="Names" title="NAMES"

location="https://66.124.64.38:8443/vicWebPim/do/sear
chAct"/>
```




The Jakarta Project

<http://jakarta.apache.org>

World's greatest standard PIM portal application

Menu

NAMES

Family's SPA Personal Information Manager ASP

ITEM

SKIN1

For now we play with a standard content syndication framework, but ..

SKIN2

PRINT

[Search](#)

LOGOUT

Copyright 2002 My Org Inc.

- Test it out.

You can have several menus, images, and drop-down menus. And like everything else in this framework, it comes with source code.

What other framework comes with source code? Thanks, Jakarta.

And another thing, when we write web applications, we capitalize on the fact that we won't need to have installation or upgrade support on the client machines. Because we are using a

centralized application server, those client software support costs are eliminated over the lifetime of the application.

Review

We have added a simple menu.

Chapter 23: Deployment

Goals

Cross-deploy to another J2EE container on Linux.

Linux

There are few places to download Linux free-of-charge on the web, including the RedHat and Suse sites. However, if you don't have high bandwidth, you may consider purchasing a package version from your local retailer or off of the web.

You should consider staying on Windows server:

- If you consider Windows licensing costs reasonable.
- You consider Windows to be scalable, secure, and stable.
- You consider the cost of resources for remote administration of multiple Windows servers reasonable.
- You do not mind paying for standard development tools.
- You do not mind installing the same application to many users and periodically upgrading that application.

Are you ready for Linux server?

- If you are using Mozilla for 6 months or more instead of Outlook, or another cross platform browser and e-mail client.
- If you are using WordPerfect Office for 6 months or more (or StarOffice) instead of MS Office.

- If you use cygwin and vim.
- You like to use a thin client X Windows, such as labF.com
- Just plain sick and tired of Windows.

If you are using these or similar cross-platform tools, you can go to Linux, because these tools run on Linux, and when you switch to Linux, it won't look any different.

If you are using MS tools than you should considering staying with Windows because you may not be familiar with cross-platform applications.

NetBeans and JBuilder run on Linux, but they also run on Windows. Being cross platform gives you choices and room to negotiate with a vendor so you are not locked in.

- That is why you code to ANSI SQL, so you can have choice of SQL servers vendors and costs.
- That is why you code to J2EE, so you can have choice of application servers.
- That is why you use a standard portal framework based on Struts.

Alternatively, you can beg your sales rep for a price break and improved support.

Linux ships with PostgreSQL for example. PostgreSQL is ANSI SQL and can handle large loads of users and is FREE for any use, and you can have the source code. No annual fee. No upgrade fee. No per CPU, per server, or per user costs.

NetBeans is a free IDE. Some of the best programmers on the planet code open source. Only mushrooms can grow in the dark.

We have developed on the Tomcat server. It is free. It is fine for most applications you will ever build.

But this is a J2EE class, so let's practice deploying to another J2EE server.

Your web application should be cross-application server capable. (This kind of eliminates Microsoft and their activation code and predatory licensing).

Security Alert

- Linux should be set up with a fire wall and a secured shell (ssh).
- Never install software as root, if possible.
- I have a user named "app" that I use for installs if possible.
- Never use the software using the same account that created it, so programmers should not login in as "app". Programmers should only have access to the application folders.
- In databases, never create tables as the DBA, but rather create the tables using a special login, like "model". Also, never access the tables using the table owner account. Programmers, and applications, should use a login that doesn't have permissions to create or destroy database objects.

Remove Application Security Temporarily

Before porting the application, we will remove the application security temporarily, so we can easily test the port.

We need to remove:

- HTTPS encryption
- JDBC realm authentication
- Row-based security

Menu has the HTTPS call, Tomcat has the JDBC, beans have row-based security. Set retrieval to all nulls. Comment out the security section in “web.xml”.

Test that the application still runs without these. You want to debug it in this development environment.

After we port, we will turn security back on.

X Windows

- Start up cygwin or telnet.

```
ssh -l vic 198.144.194.96
```

Where “vic” is your username and use IP of your server.

This should connect to Linux from your Windows PC via a secure connection. Note we are using Linux as server and Windows as client, via X-Windows thin client. All the software executes on Linux and the X-Windows GUI executes on the Windows PC. This way the only thing that needs to be installed on a PC is X-Windows. NetBeans, Orion, PostgreSQL, jikes, Jwhich, JRockit, and the CLASSPATH are all on Linux. Each user can share the applications and CLASSPATH settings.

- Start LabF X-Windows.
- Switch back to cygwin shell. We have an X Windows screen but it is not doing anything other than sitting there. Let’s connect our cygwin to the Linux with the X Windows.

```
Xterm -display 66.124.64.35:0&
```

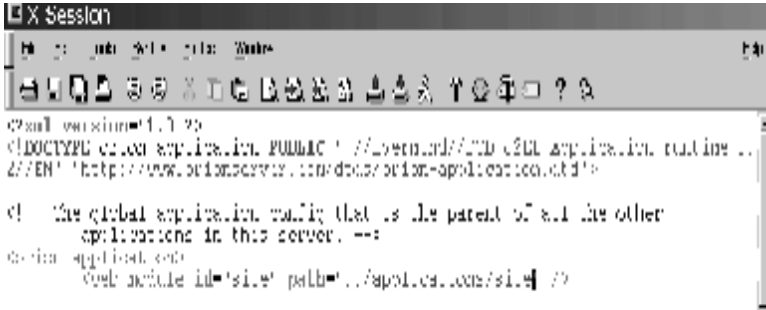
Use the IP of your PC

Now we have a X session.

- And start vi from X.

```
gvim
```

As you can see, vi is a Visual editor that has syntax highlighting for a variety of languages, including Java, and is cross-platform. You can download vi from <http://vim.org>.



Of course, Java and, therefore, the NetBeans IDE, runs on Linux.

JVM

- Type:

```
java -version
```

- You should have downloaded the latest version of the J2SDK from <http://java.sun.com>.
- Once you have the download, type the following as “root”:

```
chmod a+x j2sdk-1_3_1_linux-i386-rpm.bin
./j2sdk-1_3_1_linux-i386-rpm.bin
rpm -iv jdk-1.3.1.i386.rpm
```

- As root, modify the “/etc/profile” file, adding the following environment variable declarations:

```
export JAVA_HOME=/usr/java
export PATH=$PATH:$JAVA_HOME/bin
```

- Copy “tools.jar” from the lib/ JDK directory to “/opt/orion” (or wherever you installed Orion).

There is more on installing software onto Linux at <http://www.linuxdoc.com>.

Install and Configure the Orion J2EE Application Server

- You can use “gnorpm” to download and install Orion, by using the webfind menu command to find Orion.
- Download the Orion RPM version from OrionServer.com. I put Orion in “/home/apps/orion” and soft linked it to “/orion” or “/opt/orion”.
- You could also turn off Apache, since Orion (and Tomcat) can serve HTML pages as well as dynamic J2EE applications.
- Initialize Orion and select a password for the server administrator.

```
cd /opt/orion
java -jar orion.jar -install
```

- Start Orion

```
cd /opt/orion
```

```
java -jar orion.jar
```

- Shutdown Orion.

```
cd /opt/orion
```

```
java -jar admin.jar ormi://localhost admin password -  
shutdown force
```

- [OPTIONAL] Update Orion with the latest build.

```
java -jar autoupdate.jar
```

- Restart Orion and test out that the server is running via a browser to the server port. Open up the documentation page.



Introduction

This is the main documentation index for the Orion Application Server different sections ordered somewhat "chronologically" starting with in:

- [Introduction](#)
- [Installing Orion](#)
- [Learning the technologies](#)
- [Configuring Orion](#)
- [Developing with Orion](#)
- [Debugging applications in Orion](#)
- [Deploying Orion into a production environment](#)
- [Orion FAQs](#)
- [Orion Distribution overview](#)
- [Examples](#)
- [Orion tools Reference](#)
- [Configuration Reference](#)
- [Deployment settings Reference](#)
- [XML tag Reference](#)
- [XML attribute Reference](#)
- [Orion API Documentation](#)
- [J2EE API Documentation](#)
- [J2EE Specifications](#)
- [External links](#)
- [TODO](#)

You can download the documentation package [here](#).

▶▶ If you just want access to the reference manuals, y

Installing Orion

[This guide](#) explains how to install Orion using a few simple steps

Learning the technologies

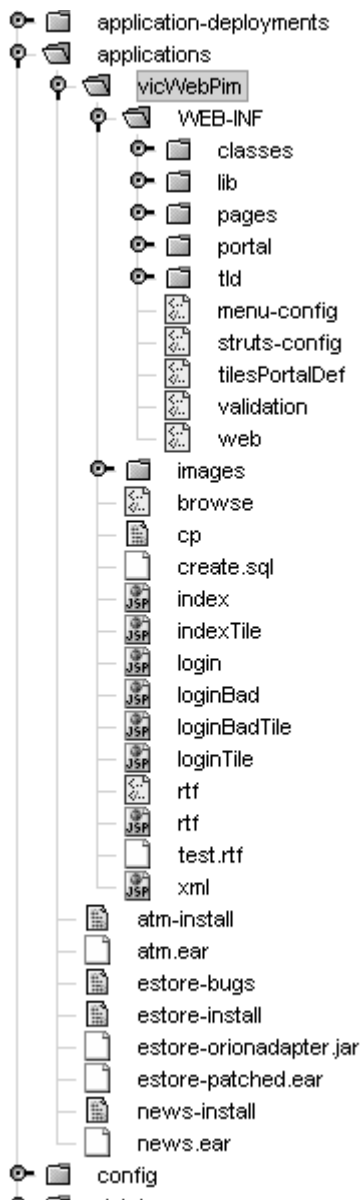
With Orion being an implementation of the J2EE specification, underst many sources of information about this are available [externally](#). Howe subjects. These are available on the [Tutorials](#) page.

The technologies that are of most interest are the following:

- J2EE
- EJB
- Servlets/JSP
- JMS
- JNDI
- JDBC
- JTA
- XML/XSL

- Under “/orion/applications” folder, create your Pim name (ie “pim”). Copy or FTP your project into the newly created folder.

- In NetBeans, mount the “/opt/orion” folder.



- Review the “application-creation-how-to.html” in the Orion “docs\” folder and the Struts documentation for deploying to Orion.
- In “/opt/orion/config/default-web-site.xml” (we go to localhost/pim to view):

```

    <default-web-app application="default"
name="defaultWebApp" />
    <web-app application="default" name="pim"
root="/pim" />

```

- In “/opt/orion/config/application.xml” (use jikes as well if you have it):

```

<orion-application>
    <web-module id="defaultWebApp" path="../default-web-
app" />
    <web-module id="pim"
path="../applications/vicWebPim" />

```

- And you should have this working now:



The Jakarta Project

<http://jakarta.apache.org>

World's greatest standard PIM portal application

PIM Menu

Names

Family's SPA Personal Information Manag

Items

Skin1

For now we play with a standard content s

Skin2

Print

[Search](#)

LogOut

Copyright 2002 My Org Inc.

- Now configure the development mode so we can fix up things. This is “../config/global-web-applications.xml”:

```
<orion-web-app
  jsp-cache-directory="./persistence"
  servlet-webdir="/servlet"
  development="true"
  persistence-path="./persistence/state.ser">
```

Make sure that “poolman.xml” is in the CLASSPATH. I keep it in “/opt/orion” . Alternatively, you can use Orion’s pool manager, but this would require another level of configuration.

Review

We have deployed our application to a server and a faster VM (if you used and JRockit).

Chapter 24: Performance Assurance

Performance Testing

In addition to 8 basic areas of the Project Management Institute (PMI), there are 2 ways to accelerate the project development and save money. One is refactoring to get object-orientation (OO). The other is quality assurance (QA).

The main part of QA is assuring that you have good requirements that the organization will find useful. The other is stress testing your application early, and often, and at the end of every iteration.

Testing the application with a single user or a few dozen of users has little value. The application needs to be tested with several thousands of concurrent users. An application that is performing fine with 10 users might not be able to handle 2,000 concurrent users. Your performance tuning and optimization must be done only under load.

However, if your application has limited concurrent usage, or will never require more than a handful of users, you may not need to put the resources into stress testing under load.

One stress testing tool is from Microsoft: <http://webtool.rte.microsoft.com>. A stress testing engineer is one of the critical roles that should be assigned and is usually at least a full-time engineer on projects with four (4) or more developers. Each deliverable we did in the workbook could be stress tested individually, and then be given a very detailed integrated stress test at the end of each iteration.

When you stress test, you can save money on the number of servers you operate. Stress testing happens in an iterative fashion. The first time you stress test you might only get to 50 users before the server crashes. I have seen servers that handle about 1,000 concurrent web users per CPU, but you need to test your application to see what it can handle. When the server is in production, you should not allow more connections on the server than what it was tested for.

Each PC that is doing stress testing can simulate multiple users, 50-200. If you need to handle 1,000 concurrent users, you might need more than 10 PCs, possibly multiple network cards per machine, and a fast network.

- First you need to record a session:



The scripts you should record should come from your use cases. There are typically several classes of users. You should know how many users there are for each class. There may be 600 users that do X and 150 users that do Y, etc.

QA saves money on many fronts and always gives a great ROI, especially if the application is commercially distributed and requires external support infrastructure. If you catch a single bug before deployment/shipment, you have paid for the QA effort.

If you have more than 10 million rows, a large portion of the performance will be JOINS. Removing a single JOIN can exponentially increase performance. To remove joins, see the

chapter on RDBMS. SQL SELECT iterates for each row in each table. Removing joins on a small RDBMS has no effect.

Performance Enhancers

- Enhanced JVM, such as JRockit
- Optimized application servers (such as Orion or BEA).
- Cached disk IO, such as Mylex.
- Fast JDBC driver with published test results.
- For cost and performance reasons, you might want to use Linux with multiple CPUs because of blocked threading.

Review

Performance is the key indicator as to whether the application is accepted by the user community. QA and stress testing are critical to insure application stability and performance metrics.

Database access usually contributes the most to application lag and the dynamics of the database functions should be streamlined as much as possible, including reducing the number of joins in SELECT statements.

Chapter 25: Audit Queue

Goals

Be able to track what the users clicks on and where they go or come from.

Audit

There are clients who track everything a user does at their site. There are times when you want to track click-throughs of your skyscraper tile (east or right hand vertical).

It is not a good idea to write to the database immediately because it is slow, but we must collect that information in real-time. We will write to the database later, using a separate thread. We need to use a thread safe and fast collection queue for asynchronous processing.

Instead of explaining threads or collection, let's use Doug Lea's Linked List Queue from <http://gee.cs.oswego.edu/dl/classes/EDU/oswego/cs/dl/util/concurrent/intro.html>.

- We will start the LinkedQueue as a singleton on init.

Audit Queue Lab

When a user clicks on something in our “SpaBaseBean” action

class, we can have a method such as `que.put(user,action)` that will capture the link information. Since there could be millions of simultaneous users on our site, this method could be rather busy and the queue would grow rather quickly.

To shrink the queue, we will need to write a record from the queue to the database and delete it from the queue without too much locking and blocking.

This is a pseudo code fragment:

```
ActionForward af = spaPerform( mapping, form,
request, response);
// here we could code after call
// singleton of LinkedQueue
try {
    LinkedQueue q= new LinkedQueue();
    q.put(request.getRequestURI());
} catch (Exception e) {}
    //= new LinkedQueue();
    return af;
} //perform
```

Now we need an init servlet that will read the queue, write to the database, and remove the record from the queue.

Chapter 26: Content and Syndication

Goals

To be able to have a company newsletter on the web. Reivew the example that ships with “struts-tiles” under the RSS folder.

Simple Content Displaying

In order to publish an article on the web, you should not do it in JSP if you are using an MVC framework.

You should create a table (CONTENT):

- PK
- Title
- Meta Data (key words)
- Short Copy
- Image (Blob)
- Long Copy
- Status (Approved)
- Rank
- Click-Through Count

Then create a list page that lists articles with a link in Title.

When a user clicks on a title, they get to body of the article.

Rank should be an integer that lists the order of articles should be displayed. Higher ranks get displayed on top, such as newer or more relevant copy.

Content Syndication Lab

- Create the above CONTENT table.
- Create MVC to List.
- Create MVC to Zoom.
- Insert a few articles.
- [OPTIONAL] Use audit queue to count click-throughs.

Content Syndication

The problem with publishing a newsletter is acquiring content. The CONTENT table can be fed by RSS service providers such as MoreOver and Corante.

The RSS providers charge between \$50 to \$500 per month to receive channels (specific content) and the content may be captured in your table for display and archiving.

```
<SCRIPT  
src="http://www.corante.com/data/personal/js/fullpersonal.js">  
</SCRIPT>
```

or

```
http://p.moreover.com/cgi-local/page?c=Java%20news&client\_id=basebeans\_showcase&o=rs  
s
```

The examples above are not MVC-based. Both Moreover.com and Corante.com will customize an RSS feed to your table

structure. In your table, you can then assign rank based on what you want to display because now you have a stable supply of content.

Content Retrieval Servlet Lab

- Create a servlet that at startup creates a form bean that reads the above feed and writes to your database table.

Chapter 27: Review

You've seen how to leverage the Struts framework to develop MVC web applications with database access, searching, CRUD, iteration, drill-down, master-detail processing, menus, tiles, and security.

We have also demonstrated object-oriented approaches and techniques that should make your development efforts more productive. This approach should also allow you to focus on the business requirements in a productive manner as opposed to exhausting resources on technical challenges.

We have also discussed the implementation and benefits of open standards that allow for flexibility, extensibility, and freedom of choice.

If you do build a sample application using this framework, considering posting it to SourceForge (www.sourceforge.net) so that it may benefit from peer review

Appendix A: Downloads

Software Downloads

- Jakarta-Tomcat
- PostgreSQL RDBMS
- NetBeans IDE
- Mozilla browser
- Struts framework
- PoolMan connection pool manager
- Java Developer Kit (JDK)

In general you should always have a choice of two RDBMs (PostgreSQL and ASA), two application servers (Orion and Tomcat), two editors (gvim and NetBeans), etc.

Installation and Configuration

Workstation and Server Configurations

If the machines you will be using for development have been used for something else in the past or have another JDK installed, ideally you would want to reformat them and install a fresh OS, with all the latest patches, although that isn't an option for the vast majority of developers.

If you decided to do a clean install, I would create a D: partition and put all important files there, just in case you need to re-install quickly. I always put my mail folder in "d:\mail", for example, the OS and applications on the "c:" drive.

Now install any Windows or other application you will need,

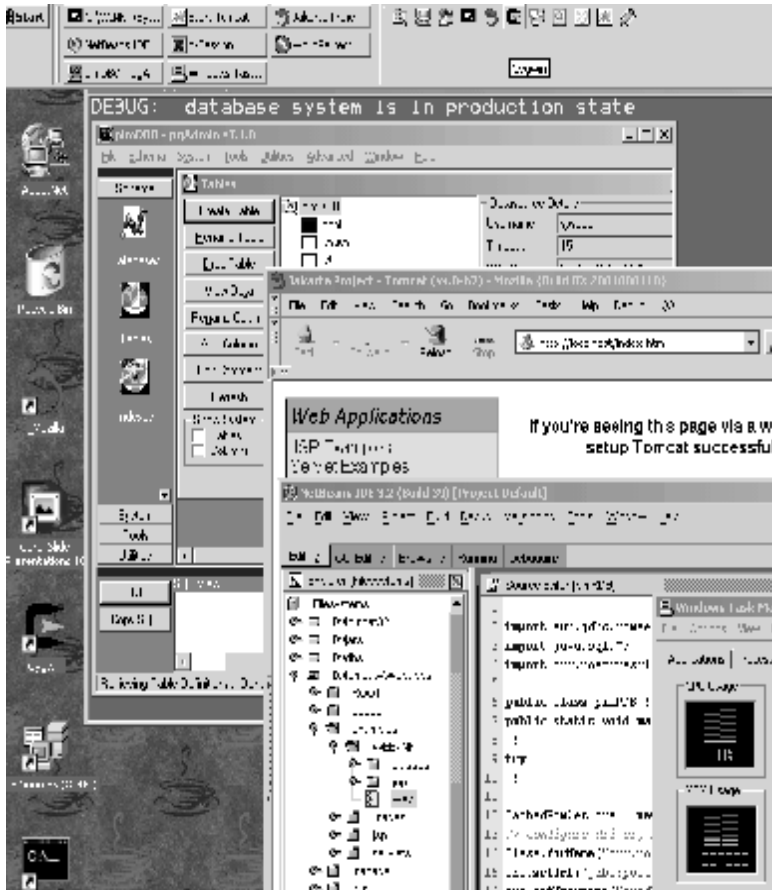
before the rest of the installs (i.e. StarOffice, WinZip, etc). Anything you normally use during development should be installed now.

It is possible to use both development and server software on one machine (as below) under lab conditions, but you will find that it is not practical for real world projects.

You do not want the development environment to crash when you are knee deep in the alligators during the project so I recommend at least one (1) separate server (for the application server and RDBMS), even if you have one or two developers. If this is not practical, make your best judgement.

Windows 2000 should be sufficient for development workstations. You should have 19" or 21" monitor on development workstations. I also recommend that you use Linux or FreeBSD for your development application server (Tomcat) and development RDBMS server (PostgreSQL).

Download and Install Core Software Packages



- Download JDK 1.4 and install on both Development Server and developers Workstations (I used d:\jdk1.4 folder, Linux people can follow Windows directory)

conversions and adopt to Linux). Some software engineers like Linux, easier and faster. However, I will show all my examples using Windows (even on the server side), for the people who do not use Linux.

We are not going to solve technology problems, we will solve business problems, so always use the latest binary build when downloading and avoid intermediate source code builds. (But feel good that source code is available; it gives you choices.)

- Download and install Tomcat 4 (I used d:\Tomcat4. Jakarta web site is under www.apache.org) on development server. You can use any J2EE compliant application server.
- Download and install NetBeans IDE (from Netbeans.org) to workstation. (I used d:\Netbeans). You can use any IDE or Text Editor you like. In fact, you do not need GUI, you can develop under Linux text command line without ever starting X-Windows. If you are one of the more expert readers, then do this whole book in console mode .
- Download and install Mozilla browser (from Mozilla.org) on the workstation. You should not use IE for development, since things that work on it, might not work on other browsers your clients might use. Feel free to use any browser other than IE.
- Download Struts from Jakarta. I select this framework for the sample project but you may want to use another framework for your project. On the development workstations and server, expand Struts under d:\jars. I also install d:\Tomcat4 on development (which I never start), so that I can use similar classpath strings.

- Download and install PostgreSQL on the server. (Ideally you would have a second server machine for you RDBMS server, so that you can more easily tune your code during the performance stress testing.) PostgreSQL comes with Linux. On Windows 2000, you can go to <http://courseweb.xu.edu.ph> ; download and install PostgreSQL. More on database installation in the RDBMS chapter. I installed it under “d:\dbs\PostgreSQL”. *Optional alternative:* You can install Cygwin for Windows from Cygwin.com, which has PostgreSQL included. You can feel free to use any database you like.
- *Optional:* Download the WinAXE X-Terminal software from www.labF.com or other X-Window software so that you can remotely X-Window into Linux, or if you need thin client development for a group. This way the GUI is processed on Windows and everything is on one central Linux. This tremendously speeds up configuration for a group, since it is only done once. This is my favored configuration.
- Set up JAVA_HOME environment variable and CATALINA_HOME environment variable on development servers and workstations.
- Start Tomcat. From Mozilla, display the home page of Tomcat. Works? Test out the Tomcat examples.

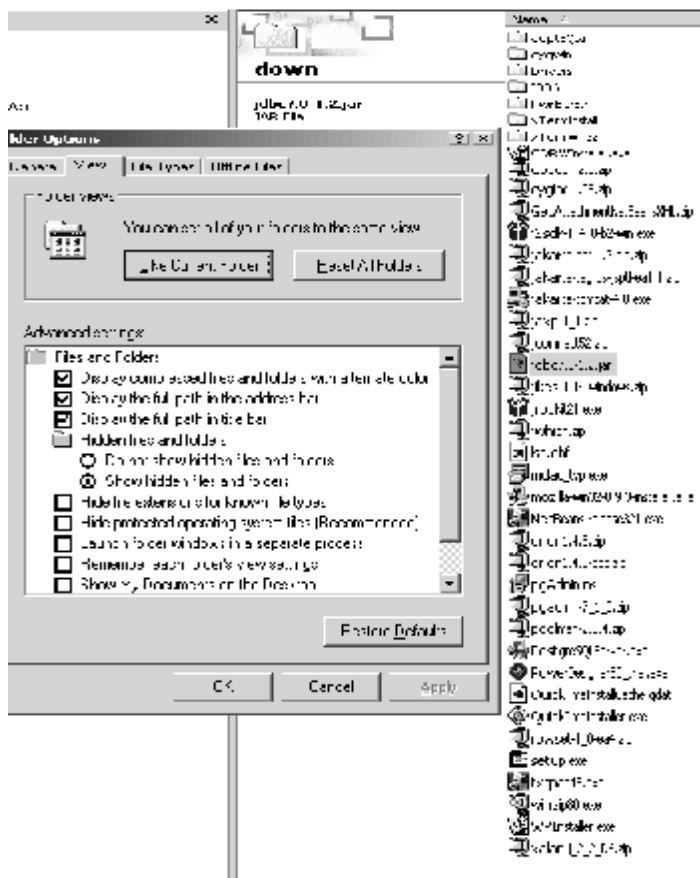
There are many books that are more introductory and step-by-step. I am trying to aim this book more at people who are experienced programmers and most should be able to configure the above environment.

You have installed a J2EE-compliant application server (Tomcat; Orion), an RDBMS (PostgreSQL), a browser (Mozilla), and an

IDE (NetBeans) or text editor. Hopefully, the application server is up and running and you can browse the sample page from your browser. In addition, you have downloaded and installed Struts.

Congratulations on your first major milestone! Pat yourself on the back.

I use WinZip for Jars and Wars. Here are some of my settings and downloaded files, which I will explain below (if you know what they are, then skip ahead):



You should show the file extensions for known file type. You should also set up the command line environment to your liking (colors, fonts, etc.) since you will be spending some quality time there, even in Windows.

Download the RDBMS utilities

- The PostgreSQL graphical administration tool may be downloaded from the PostgreSQL organization (<http://www.postgresql.org>). Download and install the pgAdmin tool for Windows. *Hint:* You install it by right-clicking on the downloaded file.

You will also need (if you are using Windows) some additional ODBC files for use with the pgAdmin tool. You may download the latest MAC's from Microsoft (www.microsoft.com/data).

PostgreSQL comes pre-installed on most Linux distributions and the pgAdmin tool is included.

- Download and install the JDBC drivers for PostgreSQL (<http://jdbc.fastcrypt.com/>). Cygwin also has JDBC drivers for PostgreSQL.
- Download the latest PoolMan connection pool JAR from www.codestudio.com. You are free to use any connection pooling that you like, but the labs and examples will use PoolMan.

From the Developer's Connection (<http://developer.java.sun.com/developer/earlyAccess/crs/>), download the JDBC RowSet classes. We will use this for our result sets and not the older JDBC ResultSet object. You will have to register for free to get access to it at the above link.

Other files you will need in you d:\down folder:

- JAXP Java XML handler from sun.java.com/xml
- <http://jakarta.apache.org/taglibs/doc/jsptl-doc/intro.html> has the Standard Tag Lib
- Xalan XSL processor from xml.apache.com. (Xalan is not compatible with Tomcat4 but we can use it to manually test XML files).
- Download “struts-menu.zip” from Husted.com .
- Download the regular expression “struts-validator.jar” from Jakarta
- Download the portal tiles from <http://www.lifl.fr/~dumoulin/tiles/>
- Download the Struts validation JAR from <http://home.earthlink.net/~dwinterfeldt/index.html>
- Jikes compiler from (faster than Sun’s) from www.ibm.com/developerworks/oss, you may use it instead of javac. Install it to d:\jdk1.4 \ bin, or put it in your path.
- *Optional:* JRocket VM / p-code interpreter from www.JRocket.com (Faster than Sun’s VM), use instead of Suns.
- *Optional:* Download the standard JSP tags from www.apache.com/Jakarta
- Create a folder under “d:\jars” called “classes”. Place JWhich.java there. Run javac and jikes on JWhich to generate a class and test our development environment.

Can you compile?

System Configuration

- Expand all the other JARs under the JAR folder, such as JAXP, Xalan, Poolman, etc.
- Install other files as needed.
- Go back to NetBeans.org and download optional XML module and the DB/SQL module and install them. Also, other modules you might like, maybe text compare.
- Share the Tomcat folder and the webapps folder under Tomcat, if running under Windows. Share other folders as needed.

Now let's set up the CLASSPATH.

Your path might look like this:

c:\winnt\system32;d:\winnt;d:\pb8\Shared\PowerBuilder;d:\jdk1.4\bin

This way we can go to a command prompt and do a javac. You should copy jikes to d:\jdk1.4\bin, so you can use jikes instead of javac.

The CLASSPATH is basically a PATH for Java.

<!-- And Java is just P-code, JAR is just a TAR, SQL is just relational algebra, exceptions are used instead of if > 0 for each line of code -- >

- Expand the Poolman from downloads to the “jars” folder and copy “poolman.jar” to “d:\jars”.
- Expand RowSet from downloads to Jars folder and copy

RowSet.jar to d:\jars;

We will later have to add JDBC driver files so that we can build our database application.

Also, here is a sample CLASSPATH environment variable I use, for those who just want to dive in. We will fill in the jars later. You can use it as a reference, but your CLASSPATH might be slightly different.

For example, this is more for a server. On a workstation, most jars would be under jars folder, not under Tomcat\lib.

```
. ;D:\Tomcat4\common\lib\servlet.jar;D:\jdk1.4\lib\tools.jar;d:\jars\classes;d:\Tomcat4\jasper\jasper-compiler.jar;W:\vicWebPim\WEB-INF\lib\struts.jar;w:\vicWebPim\WEB-INF\lib\poolman.jar;w:\vicWebPim\WEB-INF\lib\rowset.jar;w:\vicWebPim\WEB-INF\lib\jdbc2_0-stdext.jar;w:\vicWebPim\WEB-INF\lib;w:\vicWebPim\WEB-INF\lib\postgresql.jar;d:\jdk1.4\jre\lib\rt.jar;W:\vicWebPim\WEB-INF\classes
```

// Xerces must be loaded first, so we must keep in mind to do that. Also, struts.jar will need to be in every webapps\lib folder.
// Poolman\lib needed to access the Poolman.XML pool configuration on server only.

- We should set up the PATH and CLASSPATH and the JAVA_HOME and CATALINA_HOME on both server and each workstation.

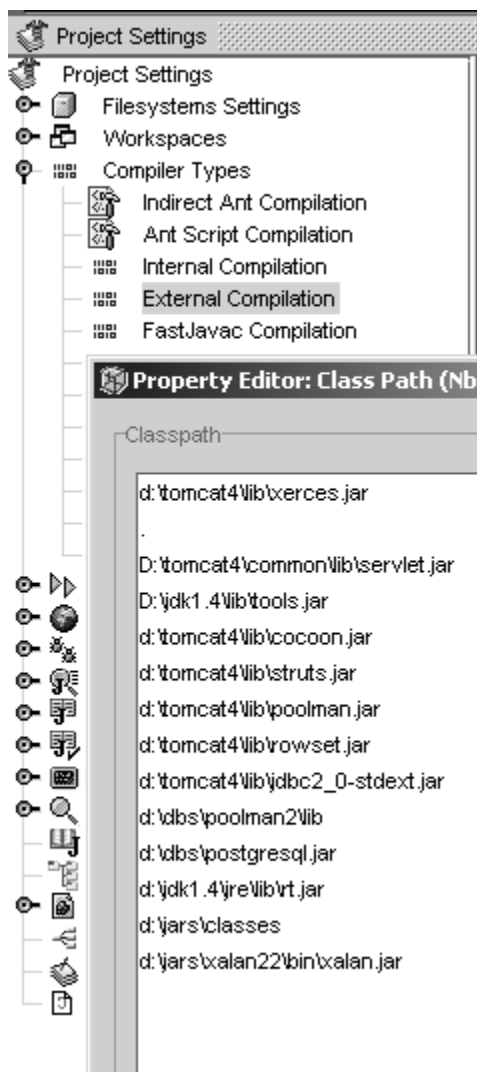
For Windows, right click on Computer Icon, then Advanced tag, then Environment Variables. In the system properties, add a

CLASSPATH variable, with above as the value, like below:

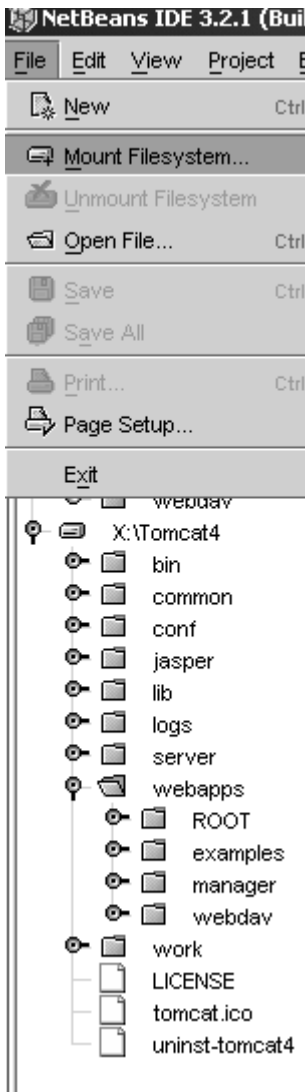


NetBeans IDE CLASSPATH

- Now start up the NetBeans IDE, so we can set up the CLASSPATH. (If using text editor, you are fine.)
- Click on Project, Settings, expand on compiler types like this:



- For each Compilation tab, expand it and click on Expert mode towards the bottom. Paste the CLASSPATH string.
- You remember that we shared the folders on the server? We want to be able to get to Tomcat and webapps from IDE, so mount the directory drive using File, Mount:



We will spend some more time on setting up Struts in the Setup chapter. (*Optional:* You could copy the war files from “jars\struts” to your “.\tomcat\webapps” folder and restart Tomcat. Take a look at some sample struts applications, including the “struts-blank.war” The “struts-blank.war” is a good starting template to create applications from since the “web.xml” and the TLD’s (tag libraries) are set up. More later...)

Alternatives

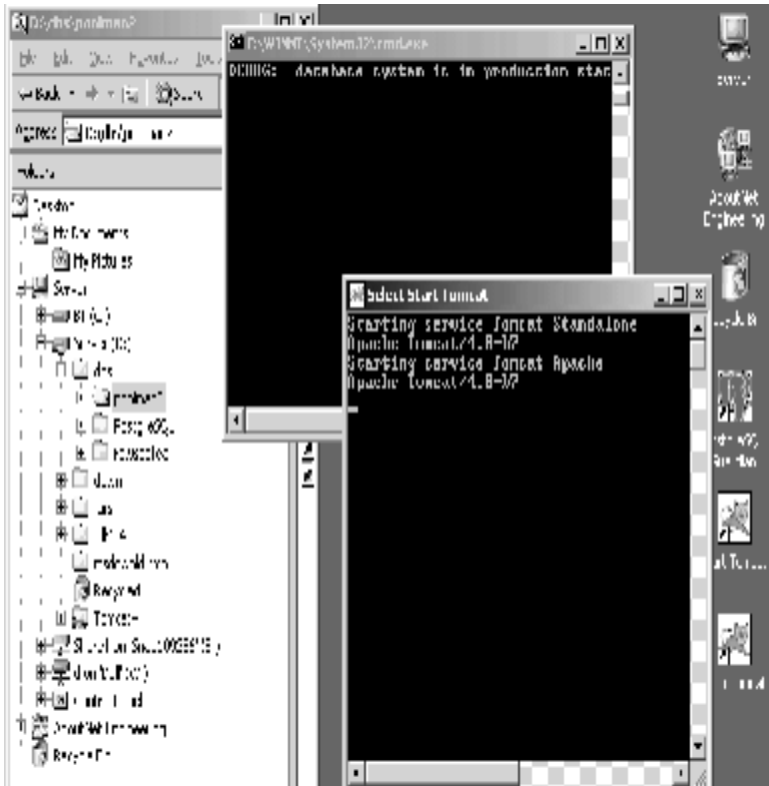
You can use any IDE or editor you like, and any J2EE application server or servlet server, any RDBMS and any OS that has a JRE you like, for example Mac OS X.

We are using Tomcat since it is the reference J2EE server, and it is free for development AND deployment. Any application that runs on Tomcat should work on other application servers in order for the other servers to be J2EE certified. Some people like Orion (<http://www.orionserver.com>; commercial) for production.

PostgreSQL is free and it is ANSI SQL-92 compliant as well as fast in a large scale production environment.

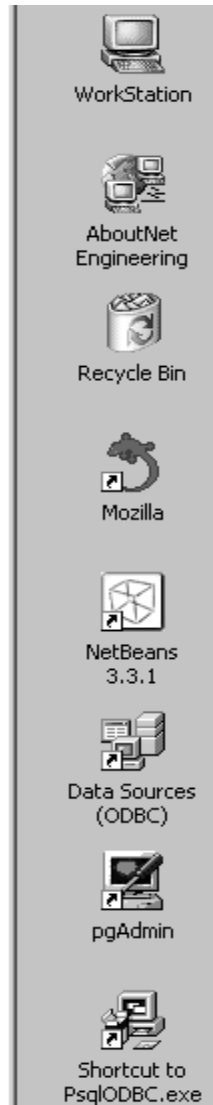
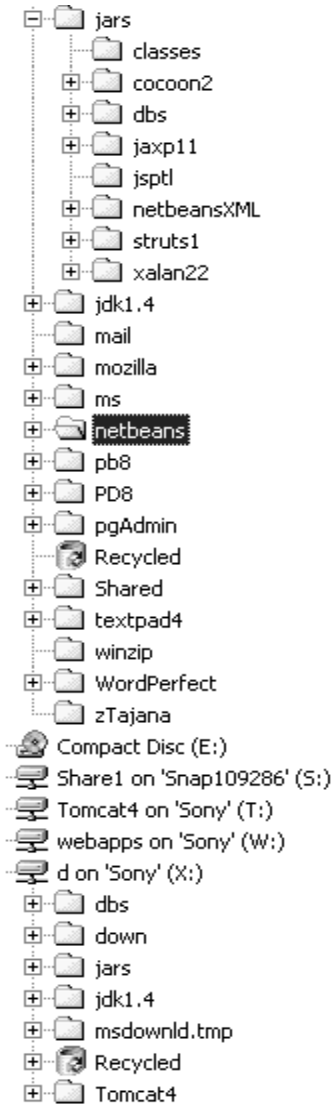
NetBeans is a free IDE, although you may choose any IDE you like (including JBuilder, VisualAge, et al).

Ideally, you should be able to do these exercises and labs with any framework, but I selected Struts as a base foundation and most examples and labs are Struts-specific. However, I think there is still some relevant material in this book if you are using another framework. You can choose any framework you would like for any of your projects and the concepts in this book should still be applicable, if not the specific source code.



Your server may not look exactly as the figure above since we only have the RDBMS and Tomcat server installed.

And your development workstation might look the figure below. Note that the NetBeans IDE and Mozilla browser are here, but the RDBMS and application server on the the server machine.



Once again, if you only have a single machine to do these exercises and labs then your setup should look similar to these two figures combined.

Review

You should have a “jars” folder, a “download” folder, and a “webapps” (Tomcat) or “applications” (Orion) folder.

We have installed Tomcat, NetBeans, Mozilla, and PostgreSQL and configured our system for the labs and exercises.

Appendix B: Technology Architecture

We have spent the whole book talking about application architecture. With a framework, it is more difficult to mess up an application. You can improve the framework, and use more helper objects always, but....

Let's touch base on technology architecture. First of all, when deciding what to buy, review www.tpc.org.

What to buy?

There are companies out there who do not like to use open standards. Several reasons.

One is cultural, organizations have purchasing departments, and they are set up to buy. People feel better when they say I have spent X number of dollars on this software.

The second reason is legal culture. Most organizations have older legal counsel not familiar with IT and Intellectual Property. It goes like this.

There are many open source licenses out there. Apache, GPL, LGPL, etc. Some of them say that if you modify the software, the improvements are public domain or something like that. So lawyers mistakenly advise companies not to use open standard since they go to public domain, just to be safe.

But if they really read the license they would know they are not modifying software, they are writing applications, and you own the application.

For example, let's say you purchase the source code for WebLogic. The license might say changes to source code of WebLogic are property of BEA, and you just have a license. They are in business of creating J2EE systems software.

Say you write a purchase order application that runs on WebLogic. Who owns that?

If you find that you get good support and the annual upgrade assurance and license cost give your organization a good profit margin, don't worry.

I like/prefer products that are less expensive. For example, using Struts for portals, content syndication, and application development, as opposed to buying an expensive content management portal.

One reason is that business change and the vendor might not support my favorite product in a few years. So I want the source code, but that is either prohibitively expensive or unavailable for commercial frameworks.

Also, I might not work for the same company in a few years so I want to develop my skills on industry standard technology.

For example, my next company might use iPlanet and they will not give me a good salary if first they have to train me. So I like to learn something that keeps me technically adroit.

A good organization might also give out bonuses if an employee saves them money.

Large applications

This comes more into a play on larger applications. If you only have a 1,000 concurrent users with 5 million master rows, this fits on a laptop. Costs and support needs are relatively small.

But what if you need to have 3 sites with 5 servers each. That is 15 production machines and all the software.

Plus a staging and development environment. Plus support. Costs go up considerably unless you are not proprietary.

How would you set up a large environment with 3 sites?

Typically you want one master site that has a read-write database server. This is your master and a bottleneck machine. Use many caching disk controllers and look at the TPC.org site.

You would then replicate the data to 2 “slave” database sites. This would be read-only, with only queue logging tables writeable, and anything temporary like that.

Your application servers would then connect to these servers, creating 2 connection pools each. One connection pool would talk to the read-write database and would only be used for updates and to verify business rules (“Did the client exceed the credit line?”).

The other pool is for the read-only database, used for SELECTs and reports.

Each database server could have many application servers connect to it.

The user would connect through a Cisco router that would hide the name of the box. Most J2EE servers, such as Orion, have application server fail over and clustering so a user would not know that a server crashed.

When a user connects to a server to start a session, the application server would invoke a load helper object that would determine the closest geographically application server to the user (based on IP) with a low load and connect to that server.

I also create a cron job (in Unix/Linux) to reboot all the production server machines every Friday night (or another predetermined bounce time). I would also hold quarterly drills at each site to rebuild the database server on a new box and put it back on-line. This way, if a server ever does crash, it is routine for a site to make a new one. (All backed up row-by-row to a flat file).

Appendix C: Struts Tiles

Printed by permission of the author [Cedric Dumoulin](#).

Copyright © 2000, Apache Software Foundation
and Cedric Dumoulin

Tiles (Components)
Tutorial v0.4
Author : [Cedric Dumoulin](#)
Date : 14 Nov. 2000
Rev : 9 Sep. 2001

Tutorial overview

This tutorial shows you how to write and use Tiles to define a Web site. Use this tutorial to learn the basic skills you'll need to develop a Web site and to learn about some of the features of the Tiles Framework.

In this tutorial, you will create a fictitious web site, with pages illustrating Tiles features : basic page, portal page, definitions, i18n.

This tutorial cover the view part of the UI, it doesn't cover the model and controller part of the MVC 2 model.

Along the way, you'll learn how to perform the following tasks:

- Build a page
 - Identify tiles
 - Write your tiles
 - Assemble tiles
 - Write a layout tiles
- Reuse existing layouts / templates
 - Reuse classic layout
 - Reuse portal layout
 - Reuse vertical boxes layout
- Use screen or tiles definitions
 - Declare and use definitions
 - Extends definitions
 - Definitions as tile parameters

- Struts and definitions
- Design intelligent tiles
 - Build new layouts
 - Reusable sub-menu
 - Reusable tiles (included twice)
 - View sources tiles and menu
- Create i18n tiles

3 Setup your development environment

In order to run examples of this tutorial, you need to install a JSP server, like Tomcat. You also need the Tiles Library, the Struts Library, and some jar files from the Jakarta Commons project. If you have downloaded Tiles binary distribution, you already got all needed files.

We suppose that you have some basic knowledge of web development, JSP and JSP tags.

Examples of this tutorial are in the Tiles distribution file.

However, we advise you to write yourself the examples. For that, you can create a directory under the web server webapps directory (call it tutorial), and put all your files and sub-directories under this directory.

In order to avoid rewriting configurations files, you can copy the complete WEB-INF directory from the tutorial to your directory.

4 Write your first page

Your first page is a simple page that will be improved later in this tutorial.

4.1 Plan your tiles and main layout

Your first page look like the following picture :



It is divided in header, footer, menu and body. Such division identifies our main tiles :

- header : containing company logo
- footer : containing copyrights
- menu : containing direct links to others pages
- body : the main area of the page

These tiles are assembled together using a layout or template. This layout is also a tile. As layouts are often used, there exists a library of layouts. But, in this tutorial, you will learn how to write simple layout. Once you have your main tiles and your layout, you need to set up your web page main entrance (i.e. : define the web page that will process the http request). This can simply be done in a jsp file.

Starting from the page main entrance, we can draw a tree of tiles, like the following :



You will next learn how to write these tiles, the layout, and the page entrance.

4.2 Write Tiles

Covered topics : write simple tiles; the `?tml link?problem`.

4.2.1 Header

Your header will contain the company logo.

Copy directory `tutorial/images` from the distribution tutorial directory to your examples root directory.

Create a new blank file, and save it under `tutorial/common/header.jsp`.

Edit your new file, and place your images, as in following :

```
<a href="http://www.mycompany.com">
  
</a>

```

You can note that images path is absolute (starting with `??`). You need to use an absolute path because this component can be included from pages residing at different places, not always at your site root.

4.2.1.1 The `?tml link?problem`

Using simple absolute path work fine if your web site is placed directly at the root of your domain name (i.e. : at www.mycompany.com). If your web site is under a subdomain (i.e. : at www.mycompany.com/my_site), as it is often the case, you need to prefix this path with the subdomain name (`my_site`). In order to have a portable site, you can retrieve the subdomain name from the http request.

Our component becomes :

```
<a href="http://www.mycompany.com">
  
</a>

```

It is a good habit to prefix absolute links used in client side (html links) by the subdomain name, retrieved from the http request. In this tutorial, you will always prefix html links with subdomain name.

If you also plan to use Struts, you can use the `<link>` or `` tag defined in html tags library.

```
<% @ taglib uri="/WEB-INF/struts-html.tld" prefix="html" %>

<a href="http://www.mycompany.com">
  <html:img src="/tutorial/images/headerLeft-logo.gif" align="left" border="0">
</a>
<html:img src="/tutorial/images/headerRight-logo.gif" align="right" border="0">
```

4.2.2 Footer

The footer contains company copyrights.

Create a new file, and save it under `tutorial/common/footer.jsp`.

Write copyrights notice, as in following :

```
<div align="center">
  <font color="#023264" size="-1">
    <em>      Copyright &copy; 2001, MyCompany      </em>
  </font>
</div>

```

4.2.3 Menu

The menu contains links to other pages. For now, you just write a simple menu. You will create more sophisticated menus later in this tutorial.

Create a new file, and save it under `tutorial/basic/menu.jsp`.

Now, you can create your menu : create a table, with three rows. Normally, each row contains a link to a page. Here, only first row contains link to the main page. Your code should look like the following :

```
<table>
<tr bgcolor="Blue">
  <td>
    Menu
  </td>
</tr>
<tr>
  <td width="120" valign="top">
    <font size="-1">
      <a href="<%=request.getContextPath()%>/tutorial/index.jsp">Home</a>
    </font>
  </td>
</tr>
<tr>
  <td width="120" valign="top">
    <font size="-1">Item 2</a></font>
  </td>
```

```

</tr>
<tr>
  <td width="120" valign="top">
    <font size="-1">Item 3</a></font>
  </td>
</tr>
</table>

```

4.2.4 Body

The body is the active part of your pages. For now, you will develop a simple body writing the classic sentence ?ello the World? You will develop other bodies later.

Create a new file, and save it under tutorial/basic/hello.jsp.

In your file, write ?ello the World?

The code should look like the following :

```

<div align="center">
  <font size="+1"><b>Hello the World</b></font>
</div>

```

4.3 Assemble Tiles in a Page

Covered topics : passing parameters, basic layout understanding, assembling tiles in layout, insert tag mechanism understanding. Retrieving parameter values.

Now, you will assemble your tiles. For that, you will develop a tile taking in charge the layout, and including previously developed tiles where you want them to reside. This tile can be seen as a Template.

In order for your layout tile to be reusable, you will pass it tiles to be inserted as parameters.

4.3.1 Passing Parameters

To pass parameters to the tile that will do the layout, you write something like that :

```

<% @ taglib uri="/WEB-INF/tiles.tld" prefix="tiles" %>

<tiles:insert page="/basic/myLayout.jsp" flush="true">
  <tiles:put name="title" value="My first page" />
  <tiles:put name="header" value="/tutorial/common/header.jsp" />
  <tiles:put name="footer" value="/tutorial/common/footer.jsp" />
  <tiles:put name="menu" value="/tutorial/basic/menu.jsp" />
  <tiles:put name="body" value="/tutorial/basic/helloBody.jsp" />
</tiles:insert>

```

Or like the following, which is the same, but looks like more familiar to people already using Struts Template :

```

<% @ taglib uri="/WEB-INF/tiles.tld" prefix="template" %>

<template:insert page="/basic/myLayout.jsp" flush="true">

```

```

<template:put name="title" value="My first page" />
<template:put name="header" value="/tutorial/common/header.jsp" />
<template:put name="footer" value="/tutorial/common/footer.jsp" />
<template:put name="menu" value="/tutorial/basic/menu.jsp" />
<template:put name="body" value="/tutorial/basic/helloBody.jsp" />
</template:insert>

```

Create a new file, write previous line of code and save it under tutorial/basicPage.jsp.

The first line instructs the web server to use the tiles tag library. It also specifies that all tags from this library will start with the prefix tiles.

The tag itself specifies the tiles page to insert : /tutorial/basic/myLayout.jsp. You will develop this tile in a few moments. We pass parameters to this tile, each one is identified by a name. The value can be of any type. Here we use hard-coded strings.

4.3.2 Write Layout / Template

First, create a new file, and save it under tutorial/basic/myLayout.jsp.

In this file, create a table reflecting your layout, as describe in [Plan your components and main layout](#).

The code looks like this :

```

<html>
<head>
  <title></title>
</head>

<body>
<TABLE width="100%">

  <TR>
    <TD colspan="2">header</TD></TR>
  <TR>
    <TD width="120">menu</TD>
    <TD>body</TD></TR>
  <TR>
    <TD colspan="2">footer</TD>
  </TR>
</TABLE>

</body>
</html>

```

Note : This layout contains <html>, <head> and <body> tags. This tile is used to define the html page.

4.3.3 Insert Tiles

Now, you will insert (include) tiles by using special tags.

Put the following code at the beginning of your page :

```
<% @ taglib uri="/WEB-INF/tiles.tld" prefix="tiles" %>
```

This instructs jsp processor to use the special tags library "/WEB-INF/tiles.tld", and that tags will be prefixed by `?/span>tiles?`

Next, you can insert your tiles by using following tag :

```
<tiles:insert attribute=?ody?flush=?rue?/>
```

This tag inserts a tile whose name is taken from the value of the attribute `?ody?`. It is why you must use `attribute=?`.

Do the same for each of your tile. The code should look like :

```
?/span>
<TABLE width="100%">

  <TR>
    <TD colspan="2"><tiles:insert attribute="header" /></TD></TR>
  <TR>
    <TD width="120"><tiles:insert attribute="menu" /></TD>
    <TD><tiles:insert attribute="body" /></TD></TR>
  <TR>
    <TD colspan="2"><tiles:insert attribute="footer" /></TD>
  </TR>
</TABLE>
?/span>
```

4.3.4 Set Page Title

Your layout tile will be used several times. But you certainly want a different title for each page using this layout. For that, you will pass the page title as a parameter. In the layout, you extract the title from tile? parameters, and put the value in the correct place :

```
?/span>
<head>
  <title><tiles:getAsString name="title"/></title>
</head>
?/span>
```

Tag `getAsString` (formerly `getAttribute`) retrieves the value of a tile? attribute, and write it directly as a string. So, the tag will be replaced by the value you passed to the tile.

4.3.5 Complete code

Following is the complete code for your first layout tile :

```
<% @ taglib uri="/WEB-INF/tiles.tld" prefix="tiles" %>
```

```
<html>
<head>
  <title><tiles:getAsString name="title"/></title>
</head>
```

```

<body>
<TABLE width="100%">

  <TR>
    <TD colspan="2"><tiles:insert attribute="header" /></TD></TR>
  <TR>
    <TD width="120"><tiles:insert attribute="menu" /></TD>
    <TD><tiles:insert attribute="body" /></TD></TR>
  <TR>
    <TD colspan="2"><tiles:insert attribute="footer" /></TD>
  </TR>
</TABLE>

</body>
</html>

```

4.4 Try Your Page

You can test your page by starting your web server, and pointing to your newly created page `?asicPage.jsp?`

Note : your configuration files must be set correctly before starting the web server. For now, you can simply copy the entire WEB-INF directory in your tutorial directory.

5 Reuse Existing Layouts

Covered topics : Reuse layouts.

Assembling of tiles is often done in the same way. It is possible to develop some basic layouts and reuse them inside a web site, or even from site to site.

Provided examples come with some layouts : classic layout, portal (columns), vertical boxes.

You will first learn how to reuse such layouts. In a later chapter, you will learn how to build your own layouts.

5.1 Classic Layout / template

The most common layout for a page is the header / menu / body / footer layout.

We provide one of this layout under `/tutorial/layout/classicLayout.jsp`. You can use it in place of your previously defined layout : simply change the name of included tile in `basicPage.jsp` :

```

<% @ taglib uri="/WEB-INF/tiles.tld" prefix="tiles" %>

<tiles:insert page="/layout/classicLayout.jsp" flush="true">
  <tiles:put name="title" value="My first page" />

```

```

<tiles:put name="header" value="/tutorial/common/header.jsp" />
<tiles:put name="footer" value="/tutorial/common/footer.jsp" />
<tiles:put name="menu" value="/tutorial/basic/menu.jsp" />
<tiles:put name="body" value="/tutorial/basic/helloBody.jsp" />
</tiles:insert>

```

Note that there is no change in the passed parameters. In the same way, you can change the way a layout component laid sub-components. If well designed, you can change all site look and feel simply by changing main layout component !

5.2 Portal

Covered topics : use of layout inside tiles, list as parameter to a tile.

Here, you will learn how to build a portal by assembling tiles.

A portal is made of tiles, ?tacked ?into columns. Portal page use the layout common to all site. So, you only need to define the body of your portal page.

This body uses a tile that lay sub tiles in columns. You need to pass as parameters the number of columns you want, and a list of tiles for each column.

5.2.1 Prepare Portal Tiles

First, you need to define some tiles to assemble in your page. You can define your own tiles, or copy the one defined in the tutorial (under tutorial/portal/*). Copy tiles from directory ?/span>portal/*?in your tutorial directory (keep the sub directory ?/span>portal?).

5.2.2 Assemble Portal Tiles

Create a new file and save it under tutorial/portal/portalBody.jsp.

Include the portal layout (called /tutorial/layout/columnsLayout.jsp), passing it the requested number of columns, and a list of tiles for each column. List names must be ?/span>listX? where ?/span>X?is the index of the list, starting at zero.

Your code looks like following :

```

<%@ taglib uri="/WEB-INF/tiles.tld" prefix="tiles" %>

<tiles:insert page="/tutorial/layout/columnsLayout.jsp" flush="true">
  <tiles:put name="numCols" value="2" />
  <tiles:putList name="list0" >
    <tiles:add value="/tutorial/portal/login.jsp" />
    <tiles:add value="/tutorial/portal/messages.jsp" />
    <tiles:add value="/tutorial/portal/newsFeed.jsp" />
    <tiles:add value="/tutorial/portal/advert2.jsp" />
  </tiles:putList>
  <tiles:putList name="list1" >
    <tiles:add value="/tutorial/portal/advert3.jsp" />
    <tiles:add value="/tutorial/portal/stocks.jsp" />
    <tiles:add value="/tutorial/portal/whatsNew.jsp" />
    <tiles:add value="/tutorial/portal/personalLinks.jsp" />
    <tiles:add value="/tutorial/portal/search.jsp" />
  </tiles:putList>

```

```
</tiles:insert>
```

In this code, you declare three attributes : one variable and two lists. Values are added to the list in the specified order. Here, you add sub-tiles URLs. It is possible to add definitions name instead (see definitions).

5.2.3 Set up web page

You can now include your portal body in any tiles. Preferably, you will include it in a page using the main layout.

Create the page in a new file called `/tutorial/portalPage.jsp` :

```
<%@ taglib uri="/WEB-INF/tiles.tld" prefix="tiles" %>
```

```
<tiles:insert page="/tutorial/layout/classicLayout.jsp" flush="true">
  <tiles:put name="title" value="My First Portal Page" />
  <tiles:put name="header" value="/tutorial/common/header.jsp" />
  <tiles:put name="footer" value="/tutorial/common/footer.jsp" />
  <tiles:put name="menu" value="/tutorial/basic/menu.jsp" />
  <tiles:put name="body" value="/tutorial/portal/portalBody.jsp" />
</tiles:insert>
```

Note that you use `?classicLayout.jsp?` rather than your `?yLayout.jsp?` Both layouts are nearly the same, except than `classicLayout` include a style sheet used in the entire site.

Check result by pointing your browser on the page.

5.3 Vertical Boxes

We often need to put tiles one under the others. The vertical boxes layout (`/tutorial/layout/vboxLayout.jsp`) does exactly that. You pass it a list of tiles, and it laid them vertically.

As an example, we will improve the menu : it will now be made of three tiles : one showing an image, one providing links to pages and one providing links to a tile writing sources.

5.3.1 Assemble Tiles

Create a new file and save it under `/tutorial/common/menu.jsp`. In this file, include the `vboxLayout` tile, and pass it a list with sub-tiles URLs.

Your code should looks like :

```
<%@ taglib uri="/WEB-INF/tiles.tld" prefix="tiles" %>
```

```
<tiles:insert page="/tutorial/layout/vboxLayout.jsp" flush="true" >
  <tiles:putList name="componentsList" >
    <tiles:add value="/tutorial/common/menu/menuLogo.jsp" />
    <tiles:add value="/tutorial/common/menu/menuLinks.jsp" />
    <tiles:add value="/tutorial/common/menu/menuSrc.jsp" />
  </tiles:putList>
</tiles:insert>
```

The `vboxLayout` tile requires one parameter called `?componentList?` It is a list of URLs referencing sub-tiles to insert. Of course, you need to have such tiles.

`MenuLogo.jsp`

Following is the code of this tile :

```

```

MenuLinks.jsp

Following is the code of this tile :

```
<% @ taglib uri="/WEB-INF/tiles.tld" prefix="tiles" %>

<tiles:insert page="/tutorial/common/submenu.jsp" flush="true">
  <tiles:put name="title" value="Main Menu" />
  <tiles:putList name="items" >
    <tiles:add value="Home" />
    <tiles:add value="Basic Page" />
    <tiles:add value="First Portal" />
  </tiles:putList>
  <tiles:putList name="links" >
    <tiles:add value="/tutorial/index.jsp" />
    <tiles:add value="/tutorial/basicPage.jsp" />
    <tiles:add value="/tutorial/portalPage.jsp" />
  </tiles:putList>
</tiles:insert>
```

This tile uses another tile, submenu.jsp. This later takes a list of items and a list of links as parameters, and shows them in a menu fashion. You will learn later how to write the submenu tile. For now, copy it from the original /tutorial/common directory to your /tutorial/common/submenu.jsp

MenuSrc.jsp

Following is the code of this tile :

```
<% @ taglib uri="/WEB-INF/tiles.tld" prefix="tiles" %>

<tiles:insert page="/tutorial/common/menuViewSrc.jsp" flush="true" >
  <tiles:putList name="list" >
    <tiles:add value="/tutorial/basicPage.jsp" />
    <tiles:add value="/tutorial/portalPage.jsp" />
    <tiles:add value="/tutorial/portal/portalBody.jsp" />
    <tiles:add value="/tutorial/common/header.jsp" />
    <tiles:add value="/tutorial/common/menu.jsp" />
    <tiles:add value="/tutorial/common/footer.jsp" />
    <tiles:add value="/tutorial/layout/classicLayout.jsp" />
    <tiles:add value="/tutorial/layout/vboxLayout.jsp" />
  </tiles:putList>
</tiles:insert>
```

This tile uses another tile, menuSrc.jsp. This later takes a list of pages URLs, and shows links pointing to a tile writing source of an URL. You will learn later how to write needed tiles. For now, copy them (viewSrc.jsp; viewSrcBody.jsp, menuViewSrc.jsp) from the original /tutorial/common directory to your /tutorial/common/.

5.3.2 Set up web page

You can now use your new menu.

Edit the `/tutorial/portalPage.jsp`, and specify the path to your new menu :

```
<%@ taglib uri="/WEB-INF/tiles.tld" prefix="tiles" %>

<tiles:insert page="/tutorial/layout/classicLayout.jsp" flush="true">
  <tiles:put name="title" value="My First Portal Page" />
  <tiles:put name="header" value="/tutorial/common/header.jsp" />
  <tiles:put name="footer" value="/tutorial/common/footer.jsp" />
  <tiles:put name="menu" value="/tutorial/common/menu.jsp" />
  <tiles:put name="body" value="/tutorial/portal/portalBody.jsp" />
</tiles:insert>
```

Check result by pointing your browser to the page.

You have certainly notice that the vertical boxes example require a lot of small tiles whose only purpose is to declare parameters used by other tiles. In the next chapter, we will see how to remove such tiles, replacing them by definitions declared in a central file.

6 Definitions

Note 1 : In previous Components versions, Definitions was called Instances.

Note 2 : It is now possible to define a Definition into a jsp page (not yet described in tutorial. See `/test/testDefinition.jsp` for an example).

A tile definition is a tile associated with parameters, and identified by a definition name.

Definitions allow :

- Screen definitions
- Centralize declaration of page description
- Avoid repetitive declaration of nearly the same pages (by using definitions inheritance)
- Avoid creation of intermediate components used to pass parameters
- Specify the name of a definition as forward in the struts-config file
- Specify the name of a definition as component parameters
- Overload definition attributes
- Use a different copy of a component, depending on the local (i18n)
- Use a different copy of a component, depending on a key like the channel (see multi-channels)

Definitions are declared in a description file in XML. You can place this file anywhere and call it as you want, but its preferred place is under WEB-INF/componentDefinitions.xml. There is a DTD for this file, located in lib/components-config.dtd. Actually, parser doesn't use the dtd to validate file.

6.1 Declare Definitions

Create a new file, and save it under WEB-INF/componentDefinitions.xml (You can erase any existing file with this name if any).

Write or copy following line of code :

```
<!-- html definition Mappings -->

<component-definitions>

  <!-- Definition description -->
  <definition name="myFirstDefinition" path="/tutorial/layout/classicLayout.jsp">
    <put name="title" value="My First Definition Page" />
    <put name="header" value="/tutorial/common/header.jsp" />
    <put name="footer" value="/tutorial/common/footer.jsp" />
    <put name="menu" value="/tutorial/common/menu.jsp" />
    <put name="body" value="/tutorial/basic/hello.jsp" />
  </definition>

</component-definitions>
```

This declares a definition called myFirstDefinition, using component /layout/classicLayout.jsp.

Definition declaration syntax is nearly the same as including a component with parameters. You must specify a name, and the path of the component that you defined. After that, you can add all parameters that you need. Here, we pass all required parameters, but it is possible to pass only some of them.

6.2 Use Definitions

To use a definition in a web page, you include it with the insert tag.

You will see later that you can also use definitions as component parameters or as forward in the struts-config file.

6.2.1 Insert Definition

Create a new file, and save it under definitionPage.jsp.

Write following lines :

```
<% @ taglib uri="/WEB-INF/tiles.tld" prefix="tiles" %>
```

```
<tiles:insert definition="myFirstDefinition" flush="true" />
```

These lines include specified component definition.

Web server look in its definitions list declaration for requested definition, and then call this definition.

In order to work, the definitions list must be initialized. You will learn how in a few moment.

6.2.2 Override Parameters

You can override any parameters of a definition. For example, override the title parameter as following :

```
<% @ taglib uri="/WEB-INF/tiles.tld" prefix="tiles" %>
```

```
<tiles:insert definition="myFirstDefinition" flush="true" >
  <tiles:put name="title" value="My First Definition With Overloaded Title" />
</tiles:insert>
```

In fact, overriding a parameter as the same syntax as passing parameter. Overriding parameter is often used to specify the body and title used by a new page.

Overriding parameter can also be done in the definition description file.

6.3 Set Web Application Configuration

Before using definitions in a page, you need to load the definitions list in your application. Using a special servlet, extending the Struts `?ction` servlet? can do this.

Edit the WEB-INF/web.xml file, and add following lines :

```
?
<!-- Action Servlet Configuration -->
<servlet>
  <servlet-name>action</servlet-name>
  <servlet-class>s1.struts.component.ActionComponentServlet</servlet-class>
  <init-param>
    <param-name>definitions-config</param-name>
    <param-value>/WEB-INF/componentDefinitions.xml</param-value>
  </init-param>
  <init-param>
    <param-name>definitions-debug</param-name>
    <param-value>1</param-value>
  </init-param>
  <init-param>
    <param-name>config</param-name>
    <param-value>/WEB-INF/action.xml</param-value>
  </init-param>
  <init-param>
    <param-name>debug</param-name>
    <param-value>2</param-value>
  </init-param>
  <init-param>
    <param-name>detail</param-name>
    <param-value>2</param-value>
  </init-param>
  <load-on-startup>2</load-on-startup>
</servlet>
?/span>
```


If you already have a servlet called `ActionServlet` replace it with the one provided. These lines declare a servlet called `StrutsActionServlet`, which is a definition of `org.apache.struts.action.ActionServlet`. This servlet extends `StrutsActionServlet` by adding definitions capabilities. All struts parameters still work. There is additional optional parameters :

- `definitions-config`, which specifies the name of the definitions file. If not specified, `WEB-INF/componentDefinitions.xml` will be used, if exist.
- `definitions-debug`, which specifies debug level. `?` (default) means no debug information will be output, `??` means debug information.

Now you can try your first definition page : restart the web server, and point your browser on `definitionPage.jsp`.

Warning : in the current version (010429), the definition description file is not automatically reloaded. This mean that you have to restart the server each time you change the description file. This will change in future versions.

6.4 Definitions as Component Parameters

A definition name can be used as parameter of a component, as long as this parameter is used to include something.

As an example, we will rewrite our previous portal page entirely by using definitions, with some passed as component parameters.

We first need to declare the portal definition, and then describe the menu and the portal body.

6.4.1 Portal Definition

Definition declaration is done in the `componentDefinitions.xml` file. Declare the portal definition as following :

```
?
<!-- Main Layout Definition description -->
<definition name="mainLayout" path="/tutorial/layout/classicLayout.jsp">
  <put name="title" value="World Financial Online" />
  <put name="header" value="/tutorial/common/header.jsp" />
  <put name="footer" value="/tutorial/common/footer.jsp" />
  <put name="menu" value="menu.main" />
  <put name="body" value="main.portal.body" />
</definition>
?/span>
```

You can note that values of `menu` and `body` attributes are not URLs. They are definition names that we now need to define.

6.4.2 Portal Body Definition

You can write the portal body definition as following :

```
?
<!-- Portal Body declaration-->
<definition name="main.portal.body" path="/tutorial/layout/columnsLayout.jsp">
  <put name="numCols" value="2" />
  <putList name="list0" >
```

```

    <add value="/tutorial/portal/login.jsp" />
    <add value="/tutorial/portal/messages.jsp" />
    <add value="/tutorial/portal/newsFeed.jsp" />
    <add value="/tutorial/portal/advert2.jsp" />
  </putList>
  <putList name="list1" >
    <add value="/tutorial/portal/advert3.jsp" />
    <add value="/tutorial/portal/stocks.jsp" />
    <add value="/tutorial/portal/whatsNew.jsp" />
    <add value="/tutorial/portal/personalLinks.jsp" />
    <add value="/tutorial/portal/search.jsp" />
  </putList>
</definition>
?/span>

```

This declares a definition of columnsLayout, with specified parameters. You can see that the syntax is similar as the declaration in the jsp file.

6.4.3 Menu Definitions

You will now rewrite menu as definitions. Remember : the main menu is made of logo menu, links menu and sources menu. Links menu uses submenu.jsp components, and sources menu uses menuViewSrc.jsp component.

Main Menu

Following is the code of definition :

```

?/span>
<!-- Main menu definition -->
<definition name="menu.main" path="/tutorial/layout/vboxLayout.jsp" >
  <putList name="componentsList" >
    <add value="menu.logo" />
    <add value="menu.links" />
    <add value="menu.src" />
  </putList>
</definition>
?/span>

```

Again, definition declaration is similar to declaration in the jsp file. But now, you don't need the intermediate component anymore.

Menu Logo

Following is the code of definition :

```

?/span>
<!-- menu logo definition -->
<definition name="menu.logo" path="/tutorial/common/menu/menuLogo.jsp" />
?/span>

```

Here, we include the menuLogo.jsp file. Note that there are no parameters. In fact it is possible to specify directly the file to use in the menu.main definition. Another possibility is to define a component taking an image URL as parameter, and showing this image. Try to write it as an example !

Menu Links

Here again, definition code is similar as component code :

```
?/span>
<!-- menu logo definition -->
<definition name="menu.links" path="/tutorial/common/submenu.jsp" >
  <put name="title" value="Main Menu" />
  <putList name="items" >
    <add value="Home" />
    <add value="Basic Page" />
    <add value="First Portal" />
    <add value="First Definition" />
    <add value="Overloaded Definition" />
    <add value="Extended Definition" />
  </putList>
  <putList name="links" >
    <add value="index.jsp" />
    <add value="basicPage.jsp" />
    <add value="portalPage.jsp" />
    <add value="firstDefinition.jsp" />
    <add value="overloadedDefinitionParameters.jsp" />
    <add value="extendedDefinition.jsp" />
  </putList>
</definition>
?/span>
```

If you want to add entry, you need to add entry name, and corresponding URL.

Menu Sources

Following is an abstract of the menu source definition :

```
?/span>
<!-- Menu sources definition -->
<definition name="menu.src" path="/common/menuSrc.jsp" >
  <putList name="list" >
    <add value="index.jsp" />
    <add value="basicPage.jsp" />
    <add value="portalPage.jsp" />
    <add value="firstDefinition.jsp" />
    <add value="overloadedDefinitionParameters.jsp" />
    <add value="extendedDefinition.jsp" />
    <add value="/tutorial/WEB-INF/componentDefinitions.xml" />

    <add value="/tutorial/basic/myLayout.jsp" />

    <add value="/tutorial/layout/classicLayout.jsp" />
    <add value="/tutorial/layout/columnsLayout.jsp" />
    <add value="/tutorial/layout/vboxLayout.jsp" />
  </putList>
```

```
</definition>
  ?/span>
```

You can add entry by adding its URL.

6.4.4 Try Your Definition

You are now ready to try your definition :

Write a JSP page including it, restart the server, and point your browser on the page.

Create a new file and save it under index.jsp :

```
<%@ taglib uri="/WEB-INF/tiles.tld" prefix="tiles" %>
```

```
<tiles:insert definition="mainLayout" flush="true" />
```

6.5 Extended Definitions

You can do inheritance with definitions. This means that a definition can extend another definition, inheriting all attributes defined in the parent. Of course, child can overload any of the attributes.

As an example, you will extend the mainLayout definition, changing the body and the title. As body, you will use one of the portal components.

Your definition declaration looks like following :

```
  ?/span>
  <!-- Extended definition example -->
  <definition name="extended.definition.example" extends="mainLayout" >
    <put name="title" value="Extended Definition" />
    <put name="body" value="/tutorial/portal/newsFeed.jsp" />
  </definition>
  ?/span>
```

You declare a definition, specifying its name, and the name of the definition that it extends. For that, you use `extends=?efinitionName?`

To try your extended definition, you need to create a web page including the definition. Create a new file and save it under `extendedDefinition.jsp` :

```
<%@ taglib uri="/WEB-INF/tiles.tld" prefix="tiles" %>
```

```
<tiles:insert definition="extended.definition.example" flush="true" />
```

6.6 Definitions as Forward

Definitions can be used as forward targets in the `struts-config.xml` file.

As an example, you will write a page with two buttons : success and failure. A form, associated to a Struts action, surrounds these buttons. The Struts action forwards to a logical name, success or failure.

6.6.1 Form page

You first need to write the form page. In fact, you will write the body, declare a definition, and write a page including this definition.

Body

Create a new file and save it under `forward/forwardBody.jsp` :

```
<form action="forwardExampleAction.do">
```

Select an action :

```
<br><input type="submit" name="success" value="success">
```

```
<br><input type="submit" name="failure" value="failure">
</form>
```

This file contains a form with two buttons. You can use Struts tag form instead.

Definition

Declare a new definition including this body :

```
?/span>
<!-- Struts forward definition example -->
<definition name="forward.example.choice.page" extends="mainLayout" >
  <put name="title" value="Struts Forward Test" />
  <put name="body" value="/tutorial/forward/forwardBody.jsp" />
</definition>
?/span>
```

Page

Write a page including this definition.

Create a new file and save it under /strutsForward.jsp :

```
<%@ taglib uri="/WEB-INF/tiles.tld" prefix="tiles" %>
```

```
<tiles:insert definition="forward.example.choice.page" flush="true" />
```

You can try the page, but you still need to write the Struts action before submitting !

6.6.2 Struts Action

Struts action is a java class. Following is an abstract of the code of this class :

```
public class ForwardExampleAction extends Action {
```

```
    public ActionForward perform(
        ActionMapping mapping,
        ActionForm form,
        HttpServletRequest request,
        HttpServletResponse response)
        throws IOException, ServletException
    {
        String success = request.getParameter( "success" );
        if( success != null )
            return (mapping.findForward("success"));

        return (mapping.findForward("failure"));
    }
}
```

You can find complete code in documentation. For now, you have nothing particular to do, because compile class is included in components.jar.

6.6.3 Struts Configuration File

You need to setup the Struts configuration file, in order to register the new action.

Add something like following in the action.xml or struts-config.xml :

```
<!-- Global Forward Declarations -->
```

```

<forward name="failure" path="forward.example.failure.page"/>

<!-- struts forward example -->
<action path="/forwardExampleAction"
  actionClass="s1.struts.example.tutorial.ForwardExampleAction">
  <forward name="success" path="forward.example.success.page"/>
</action>
?/span>

```

If you use the struts-config.xml file, you need to put the global forward between <global-forwards> and </global-forwards>.

You can note that the forwards?paths specified definitions, rather than URLs. Now you need to declare these two definitions.

6.6.4 Definition Declarations

Declare the success and failure definitions :

```

<!-- Struts forward definition example -->
<definition name="forward.example.success.page" extends="mainLayout" >
  <put name="title" value="Struts Forward to 'success'" />
  <put name="body" value="/tutorial/forward/successBody.jsp" />
</definition>

```

```

<!-- Struts forward definition example -->
<definition name="forward.example.failure.page" extends="mainLayout" >
  <put name="title" value="Struts Forward to 'failure'" />
  <put name="body" value="/tutorial/forward/failureBody.jsp" />
</definition>
?/span>

```

Write the corresponding bodies.

Create a new file and save it under /forward/successBody.jsp :

```
<% @ taglib uri="/WEB-INF/tiles.tld" prefix="tiles" %>
```

Struts forward to ?uccess?

```
<br>
<tiles:insert page="/forward/forwardBody.jsp" flush="true" />
```

Create a new file and save it under /forward/failureBody.jsp :

```
<% @ taglib uri="/WEB-INF/tiles.tld" prefix="tiles" %>
```

Struts forward to 'failure'.

```
<br>
<tiles:insert page="/forward/forwardBody.jsp" flush="true" />
```

Note that you don? need to write pages including the success or failure definitions. Inclusion is done directly by the value returned by Struts action.

6.6.5 Try Your Page

You can now try the new page. Restart the web server to reload definitions. Point your browser on /strutsForward.jsp, and click on one button.

7 Write More Complex Components

It's now time to write more complex components. You have already used such components : classicLayout, submenu, viewSrc, ?/span>

Such components are intelligent components containing some logic. You can put logic inside a component, as long as it is UI logic. You must avoid some business logic inside a component.

7.1 Submenu Component

Covered topics : dynamic component, passing list as parameters, using list in components, using conditional tags.

Submenu takes as parameters : a list of items, a list of links corresponding to items, a title and value of selected item. It shows the title, followed by items.

When you click on an item, you follow corresponding link. If selected value correspond to one item, this item is disable (no link).

Create a new file and save it under common/submenu.jsp.

7.1.1 Iterate on List

First, you will show the list of items. Create a table with two rows. In the first row, write the title parameter. Around the second row, place ?iterate?tags, and write the iteration value in the row.

Code is as following :

```
<%@ taglib uri="/WEB-INF/struts.tld" prefix="struts" %>
<%@ taglib uri="/WEB-INF/struts-logic.tld" prefix="logic" %>
<%@ taglib uri="/WEB-INF/struts-bean.tld" prefix="bean" %>
<%@ taglib uri="/WEB-INF/tiles.tld" prefix="tiles" %>
<%@ page import="java.util.Iterator" %>

<!-- Push component attributes in page context -->
<tiles:importAttribute />

<table>
<tr bgcolor="blue">
<td width="120" valign="top" colspan="2">
<bean:write name="title"/>
</td>
</tr>

<!-- Prepare the links list to be iterated -->
<bean:define id="links" name="links" type="java.util.List" scope="page" />
<% Iterator i = links.iterator(); %>

<!-- iterate on items list -->
<logic:iterate id="item" name="items" type="String" >
<tr>
```

```

<td width="10" valign="top" ></td>
<td valign="top" >
  <font size="-1">
    <a href="<%=request.getContextPath()%>/<%=i.next()%>"><%=item%></a>
  </font>
</td>
</tr>
</logic:iterate>

```

```
</table>
```

You start by importing components attributes (items, links, title, selected) to the page context. This allows you to use such attributes.

Then, you show the title, and you iterate on each items. As you can iterate only on one list with the iteration tag, you need to iterate yourself on second list. It is why you declare an `?erator i`?

7.1.2 Conditional code

One specification is to disable link on selected item.

You need to check if an item is equal to `?elected?value`, and write appropriate value.

Your code become :

```

?/span>
<!-- iterate on items list -->
<logic:iterate id="item" name="items" type="String" >
<tr>
  <td width="10" valign="top" ></td>
  <td valign="top" >
    <!-- check if selected -->
    <logic:notEqual name="selected" value="<%=item%>">
      <font size="-1">
        <a
href="<%=request.getContextPath()%>/<%=i.next()%>"><%=item%></a></font>
      </logic:notEqual>
      <logic:equal name="selected" value="<%=item%>">
        <font size="-1" color="fuchsia"><%=item%></font>
      </logic:equal>
    </td>
  </tr>
</logic:iterate>
?/span>

```

7.1.3 Check Parameters Presence

The specification says that title and selected are optional. So, you need to check their presence. For that, you use Struts tags `?resent?and` and `?otPresent?`

Check for title presence, and show it only if present.

Check for selected presence, and declare a dummy selected variable if not.

Your code must be changed to :

```
    ?/span>
</table>
<logic:present name="title">
<tr bgcolor="blue">
  <td width="120" valign="top" colspan="2">
    <bean:write name="title"/>
  </td>
</tr>
</logic:present>

<logic:notPresent name="selected" >
  <% pageContext.setAttribute( "selected", "" ); %>
</logic:notPresent>
?/span>
```

7.2 View Sources Component

[to do 001115]

You can check code in /common/menuViewSrc.jsp. Nearly the same as submenu.

7.3 Body Including Twice (or more) the Same Tile

Covered topics : use of sub-Tiles in a Tile.

In this chapter you will learn how to write Tiles that can be reused more than one time in a page. As example, we develop an ?address?Tile, included two times in an ?invoice?

Problems come when you have input fields in the address tile : how to have two different names for input while using the same tile description ? As a solution, you will prefix input field names by a prefix pass as parameter to the Tile. Second difficulty is how to retrieve data to be shown in the address Tile ? Here, you will pass Java bean (Java object), and deal with it in the address component. Data to be edited must be accessible from the bean using the prefix and the property name/

All Java classes are already written and compiled for you. You can check sources from the Tile Library distribution, under directory src/org/apache/struts/example/invoice.

7.3.1 Address Tile

Create a new file and save it under invoice/editAddress.jsp.

In this file, create a table with two columns : one containing fields names, one containing html input fields. Use Struts tags ?ext?for the input fields.

Following is the code for this tile :

```
<%@ taglib uri="/WEB-INF/tiles.tld" prefix="tiles" %>
<%@ taglib uri="/WEB-INF/struts-html.tld" prefix="html" %>
<%@ taglib uri="/WEB-INF/struts-bean.tld" prefix="bean" %>
```

```

<%-- Edit an Address object
  @param bean An address object to edit.
  @param beanName The path to add between the bean and the properties to edit.
--%>
<%-- Retrieve parameters from component context, and declare them as page variable --
%>
<files:useAttribute id="beanName" name="property" classname="java.lang.String"
ignore="true" />
<files:importAttribute name="bean" />

<%-- Add a '.' separator to the path (beanName), in order to access the property from the
given bean --%>
<% if( beanName == null ) beanName = ""; else if (beanName != "" ) beanName =
beanName + "."; %>

<table border="0" width="100%">

  <tr>
    <th align="right" width="30%">
      Street
    </th>
    <td align="left">
      <%-- Declare an html input field. --
%>
      <%-- We use the bean passed as parameter. --
%>
      <%-- Property name is prefixed by the sub-bean name if any. --%>

      <html:text name="bean" property='<%=beanName+"street1"%>' size="50"/>

    </td>
  </tr>

  <tr>
    <th align="right">
      Street (con't)
    </th>
    <td align="left">
      <html:text property='<%=beanName+"street2"%>' size="50"/>
    </td>
  </tr>

  <tr>
    <th align="right">
      City

```

```

</th>
<td align="left">
  <html:text name="bean" property='<%=beanName+"city"%>' size="50"/>
</td>
</tr>

<tr>
<th align="right">
  Country
</th>
<td align="left">
  <html:text property='<%=beanName+"country"%>' size="50"/>
</td>
</tr>

<tr>
<th align="right">
  Zip code
</th>
<td align="left">
  <html:text property='<%=beanName+"zipCode"%>' size="50"/>
</td>
</tr>

</table>

```

This Tile takes two parameters :

- **beanName** : the path from the bean to the properties to edit. This path is the name(s) of sub-bean(s) containing properties.
- **bean** : the root bean exposing data to edit

First, we retrieve parameters, then we compute the prefix if any.

Names of generated html input tags will be ?eanName.fieldname? This will allow to retrieve value in the controller.

7.3.2 Invoice Tile

Create a new file and save it under invoice/editInvoice.jsp.

This invoice contains two times the address tile, and two html input fields :

```

<% @ page language="java" %>
<% @ taglib uri="/WEB-INF/tiles.tld" prefix="tiles" %>
<% @ taglib uri="/WEB-INF/struts-html.tld" prefix="html" %>
<% @ taglib uri="/WEB-INF/struts-bean.tld" prefix="bean" %>

<html:errors/>

<html:form action="/tutorial/invoice/editInvoice.do" >

<font size="+1">Edit Customer Informations</font>

```

```

<table border="0" width="100%">

<tr>
  <th align="right" width="30%">
    First Name
  </th>
  <td align="left">
    <html:text property="firstname" size="50"/>
  </td>
</tr>

<tr>
  <th align="right">
    Last Name
  </th>
  <td align="left">
    <html:text property="lastname" size="50"/>
  </td>
</tr>

<tr>
  <th align="right" >
    Shipping Address
  </th>
  <td align="left">&nbsp;  </td>
</tr>
<tr>
  <td align="center" colspan="2">
    <!-- Include an "address editor" component.          --%>
    <!-- Pass the component name and component value as parameter --%>
    <!-- Value comes from the form bean --%>
    <tiles:insert page="/tutorial/invoice/editAddress2.jsp" >
      <tiles:put name="property" value="shippingAddress" />
      <tiles:put name="bean" beanName="invoiceForm" />
    </tiles:insert>
  </td>
</tr>

<tr>
  <th align="right" >
    Billing Address
  </th>
  <td align="left">

```

```

        &nbsp;
    </td>
</tr>
<tr>
    <td align="center" colspan="2">
        <tiles:insert page="/tutorial/invoice/editAddress2.jsp" >
            <tiles:put name="property" value="billAddress" />
            <tiles:put name="bean" beanName="invoiceForm" />
        </tiles:insert>
    </td>
</tr>

<tr>
    <td align="right">
        <html:submit>
            save
        </html:submit>
        <html:submit>
            confirm
        </html:submit>
    </td>
    <td align="left">
        <html:reset>
            reset
        </html:reset>
        &nbsp;
        <html:cancel>
            cancel
        </html:cancel>
    </td>
</tr>
</table>

```

```
</html:form>
```

You insert an address Tile where you want it to reside. You pass it its name, and the Java bean containing values. This bean comes from ?nvoiceForm?object, and is retrieved using getter method. Don? forget to provide two different name for each Tile inserted !

7.3.3 Try Your Page

You can use your edit invoice form in a page using main layout.

Create a new file and save it under invoice/index.jsp.

```

<%@ taglib uri="/WEB-INF/tiles.tld" prefix="tiles" %>

<tiles:insert definition="mainLayout" flush="true">
    <tiles:put name="body" value="/tutorial/invoice/editInvoice.jsp" />

```

</tiles:insert>

7.3.4 Variant

If you don't like to use scriptlets (<%= ?>) inside your tags, you can extend Struts tags to add a 'prefix' attribute. This solution was used in an older example accessible in the invoice directory (editAddress2.jsp and editInvoice2.jsp). As example, we provide an extended version of tag text, available in library "extensions.tld".

8 Internationalization (i18n)

Covered topics : use of i18n description files.

Components Library allows having different copies of a component, each one suited for a language or locale. When the component is included, the appropriate copy is chosen.

Now come the problems : where to put copies ? There are two main approaches :

- Put them in the same directory that the component, and add language code as suffix (_fr, _en, ?).
- Put them in a separate directory, named with the language code (/fr/, /en/, ?).

Each approach has advantage and drawbacks, depending of your application. So we have chosen to let yourself determine where to place copies.

The Component i18n mechanism let you write different definitions file, one for each language. Choice of the appropriate definition file is done following the same rules as the Java properties bundle, adding language suffix to file name. You must write a default definition file, and you can write copies of this file, each copy name suffixed by the language code.

In a copy, you don't need to rewrite all definition descriptions : you just rewrite definitions that differ. All copies automatically inherit from the default definition file.

This i18n mechanism is intended to be a complement of the key-properties bundle provided by Struts. You should use it when you have big components to internationalize. If you have only small sentences or words, use the key mechanism. Both mechanisms can be used in one site.

Let see how all this work on an example : you will add a new menu giving choice between several languages. When you change language, the menu change, displaying only the other possible languages. Titles of some pages also change. This is not a real i18n example : it just illustrates how we can do.

You will start by writing the new menu, and add it to the main menu. This menu is linked to a Struts action switching the user Locale object. Then, you will write copies of the definitions file.

8.1 *Select Language Menu*

This menu shows different available languages. You will write it has a definition using a submenu component :

```
<!-- -->
<definition name="menu.lang" path="/common/submenu.jsp" >
  <put name="title" value="Languge" />
  <putList name="items" >
    <add value="Francais" />
    <add value="English" />
    <add value="Deutch" />
  </putList>
  <putList name="links" >
    <add value="lang.do?language=FR" />
    <add value="lang.do?language=UK" />
    <add value="lang.do?language=DE" />
  </putList>
</definition>
```

You specify the item names, and the corresponding URLs. Here, items are linked to a Struts action switching the current language.

Don't forget to add the new sub-menu to the main menu definition :

```
<!-- Main menu definition -->
<definition name="menu.main" path="/layout/vboxLayout.jsp" >
  <putList name="componentsList" >
    <add value="menu.logo" />
    <add value="menu.links" />
    <add value="menu.src" />
    <add value="menu.lang" />
  </putList>
</definition>
```

8.2 *Select Language Action*

The action Java code is as follow :

```
public final class SelectLocaleAction extends Action {

    public ActionForward perform(
        ActionMapping mapping,
        ActionForm form,
        HttpServletRequest request,
        HttpServletResponse response)
        throws IOException, ServletException {

        // Extract parameters we will need
        String requested = (String)request.getParameter( "language" );

        if( requested == null )
```

```

        return (mapping.findForward("failed"));
    if( requested.equalsIgnoreCase( "FR" ) )
        setLocale( request, Locale.FRANCE );
    if( requested.equalsIgnoreCase( "UK" ) )
        setLocale( request, Locale.UK );
    if( requested.equalsIgnoreCase( "DE" ) )
        setLocale( request, Locale.GERMAN );

    return (mapping.findForward("success"));
}

protected void setLocale( HttpServletRequest request, Locale locale )
{
    HttpSession session = request.getSession(false);
    if (session != null)
        session.setAttribute(Action.LOCALE_KEY, locale);
}
}

```

It could certainly be improved, but it is not the purpose of this tutorial. It simply checks requested language, and set the user locale attribute appropriately (under the name defined by Action.LOCALE_KEY).

You can now try your pages, but nothing new happens because you need to define localized copies of definitions description file.

8.3 Localized Definition Descriptions Files

You can write a definitions file for each of your languages. Remember that you must have a default file (with no language code suffix).

It is better to start from a new empty file, as you don't need to rewrite all definitions.

Create a new file and save it under WEB-INF/componentDefinitions_fr.xml.

Copy the menu.lang definition from the default file. Translate title value and erase the ?rench? items and its link.

Also copy the mainLayout definition from the default file. Translate title, and change footer value to ?fr/common/footer.jsp?/span>. Also write such file.

Your definition description file should look like :

```

<!-- html definition Mappings -->
<component-definitions>

    <!-- Main Layout Definition description -->
    <definition name="mainLayout" path="/layout/classicLayout.jsp">
        <put name="title" value="Le Monde Financier En Ligne" />
        <put name="header" value="/tutorial/common/header.jsp" />
        <put name="footer" value="/tutorial/fr/common/footer.jsp" />
        <put name="menu" value="menu.main" />
        <put name="body" value="main.portal.body" />
    </definition>

```



```
<!-- select language menu -->
<definition name="menu.lang" path="/common/submenu.jsp" >
  <put name="title" value="Languge" />
  <putList name="items" >
    <add value="English" />
    <add value="Deutch" />
  </putList>
  <putList name="links" >
    <add value="lang.do?language=UK" />
    <add value="lang.do?language=DE" />
  </putList>
</definition>
```

```
</component-definitions>
```

Do the same for others language.

8.4 Try Your Pages

You can now try your i18n pages. Point your browser on the index.jsp page, and select another language. Watch the window title, and the footer.

In the menu, select a page extending the mainLayout definition, like `?i>Extended Definition?` Check the footer : it is the one defined by mainLayout definition in the localized description file because shown definition extends mainLayout.

9 Multi-Channels

The same mechanism than i18n can be used for channels (wml, different browser support, ?). Each channel components are written under a different directory.

Work in progress. Example available in comps-channel.war.

The idea for channel system is to write another definition factory that will choose the component to insert according to the provided definition name, and a channel key extracted from wherever you want (value from session, , ?). Actually, factory implementation is clearly separated from the rest. If you are interested, you can write your own factory. It needs to implement `ComponentDefinitionsFactory.java` (1 method). You can let `ComponentActionServlet` create your factory by setting the servlet init parameter `definitions-factory-class` to the name of your factory. You can also define init parameters for your factory. They will be passed in a `Map`. Again, check example in `comps-channel.war`.

Appendix D: Struts Validation

Used by permission of the author David Winterfeldt.

Copyright © 2001 by David Winterfeldt and the Apache Software Foundation.

Overview

The Validation Framework was made to work with [Struts](#), but can be used for validation of any java bean. It can perform basic validations to check if a field is required, matches a regular expression, email, credit card, and server side type checking and date validation. Different validation rules can be defined for different locales. The framework has basic support for user defined constants which can be used in some field attributes. The validation routines are modifiable in the validation.xml file so custom validation routines can be created and added to the framework.

Setup

In Struts once you have defined the ValidatorServlet in the web.xml so it can load your ValidatorResources you just have to extend `com.wintecinc.struts.action.ValidatorForm` instead of `org.apache.struts.action.ActionForm`. Then when the validate method is called the action's name attribute from the struts-config.xml is used to load the validations for the current form. So the form element's name attribute in the validation.xml should match action element's name attribute.

Another alternative is to use the action mapping you are currently on by extending the `ValidatorActionForm` instead of the `ValidatorForm`. The `ValidatorActionForm` uses the action element's 'path' attribute from the `struts-config.xml` which should match the form element's name attribute in the `validation.xml`. Then a separate action can be defined for each page in a multi-page form and the validation rules can be associated with the action and not a page number as in the example of a multi-page form in the validator example.

Internationalization

Validation rules for forms can be grouped under a `FormSet` in the `validation.xml` file. The `FormSet` has language, country, and variant attributes that correspond with the `java.util.Locale` class. If they are not used, the `FormSet` will be set to the default locale. A `FormSet` can also have constants associated with it. On the same level as a `FormSet` there can be a global element which can also have constants and have validator actions that perform validations.

The default error message for a pluggable validator can be overridden with the `msg` element. So instead of using the `msg` attribute for the mask validator to generate the error message the `msg` attribute from the field will be used if the name of the field's name attribute matches the validator's name attribute.

```
ex: <field property="lastName"
      depends="required,mask">
  <msg name="mask" key="registrationForm.lastname.maskmsg"/>
  <arg0 key="registrationForm.lastname.displayname"/>
  <var>
    <var-name>mask</var-name>
    <var-value>^[a-zA-Z]*$</var-value>
  </var>
</field>
```

The arguments for error messages can be set with the arg0-arg3 elements. If the arg0-arg3 elements' name attribute isn't set, it will become the default arg value for the different error messages constructed. If the name attribute is set, you can specify the argument for a specific pluggable validator and then this will be used for constructing the error message.

```
ex: <field property="lastName"
      depends="required,mask">
  <msg name="mask" key="registrationForm.lastname.maskmsg"/>
  <arg0 key="registrationForm.lastname.displayname"/>
  <var>
    <var-name>mask</var-name>
    <var-value>^[a-zA-Z]*$</var-value>
  </var>
</field>
```

By default the arg0-arg3 elements will try to look up the key attribute in the message resources. If the resource attribute is set to false, it will pass in the value directly without retrieving the value from the message resources.

```
ex: <field property="integer"
      depends="required,integer,range">
  <arg0 key="typeForm.integer.displayname"/>
  <arg1 name="range" key="{var:min}" resource="false"/>
  <arg2 name="range" key="{var:max}" resource="false"/>
  <var>
    <var-name>min</var-name>
    <var-value>10</var-value>
  </var>
  <var>
    <var-name>max</var-name>
    <var-value>20</var-value>
  </var>
</field>
```

Constants/Variables

Global constants can be inside the global tags and FormSet/Locale constants can be created in the formset tags. Constants are currently only replaced in the Field's property attribute, the Field's var element value attribute, the Field's msg element key attribute, and Field's arg0-arg3 element's key attribute. A Field's variables can also be substituted in the arg0-arg3 elements (ex: `${var:min}`). The order of replacement is FormSet/Locale constants are replaced first, Global constants second, and for the arg elements variables are replaced last.

```
ex: <global>
    <constant name="zip" value="^\d{5}(-\d{4})?$" />
</global>

<field property="zip"
    depends="required,mask">
    <arg0 key="registrationForm.zippostal.displayname"/>
    <var>
        <var-name>mask</var-name>
        <var-value>${zip}</var-value>
    </var>
</field>
```

The var element under a field can be used to store variables for use by a pluggable validator. These variables are available through the Field's `getVar(String key)` method.

```
ex:
<field property="integer"
    depends="required,integer,range">
    <arg0 key="typeForm.integer.displayname"/>
    <arg1 name="range" key="${var:min}" resource="false"/>
    <arg2 name="range" key="${var:max}" resource="false"/>
    <var>
        <var-name>min</var-name>
        <var-value>10</var-value>
    </var>
```

```
<var>
  <var-name>max</var-name>
  <var-value>20</var-value>
</var>
</field>
```

See type form's integer field in the example web app for a working example.

Pluggable Validators

Validation actions are read from the validation.xml file. The default actions are setup in the validation.xml file. The ones currently configured are required, mask ,byte, short, int, long, float, double, date (without locale support), and a numeric range. The 'mask' action depends on required in the default setup. That means that 'required' has to successfully completed before 'mask' will run. The 'required' and 'mask' action are partially built into the framework. Any field that isn't 'required' will skip other actions if the field is null or has a length of zero. If the Javascript Validator JSP Tag is used, the client side Javascript generation looks for a value in the validator's javascript attribute and generates an object that the supplied method can use to validate the form. For a more detailed explanation of how the Javascript Validator Tag works, see the [JSP Tags](#) section.

The 'mask' action let's you validate a regular expression mask to the field. It uses the [Regular Expression Package](#) from the [jakarta site](#). All validation rules are stored in the validation.xml file. The main class used is [org.apache.regexp.RE](#).

Example Validator Configuration from validation.xml.

```
<validator name="required"
  classname="com.wintecinc.struts.validation.Validator"
  method="validateRequired"
  msg="errors.required"/>
```

```
<validator name="mask"
  classname="com.wintecinc.struts.validation.Validator"
  method="validateMask"
  depends="required"
  msg="errors.invalid"/>
```

Creating Pluggable Validators

The ValidatorAction method needs to have the following signature. See the com.wintecinc.struts.validation.StrutsValidator class for examples.

```
(java.lang.Object,
 com.wintecinc.struts.validation.ValidatorAction,
 com.wintecinc.struts.validation.Field,
 org.apache.struts.action.ActionErrors, ,
 javax.servlet.http.HttpServletRequest, javax.servlet.ServletContext)
```

- java.lang.Object – Bean validation is being performed on.
- com.wintecinc.struts.validation.ValidatorAction – The current Validator Action being performed.
- com.wintecinc.struts.validation.Field – Field object being validated.
- org.apache.struts.action.ActionErrors – The errors object to an ActionError to it the validation fails.
- javax.servlet.http.HttpServletRequest – Current request object.
- javax.servlet.ServletContext – The application's ServletContext.

Multi Page Forms

The field element has an optional page attribute. It can be set to an integer. All validation for the any field page value less than or equal to the current page is performed server side. All validation for the any field page equal to the current page is generated for the client side

Javascript. A multi-part form expects the page attribute to be set.

ex: `<html:hidden property="page" value="1"/>`

Comparing Two Fields

This is an example of how you could compare two fields to see if they have the same value. A good example of this is when you are validating a user changing their password and there is the main password field and a confirmation field.

```
<validator name="twofields"
  classname="com.mysite.StrutsValidator"
  method="validateTwoFields"
  msg="errors.twofields"/>

<field property="password"
  depends="required,twofields">
  <arg0 key="typeForm.password.displayName"/>
  <var>
    <var-name>secondProperty</var-name>
    <var-value>password2</var-value>
  </var>
</field>
```

```
public static boolean validateTwoFields(Object bean,
  ValidatorAction va, Field field,
  ActionErrors errors,
  HttpServletRequest request, ServletContext application) {

  String value = ValidatorUtil.getValueAsString(bean,
  field.getProperty());
  String sProperty2 = field.getVarValue("secondProperty");
  String value2 = ValidatorUtil.getValueAsString(bean, sProperty2);

  if (!GenericValidator.isBlankOrNull(value)) {
```

```

    try {
        if (!value.equals(value2)) {
            errors.add(field.getKey(),
ValidatorUtil.getActionError(application, request, va, field));

            return false;
        }
    } catch (Exception e) {
        errors.add(field.getKey(),
ValidatorUtil.getActionError(application, request, va, field));
        return false;
    }
}

return true;
}

```

Validating Outside of Struts

Here is a short example of validating something outside of the Struts Framework. The validator element's `methodParams` attribute have a default method signature that all the `StrutsValidator` validation methods use.

```

<form-validation>
  <global>
    <validator name="capLetter"
      classname="com.mysite.Validator"
      methodParams="java.lang.Object,com.wintecinc.struts.validati
on.Field,java.util.List"
      method="isCapLetter"
      msg="Letter is not in upper case."/>
  </global>
</formset>
<formset>
  <form name="testForm">

```

```

    <field property="letter"
          depends="capLetter">
    </field>
</form>
</formset>
</form-validation>

```

The method signature and parameters are dynamically created based on the methodParams and the resources added to the Validator using the class name as a key. The class "java.lang.Object" is reserved for the bean that is being validated and there can't be any duplicate class names because they are used as the key when associating the actual instance of the class when setting up the Validator.

The ValidatorAction and the Field are automatically passed in if specified in the methodParams attribute. The other instances of classes need to be added to the Validator using the addResource method along with the class they represent from the validator element's methodParams attribute. Bean is the object being validated. The ValidatorResourcesInitializer can be used to load the validation.xml file and return an instance of ValidatorResources based on the xml file. So based on the validation.xml file defined above the getLetter method will be called on the bean variable.

```

ValidatorResources resources =
ValidatorResourcesInitializer.initialize("validation.xml", debug);
Validator validator = new Validator(resources, "testForm");
validator.addResource(Validator.BEAN_KEY, bean);
validator.addResource("java.util.List", lErrors);

try {
    validator.validate();
} catch (ValidatorException e) {
    // Log Exception
}

```

This is the validation method being used by the capLetter validator. The validation fails if the value retrieved is null, has a length other than one, and the character doesn't fall in the range A-Z. Error messages are added to the java.util.List that is passed into the method.

```
public static boolean isCapLetter(Object bean, Field field, List l) {
    String value = ValidatorUtil.getValueAsString(bean,
field.getProperty());

    if (value != null && value.length() == 1) {
        if (value.charAt(0) >= 'A' && value.charAt(0) <= 'Z') {
            return true;
        } else {
            l.add(field.getMsg);
            return false;
        }
    } else {
        l.add(field.getMsg);
        return false;
    }
}
```

Appendix E: WebLogic Deployment

Specific steps to deploy applications on WebLogic may be downloaded from the BaseBeans Engineering web site (www.basebeans.net).

Appendix F: Why not use ... ?

In the final analysis, you can use whatever application server, IDE, text editor, RDBMS, and/or connection pool that you want.

What not use ... EJB's?

There are those who say EJB's are slow in production. I have had some of those experiences. Therefore, the best use of EJB is for smaller applications with a low volume, or applications that required a truly distributed environment. Sun has pushed EJBs in the past, but, as many know, Sun also pushed for client-side Java.

If you have a low-volume, or a distributed, application and don't mind technical complexity, each module, or the entire application, should use EJB's.

If you are already using EJB's, one way to speed them up is to use an astral clones design pattern, basically a cache of rows. The SpaBaseBean could extend the AstralBean and then the AstralBean can make EJB calls a bit more effectively. Sooner or later, you will have to refactor to use Java or Form Beans, such as the "SpaBaseBean".

For distributed processing, SOAP will be the dominant architecture, and not EJB. If you want to use functions from a distributed servers in your application, use SOAP. In the next edition I will have a chapter on leveraging SOAP.

Why not use ... MySQL?

MySQL has a significant performance run-time cost on Windows and is generally proven to be slow under load. In addition, the

current version, as of this writing, is not ANSI SQL-compliant.

Why not use ... MVC Frameworks other than Struts?

Struts is the only production-proven JSP framework that I am aware of and it is going to be a dominant architecture. If you leverage the Struts framework with `CachedRowSet`'s, XML/XSLT, and content syndication, there is nothing that is comparable. The source code is free and developers with Struts expertise will be, and actually are, more marketable.

Vic Cekvenich is a certified instructor for several vendors in technologies such as Java, SQL, object-orientation, and performance tuning. He has over 14 years of software development experience working on large-scale projects and applications. He works for BaseBeans Engineering (www.basebeans.com) where he does mentoring, training, architecture, and project recovery for their clients.

You may contact him at "vic@basebeans.com".



Coast of Dalmacija:



Struts Fast Track: J2EE / JSP Framework teaches good practices in developing large base Struts web applications with MVC partitioning, database access, security, and content syndication.

Java 2, Enterprise Edition, is a Java standard and is similar in principle to the ANSI SQL standard. The ANSI SQL standard ensures that you have a choice of database vendors. In the same way, if you develop a web application that is J2EE-compliant you will have a choice of application server vendors and products, both open source and commercial. Sun has defined a standard for good designs and practices for J2EE called Blueprints.

The main point of Blueprints is that your application should be modular and written in three tiers called Model-View-Control (MVC), or Model 2. This requires that in your JSP presentation you do not write any SQL or application code. This also requires that all the data access, such as SQL, is in the data layer. MVC is fast becoming the standard for software design.

Struts was developed by the same people who developed Tomcat. Tomcat is the reference J2EE container for servlets and JSPs. This guarantees that applications developed leveraging the Struts framework will run on any J2EE server, such as WebLogic, WebSphere, iPlanet, JRun, and Orion Server.

Companies are now looking to implement standard and inexpensive cross-platform web applications and **Struts Fast Track: J2EE / JSP Framework** will cover all the necessary topics to prepare developers to build these standard, Struts-based web applications.